Verisk™

Sequel Security

Version: 3.129.23132.1

12 May 2023

Table of Contents

1. Sequel Security Services 6
   1.1 Features 7
      1.1.1 Authentication as a Service 7
      1.1.2 Single Sign-on / Sign-out 7
      1.1.3 Access Control for APIs 7
      1.1.4 Federation Gateway 7
      1.1.5 Single master domain 7
      1.1.6 Role-based access control 7
      1.1.7 Cross-domain identity management 7
      1.1.8 Delegated entities and user management 7
      1.1.9 Import and Export tools and APIs 7
   1.2 Why 8
   1.3 About how this documentation is organized 8
2. Architecture 9
   2.1 Architecture 9
      2.1.1 Sequel Security Service 9
      2.1.2 Sequel.Security.Integration NuGet package 11
      2.1.3 User web component 11
      2.1.4 Legacy Security Sync Services 11
      2.1.5 Service bus integration 12
3. Developers handbook 13
   3.1 Overview 13
   3.2 Developer's guide 14
      3.2.1 .NET Core version 14
      3.2.2 Use of a Security FVM for local development 14
   3.3 UI Style guide 16
      3.3.1 Style guide 16
      3.3.2 Prototypes 16
   3.4 API best practices 17
      3.4.1 API design 17
      3.4.2 API documentation 17
   3.5 Message Bus 19
      3.5.1 Overview 19
      3.5.2 1. Message contracts 20
      3.5.3 2. Publisher 23
      3.5.4 3. Consumer for internal Security Services 24
   3.6 Sequel Security Tool for developers 26
      3.6.1 Installation 26
      3.6.2 Deployment Manager token replacement 26

3.6.3 Functionality 27
3.7 Integration 28
3.7.1 Starting with security 28
3.7.2 Integration guide 29
3.7.3 Migration guide 48
3.7.4 Migration guide: database 54
3.7.5 Single Sign-out 57
3.7.6 Single Session Management 59
3.7.7 User Session Avatar 61
3.7.8 How to request a token from a client application 65
3.7.9 Reset password 66
3.8 OpenIdConnect 68
3.8.1 Client settings 68
3.8.2 Grant Types 75
3.9 Performance 76
3.9.1 Load Tests 76
3.9.2 Database Performance 80
3.9.3 Logging Performance 89
3.9.4 Setup Application Request Routing (ARR) Environment 98
4. Operations handbook 99
4.1 Overview 99
4.1.1 Release notes 99
4.2 Platform specs 100
4.2.1 Architecture overview 100
4.2.2 Database Server 102
4.2.3 Application Server 103
4.3 Installation guide 106
4.3.1 Pre-Requisite Information 106
4.3.2 Installation Information 106
4.3.3 Deployment Manager 112
4.4 Database installation 117
4.4.1 Products 117
4.4.2 Global settings 117
4.4.3 Modules 118
4.4.4 Appendix 119
4.5 Security Apps Installation 121
4.5.1 Products 121
4.5.2 Global settings 121
4.5.3 Modules 123
4.5.4 Appendix 133
4.6 Synchronization Services Installation 134
4.6.1 Products 134

4.6.2 Global settings 134
4.6.3 Modules 135
4.6.4 Appendix 142
4.7 Housekeeping & maintenance 143
4.7.1 Introduction 143
4.7.2 General housekeeping 143
4.7.3 Database maintenance 146
4.8 Advanced 149
4.8.1 Resilience in Security 149
4.8.2 Docker 152
4.8.3 Paas 164
4.8.4 Saas 180
4.8.5 Scale out 194
4.9 Registration forms 196
4.9.1 LDAP Sync Registration 196
4.9.2 Azure AD Authentication Registration 198
4.9.3 Azure AD Sync Registration 206
4.9.4 JumpCloud Authentication Registration 213
4.9.5 Okta Authentication Registration 222
4.9.6 Verisk's Okta Authentication Registration 227
4.10 Troubleshooting 231
4.10.1 Starting point 231
4.10.2 General common issues 232
4.10.3 Admin site issues 233
4.10.4 Security API issues 234
4.10.5 Authentication issues 235
4.10.6 Authorization issues 238
4.10.7 Security tool issues 239
4.10.8 LDAP Sync issues 240
4.10.9 Legacy Sync Service 242
5. Product handbook 243
5.1 Product's handbook 243
5.1.1 Authentication and Authorization 243
5.1.2 Definition of Application 243
5.1.3 Audit logs 245
5.2 Security Rest API 246
5.2.1 API organisation 246
5.3 Authentication 247
5.3.1 Authentication Model 247
5.3.2 API resources 252
5.3.3 Identity resource 254
5.3.4 Client 255

5.3.5 Persisted Grants 263
5.3.6 Authentication of users 264
5.3.7 Password-based AuthN 266
5.3.8 Windows AuthN 272
5.3.9 Azure AD AuthN 273
5.3.10 ISO ClaimSearch AuthN 276
5.3.11 Okta AuthN 280
5.3.12 JumpCloud AuthN 281
5.3.13 User session 282
5.4 Authorization 284
5.4.1 Authorization Model 284
5.4.2 Groups 287
5.4.3 Roles and permissions 289
5.4.4 User Types 294
5.4.5 Entities 295
5.4.6 Cross domain management 313
5.5 Users 350
5.5.1 Users 350
5.5.2 Search for users 352
5.5.3 Create new users 354
5.5.4 Edit users 356
5.5.5 Inactive and delete a user 358
5.5.6 Clone existing users 359
5.6 Data exchange 361
5.6.1 Data exchange 361
5.6.2 Data exchange at UI 363
5.6.3 Data exchange at API 367
5.6.4 Sequel Security Tool 368
6. Release notes 376
6.1 Current version 376
6.1.1 Release Notes 376
6.1.2 Breaking Changes notes 376
6.1.3 Changelog 377

1. Sequel Security Services

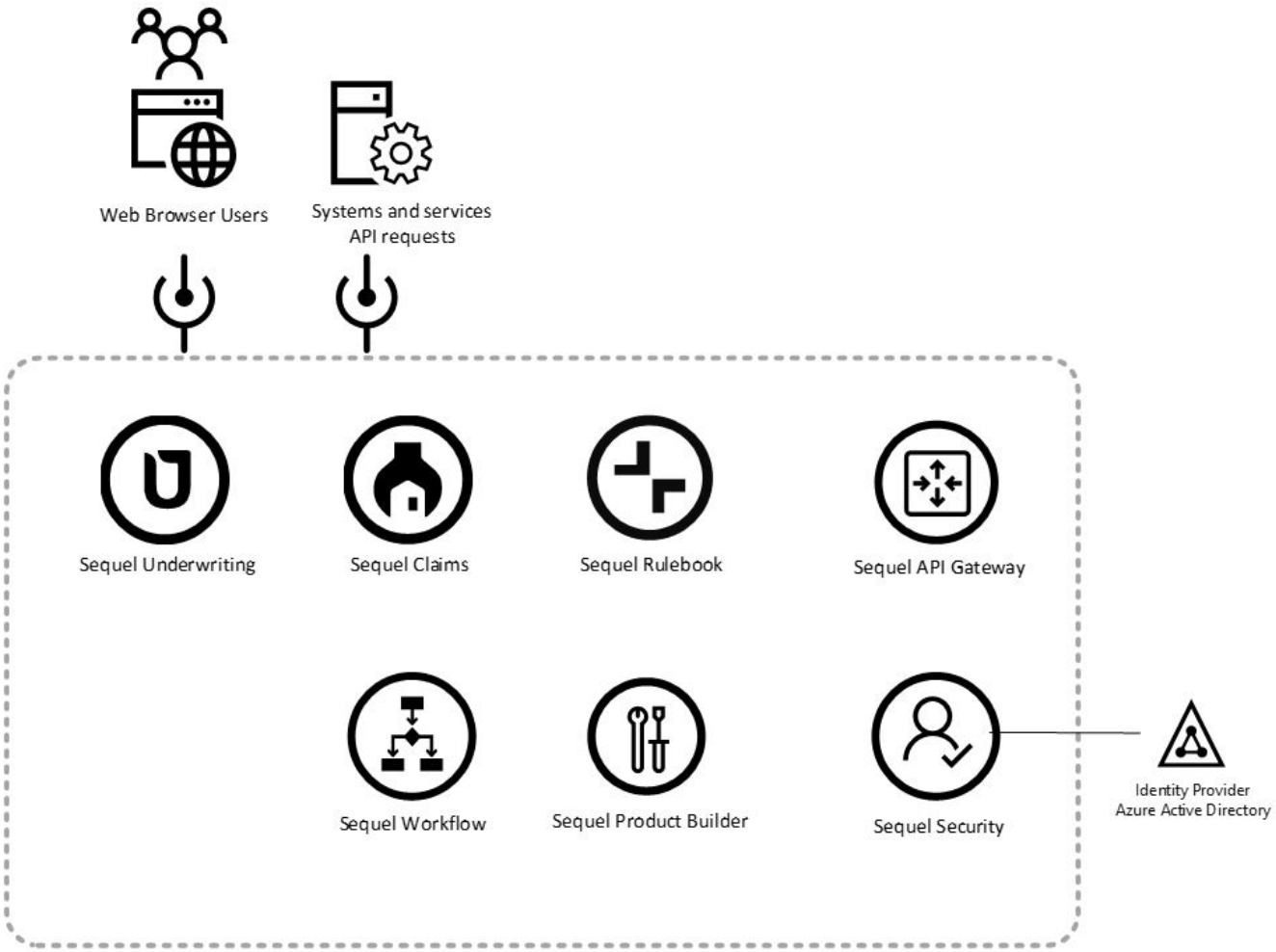
The Sequel Security Service is responsible (in part) for providing authentication and authorisation services to Sequel products. It provides three primary capabilities (directly or indirectly)

- *Authentication* – identifying a user
- *Authorisation* – identifying a user's permissions/roles/rights
- *User Information* - store and cache

Some principles have been applied on the design and development, making *Sequel Security Service*:

- the *single master domain* for managing security concerns, shared across all our frontend applications and APIs,
- based on *modern and secured standards*: oAuth 2.0 and OpenID Connect protocols
- *open to connect to other identity providers (aka IdP) systems*: Azure, Okta, Jumpcloud,...
- *scalable*
- *cloud ready*
- *API first*

Whilst the Sequel Security Service can perform Authentication & Authorisation capabilities itself, it is typically used as a proxy to identity providers who have responsibility for identifying a user and their permissions/roles/rights. This approach enables Sequel products to leverage the benefits and vendor implementations of multi-factor authentication (MFA), single sign-on (SSO), identity protection & monitoring etc., providing customers with a secure, trusted, tried and tested mechanism to authenticate and authorise users.



Critically, when used as a proxy/middleware Sequel Security Service does not store user credentials (eg username, password), nor does it have knowledge of these sensitive credentials – the responsibility of identifying a user lies with the identity provider (eg Microsoft Azure, Okta,...). At no point are these sensitive credentials communicated, processed or stored by the Sequel Security Service. The Sequel Security Service is responsible for storing non-sensitive user information, within its User Information Store and Cache, and providing this user information to other products within the Sequel suite. Section User Information Store and Cache provides an overview of this functionality embedded within the Sequel Security Service.

> **Important**
>
> Sequel Security Service v3 is designed to be hosted only by Verisk's AWS Managed Service. For other hosting options, like on-premises, please contact with SuppApp Product Team.

1.1 Features

1.1.1 Authentication as a Service

Centralized login logic for all our integrated products; based on Duende IdentityServer, an officially certified implementation of OpenID Connect.

1.1.2 Single Sign-on / Sign-out

Single sign-on (and out) over our products.

Auto sign-out on inactive session detection.

1.1.3 Access Control for APIs

Issue access tokens for APIs for various types of clients, e.g. server to server, web applications, SPAs and native/mobile apps. Used for protecting our web applications, public APIs, internal communications and also Sequel API Gateway.

1.1.4 Federation Gateway

Support for external identity providers like Azure Active Directory, Okta, JumpCloud, Windows Integrated Security, Verisk ISO ClaimSearch and opening federation to other external identity providers.

1.1.5 Single master domain

All security concerns like users, permissions, roles, groups,... are centralized in a central service: open via an API and also an SPA Administration site.

1.1.6 Role-based access control

A role-base access control model based on the concepts of groups and roles; that models all Sequel's products integrated with the Security Service.

1.1.7 Cross-domain identity management

Management and synchronization of cross-domain identity provider is supported; allowing exchange and translate users and permissions from different identity providers to Sequel Security Service and Sequel products. Currently, this is supoprted for Active Directories (with LDAP) and Microsfot Azure (with MsGrpah).

1.1.8 Delegated entities and user management

The Entity Delegated feature allows to define entities (team, organizations,...) and delegate the management of permissions of those users.

1.1.9 Import and Export tools and APIs

A console tool and an API allows to import and export configurations by domains (authentication, authorization, users, entities,...) and/or applications.

1.2 Why

At Sequel we are trying to move our applications to an architecture oriented to services, and with great influences from microservices. In this process, we need to decouple services and also improve communications and delegate responsibilities. Regardless to this point security is one of our first challenges.

Our legacy architecture (in terms of security) presented some problems like:

- Security model/technology is not the same across products.
- Security management is not centralized. Users have to be created and managed in multiple applications.
- Security is closed and not based on modern standards.
- Security features are not supported equally across all our products.
- Security is coupling heavily our services.

This is causing us (and to our clients) some problems:

- Duplicated efforts: developing security
- Duplicated efforts: configuring duplicated user accounts for each service.

1.3 About how this documentation is organized

The documentation has been organised in different sections:

Architecture

Architecture. Introduction to the Security architecture.

Developer's Handbook

Developer's Handbook. Documentation useful for developers: builds, continuous integration, branches, API documentation, ...

Operations' Handbook

Operations' Handbook. Documentation about platform specifications, installation guides, troubleshooting, different types of deployment,...

Product's Handbook

Product's Handbook. Functional and user guides.

2. Architecture

2.1 Architecture

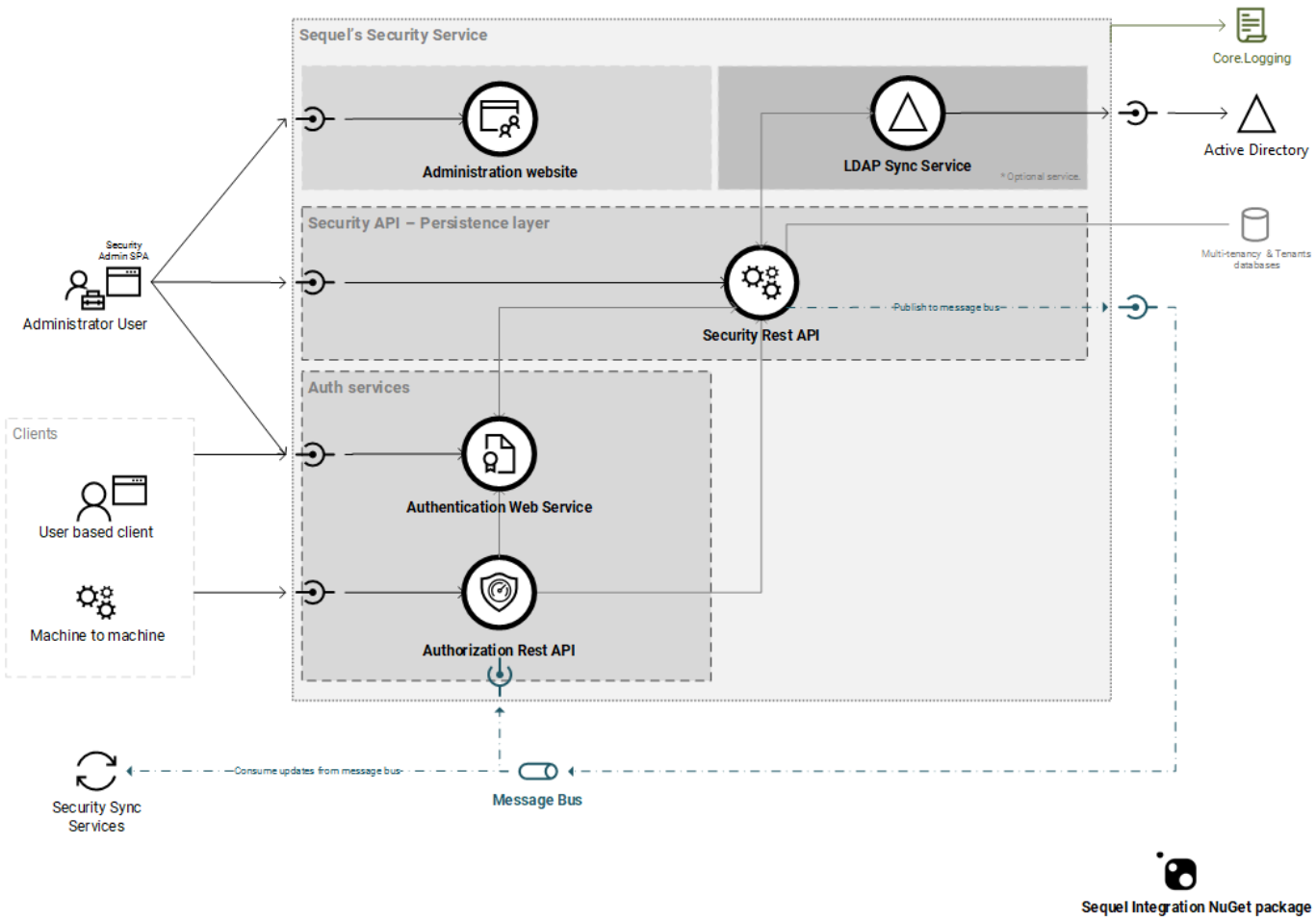
> **Important**
>
> Sequel Security Service v3 is designed to be hosted only by Verisk's AWS Managed Service. For other hosting options, like on-premises, please contact with SuppApp Product Team.

2.1.1 Sequel Security Service

The Sequel Security Service has been designed with some principles in mind:

- Modern and standard security protocols.
- API first
- Cloud ready
- Scalability.

Inspired on above principles, the security service is composed on a set of smaller services:



The boundaries of the Security services are coloured in grey; so we consider external dependencies of the service shared resources as message bus, Sequel.Core.Logging repositories and SQL servers.

All web services that composes the Security Service share some principles:

- API first: all operations are exposed in the APIs.
- APIs documented with Swagger.
- Support deployments in IIS and containers.
- Log all events using *Sequel.Core.Logging*.
- Expose health checks endpoints.
- Fully integration with the enterprise message bus.
- Support scale-out.

Authentication Web Service

Provides authentication using the OAuth2 and OpenID Connect protocols for interactive clients and machine to machine clients. Apart of the authentication protocol endpoints, this service offers the web forms for the users to enter their credentials in order to obtain an access token that will be reused across multiple applications in the suite; this service will serve the login and reset password screens, reused by all the products.

This service is a web application built with:

- ASP.NET Core 6.0 + Razor views
- EntityFramework Core 6.0
- Duende IdentityServer 6.1
- Sequel.Core.Logging libraries

Authorisation Rest API

This REST API offers a fast access to all queries related to authorization: in other words to effective permissions that a user or groups has. Information provided by this service is retrieved from *Security Rest API* and **cached**; in other words, we can consider this service a *proxy that caches effective permissions* information from Security Rest API. It is required to be authenticated for accessing this service.

This service is a web application built with:

- ASP.NET Core 6.0 + Razor views
- EntityFramework Core 6.0
- Swagger
- Sequel.Core.Logging libraries
- Sequel.Core.MessageBus libraries

Security Rest API

REST API for managing all the resources related to security (users, roles, etc.) and to query security data (list of users, list of roles, etc.). It is the unique component in the system with access to security databases. This API will be responsible for publishing any changes in the security data to the service bus, so other applications could consume those messages and keep their data updated.

This service is a web application built with:

- ASP.NET Core 6.0 + Razor views
- EntityFramework Core 6.0
- Swagger
- Sequel.Core.Logging libraries
- Sequel.Core.MessageBus libraries

MULTI-TENANCY SUPPORT

The persistence layer has been built to support multi-tenancy in the future versions; but in the current version the system is no multi-tenant.. So, the current state of project doesn't support real multi-tenancy; however, all persistence layer and security API is multi-tenant.

As result of this work; the databases are managed following the below diagram. However, all request are redirected to the default tenant internally; so, we will have two databases:

- Multi-tenancy database: Single database per system; used to store specific information (like connection string) for each tenant.
- Security database: one per tenant; stores all authentication and authorization information.

Administration website

Static website application for managing security data, depends on *Security Rest API* and *Authentication Service*. It's a *single page application (SPA)* using:

- ASP.NET Core 6.0
- ReactJS

LDAP Sync Windows Service

Windows Service for synchronizing users and groups from an Active Directory; this is an optional component. Depends on *Security Rest API*. It is built with:

- ASP.NET Core 6.0
- LDAP
- Sequel.Core.Logging libraries
- Sequel.Core.MessageBus libraries

Azure AD Sync Windows Service

Windows Service for synchronizing users and groups from Azure AD; this is an optional component. Depends on *Security Rest API*. It is built with:

- ASP.NET Core 6.0
- MS Graph
- Sequel.Core.MessageBus libraries

2.1.2 Sequel.Security.Integration NuGet package

Package useful for other teams to integrate their applications with the **Sequel Security Service**. Helps to register a resources, clients,... This component is not part, purely talking of the Security Services and evolves with different versions and all versions are backward compatible between the security services and the integration package; however, using the latest version will ensure the best user experience.

2.1.3 User web component

It is a *web component* for managing concepts of the user session like current user information, inactivity and single sign-out. This component is provided with the aim of being integrated in all Sequel's applications in order to improve UI/UX consistency in user info/session management.

2.1.4 Legacy Security Sync Services

Some applications like Claims, Origin and Workflow need to keep the legacy security schema in sync with the Security Service; this is done with a windows service that consumes all messages with changes published by security. The owner of those consumers are those applications; however, the first implementation was provided by Sequel's Security Team.

The legacy security sync service is built using:

- ASP.NET Core 6.0
- Sequel.Core.Logging libraries
- Sequel.Core.MessageBus libraries

2.1.5 Service bus integration

Security services emit messages to the bus (RabbitMQ) for different reasons; that we will cover in the next sections.

3. Developers handbook

3.1 Overview

Developers' Handbook contains documentation useful for developers:

- Developer's guide, with tips for developing and debugging applications with security services.
- UI Style guide.

And more useful topics: api best practices, sequel-security tool for developers, integration, message bus, performance,...

3.2 Developer's guide

The **Sequel Security Service** is a system built on the .NET Core stack (among other technologies). This guide describes some useful tips for developers in order to configure the dev box.

3.2.1 .NET Core version

The applications of the **Sequel Security Service** has been implemented using .NET6; Make sure that you have the right version by just running the following command:

```
> dotnet --version
```

If the previous command doesn't output a version higher or equal to 6.0, then you should install the .NET Core SDK from https://dot.net.

Local HTTPS configuration for 'dotnet' command

The applications that compose the **Sequel Security Service** must always be running on HTTPS. This is mandatory for production environments, but also for local environments during the development stage. As described in the previous section, ASP.NET Core 6.0 is the current version of the framework that is being used; there's a dotnet global tool built into .NET Core 6.0 to help with certificates at dev time, called "dev-certs." In order to configure the dev certificates for localhost into the machine being used for development the following command should be executed:

```
> dotnet dev-certs https --trust
```

A popup asking if the localhost certificate should be trusted will be shown and, after accepting it, then the applications will be executed on HTTPS without issues on Chrome or any other browser. Now the applications can be safely started on HTTPS using:

```
dotnet run
```

Local HTTPS configuration for ISS Express

When running/debugging the applications from Visual Studio they can be run on IIS Express. The first time you execute the applications a popup will appear to trust the IIS Express dev certificate; by accepting it the applications will run on HTTPS without issues.

3.2.2 Use of a Security FVM for local development

This document will guide you on how to configure a Security FeatureVM as a Security server for local development in a team. For this, you need the following:

- Create a Security FVM
- Exported configuration for the application which will use Security (Workflow, Claims, Origin, Impact...)
- Security Tool (see docs)

Create the FeatureVM

Just create it. You may use `master` as source branch. But if you like to be in the edge, you may use `dev`. The name of the build contains the machine name of the VM (`FVMSECXXXXXX.office.sbs`).

Install configuration

The configuration to deploy in Security of the applications should be tweaked to work on the development machines. For each application, check the `Clients.json` files to ensure the fields `AllowedCorsOrigins` have URLs that points to `localhost` (or `localhost.office.sbs` - see this) and ensure the `ClientSecrets` are correct and properly encrypted.

Then run the security tool to deploy the applications:

```
sequel-security import -c 'Data Source=FVMSECXXXXXX.office.sbs;Initial Catalog=SecurityDatabase;Integrated Security=True;Multi pleActiveResultSets=True;' --domain
authentication -a ${APP_KEY} -i /path/to/the/package
```

```
sequel-security import -c 'Data Source=FVMSECXXXXXX.office.sbs;Initial Catalog=SecurityDatabase;Integrated Security=True;Multi pleActiveResultSets=True;' --domain
authorization -a ${APP_KEY} -i /path/to/the/package
```

The `${APP_KEY}` token is the application's identifier (like `WF` for Workflow or `ORI` for Origin) which will try to import in the two commands. This will import Authentication and Authorization configuration for each.

After importing all configuration, an *IIS Reset* is recommended to clean all caches in Security:

```
iisreset FVMSECXXXXXX
```

Give admin their rights

By default, an admin user is created with the Feature VM. So, in order to the admin user to access the apps, it must have the right roles. So follow this steps to give it the permissions:

- Go to the Administration panel of security (`https://fvmsecXXXXXX.office.sbs/Administration`).
- Once there, go to the *Users* page.
- Select the `admin` user, which will open a drawer at the right.
- Select the *MEMBERSHIP* tab and then press the pencil icon near the admin's name (this will change to edit mode).
- Add the roles for each application.

Now you may be able to log in into the application in your local machine.

Change Security URLs in configuration

Now it is time to point the apps in the development machine to the Feature VM. Go to the `web.config` or `appsettings.json` of your .NET Framework/.NET Core app and change these values as following:

Key	Value
`SecuritySettings:Authorization:ServerUri`	`https://fvmsecXXXXXX.office.sbs/Authorization`
`SecuritySettings:Authentication:ServerUri`	`https://fvmsecXXXXXX.office.sbs/Authentication`
`SecuritySettings:Authentication:External:ServerUri`	`https://fvmsecXXXXXX.office.sbs/Authentication`

Now, give it a try to see if it is working. Access to the application on your development machine, check if it starts the Authentication Flow, and the app can get the information about the session (aka logged in properly).

The domain and the cookie

The session is maintained using a cookie which is available only under the domains that ends with `.office.sbs` . So, you may need to use `localhost.office.sbs` for the configuration to ensure the cookie is present in the development machines. To make it work, edit the hosts file (`C:\Windows\System32\drivers\etc\hosts` in Windows, `/etc/hosts` on *nix) and add `127.0.0.1 localhost.office.sbs` . The use this domain to access to the apps in your machine.

3.3 UI Style guide

3.3.1 Style guide

The style guide for security and all Supporting Apps configuration sites is https://bit.ly/2J4X2c9. The main library used for building the UI in security is **react-md** library (https://react-md.mlaursen.com/).

Content	Slide
Fonts	2
Colours	3
Buttons	4
Cards	5
Dialogues	6
Side panels	7,8
Notifications	9
Tabs and empty messages	10
Time stamps	11
UI elements	12-15
Wizards	16
User avatars	17
Application drawer	18

3.3.2 Prototypes

- **Security area Prototype**: https://bit.ly/2IOH8PE
- **Login area prototype**: https://bit.ly/2xcsvEr

3.4 API best practices

3.4.1 API design

- Use Two URLs per Resource. One URL for the collection and one for a single resource
- Use Consistently Plural Nouns.
- Use HTTP Methods to Operate on your Resources.
- **Wrap the Actual Data in a data Field**. (Include in current documentation!!)
- POST actions will return a 201 (resource created) and location header included with the location of the new resource.

More info at:

- https://blog.philipphauer.de/restful-api-design-best-practices/
- https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design
- https://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api

3.4.2 API documentation

We use OpenAPI specification (Swagger implementation) for documenting our APIs. In this section, we expose some tips for documenting properly those resources.

Resource description

"Resources" refers to the information returned by an API. Most APIs have various categories of information, or various resources, that can be returned. The resource description provides details about the information returned in each resource. The resource description is brief (1-3 sentences) and usually starts with a verb. Resources usually have a number of endpoints to access the resource, and multiple methods for each endpoint. Thus, on the same page, you usually have a general resource described along with a number of endpoints for accessing the resource, also described.

Endpoints and methods

The endpoints indicate how you access the resource, and the method used with the endpoint indicates the allowed interactions (such as GET, POST, or DELETE) with the resource. The endpoint shows the end path of a resource URL only, not the base path common to all endpoints. The same resource usually has a variety of related endpoints, each with different paths and methods but returning variant information about the same resource. Endpoints usually have brief descriptions similar to the overall resource description but shorter.

Parameters

Parameters are options you can pass with the endpoint (such as specifying the response format or the amount returned) to influence the response. There are four types of parameters: header parameters, path parameters, query string parameters, and request body parameters. The different types of parameters are often documented in separate groups. Not all endpoints contain each type of parameter.

Request example

The request example includes a sample request using the endpoint, showing some parameters configured. The request example usually doesn't show all possible parameter configurations, but it should be as rich as possible with parameters. Sample requests sometimes include code snippets that show the same request in a variety of languages (besides curl). Requests shown in other programming languages are optional and not always included in the reference topics, but when available, users welcome them.

Response example and schema

The response example shows a sample response from the request example. The response example is not comprehensive of all parameter configurations or operations, but it should correspond with the parameters passed in the request example. The response lets developers know if the resource contains the information they want, the format, and how that information is structured and labeled. The description of the response is

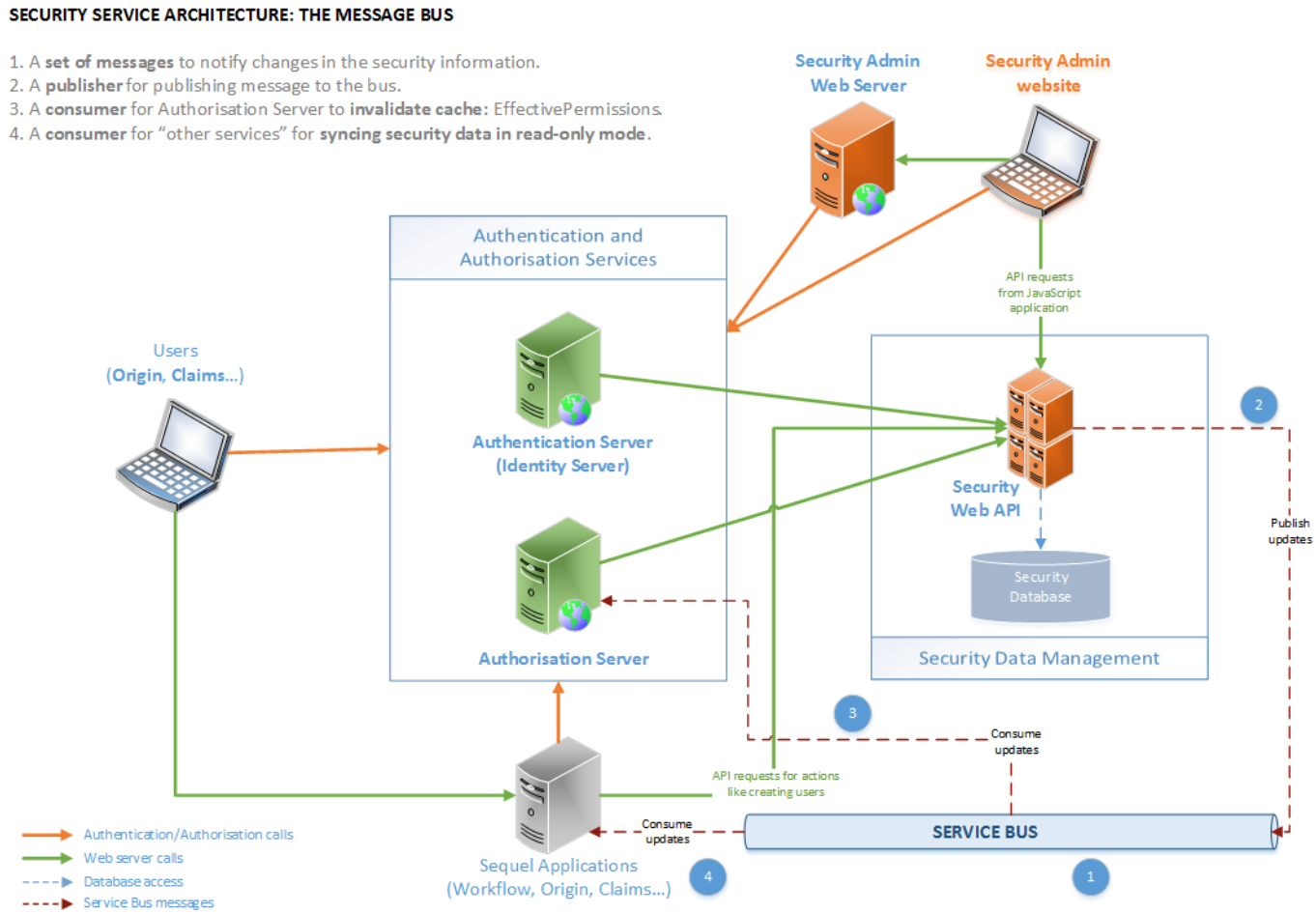
known as the response schema. The response schema documents the response in a more comprehensive, general way, listing each property returned, what each property contains, the data format of the values, the structure, and other details.

3.5 Message Bus

3.5.1 Overview

In the distributed architecture we are building; in one hand, we want to keep a single master domain for Security concerns, but in the other hand we want to allow other services to use this information without affecting performance and keeping services decoupled. This information usually is quite stable in production and quite simple; if we ignore the calculation of the effective permissions (permissions for a given user of group). Based on above premises, we can assume that allowing other services to manage a local read-only copy of this information is reasonable and quite important for performance reasons (almost required); we will refer to this local read-only version in some applications like the *legacy security schema*.

This integration is achieved using our service bus. All changes and deletions of groups, roles, user types, securables and users will be notified with a message; also, each time than an effective permissions has changed will be notified.



In the previous diagram we can see how the service bus is used in the Security services. We will need to develop four main components or artifacts:

1. **Message contracts**. A set of messages for defining the different events that can occur in the security service and we need to expose to external services.
2. **Publisher**. We need a mechanism in the Security API that it will react to changes in security configuration and it will send the specific message to the bus.
3. **Consumer for security services**. Authorization server needs to cache the effective permissions provided by Security API; so, it's required a mechanism to invalidate this cached information. So, we need a bus consumer to react to events for invalidating the cached information. Also, Authentication services could have cached data and it will need to react to changes in order to invalidate this information.
4. **Consumer for "external services"**. External services (as Workflow or Product Builder) will need to keep some security information in sync. So, they will need to build a consumer for reacting to those events. As part of the work done in Supporting Apps Team we will develop this consumer in Product Backlog Item 113260:Security Messages Consumer for WF & PB

3.5.2 1. Message contracts

Message contracts will be defined in Sequel.Security.MessageBus.Contracts project and this will be published via NuGet.

General concerns

There are some concerns when we define the contracts that we have to cover:

- A **message contract per event**. In our case, that means that we will have two different messages for each entity: changed and deleted. Changed includes created event; as both events must be managed in the consumer with the same approach: *create if not exists or update*.
- Contracts must be **versioned** based on *different interface* implementations; we will use namespaces to achieve it, following this name convention:

```
Sequel.Security.MessageBus.Contracts.{MessageName}.v{Version}

Sequel.Security.MessageBus.Contracts.Securables.v1
Sequel.Security.MessageBus.Contracts.Securables.v2
Sequel.Security.MessageBus.Contracts.Groups.v1
```

- Don't assume message will be consumed in **sequence**: We need to add a SequenceId.

```
    /// <summary>
    /// Defines a class that contains a SequenceId for sequencing messages
    /// </summary>
    /// <typeparam name="T">Class type of the object created or changed</typeparam>
    public interface ISequencedMessage
    {
        string SequenceId { get; set; }
    }
```

The sequence id for some messages it could be a version number that we can manage in the underlying entity (maybe at database level). However, some scenarios, like EffectivePermissionChanged will be quite complex to define a sequence id. So, we will need to define it in a different way; and with the aim of keep consistency I would suggest to use the same approach for all messages. There are different approaches for solving it; and some of them are described in this article: https://stackoverflow.com/questions/2671858/distributed-sequence-number-generation. However, for our current scenario I would suggest to generate a timestamp based on current data. It could be something like:

```
DateTime.UtcNow.ToString("yyyyMMddHHmmssffffff", CultureInfo.InvariantCulture)
    //20180525143621434686  -- Lenght 20
```

- **Multi-tenant** messages.

As security information is divided by tenants; this information must be included for all messages, as this will allow to the subscribers to filter by some types of messages. We can resume common information in below interface:

```
    public interface ITenantAwareMessage
    {
```

```
        string TenantId { get; set; }
    }
```

Architecture catch-up required: If we want to deploy service bus as multi-tenant resources; we can implemented it in RabbitMQ using virtual host; and in Microsoft using namespaces. However, this will required to implement our consumers to connect to multiple hosts. This will required some input some Architecture team.

- **Application** base messages.

As most of the security information is divided by applications; this information must be included for some messages, as this will allow to subscribers to filter by some types of messages. We can resume common information in below interface:

```
public interface ISecurityMessageContractBase {
    string ApplicationKey { get; set; } //For filtering
    string Code { get; set; }
    string Key { get; set; } //Key = ApplicationKey + . + Code
}
```

Architecture catch-up required: Current implementations of Sequel.Core.MessageBus doesn't provide support for filtering messages.

Messages

EFFECTIVE PERMISSIONS

- **IEffectivePermissionChangedMessage**. *Effective permissions for a specific application has changed*. Change could affect to a single user or a list of groups. This message is published:
- When membership collection changes for a user: publish a message with the affected user, group and application.
- When permissions associated to a role changes: publish a message with the application affected and the groups affected; UserName is null. To calculate the affected groups we need to detect all different groups associated to this role in the membership table.

The content of this message is directly defined by the endpoints offered by SecurityAPI Service (at the EffectivePermission); where effective permissions are mainly accessed by application, user and a list of groups. Managing this message in this way we can allow to other applications to cache responses and invalidate them; and at the same time, keep all the logic for calculating effective permissions internally at the security services. The time to live for this messages is set to 30 minutes by default.

```
public interface IEffectivePermissionChangedMessage : ITenantAwareMessage, ISequencedMessage
{
    /// <summary>
    /// Determines ApplicationKey where permissions has changed.
    /// </summary>
    string ApplicationKey { get; set; }

    /// <summary>
    /// Determines the groups affected by the change in effective permissions.
    /// </summary>
    List<string> GroupKeys { get; set; }

    /// <summary>
    /// Determines the user affected by the change in effective permissions.
    /// If null means that affects to all applications.
  /// </summary>
    string UserName { get; set; }
}
```

FORGOT PASSWORD

- **IForgotPasswordChangedMessage**. This message is published when a user make a request to change the password.

The content of this message is defined by the user data. The time to live for this messages is set to 10 minutes by default.

```
public interface IForgotPasswordChangedMessage : ITenantAwareMessage, ISequencedMessage
  {
      string Name { get; set; }
      string EmailAdress { get; set; }
      string FirstName { get; set; }
      string LastName { get; set; }
      string UrlBack { get; set; }
  }
```

ENTITIES CHANGED OR DELETED: GROUPS, ROLES, SECURABLE, USER AND USER TYPES.

The purpose of this messages is to allow other systems to synchronize security information. Time to live parameter is set to 12 hours by default.

Messages related to creation and modification are based on:

```
public interface IObjectChangedMessage<T> : ITenantAwareMessage, ISequencedMessage
{
    /// <summary>
    /// Gets or sets the data of the
    /// </summary>
    T Data { get; set; }
}
```

Deletion messages are based on:

```
public interface IObjectDeletedMessage<K> : ITenantAwareMessage, ISequencedMessage
{
    K Key { get; set; }
}
```

With above abstractions, we can define below messages; inheriting from `IObjectChangedMessage<T>` or `IObjectDeletedMessage<K>`:

- **IGroupChangedMessage**. A new group has been created or an existing has been modified. Where Data is:

```
public interface IGroup : ISecurityMessageContractBase
{
    bool IsSystem { get; set; }
    string Name { get; set; }
    bool AutomaticallyCreated { get; set; }
}
```

As a sample, an implementation of this message will look like:

```
class GroupChangeSampledMessage : IObjectChangedMessage<Group>, v1.IGroupChangedMessage
{
    public Group Data { get; set; } //Group implements IGroup
    public string TenantId { get; set; }
    public string SequenceId { get; set; }
}
```

- **IGroupDeletedMessage**. A group has been deleted.
- **IRoleChangedMessage**. A new role has been created or an existing has been modified. Where Data is:

```
public interface IRole : ISecurityMessageContractBase
{
    bool IsSystem { get; set; }
    string Name { get; set; }
}
```

- **IRoleDeletedMessage**. A role has been deleted.
- **ISecurableChangedMessage**. A new securable has been created or an existing has been modified. Where Data is:

```
public interface ISecurable : ISecurityMessageContractBase
{
    bool IsSystem { get; set; }
    string Name { get; set; }
    string CreateDescription { get; set; }
    string DeleteDescription { get; set; }
    string Description { get; set; }
    bool IsCreateAllowed { get; set; }
    bool IsDeleteAllowed { get; set; }
    bool IsReadAllowed { get; set; }
    bool IsUpdateAllowed { get; set; }
    string ReadDescription { get; set; }
    string UpdateDescription { get; set; }

}
```

- **ISecurableDeletedMessage**. A securable has been deleted.
- **IUserChangedMessage**. A new user has been created or an existing has been modified. This message is also published when membership or the user types assigned for the user has changed. Where Data is defined by:

```
public interface IUser
{
```

```
    string Name { get; set; }

    string EmailAddress { get; set; }

    string FirstName { get; set; }
    string LastName { get; set; }
    string PhoneNumber { get; set; }

    bool AbsenceInd { get; set; }
    bool AutomaticallyCreated { get; set; }
    bool LockoutEnabled { get; set; }

    DateTime? LastLoginDateUtc { get; set; }
    DateTime? ActiveEndDateUtc { get; set; }
    DateTime? PasswordExpiryDateUtc { get; set; }
    DateTime? LockoutEndDateUtc { get; set; }

    List<IMembership> Memberships { get; set; }
    List<IAssignedUserType> AssignedUserTypes { get; set; }
}

public interface IMembership
{
    string GroupKey { get; set; }
    string RoleKey { get; set; }
}

public interface IAssignedUserType
{
    string UserTypeKey { get; set; }
}
```

- **IUserDeletedMessage**. A user has been deleted.
- **IUserTypeChangedMessage**. A new user type has been created or an existing has been modified. Where Data is defined as:

```
public interface IUserType : ISecurityMessageContractBase
{
    bool IsSystem { get; set; }
    string Name { get; set; }
}
```

- **IUserTypeDeletedMessage**. A user type has been deleted.
- **IConfigurationChangedMessage**. A message that contains current Authorization and/or Users configuration:

```
public interface IConfiguration
{
    IUsersConfiguration Users { get; set; }

    IDictionary<string, IAuthorizationConfiguration> Applications { get; set; }
}

public interface IUsersConfiguration : IEnumerable<IUser>
{

}

public interface IAuthorizationConfiguration
{
    IEnumerable<ISecurable> Securables { get; }
    IEnumerable<IUserType> UserTypes { get; }
    IEnumerable<IRole> Roles { get; }
    IEnumerable<IGroup> Groups { get; }
}
```

3.5.3 2. Publisher

At the Security API we need to develop a publisher for sending above messages to the bus. Triggering scenarios are described above in the message section.

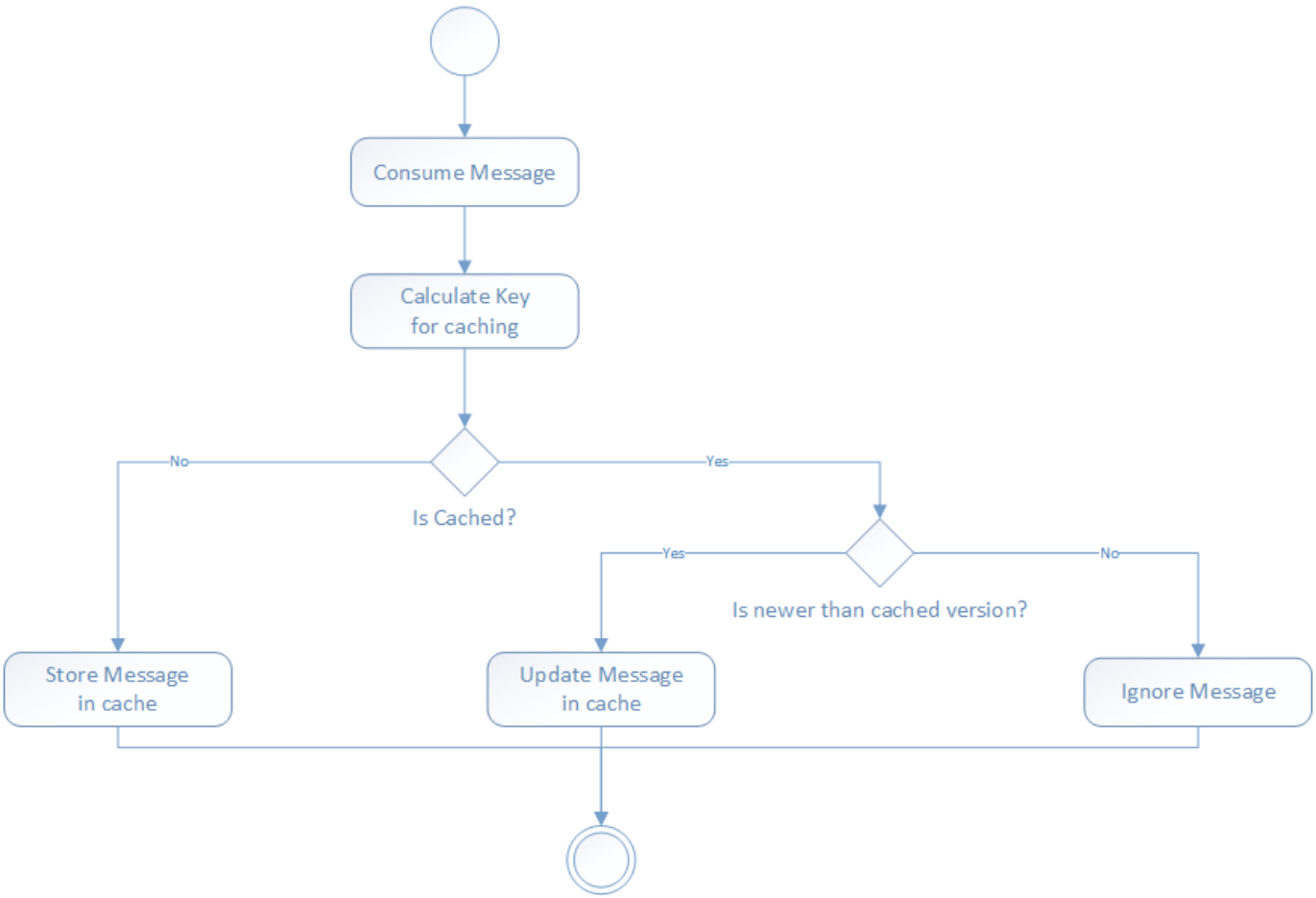
> **Architecture catch-up required:** Concerns here: exchange naming convention and how to support filtering by application and multi-tenant (previously exposed).

3.5.4 3. Consumer for internal Security Services

Consumers and caching

One purpose of consumers in security is caching information. We will follow below approaches:

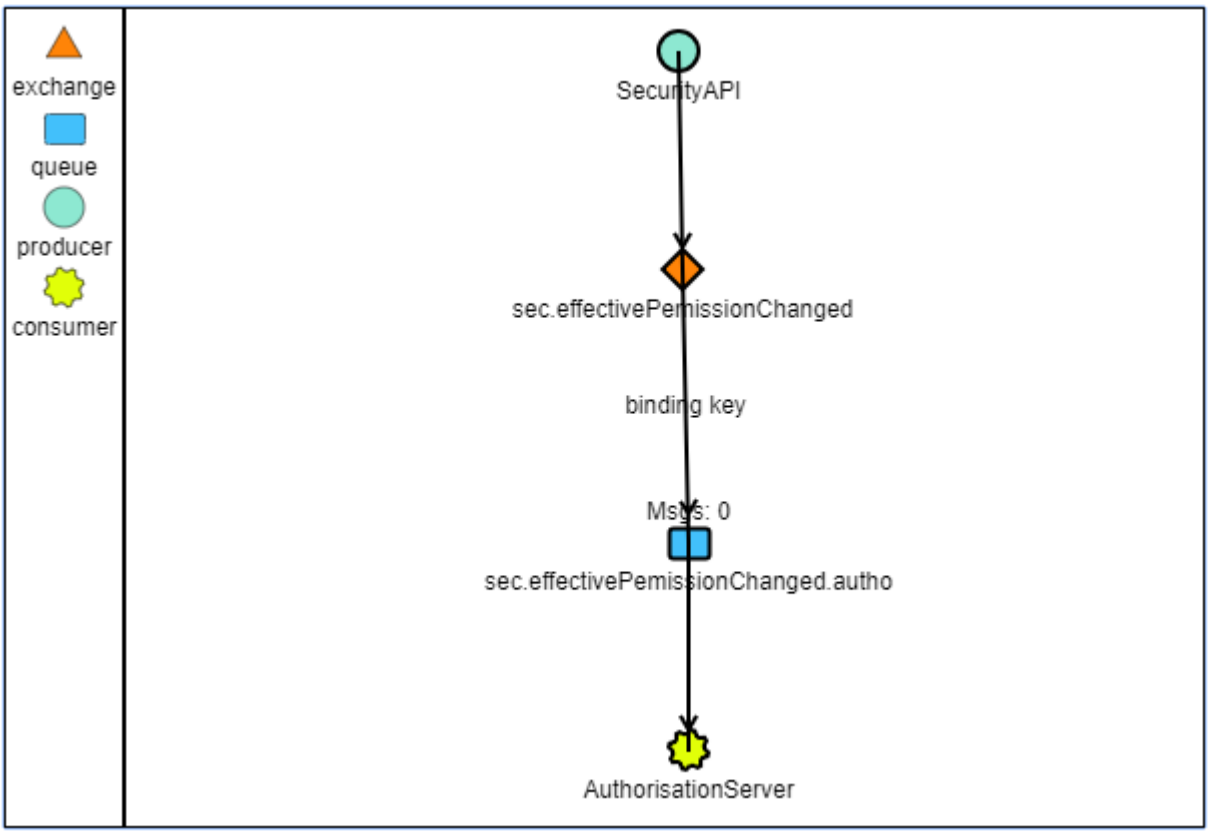
- We will use in memory cache from ASP.NET Core.
- Cache will be enabled by default via configuration (cache:enabled = true). So, we will be able to disable if required.
- Cache will be invalidated consuming messages from the bus; the logic to invalidate the bus is:



The logic for calculating the key depends of each message; for messages inheriting from `ISecurityMessageContractBase` the Key is clearly defined; for User messages the Key is the Id; and for Effective Permissions the key is based on ApplicationKey, UserId and GroupIds (it will depend on how this application is managing the cache).

Consumer for Authorisation service

EffectivePermissionsChanged will be consumed by Authorisation service, in order to invalidate the EffectivePermissions cached.



Consuming a message will need to ensure this message is newest than the previous one; and, in this case invalidate the changed data. The information cached will be based on TenantId, ApplicationKey, UserId and GroupIds.

3.6 Sequel Security Tool for developers

3.6.1 Installation

`sequel-security` is a global tool implemented using the Global Tools feature in .NET Core 6.0. The .NET Core SDK 6.0.0 or newer must be installed for development, and the runtime for servers where it needs to be installed/executed.

For **production**, get the version of this tool (Sequel.Security\Packages\Tools\sequel-security\sequel-security.exe) from the same **Sequel.Deployment.Manager** you work with, for example:

```
\\buildoutput.office.sbs\drops\Security\stable\1.72.21057.01.stable\Sequel.Security\Packages\Tools\sequel-security\sequel-security.exe
```

For **testing** purposes, the security tool can be installed manually as follows:

1. Clone Sequel Security using the master branch.

2. Open a command line prompt and navigate to the folder where the `sequel-security.csproj` file is.

3. Pack the tool into a NuGet package using:

```
dotnet pack -c release -o nupkg
```

4. Globally install the tool with:

```
dotnet tool install --add-source .\nupkg -g sequel-security
```

5. If necessary, uninstall the tool from the global scope with:

```
dotnet tool uninstall -g sequel-security
```

3.6.2 Deployment Manager token replacement

Deployment Managers of Rating Engine and Workflow are configured with activities to replace tokens with parameters during database configuration process. Tokens examples are **ClientSecretHash**, **RatingEngineUrlExternal** or **WorkflowExternalUrl**.

RunPowerShellScript ReplaceHash.ps1 and ReplaceTokensWithParameters activities replace token in the package files and cannot be reused, so redeploying again using a different URL and secret does not replace tokens and new settings are not upload; the settings of the first deploy remains.

The solution for the above problem is:

1. Update `RunPowerShellScript ImportSecuritySettings` activity in DatabaseMetadata.xml, with arguments for UtilsPath, ClientSecretHashToken, ClientSecretHashValue and RatingEngineUrlExternalToken (for Rating Engine) or WorkflowExternalUrl (for Workflow) like:

```
<activity xsi:type="RunPowerShellScript" file="Packages\Scripts\ImportSecuritySettings.ps1">
    <argument name="ConnectionString" value="{p:SecurityConnectionString}" type="string"/>
    <argument name="ImportToolLocation" value="{var:RootFolder}Packages\Tools\sequel-security\sequel-security.exe" type="string" />
    <argument name="InputFolder" value="{var:RootFolder}Security\Vanilla" type="string" />
    <argument name="UtilsPath" value="{var:RootFolder}\lib\Sequel.Deployment.Manager.Utils.exe" type="string" />
    <argument name="ClientSecretHashToken" value="__ClientSecretHash__" type="string"/>
    <argument name="ClientSecretHashValue" value="{p:ConfigurationClientSecret}" type="string"/>
    <argument name="RatingEngineUrlExternalToken" value="{__RatingEngineUrlExternal__}{{p:RatingEngineUrlExternal}}" type="string"/>
</activity>
```

2. Delete activities `ReplaceTokensWithParameters` and `RunPowerShellScript ReplaceHash.ps1` from DatabaseMetadata.xml.
3. Delete `ReplaceHash.ps1` script
4. Update `ImportSecuritySettings` script:
    - adding parameters for new arguments such as UtilsPath, ClientSecretHashToken, ClientSecretHashValue and RatingEngineUrlExternalToken (for Rating Engine) or WorkflowExternalUrl (for Workflow).

      ```
      [string]$UtilsPath,
      [string]$ClientSecretHashToken,
      [string]$ClientSecretHashValue,
      [string]$RatingEngineUrlExternalToken
      ```

    - setting the hash value of the client secret

      ```
      $HashedValue = & "$UtilsPath" -m hash -p $ClientSecretHashValue
      ```

    - editing the security tool import authentication command adding token replacements parameters

      ```
      -tr "{$ClientSecretHashToken}{$HashedValue}" -tr "$RatingEngineUrlExternalToken"
      ```

Check the script and activity details for Rating Engine as sample.

3.6.3 Functionality

Functionality and commands details are here

3.7 Integration

3.7.1 Starting with security

In this section we will cover how to integrate your application with the new Security Service:

- For Claims and Origin please follow the migration guide.
- For any other application please follow the integration guide.
- Also checkout User Session Avatar for SPAs and similar web apps.

3.7.2 Integration guide

Overview

Sequel Security Services aim is provide a single master domain for managing security concerns, shared across all our frontend applications (at least), open to connect to other systems and based on modern and secured standards.

With this goal, we have defined a set of services in charge of managing security (Authentication and Authorization) based on oAuth 2.0 and OpenID Connect protocols.

This will allow us to decouple services in terms of security, while providing a SSO experience to the user and reducing duplicates effort in different teams for implementing security topics.

We will cover topics like:

- Authentication
- Register API Resources and Clients.
- Single Sign in and Single Sign out.
- Authorization
- Securables, Roles and Groups
- User Types
- Users
- Entities

It is highly recommended to be familiar with other Security documentation:

- Technical documentation
- Platform Specification
- Installation Guide

This document will try to define the steps to integrate the new Sequel Security Model and Services in a web application or Web API.

During this document we will use as sample the Security integration into Sequel Rating Engine.

Design

APPLICATION

As described in the Technical Documentation, the Security Model is driven by the concept of an *Application*. So, the first step is define how this application will be modelled in terms of authentication and authorization.

All applications needs a `Key` that must be unique and short in the sequel ecosystem. Some existing applications (in bold) are and some keys reserve for future (in italic) are:

Application	Application Key
Broking	*BRK*
Claims	**CLM**
Document Service	*DOC*
API Gateway	*GWY*
Impact	*IMP*
Origin	**ORIGIN**
Product Builder	**PB**
Rating Engine	**SRE**
Reinsurance	*RE*
RuleBook	*RB*
Security	**SEC**
Underwriting	*UW*
Workflow	**WF**

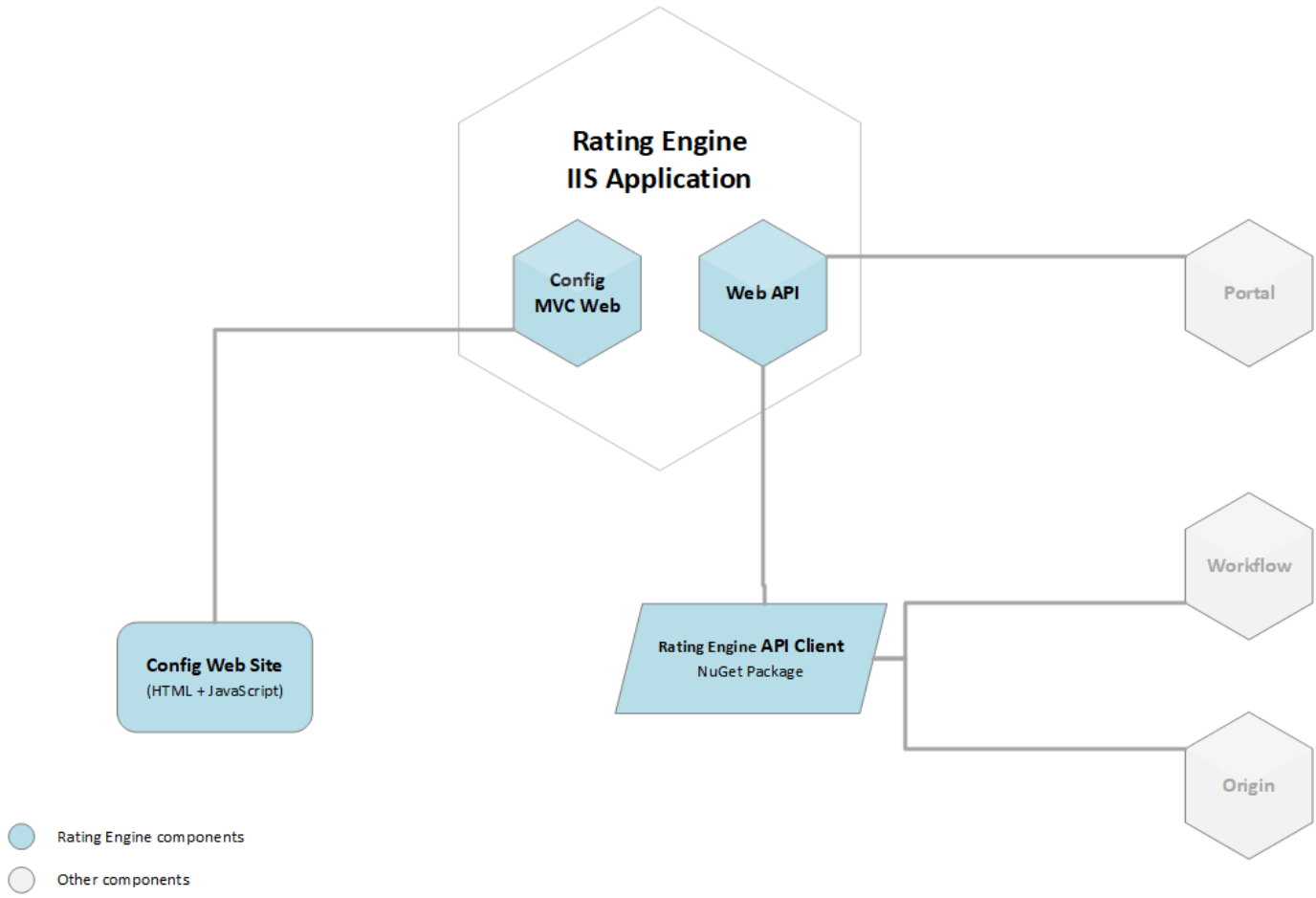
STEP 1. Naming Decide a name for the application.

In our sample, we have decided to use SRE as the key for the Sequel Rating Engine application.

AUTHENTICATION

Once we have decided a name, we need to model the authentication behaviour of this application. For doing this, we will suggest to prepare a diagram of your application, where all web applications (differentiate UI from server side) and web API are described. Also, existing interactions with other sequel services should be included.

In our sample, in Rating Engine, the diagram will look like:

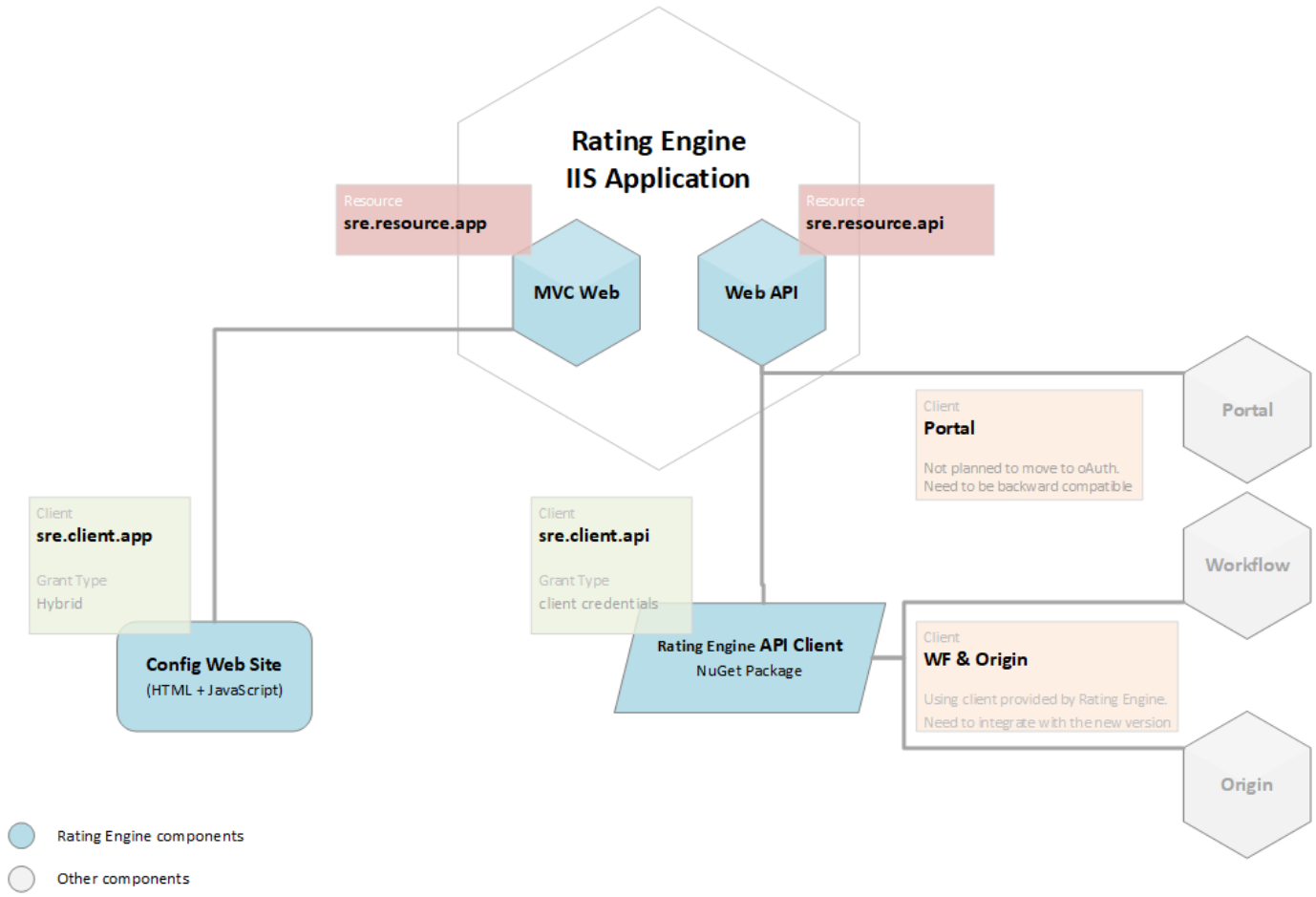


Based on our architecture we will need to discover the different API resources and clients we need to protect and we need to allow to access; plus the scopes within each resource. Also, we will provide names. For naming authentication components (resources, scopes, clients,..) we have decided to follow the name convention that all those concepts starting by an application key are owned by this application and will be managed in the import/export process as part of this application. So, you have to follow this name convention.

STEP 2. Design Authentication Determine the resources to be protected and the potential clients.

For our sample, we have discovered:

- *2 Resources that needs to be protected:* `sre.resource.app` *(with a single scope* `sre.resource.app.user_login`*) and* `sre.resource.api` *(with a single scope* `sre.resource.api.full_access`*).*
- *2 known clients:* `sre.client.app`*,* `sre.client.api` *and other existing clients.*



API Resources

We have divided the IIS application in two resources because our application have two clear areas to be protected in different ways. We have decided to model it with two different resources because, ideally, they should be two independent resources

- **Web MVC** (that we will called `sre.resource.app`) covers the Rating Engine MVC endpoints that offer service to the web application executed in the browser (this is the client `sre.client.app`). Basically, here the user is a human and how this works is:
- Users using the `sre.client.app` in theirs browsers call to `sre.resource.app`.
- `sre.resource.app` asks for authentication to `sre.client.app` and user is redirected to the Authentication Server in order to authenticate himself and allow to `sre.client.app` to interact with `sre.resource.app` in his name.

This is modelled using the hybrid grant type flow; that allows to support access tokens and refresh tokens. This is valid for MVC applications; pure SPA needs to use authorization code flow. In the first versions we were using implicit flow, that is no longer recommended due to security risks: The state of implicit flow

- **Web API** (`sre.resource.api`) covers the Rest API exposed by Rating Engine and consumed by other services like Portal, Origin or Workflow. In this scenario, the client is a service and security is based on client credentials flow.

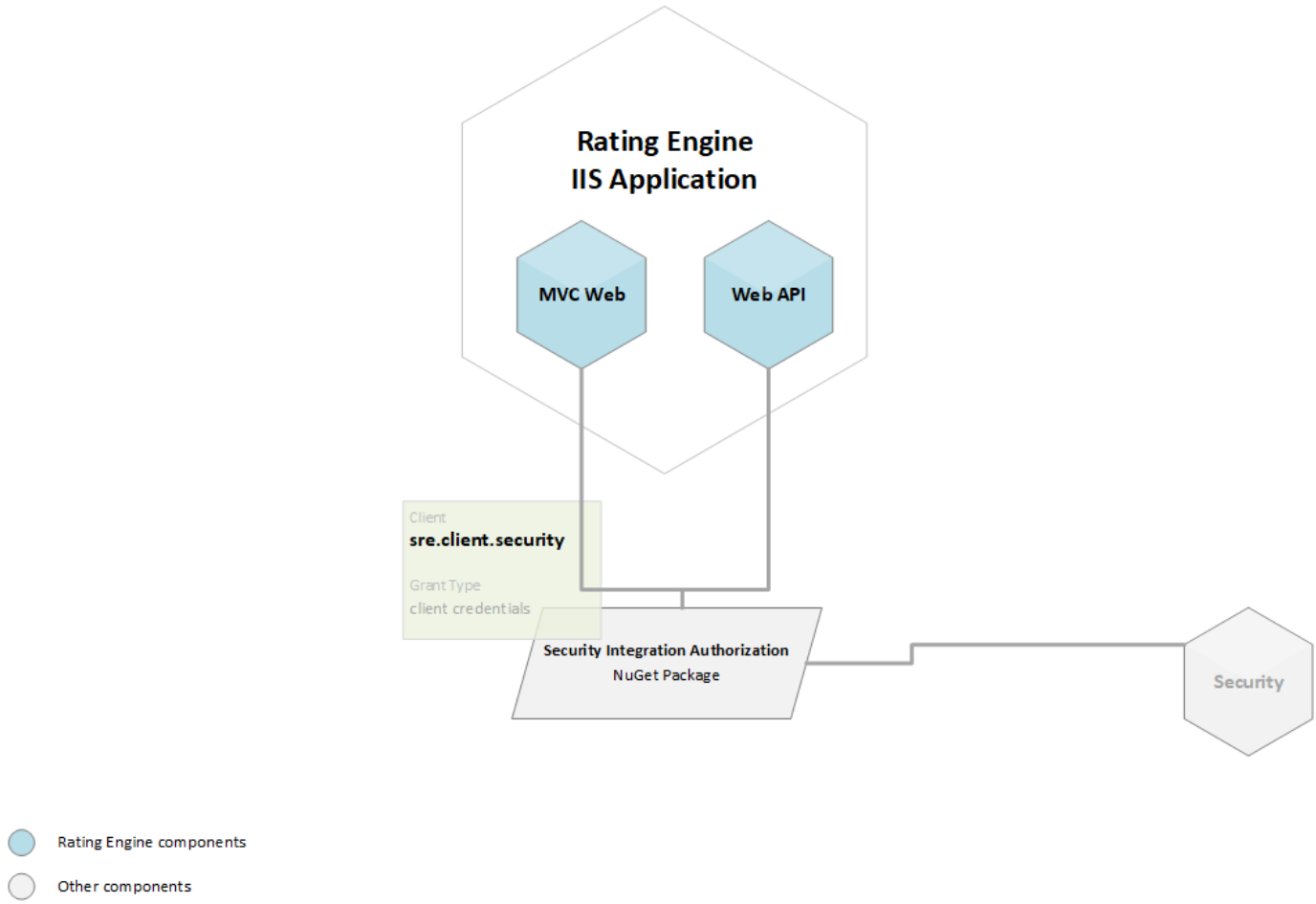
Clients

- **Rating Engine Application Client (`sre.client.app`)** executed in browser (HTML+JavaScript). This HTML client will call to the `sre.resource.app`.
- **Rating Engine API Client (`sre.client.api`)**. This is a proxy provided as a NuGet package for calling to Rating Engine Web API (`sre.resource.api`).
- **Other existing clients**. In this specific case; we have decided to provide to Rating Engine the ability of being deployed with the new security enabled or disabled; so, it is backward compatible with existing clients. However, it will no be recommended to keep using it without security enabled. For those clients, it will be required to integrate: register clients and update clients to manage access tokens.

AUTHORIZATION

The authorization model allows to an application to define the securables for protecting resources, the groups and roles.

Communication with Authorization server is provided by Sequel Security Integration NuGets:



Clients

- **Rating Engine Security Client (`sre.client.security`)**. Used by Secure attribute for calling Security Authorization API (`sec.authorization`, `sec.securityApi`).

Securables

A "securable" defines a resource or action that requires special authorization to be accessed or executed. There are mainly two types of securables: the built-in securables defined in the system, and securables created by the user. The built-in securables are the ones we need to define at this moment.

There are some patterns that we have been following for creating securables. In general, we have found different scenarios where securables are applied:

- **Protect APIs**. Based on the maturity of the API (see Richardson Maturity Model) we are protecting we can apply below rules:
- Level-2, HTTP verbs:
  - Define one securable per resource
  - Assign action to http verb for pure :
  - POST - Create action
  - PUT - Update action
  - GET - Read action
  - DELETE - Delete action
- Level-1, API resources:
  - Define one securable per resource
  - Determine which is the right securable action for each action.
- Level-0, RPC under HTTP. This is when we are using HTTP as a transport system for remote interactions, but without using any of the mechanisms of the web.
  - Lack of order forces to determine action per action the proper securable and action associated.
- **Protect data**. In other scenario, we want to apply some specific securables to the data itself, not to the action. A sample of this is in Workflow, where you can assign a specific securable to a given workflow task, transition,... In this scenario, there are no rules and must be a decision that each analysis will take while designing the security model for the application.

STEP 3. Design Authorization: Securables Define securables required for the Vanilla configuration.

After analysing the Rating Engine requirements, we have detected that we just need to protect some APIs. So, we have listed all controllers and actions

RatingEngineController

Action controller	*Verb*	*Allow Anonymous*	*Securable*	*Action*
AccessDenied	*GET*	*TRUE*	-	-
Index			*Engine*	*Read*
Details			*Engine*	*Read*
Promote			*EnginePromote*	*Update*
Create			*Engine*	*Create*
Create	*POST*		*Engine*	*Create*
Edit			*Engine*	*Update*
Edit	*POST*		*Engine*	*Update*
Delete			*Engine*	*Delete*
DeleteConfirmed	*POST*		*Engine*	*Delete*

EngineModelController

Action controller	Verb	Allow Anonymous	Securable	Action
Details			Model	Read
Help			Model	Read
Create	POST		Model	Create
Download			Model	Read
Edit			Model	Update
Edit	POST		Model	Update
Delete			Model	Delete
DeleteConfirmed	POST		Model	Delete
Execute	POST		ModelManualTest	Read

WhatIfController (aka Analytics)

Action controller	Verb	Allow Anonymous	Securable	Action
Index			Analytics	Read
CompareList	POST		Analytics	Read
Compare			Analytics	Read
GetAllDataCompare		Analytics	Read	
GetAllData			Analytics	Read

Based on this analysis, the securables that we will require for Sequel Rating Engine are:

Key	Name		Create Allowed	Read Allowed	Update Allowed	Delete Allowed
SRE.Engine	Engine		Yes	Yes	Yes	Yes
SRE.EnginePromote	Engine promotion	Promotes to live	No	No	Yes	No
SRE.Model	Engine model		Yes	Yes	Yes	Yes
SRE.ModelManualTest	Engine model manual test		No	Yes	No	No
SRE.Analytics	Analytics		No	Yes	No	No

TIP In some applications, we have developed unit test to ensure all actions at all controllers are protected as expected.

Groups and Roles

Groups are used for defining ACLs; while Roles defines set of permissions. At design time, we need to understand the ACLs requirements of our application and also the different basic roles required by our application. At this stage, we are looking at defining a Vanilla configuration for our application; a base configuration that it will be valid for all implementations. Later, this configuration can be customized by the clients or our support team; however, providing a zero configuration required the first day that you will run your application is required.

There are some requirements and also some common patterns used in Workflow, Product Builder, Origin and Claims that we describe below:

- **Public group**. All applications are required to define a `[ApplicationKey].Public` group as this is the default value used by the Security Service when the group is not provided. In general, existing application are protecting all endpoints offered to UI base clients with the `[ApplicationKey].Public` group.
- **Roles**. Some basic roles are provided for each application to facilitate the configuration; like: `[ApplicationKey].Configurator` or `[ApplicationKey].Standard`. But, this is up to each product team to define it.

STEP 4. Design Authorization: Groups & Roles Define groups required for the Vanilla configuration.

In our sample, we will keep it simple and we will define a `SRE.Public` group and a `SRE.Configurator` and `SRE.Tester` roles.

`SRE.Public` group: At Sequel Rating Engine there is no concept of ACLs, so having a single group is enough to cover our requirements.

`SRE.Configurator` role: Models a full access administrator user; this is the single and current role that exists at rating engine. Permissions for this role will be:

Key	*Create*	*Read*	*Update*	*Delete*
SRE.Engine	*Yes*	*Yes*	*Yes*	*Yes*
SRE.EnginePromote			*Yes*	
SRE.Model	*Yes*	*Yes*	*Yes*	*Yes*
SRE.ModelManualTest		*Yes*		
SRE.Analytics		*Yes*		

`SRE.Tester` role: Models a new role that represents a user that cannot create or change ratings or rating models; and the action that can do is execute manual test of rating engines (so, also can read information). Permissions for this role will be:

Key	*Create*	*Read*	*Update*	*Delete*
SRE.Engine		*Yes*		
SRE.Model		*Yes*		
SRE.ModelManualTest		*Yes*		

User Types

User Types provides a mechanism for grouping users; allowing to assign a user to a maximum of one user type per application. There are no rules related to security or permissions associated to user types.

Create security package definition

STEP 5. Create a security package definition for the application.

OVERVIEW

Security configuration can be imported and exported using a format based on folder and json files:

- **[ApplicationKey]** folder. Root folder, named with the application key
- **Application.json**. File with application definition.
- **Authentication** folder. Authentication folder contains all settings related to resources and clients; it's highly dependant of the environment where it will be deployed, as contains URI relative to this environment.
  - **ApiResources.json**. File with definition of API resources.
  - **Clients.json**. File with definition of clients.
- **Authorization** folder. Contains all settings for authorization concepts like:
  - **Groups.json**.
  - **Roles.json**.
  - **Securables.json**.
  - **UserTypes.json**.

APPLICATION

Create a folder and named with the ApplicationKey; in our sample SRE.

Inside this new folder, add a new file called Application.json and populate with below information:

```
{
  "Key": "[ApplicationKey]",
  "Code": "[ApplicationKey]",
  "Name": "[ApplicationName]"
}
```

In our sample, for Rating Engine; this file will look like:

```
{
  "Key": "SRE",
  "Code": "SRE",
  "Name": "Sequel Rating Engine"
}
```

- ApplicationKey: Unique key to refer an application. Key and Code must contain the same value.
- ApplicationName: More human friendly name used in UI when referring to this application

AUTHENTICATION

Under the application folder create a new folder `Authentication`. The files we need to model in this section have been defined to easily import and export information required by Identity Server to work.

API Resources: ApiResources.json

At the `Authentication` folder create a new file `ApiResources.json`. In this file, we will model the resources we want to protect in our service. This file contains a list of `Api resources` definition. And API resource object looks like:

```
  {
    "Enabled": true,
    "Name": [ApiResourceName],
    "DisplayName": [ApiResourceDisplayName],
    "Description": [ApiResourceDescription],
    "UserClaims": [],
    "ApiSecrets": [],
    "Scopes": [
      {
        "Name": [ScopeName],
        "DisplayName": [ScopeDisplayName],
        "Description": [ScopeDescription],
        "Required": true,
        "Emphasize": true,
        "ShowInDiscoveryDocument": true,
        "UserClaims": []
      }
```

```
    ]
  }
```

For defining a resource we need:

- ApiResourceName: Unique name assigned to this resource. Keep in mind the name convention.
- ApiResourceDisplayName: Display name for the resource. Not used yet, but we will use in future when the Security Administration UI will allow to manage resources.
- ApiResourceDescription: Description of the resource. Not used yet

And, for each scope:

- ScopeName: Unique name assigned to this scope. Keep in mind the name convention.
- ScopeDisplayName: Display name for the scope. Not used yet, but we will use in future when the Security Administration UI will allow to manage scopes.
- ScopeDescription: Description of the scope. Not used yet

For our sample in Rating Engine, the file will look like:

```
[
  {
    "Enabled": true,
    "Name": "sre.resource.api",
    "DisplayName": "Rating Engine API resource",
    "Description": "Rating Engine API resource",
    "UserClaims": [],
    "ApiSecrets": [],
    "Scopes": [
      {
        "Name": "sre.resource.api.full_access",
        "DisplayName": "Full access to Rating Engine API",
        "Description": "Full access to Rating Engine API",
        "Required": true,
        "Emphasize": true,
        "ShowInDiscoveryDocument": true,
        "UserClaims": []
      }
    ]
  },
  {
    "Enabled": true,
    "Name": "sre.resource.app",
    "DisplayName": "Rating Engine Application resource",
    "Description": "Rating Engine Application resource",
    "UserClaims": [],
    "ApiSecrets": [],
    "Scopes": [
      {
        "Name": "sre.resource.app.user_login",
        "DisplayName": "User login to Rating Engine Application",
        "Description": "User login to Rating Engine Application",
        "Required": true,
        "Emphasize": true,
        "ShowInDiscoveryDocument": true,
        "UserClaims": []
      }
    ]
  }
]
```

Clients: Clients.json

At the `Authentication` folder create a new file `Clients.json`. In this file, we will model the client we want to allow access to our service as part of our Vanilla configuration; keep it mind that later when designing a given solution other clients could be required and can be added. Configuring clients is so far the most complex task we will need to tackle in this process.

Currently, we have been configuring four types of clients that we will describe in the next sections and we will us patterns for configuring new ones. In general, those clients are based on different grant types (flows). It's true there is no limitation to allow the same client to use different grant types; however, in general, we have been using a single grant type per client.

Before going into the details, we will describe the scenarios we have already covered:

SPA web application: based on authorization code flow

In a SPA application we have a browser-based javascript client for user authentication and delegated access. This client uses the so called *authorization code flow* (recommended with `PKCE` enabled) to request an access token from Javascript. In this type of clients the user will login to IdentityServer, invoke the web API with an access token issued by IdentityServer, and logout of IdentityServer. All of this will be driven from the JavaScript running in the browser with the help of the `oidc-client`, or any other library that implements the OpenID Connect protocol.

As a sample, we recommend to look at Security Admin site, as this is a pure SPA done with React.

More information at:

Identity Server documentation about Adding a JavaScript client

MVC web application: based on hybrid flow

Interactive server side (like MVC) applications use the hybrid flow. This flow gives you the best security because the access tokens are transmitted via back-channel calls only (and gives you access to refresh tokens). Our sample uses this system and we provide NuGet packages for integrate it easily.

APIs for machine-to-machine communications: based on client_credentials flow

In this scenario no interactive user is present - a service (aka client) wants to communicate with an API (aka scope). The flow type used in `client credentials`. The client interacts with the API using a client id and a secret.

Swagger

Most of our API have been configured to use Swagger, so we have to consider Swagger a client: it will depend the type of API we are accessing we will have to configure this client for using a flow type based on user interaction (implicit or hybrid) or no interactive user (client credentials)

The Sequel Rating Engine clients.json

For our sample, the clients file will look like:

```
[
  {
    "BackChannelLogoutSessionRequired": true,
    "AlwaysIncludeUserClaimsInIdToken": true,
    "IdentityTokenLifetime": 300,
    "AccessTokenLifetime": 3600,
    "AuthorizationCodeLifetime": 300,
    "AbsoluteRefreshTokenLifetime": 2592000,
    "SlidingRefreshTokenLifetime": 1269600,
    "ConsentLifetime": null,
    "RefreshTokenUsage": 1,
    "UpdateAccessTokenClaimsOnRefresh": false,
    "RefreshTokenExpiration": 1,
    "AccessTokenType": 0,
    "EnableLocalLogin": true,
    "IdentityProviderRestrictions": [],
    "IncludeJwtId": true,
    "Claims": [],
    "AlwaysSendClientClaims": true,
    "ClientClaimsPrefix": "client_",
    "PairWiseSubjectSalt": null,
    "AllowedScopes": [
      "sre.resource.api.full_access"
    ],
    "AllowOfflineAccess": true,
    "Properties": {},
    "BackChannelLogoutUri": null,
    "Enabled": true,
    "ClientId": "sre.client.api",
    "ProtocolType": "oidc",
    "ClientSecrets": [
      {
        "Description": "Rating Engine API",
        "Value": "__ClientSecretHash__",
        "Expiration": null,
        "Type": "SharedSecret"
      }
    ],
    "RequireClientSecret": false,
    "ClientName": "Rating Engine API",
    "ClientUri": null,
    "LogoUri": null,
    "AllowedCorsOrigins": [],
    "RequireConsent": false,
    "AllowedGrantTypes": [
      "client_credentials"
    ],
    "RequirePkce": false,
    "AllowPlainTextPkce": false,
```

```
    "AllowAccessTokensViaBrowser": true,
    "RedirectUris": [],
    "PostLogoutRedirectUris": [],
    "FrontChannelLogoutUri": null,
    "FrontChannelLogoutSessionRequired": true,
    "AllowRememberConsent": true
  },
  {
    "BackChannelLogoutSessionRequired": true,
    "AlwaysIncludeUserClaimsInIdToken": true,
    "IdentityTokenLifetime": 300,
    "AccessTokenLifetime": 300,
    "AuthorizationCodeLifetime": 300,
    "AbsoluteRefreshTokenLifetime": 0,
    "SlidingRefreshTokenLifetime": 1296000,
    "ConsentLifetime": null,
    "RefreshTokenUsage": 1,
    "UpdateAccessTokenClaimsOnRefresh": false,
    "RefreshTokenExpiration": 0,
    "AccessTokenType": 0,
    "EnableLocalLogin": true,
    "IdentityProviderRestrictions": [],
    "IncludeJwtId": true,
    "Claims": [],
    "AlwaysSendClientClaims": true,
    "ClientClaimsPrefix": "client_",
    "PairWiseSubjectSalt": null,
    "AllowedScopes": [
      "openid",
      "profile",
      "sre.resource.app.user_login"
    ],
    "AllowOfflineAccess": true,
    "Properties": {},
    "BackChannelLogoutUri": null,
    "Enabled": true,
    "ClientId": "sre.client.app",
    "ProtocolType": "oidc",
    "ClientSecrets": [
      {
        "Description": "Rating Engine",
        "Value": "__ClientSecretHash__",
        "Expiration": null,
        "Type": "SharedSecret"
      }
    ],
    "RequireClientSecret": true,
    "ClientName": "Rating Engine Application",
    "ClientUri": null,
    "LogoUri": null,
    "AllowedCorsOrigins": [],
    "RequireConsent": false,
    "AllowedGrantTypes": [
      "hybrid"
    ],
    "RequirePkce": false,
    "AllowPlainTextPkce": false,
    "AllowAccessTokensViaBrowser": true,
    "RedirectUris": [
      "__RatingEngineUrlExternal__/signin-oidc"
    ],
    "PostLogoutRedirectUris": [
      "__RatingEngineUrlExternal__/signout-callback-oidc"
    ],
    "FrontChannelLogoutUri": null,
    "FrontChannelLogoutSessionRequired": true,
    "AllowRememberConsent": true
  },
  {
    "BackChannelLogoutSessionRequired": true,
    "AlwaysIncludeUserClaimsInIdToken": true,
    "IdentityTokenLifetime": 300,
    "AccessTokenLifetime": 3600,
    "AuthorizationCodeLifetime": 300,
    "AbsoluteRefreshTokenLifetime": 2592000,
    "SlidingRefreshTokenLifetime": 1269600,
    "ConsentLifetime": null,
    "RefreshTokenUsage": 1,
    "UpdateAccessTokenClaimsOnRefresh": false,
    "RefreshTokenExpiration": 1,
    "AccessTokenType": 0,
    "EnableLocalLogin": true,
    "IdentityProviderRestrictions": [],
    "IncludeJwtId": true,
    "Claims": [],
    "AlwaysSendClientClaims": true,
    "ClientClaimsPrefix": "client_",
    "PairWiseSubjectSalt": null,
    "AllowedScopes": [
      "sec.api",
      "sec.authorization"
    ],
    "AllowOfflineAccess": true,
    "Properties": {},
```

```
    "BackChannelLogoutUri": null,
    "Enabled": true,
    "ClientId": "sre.client.security",
    "ProtocolType": "oidc",
    "ClientSecrets": [],
    "RequireClientSecret": false,
    "ClientName": "Rating Engine Authorization ",
    "ClientUri": null,
    "LogoUri": null,
    "AllowedCorsOrigins": [],
    "RequireConsent": false,
    "AllowedGrantTypes": [
      "client_credentials"
    ],
    "RequirePkce": false,
    "AllowPlainTextPkce": false,
    "AllowAccessTokensViaBrowser": true,
    "RedirectUris": [],
    "PostLogoutRedirectUris": [],
    "FrontChannelLogoutUri": null,
    "FrontChannelLogoutSessionRequired": true,
    "AllowRememberConsent": true
  }
]
```

AUTHORIZATION

As described at the beginning of this section, we need to define four files with Securables, Groups, Roles and User Types for the Vanilla configuration of our application. Under the application folder create a new folder `Authorization`.

TIP: Authorization configuration can be modelled using Administration UI; exporting the configuration and later marking all entries as IsSystem.

Securables

The `Securables.json` contains a list of securable objects:

```
  {
    "Key": [SecurableKey],
    "ApplicationKey": [ApplicationKey],
    "Code": [Code],
    "Name": [Name],
    "Description": [Description],
    "IsSystem": true,
    "IsCreateAllowed": [IsCreateAllowed]],
    "CreateDescription": [CreateDescription],
    "IsReadAllowed": [IsReadAllowed],
    "ReadDescription": [ReadDescription],
    "IsUpdateAllowed": [IsUpdateAllowed],
    "UpdateDescription": [UpdateDescription],
    "IsDeleteAllowed": [IsDeleteAllowed],
    "DeleteDescription": [DeleteDescription]
  }
```

As all fields have been already described above, we will go directly to describe this file for Rating Engine, the securable file will look like:

```
[
  {
    "Key": "SRE.Engine",
    "ApplicationKey": "SRE",
    "Code": "Engine",
    "Name": "Engine",
    "Description": "Engine",
    "IsSystem": true,
    "IsCreateAllowed": true,
    "CreateDescription": null,
    "IsReadAllowed": true,
    "ReadDescription": null,
    "IsUpdateAllowed": true,
    "UpdateDescription": null,
    "IsDeleteAllowed": true,
    "DeleteDescription": null
  },
  {
    "Key": "SRE.EnginePromote",
    "ApplicationKey": "SRE",
    "Code": "EnginePromote",
    "Name": "Engine Promote",
    "Description": "Promote engine models to live",
    "IsSystem": true,
    "IsCreateAllowed": false,
    "CreateDescription": null,
    "IsReadAllowed": false,
    "ReadDescription": null,
    "IsUpdateAllowed": true,
    "UpdateDescription": null,
    "IsDeleteAllowed": false,
```

```
    "DeleteDescription": null
  },
  {
    "Key": "SRE.Model",
    "ApplicationKey": "SRE",
    "Code": "Model",
    "Name": "Engine models",
    "Description": "Engine models",
    "IsSystem": true,
    "IsCreateAllowed": true,
    "CreateDescription": null,
    "IsReadAllowed": true,
    "ReadDescription": null,
    "IsUpdateAllowed": true,
    "UpdateDescription": null,
    "IsDeleteAllowed": true,
    "DeleteDescription": null
  },
  {
    "Key": "SRE.ModelManualTest",
    "ApplicationKey": "SRE",
    "Code": "ModelManualTest",
    "Name": "Engine Model manual test",
    "Description": "Allows to test models manually from administration site",
    "IsSystem": true,
    "IsCreateAllowed": false,
    "CreateDescription": null,
    "IsReadAllowed": true,
    "ReadDescription": null,
    "IsUpdateAllowed": false,
    "UpdateDescription": null,
    "IsDeleteAllowed": false,
    "DeleteDescription": null
  },
  {
    "Key": "SRE.Analytics",
    "ApplicationKey": "SRE",
    "Code": "Analytics",
    "Name": "Analytics",
    "Description": "Analytics",
    "IsSystem": true,
    "IsCreateAllowed": false,
    "CreateDescription": null,
    "IsReadAllowed": true,
    "ReadDescription": null,
    "IsUpdateAllowed": false,
    "UpdateDescription": null,
    "IsDeleteAllowed": false,
    "DeleteDescription": null
  }
]
```

Groups

The `group.json` file contains a list of groups; a group is defined as:

```
[
  {
    "Key": [Key],
    "ApplicationKey": [ApplicationKey],
    "Code": [Code],
    "Name": [Name],
    "AutomaticallyCreated": false,
    "IsSystem": true
  }
]
```

Fields in the group object have been already defined.

For Sequel Rating Engine, the group file will look like:

```
[
  {
    "Key": "SRE.Public",
    "ApplicationKey": "SRE",
    "Code": "Public",
    "Name": "Public",
    "AutomaticallyCreated": false,
    "IsSystem": true
  }
]
```

Roles

At roles files, we will define roles and also the list of permissions. For Rating Engine, this will look like:

```
[
  {
    "Key": "SRE.Configurator",
    "ApplicationKey": "SRE",
    "Code": "Configurator",
    "Name": "Configurator",
    "IsSystem": true,
    "Permissions": [
      {
        "SecurableKey": "SRE.Engine",
        "Create": true,
        "Read": true,
        "Update": true,
        "Delete": true
      },
      {
        "SecurableKey": "SRE.EnginePromote",
        "Create": false,
        "Read": true,
        "Update": false,
        "Delete": false
      },
      {
        "SecurableKey": "SRE.Model",
        "Create": true,
        "Read": true,
        "Update": true,
        "Delete": true
      },
      {
        "SecurableKey": "SRE.ModelManualTest",
        "Create": false,
        "Read": true,
        "Update": false,
        "Delete": false
      },
      {
        "SecurableKey": "SRE.Analytics",
        "Create": false,
        "Read": true,
        "Update": false,
        "Delete": false
      }
    ]
  },
  {
    "Key": "SRE.Tester",
    "ApplicationKey": "SRE",
    "Code": "Tester",
    "Name": "Tester",
    "IsSystem": true,
    "Permissions": [
      {
        "SecurableKey": "SRE.Engine",
        "Create": false,
        "Read": true,
        "Update": false,
        "Delete": false
      },
      {
        "SecurableKey": "SRE.Model",
        "Create": false,
        "Read": true,
        "Update": false,
        "Delete": false
      },
      {
        "SecurableKey": "SRE.ModelManualTest",
        "Create": false,
        "Read": true,
        "Update": false,
        "Delete": false
      }
    ]
  }
]
```

User Types

User types is only used by Origin application. An empty file looks like:

```
[]
```

IMPORT PACKAGE TO SECURITY SERVICE

Once we have defined the files for configuring authentication and authorization following the described folder schema we can import them using the `sequel-security` tool.

Configure Security

Security configuration is provided by several NuGet packages:

- **Sequel Security Integration (** `Sequel.Security.Integration` **)** contains methods for create ISecuritySettings configuration that will be use by the rest of the package.
- **Sequel Security Integration OWIN (** `Sequel.Security.Integration.Owin` **)** contains methods for setting up authentication schemas (Cookie and Bearer) in applications using OWIN.
- **Sequel Security Integration NetCore (** `Sequel.Security.Integration.NetCore` **)** contains methods for setting up authentication schema (Cookie and Bearer) in net core applications.
- **Sequel Security Integration NetCore Http (** `Sequel.Security.Integration.NetCore.Http` **)** contains methods for setting up the authorization aspect in net core applications.
- **User Session Avatar (** `@sequel/sequel.web.usersessionavatar` **)** contains extended session management and a nice looking user avatar for your web app.

CREATING SECURITY SETTINGS

First of all it's necessary to create ISecuritySettings interface with the security configuration the application. Sequel Security Integration provided all methods for it.

For Rating Engine the configuration is stored in the appsettings.json:

```
"SecuritySettings": {
   "IsSecurityEnabled": true,
   "PermissionsCacheDurationInMinutes": 5,
   "ApplicationSettings": {
     "InternalUrl": [######],
     "ExternalUrl": [######]
   },
   "SessionSettings": {
     "TimeoutInMinutes": 15,
     "TimeoutWarningInMinutes": 3
   },
   "AuthorizationServerSettings": {
     "InternalUrl": [######],
     "ExternalUrl": [######]
   },
   "AuthenticationServerSettings": {
     "InternalUrl": [######],
     "ExternalUrl": [######]
   },
   "CookieSettings": {
     "IsSecureModeEnabled": true
   },
   "RatingEngineApplicationClient": {
     "ClientId": "sre.client.app",
     "ClientSecret": [######],
     "Scopes": "sre.resource.app.user_login"
   },
   "RatingEngineSecurityClient": {
     "ClientId": "sre.client.security",
     "ClientSecret": [######],
     "Scopes": "sec.api sec.authorization"
   },
   "RatingEngineApi": {
     "ApiName": "sre.resource.api"
   }
 }
```

Configuration is created reading that configuration and executing next methods:

```
    protected ISecuritySettings GetSecuritySettings(Settings.Security.RatingEngineSecuritySettings ratingEngineSecuritySettings)
    {
      var ratingEngineApplicationClient = SecurityClientSettingsExtensions
        .CreateClientSettings(ratingEngineSecuritySettings.RatingEngineApplicationClient.ClientId,
ratingEngineSecuritySettings.RatingEngineApplicationClient.ClientSecret)
        .IncludeScopes(ratingEngineSecuritySettings.RatingEngineApplicationClient.Scopes)
        .IncludeDefaultRedirections(ratingEngineSecuritySettings.ApplicationSettings.InternalUrl, ratingEngineSecuritySettings.ApplicationSettings.ExternalUrl);

      var ratingEngineAuthorizationClient = SecurityClientSettingsExtensions
        .CreateClientSettings(ratingEngineSecuritySettings.RatingEngineSecurityClient.ClientId, ratingEngineSecuritySettings.RatingEngineSecurityClient.ClientSecret)
        .IncludeScopes(ratingEngineSecuritySettings.RatingEngineSecurityClient.Scopes)
        .IncludeDefaultRedirections(ratingEngineSecuritySettings.ApplicationSettings.InternalUrl, ratingEngineSecuritySettings.ApplicationSettings.ExternalUrl);

      var ratingEngineApiResource = SecurityApiResourceSettingsExtensions.CreateApiResourceSettings(ratingEngineSecuritySettings.RatingEngineApi.ApiName);

      var securitySettings = SecuritySettingsExtensions.UseSecuritySettings(ratingEngineSecuritySettings.ApplicationSettings.ApplicationKey)
        .AddSessionSettings(
```

```
            timeoutInMinutes: ratingEngineSecuritySettings.SessionSettings.TimeoutInMinutes,
            timeoutWarningInMinutes: ratingEngineSecuritySettings.SessionSettings.TimeoutWarningInMinutes)
          .AddCookieSettings(
            ssoCookieDomain: string.Empty,
            isSecureModeEnabled: ratingEngineSecuritySettings.CookieSettings.IsSecureModeEnabled)
          .AddAuthorizationSettings(
            authorizationServerUri: ratingEngineSecuritySettings.AuthorizationServerSettings.InternalUrl,
            externalAuthorizationServerUri: ratingEngineSecuritySettings.AuthorizationServerSettings.ExternalUrl)
          .AddAuthenticationSettings(
            authenticationServerUri: ratingEngineSecuritySettings.AuthenticationServerSettings.InternalUrl,
            externalAuthenticationServerUri: ratingEngineSecuritySettings.AuthenticationServerSettings.ExternalUrl)
          .AddClientSettings(ratingEngineApplicationClient)
          .AddClientSettings(ratingEngineAuthorizationClient)
          .AddApiResourceSettings(ratingEngineApiResource)
          .ActivateAuthenticationConditionally(ratingEngineSecuritySettings.IsSecurityEnabled)
          .ActivateAuthorizationConditionally(ratingEngineSecuritySettings.IsSecurityEnabled);

        return securitySettings;
    }
```

AUTHENTICATION MIDDLEWARE

During application start up we need to configure the middleware which will ensure the identity of the requester and will force the redirections to Authentication server for authentications. It's mandatory create a ISecuritySettings instance before to be able to use all the methods.

.net framework applications

`Sequel.Security.Integration.Owin` contains method for configure authentication for MVC and Web API controllers in applications on .NET framework using OWIN. We can protect controllers using Cookie Authentication (using user/password as challenge method) indicating the Client and the context's conditions to accomplish and, in a similar way, we can protect controllers using Bearer Authentication indicating the Api Name and the the context's conditions to accomplish.

Supposing that Rating Engine were a net framework application with OWIN we could do this this way:

```
// app is IAppBuilder
 app.UseSequelSecurity(securitySettings)
  .UseCookieAuthentication(applicationSettings.SecuritySettings.RatingEngineApplicationClient.ClientId, context => !context.Request.Path.Value.ToLower().Contains("/
api/"))
  .UseBearerAuthentication(applicationSettings.SecuritySettings.RatingEngineApi.ApiName)
```

.net core applications

`Sequel.Security.Integration.NetCore` contains method for configure authentication schemas for MVC and Web API controllers.

In Sequel Rating Engine it's used Cookie Authentication (using user/password as challenge method) for MVC controllers and Bearer token authentication for Web API.

```
// startup.cs
 services.AddSequelSecurityAuthentication(securitySettings)
  .AddCookieAuthentication(applicationSettings.SecuritySettings.RatingEngineApplicationClient.ClientId)
  .AddBearerAuthentication(AuthenticationConstants.BearerAuthenticationScheme, applicationSettings.SecuritySettings.RatingEngineApi.ApiName);
  .AddSequelSecurityAuthorization(securitySettings);
```

For MVC controllers:

```
    [Authorize(Policy = AuthorizationConstants.IsSecurityEnabledPolicy)]
    [Authorize]
    public abstract class SecuredMvcController : Controller
    {

    }
```

For API controllers:

```
    [Authorize(Policy = AuthorizationConstants.IsSecurityEnabledPolicy)]
    [Authorize(AuthenticationSchemes = AuthenticationConstants.BearerAuthenticationScheme)]
    public abstract class ApiControllerBase : ControllerBase
    {


    }
```

Single sign-in & sign-out

For a better single sign-in and sign-out experience all applications must be deployed in the same domain. As we will be able to automatically detect single sing-in and out events and force the application to react; all information about this subject can be found in Single Sign-Out document. Also checkout User Session Avatar documentation for SPAs or similar web apps.

When applications are not in the same domain, single sign-out is not possible; and single sign-in works; however, we can not detect sign-in and out events from other applications.

AUTHORIZATION

During application start up we need to make additional configuration for interactions with Authorization server and be able to check current user permissions.

For Rating Engine this methods are included in `Sequel.Security.Integration.NetCore.Http`. It's necessary add the client that will be used for communications with Authorization server, enable Permissions' cache to improve the performance and a page for redirect users when they haven't the permissions required:

```
// startup.cs
 services
  ...
  .AddSequelSecurityAuthorization(securitySettings)
  .AddSecureAttribute(applicationSettings.SecuritySettings.RatingEngineSecurityClient.ClientId)
  .AddPermissionsCache(applicationSettings.SecuritySettings.PermissionsCacheDurationInMinutes)
  .AddNoAuthorizedAction(new ViewResult { ViewName = "NoAuthorized" });
```

Consuming effective permissions

For consuming effective permissions it's necessary decorate controllers/methods with the `SecureAttribute` indicating what Securable and Action is required for the access. In Rating Engine a static class contains all Securable names for a better reference:

```
public static class Securables
    {
        public const string Engine = "Engine";
        public const string EnginePromote = "EnginePromote";
        public const string Model = "Model";
        public const string ModelManualTest = "ModelManualTest";
        public const string Analytics = "Analytics";
    }
```

```
[Secure(Securables.Engine, Action.Read)]
public IActionResult Index()
    {

    }
```

API

Internal APIs just need to be protected for clients using client_credentials flow; and authorization is not required. When an API is publicly exposed (read it in the API gateway) we have to apply authorization based on scopes.

Caching and service bus messages

`Sequel.Security.Integration.NetCore.Http` contains a cache used by Secure attribute to store for some minutes (5 by default) the responses obtained from Authorization server to know the permissions of a user. That means if there is a permissions change for that user in the Authorization server, applications using this cache won't be notified until cache expiration.

When there is a permissions change for a user, Security API sends a message to the bus notifying this. Applications using it's own implementation of Secure attribute with a cache can use a consumer listening a queue connected to the same exchange that SecurityAPI waiting for a `Sequel.Security.MessageBus.Contracts.EffectivePermission.Changed.v1.EffectivePermissionChangedMessage` message for invalidating all cache entries for that user.

It's important to be sure that other consumers (like Authorizations server) are not consuming messages from the same queue to avoid inconsistencies between the permissions obtained for our cache and permissions obtained from Authorization server.

Action filters

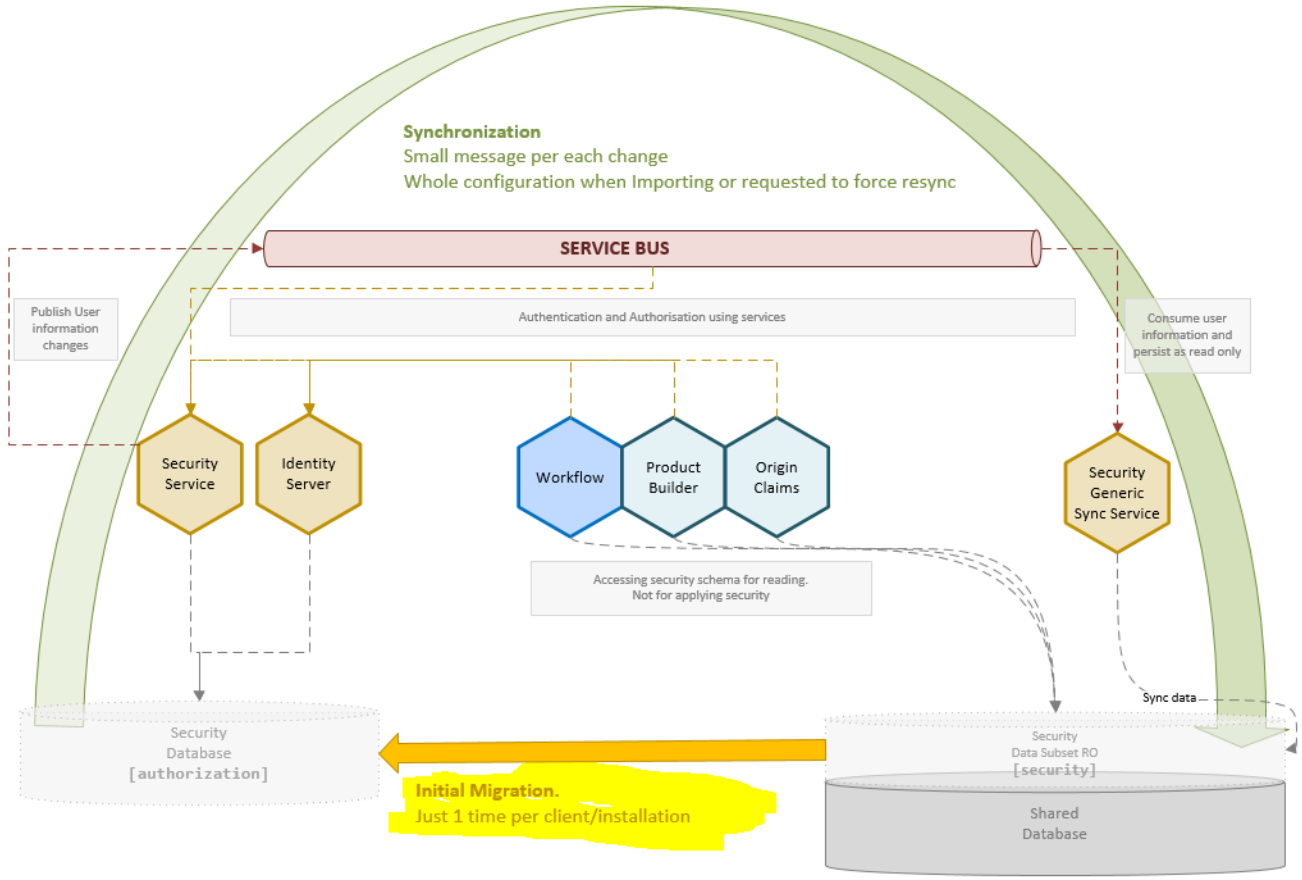
Applications using custom Secure attribute will have an action filter implementing IAuthorizationFilter to include the logic for get user permissions a accept or reject the request. Request will have two execution flows depending on if is a request from a user or a server. If the request is from another server using a Client Credentials flow the don't have to check for permissions if request if from a user we should get the username from the context, call to Authorization server (we'll need a client to get a token a do the request) and finally using de Authorization server's response check the permissions. The pseudo-code logic will be:

```
private readonly string action;
private readonly string securable;

public void OnAuthorizationAsync(AuthorizationFilterContext context)
    {
      string username= GetUserNameFromContext(context);

      if (String.IsNullOrEmpty(username)) //It's a Client Credentials request
        return;

      var permissions = CallAuthorizationServerForUserPermissions(username);

      bool hasPermissions = UserHasPermissionsForSecurableAction(permissions, securable, action);

      if (!hasPermissions)
          context.Result = GetNotAuthorizedActionResult();

    }
```

3.7.3 Migration guide

Introduction

The security service split has created a distributed system and data across different databases. In the past, we have been sorting out the data synchronization from [authorization] to [security]. Now you can normalize shared database for being compatible with the new security system and generate a security configuration package (authorization and users) for initializing the security database.



This document describes the process for migrating the information from [security] to [authorization] using the **sequel-security tool.**

Migration flow

The steps to migrate the data from [security] to [authorization] are the following:

1. Upgrade [security] legacy schema

- Apply latest security schema definition (aka dacpac). Due to some dependencies with Workflow you will need to manually delete [workflow].[UserProfile] view.
- After upgrading the [security] schema recreate manually removed objects: manually or deploying workflow database definition.

2. Check the compatibility level of the legacy database.

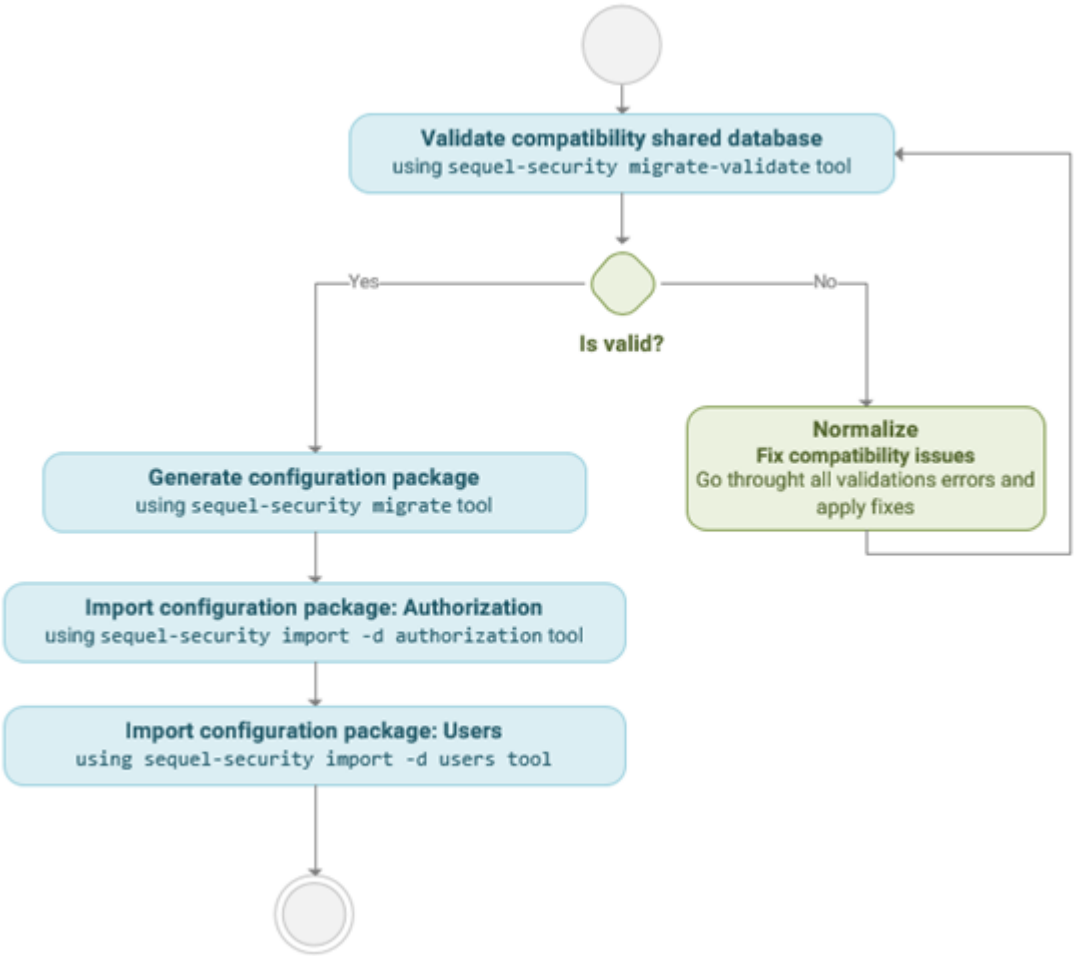
- You can include a JSON file for validating generic columns for Workflow and Product Builder if required

3. Normalize the database fixing the compatibility issues if the compatibility is not valid

4. Generate configuration package files from the legacy database

5. Import authorization entities: groups, roles, permissions, securables and user types

6. Import authorization users and memberships



1. VALIDATE COMPATIBILITY

The list of validations to be applied are:

- InvalidKey: the entity key is not like “{appKey}.{Code}”
- InvalidAppKey: the entity appKey is empty or unknown
- InvalidCode: the entity code is empty or different from the entity key code
- NameRequired: the entity name is empty
- DisplayNameRequired: the group display name is empty
- KeyDuplicated: there are two or more entities with the same key
- MissingPublicGroup: Public group does not exist in the application
- MismatchPermissionApplication: the role and securable of a permission have different appKey.
- MismatchMembershipApplication: the group and role of a membership have different appKey
- SecurableNotFound: a role has a permission whose securable does not exist
- RoleNotFound: a user has a membership whose role does not exist
- GroupNotFound: a user has a membership whose group does not exist
- UsernameNotFound: a membership user does not exist
- UsernameDuplicated: there are two or more users with the same username
- EmailAddressDuplicated: there are two or more users with the same email address
- InvalidUserTypeInUser: a user has an invalid UserType
- UserTypeNotFound: the UserType is not found in the application

The sequel-security tool allows you to generate a report with the issues detected in a legacy database running the sequel-security **migrate-validate** command:

Required parameters:

```
-c|--connection
//Connection string to the database with the legacy [security] schema to be validated.
```

Optional parameters:

```
-r|--references     /* JSON filename with references to the legacy database columns
                       of other applications (WF, PB, Origin, CLM) with security data
                       which will be validated too. */
-v|--verbose        /* Activates verbose mode */
-s|--silence        /* Activate silent mode */
```

Example 1: Validate the users and authorization entities of all applications from a legacy database called LEGACYDB located at DBSERVER server:

```
sequel-security migrate-validate -c "Server=DBSERVER;Database=LEGACYDB;Trusted_Connection=True;MultipleActiveResultSets=true" -v
```

Example 2: Allow other applications (WF, PB, Origin, CLM) to define their columns with security data, so the tool will be able to validate it.

```
sequel-security migrate-validate -c "Server=DBSERVER;Database=LEGACYDB;Trusted_Connection=True;MultipleActiveResultSets=true" -r "JSON_FILENAME"
```

The input parameter with the definition of columns to validate will be passed to the tool as a file in json format example:

```
{
  "Groups": [
    {
      "DbSchema": "workflow",
      "DbTable": "DiaryAction",
      "DbColumn": "assignedGroupId"
    }
  ],
  "Roles": [
    {
      "DbSchema": "workflow",
      "DbTable": "DiaryActionType",
      "DbColumn": "securableId"
    }
```

```
  ],
  "Securables": [
    {
      "DbSchema": "workflow",
      "DbTable": "DiaryActionType",
      "DbColumn": "securableId"
    }
  ],
  "Users": [
    {
      "DbSchema": "workflow",
      "DbTable": "Alert",
      "DbColumn": "userId"
    }
  ],
  "UserTypes": [
    {
      "DbSchema": "workflow",
      "DbTable": "DocumentUserType",
      "DbColumn": "UserTypeId"
    }
  ]
}
```

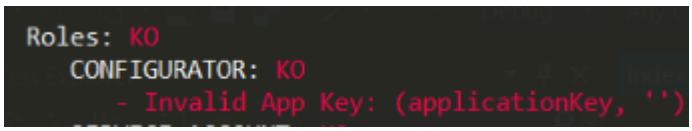
2. NORMALIZE THE DATABASE

You have to fix the validation errors found in the legacy database before the migration process. Several errors validations could be displayed because the entities do not have appKey. This should be the first issue to fix. Once the entities have appKey, many error validations will disappear.

These are the actions to normalize each issue:

InvalidKey: Fill the entity key as “{appKey}.{Code}”

InvalidAppKey: Fill the correct entity appKey. The next example display a role with code but not appKey.



InvalidCode: Fill the entity code like the entity key code

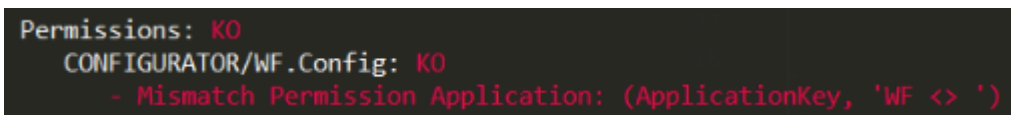
NameRequired: Fill the name

DisplayNameRequired: Fill the group DisplayName. The next example display a group where display name is empty

KeyDuplicated: Make sure the entity key is unique

MissingPublicGroup: Create the Public group in the application.

MismatchPermissionApplication: Fix the appKey both the role and securable of a permission to be the same. The next sample display a permission where role and securable have different appKey.



MismatchMembershipApplication: Fix the appKey both the group and role of a membership to be the same. The next example display a user membership where role and group appKey are different

SecurableNotFound: Find the securable and set the correct appKey. If this does not exist, create it. You can also remove the permission.

RoleNotFound: Create the role or remove the membership

GroupNotFound: Create the group or remove the membership

UsernameNotFound: Create the user or remove the membership

UsernameDuplicated: Make sure the user username is unique

EmailAddressDuplicated: Make sure the user email address is unique

InvalidUserTypeInUser: Create the UserType or remove the UserType assignment.

UserTypeNotFound: Create the UserType.

3. GENERATE MIGRATION PACKAGE

The sequel-security tool allows you to generate a data package by reading [security] schema from a legacy database and mapping to an intermediary data exchange format. The sequel-security **migrate** command performs simple mappings without any kind of transformation:

Required parameters:

```
-c|--connection     /* Connection string to the database with the legacy [security]
                       schema to be migrated. */
-o|--output         /* Output path (folder or zip file) where the migration package
                       will be generated. */
```

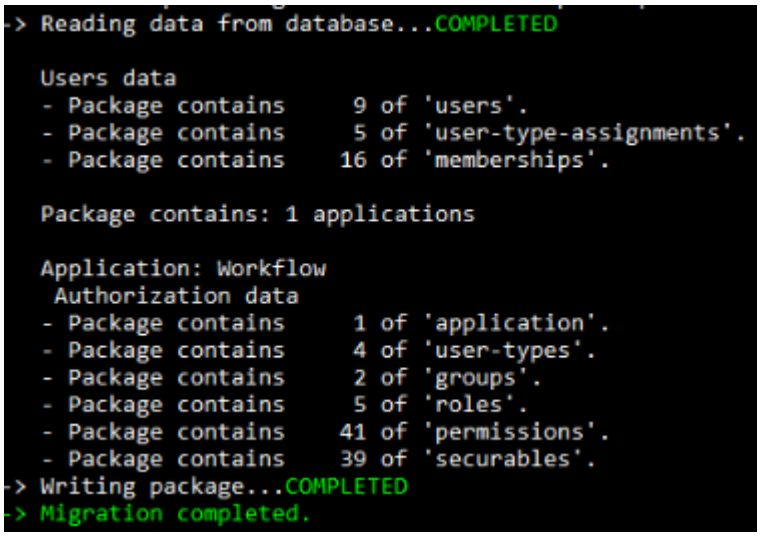
Optional parameters:

```
-a|--application    /* Application key to choose from WF, Sec, Origin, PB, CLM
                       (by default all of them) to be included in the migration package.
                       (multiple allowed)*/
-d|--domain         /* Domain to be exported (all by default).
                       Expected values (multiple allowed): authorization | users */
-val|--validate     /* Validates package before creates output file. */
-v|--verbose        /* Activates verbose mode */
```

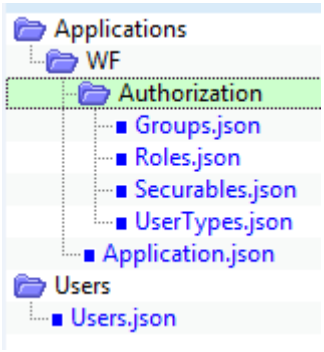
Example 1: Migrate all Workflow entities from a legacy database called LEGACYDB located at DBSERVER server and store it in a ZIP file called WorkflowMigration.zip:

```
sequel-security migrate -a WF -o WorkflowMigration.zip -c "Server=DBSERVER;Database=LEGACYDB;Trusted_Connection=True;MultipleActiveResultSets=true"
```

This is a screenshot of migrate execution result:



And this is the result of the migrate execution: a JSON file per authorization entity and user:



4. IMPORT AUTHORIZATION ENTITIES

The sequel-security **import** command import the data packages into the new Sequel Security Service.

Example: We want to import all Workflow authorization entities into a security database called SECURITYDB located at DBSERVER server from a ZIP file called Package.zip:

```
sequel-security import -a WF -d authorization -i Package.zip -c "Server=DBSERVER;Database=SECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true"
```

This is a screenshot of the result when importing authorization entities:

```
  Parameters:
       -application = WF
       -entity = All
       -connection = Server=TFSSEC153244;Database=SecurityDatabase;Trusted_Connection=True;MultipleActiveResultSets=true
       -input = migrateNORMALIZEDWFFINAL.zip
-> Reading package...COMPLETED

  Package contains: 1 applications

  Application: Workflow
   Authorization data
  - Package contains     1 of 'application'.
  - Package contains     4 of 'user-types'.
  - Package contains     2 of 'groups'.
  - Package contains     5 of 'roles'.
  - Package contains    41 of 'permissions'.
  - Package contains    37 of 'securables'.
-> Writing data into database...COMPLETED
-> Import completed.
```

5. IMPORT USERS

The sequel-security **import** command import the data packages into the new Sequel Security Service.

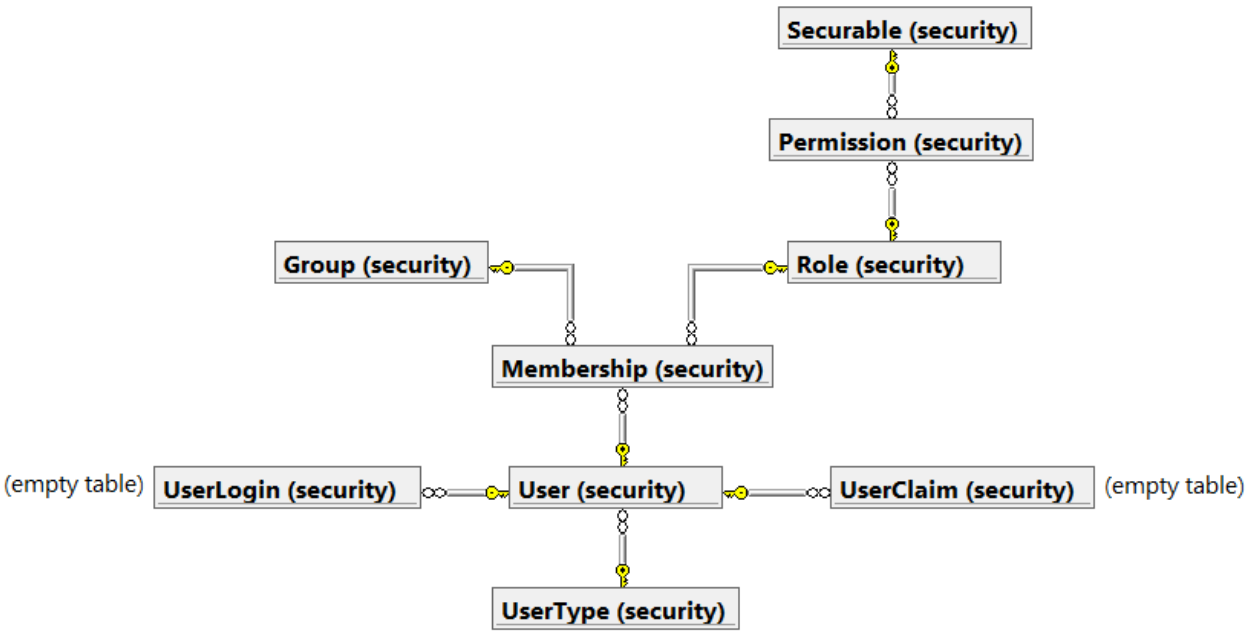
```
  Parameters:
       -application = WF
       -entity = All
       -connection = Server=TFSSEC153244;Database=SecurityDatabase;Trusted_Connection=True;MultipleActiveResultSets=true
       -input = migrateNORMALIZEDWFFINAL.zip
-> Reading package...COMPLETED

  Users data
  - Package contains     5 of 'users'.
  - Package contains     4 of 'user-type-assignments'.
  - Package contains     8 of 'memberships'.
-> Writing data into database...COMPLETED
-> Import completed.
```

3.7.4 Migration guide: database

This document describes how the database to store security data was structured in the legacy design, and how it has been implemented for the new **Sequel Security Server**.

Legacy Security Schema

The legacy security schema in a legacy shared database that contained security data looked like shown on the following diagram:



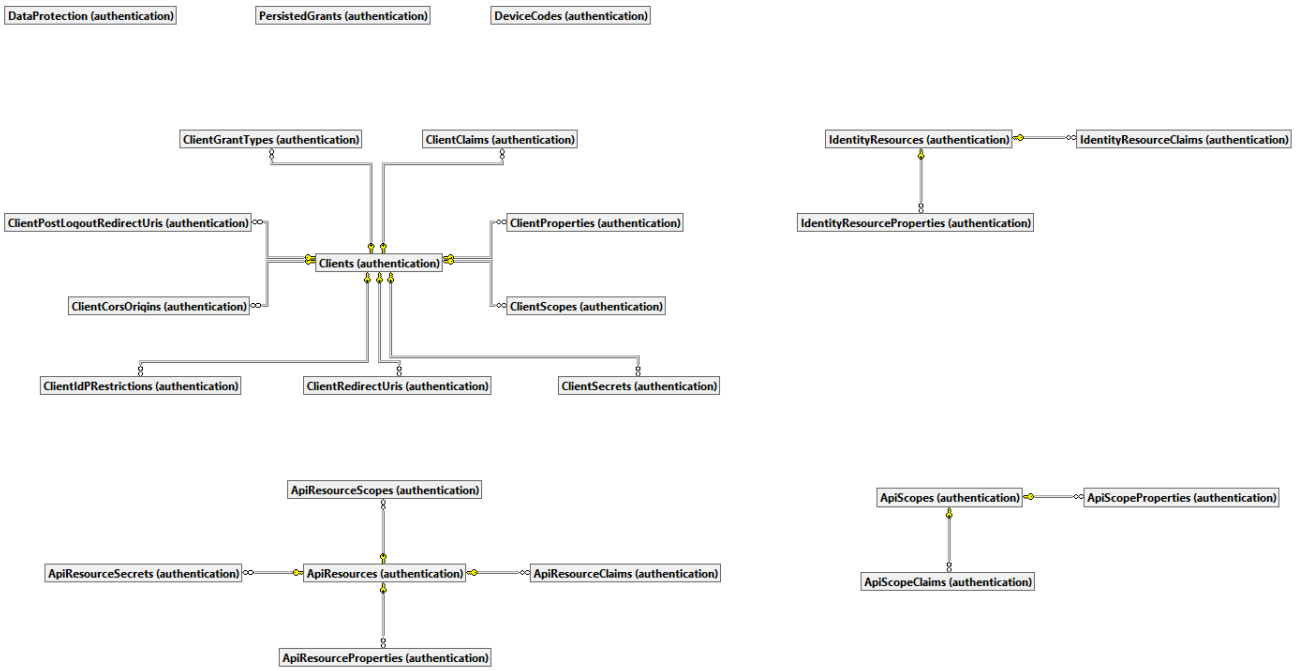
This section will not cover the details of this design (e.g.: the information stored into each table) as this design is very similar to the new one described in the next section.

New Security Schemas

The new database design, split into two database schemas, is explained below.

AUTHENTICATION SCHEMA

The `[authentication]` schema in the new Sequel Security Service database persists all the information about *authentication*. This feature is implemented by Identity Server, so the information handled by this technology is stored into the tables represented by the following diagram:

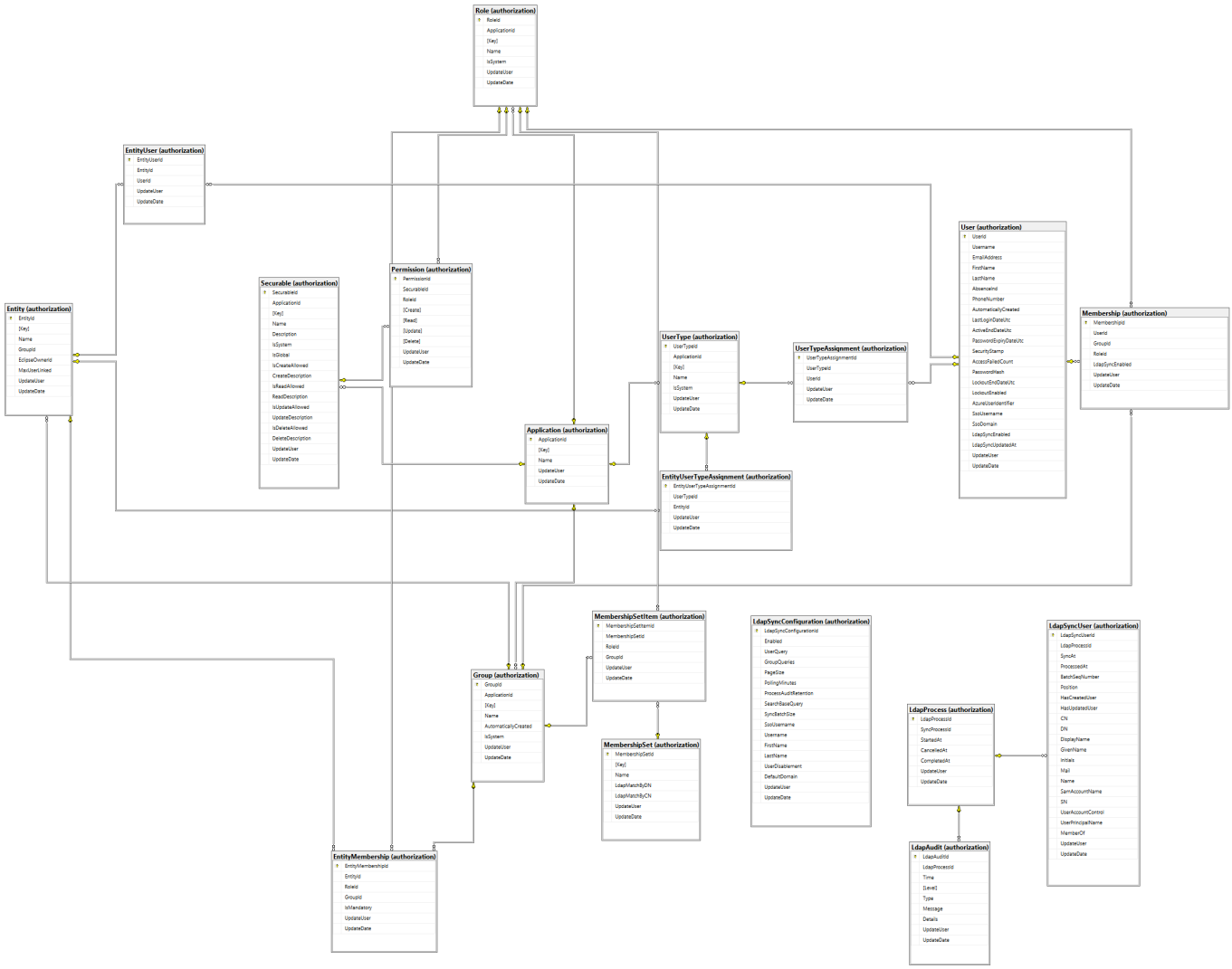


AUTHORIZATION SCHEMA

The `[authorization]` schema in the new Sequel Security Service database is intended to contain the data associated to *authorization*. The data model is quite similar to the legacy data model implemented by the `[security]` schema in the shared databases (mentioned at the beginning of this document); the main change is the addition of the new `Application` table to categorize roles, groups and user types by product in the suite.

`User` table in the legacy version contains several columns that comes from Identity and have never been used; we will use this opportunity to remove them. Columns to keep are described at Domain Model section.

The following E/R diagram shows the data model structure:



Application (security)

New table for storing information related to each application in the system, e.g. Claims, Origin, Impact, RE, Workflow, Product Builder, Rating Engine, Security Management,... Definition of this table will look like:

```
CREATE TABLE [authorization].[Application](
    [Id]   NVARCHAR (20) NOT NULL,
    [Description] NVARCHAR (50) NOT NULL,
    [UpdateUser] [nvarchar](50) NOT NULL CONSTRAINT [DF_Application_UpdateUser]  DEFAULT ('Unknown'),
    [UpdateDate] [datetime] NOT NULL CONSTRAINT [DF_Application_UpdateDate]  DEFAULT (getutcdate()),
    CONSTRAINT [PK_Application] PRIMARY KEY CLUSTERED ([Id] ASC),
    CONSTRAINT [UK_Application_Description] UNIQUE NONCLUSTERED ([Description] ASC)
);
GO
```

3.7.5 Single Sign-out

> **Note**
>
> The content described in this article is deprecated if using the **User Web Component**. See User Session Avatar for more information. Recommended approach for web apps that manages the access token (like SPAs).

In this article, we will cover Single Sign Out and also Single Session Management; as both concepts are related.

In this section, the integration of single sign-out are explained. For clarity reasons, all samples will be based on Workflow application. Also, we will cover the session timeout management. All applications must be in the same domain (ie *.office,.sbs) in order to being able to integrate the SSO.

Overview

Signing out is as simple as removing the authentication cookie, but for doing a complete federated sign-out, we must consider signing the user out of the client applications as well.

In our scenario, we need to cover two different use cases:

- **Server side integration:** Integrate single sign-in & single sign-out between applications using the back channel, for server side applications (as MVC applications).
- **Client side integration:** Improve user experience to support manual sign-out and detect sign-out events.

Server side integration

```
app.UseSequelSecurity(securitySettings)
    .UseSecurityCookie()
    .UseOidcAuthentication(
        WorkflowAppSettings.SecurityConfigurationClientId /*wf.conf*/,
        context => context.Request.Path.Value.ToLower().Contains("/api/")
            && !context.Request.Path.Value.ToLower().Contains("/publicapi/"),
        mainAppRegistration);
```

Sequel.Workflow.Security.AuthenticationAppInitializer.cs

In this sample, we are configuring Workflow application.

CONFIGURING URIS

Single sign-in: RedirectUri

For defining the single sign-in we had to define the `RedirectUri`, this was configured in the security database; at `RedirectUri` field from `[authentication].[ClientRedirect]` table. Also, same settings must be defined in the web.config (**must be exactly the same value without slash character at the end**) of each application and used as redirect URI after a successful login.

ClientIdentifier	RedirectUri
wf.conf	https://POTOROOWF.office.sbs/Workflow/sign

```
<add key="SecuritySettings:Authentication:ClientRedirect"
    value="https://POTOROOWF.office.sbs/Workflow/sign" />
```

This URI is used as a callback when single sign-in is initiated by a client application (like Workflow). When trying to access to a web base application and client requires authentication, the browser is redirected to an URL similar to:

```
https://potorooauth.office.sbs/Authentication/Login/?
returnUrl=%2FAuthentication%2Fconnect%2Fauthorize%2Fcallback%3Fclient_id%3Dwf.conf
%26redirect_uri%3Dhttps%253A%252F%252FPOTOROOWF.office.sbs%252FWorkflow%252Fsign
%26response_mode%3Dform_post
%26response_type%3Dcode%2520id_token
```

```
%26scope%3Dopenid%2520profile%2520offline_access%2520sec.authorization%2520sec.api
%26state%3DOpenIdConnect.AuthenticationProperties%253.....
```

Containing the `client_id` and `redirect_uri` that must match with the setting in the security service.

Single sign-out: PostLogoutRedirectUri

Similar to the `RedirectUri` used by sign-in, we need a `PostLogoutRedirectUri`. This URI will be used for redirecting to the client application again, in case the user performs a new login from Login Screen in security after signing-out from a client. In this scenario, we will expect the user will be redirected to the same client application.

`PostLogoutRedirectUri` must be configured in the security server, at `PostLogoutRedirectUri` field from `[authentication].[PostLogoutRedirectUri]`. Also, the same value must be stored in the web.config file (**must be exactly the same value without slash character at the end**):

ClientIdentifier	PostLogoutRedirectUri
wf.conf	https://POTOROOWF.office.sbs/Workflow

```
<add key="SecuritySettings:Authentication:ClientPostLogoutRedirectUri"
        value="https://POTOROOWF.office.sbs/Workflow" />
```

Logout from a client application

Once defined the integration of the single sign-in and sign-out in security service; we need to integrate the sign-out within the application. As mentioned before, sign-out is based on removing cookies and forcing a redirection to the Authentication server; and this is done by below code that we are placing in Logout action at `SignController`. In our sample in Workflow:

```
using Sequel.Workflow.Web.Filters;
using System.Web;
using System.Web.Mvc;

namespace Sequel.Workflow.Web.Controllers
{
    public class SignController : Controller
    {
        [SecurableNotRequired]
        public ActionResult Index()
        {
            return RedirectToAction("Index", "Home");
        }

        [SecurableNotRequired]
        [AllowAnonymous]
        public ActionResult Logout()
        {
            var client =
AppConfiguration.WorkflowAppSettings.Instance.SecuritySettings.AuthenticationSettings.ClientsSettings[AppConfiguration.WorkflowAppSettings.SecurityConfigurationClientId]

            if (Request.IsAuthenticated)
            {
                Request.GetOwinContext().Authentication.SignOut(
                new Microsoft.Owin.Security.AuthenticationProperties()
                {
                    RedirectUri = client.ExternalSettings.PostLogoutRedirectUri
                });
            }

            return Redirect(client.ExternalSettings.PostLogoutRedirectUri);
        }

    }
}
```

Client side integration

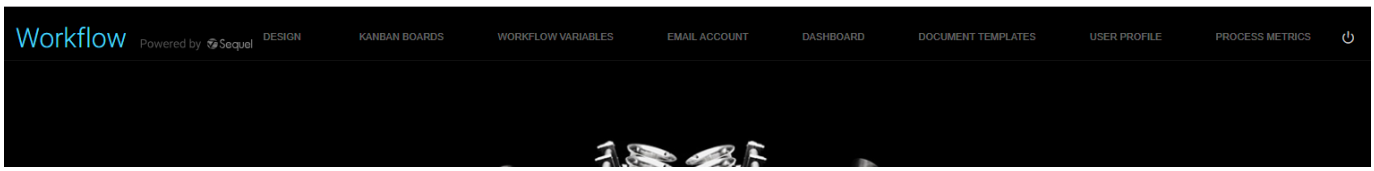
At the front-end side, we need to offer below functionalities:

- Perform a sign-out.
- Detect a single sign-out event triggered by a different application.
- Auto sign-out for inactive sessions.

> **Use the component!**
>
> There is a component that has all this functionallity implemented for SPAs and similar web apps. Checkout User Session Avatar.

PERFORM A SIGN-OUT

As a user of a client application (in this case Workflow), I will be able to click on a logout button and trigger a sign-out.



This sign-out button calls to `Sign/Logout` action, previously defined.

3.7.6 Single Session Management

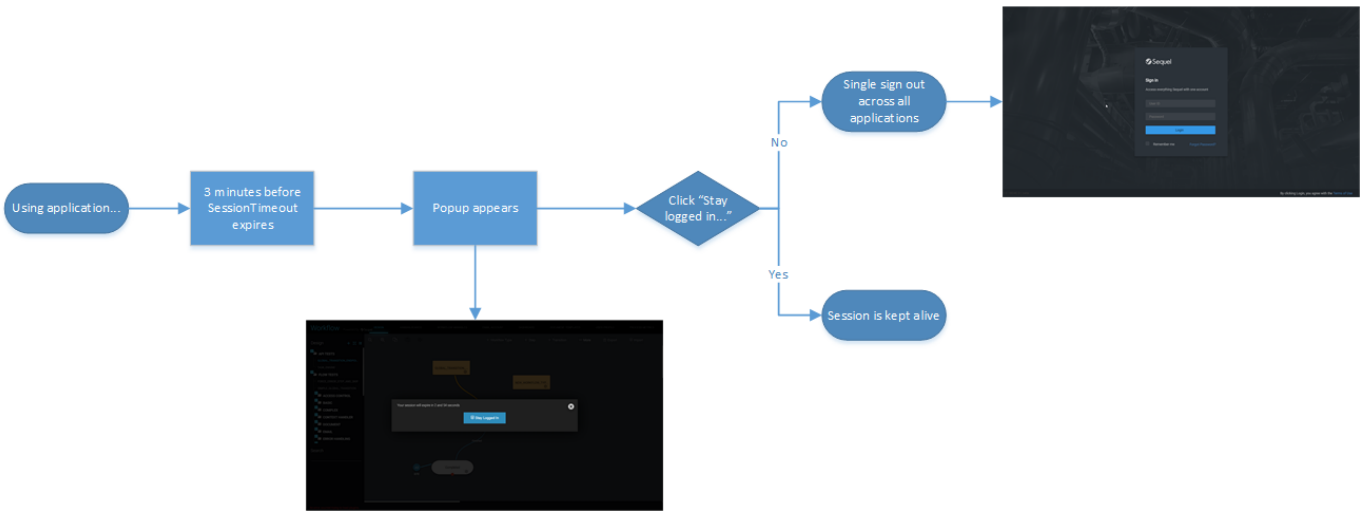
In this section we will describe how single session is managed in terms of syncing single sign outs and also for detecting inactive sessions (across all application, not just in a single window/application).

Detect a single sign-out event

For reacting to a single sign-out event in the browser, we need to check the existence of `SQ.sso` cookie; if that cookie is not found we will call to the logout endpoint. This logic was previously implemented in the layout.html; this is no longer required as is managed as part of the `CookieSsso` service described in the next section. Current implementation checks every 5 seconds if the cookie exists.

Detect inactive sessions

When a user is inactive for a long period (`SessionTimeout` setting) a pop-up is displayed to the user for confirming if is still working and wants to stay connected; in case, there are no confirmation from the user after a period of time (`TimeoutWarningInMins`) a sign-out is forced.



The implementation has been redesigned to detect, not just activity in a single window browser, also to detect activity associated with the current session in any windows running an application logged in the same session. The solution described above is based on the idea that all active sessions have the same associated `SQ.sso` cookie. We have added a timestamp to this cookie that is refreshed in every single call to the server side. When the timestamps remains the same for more than `SessionTimeout` the session is considered inactive and a single sign out is forced.

For implementing this functionality we need to apply some changes to the existing implementation (changes are using Workflow as reference):

SERVER SIDE CHANGES

'Sequel.Security.Integration' NuGet package in version 1.0.19088.1 is required for generating the new `SQ.sso` cookie with the timestamp.

Also, a new enpoint in the SignController is required in order to force a manual refresh of this cookie:

```
public class SignController : Controller
{

    .
    .
    .

    [SecurableNotRequired]
    public void RenewSsoCookie() { }
}
```

SUMMARY OF STEPS FOR UPGRADING FROM PREVIOS VERSIONS

- Upgrade Sequel.Security.Integration to at least version 1.0.19088.1
- Add new action in controller: SignController => RenewSsoCookie
- Add CookieSso and Cookie to project. Register in application
- Remove all related code in frontend and use the User Session Avatar

3.7.7 User Session Avatar

Source Code - README

The component serves several purposes:

- Nice and configurable interface for user interaction.
- Inactivity detection in client side through `SQ.sso`.
- Single Sign-Out implementation in client side.
- Cookie validation (if using signed cookie)
  - Verifies signature in Cookie to avoid external tampering.
  - Verifies several claims from the application's access token against the cookie.

The readme contains complete documentation about its implementation and configuration, but see below recommended steps for implementation.

Prerequisites

The frontend environment requires:

- `npm` for dependency management
- a bundler (like `webpack`)
- `react` and `react-dom` installed as dependencies
- have configured Nexus registry for `@sequel` packages (see this)

Your application does not need to be react, in fact is a simple component that works smooth even in plain JS or Angular. But `npm` and a bundler are mandatory in order to properly use the component and its dependencies.

Integration

The component is implementation agnostic in the **oidc** client side. It requires some mappings in order to talk with it, but also allows developers to adjust the behaviour in some scenarios.

> **README**
>
> Do not forget to read the README of the component for more detailed documentation of the configuration and theme customization.

The authentication service can be implemented with `oidc-client`, your own implementation or whatever. Just ensure to follow the steps for implementing the callbacks and profile mapping.

The `user` object must be mapped from a profile or JWT claims. Ensure to ask for `openid profile` scopes (`email` just in case it does not include it) when asking for an access token. Then, the mapping is easy as:

Property	Claim
`userName`	`sub`
`firstName`	`given_name`
`lastName`	`family_name`
`email`	`email`

oidc-client

When using `oidc-client`, you can map the properties like this:

```
const user = await userManager.getUser();
const userProp = {
  userName: user.profile.sub,
  firstName: user.profile.given_name,
  lastName: user.profile.family_name,
  email: user.profile.email,
};
```

Next step is to configure actions when several events happen (`urls` prop). First of all, configure the authentication url, this is implementation specific and it is out of scope. Next is to implement the `renewSsoCookie`, `logOut` and `refreshSession`.

- `renewSsoCookie`: URL or Callback or empty.
  - **URL**: When the component needs to refresh the `SQ.sso` cookie, will call this endpoint using `GET`. It can also be a function that returns an `string`.
  - **Callback**: Calls the function to do the renew cookie operation. Can be a simple API call that does just exactly this.
  - **Empty**: Do not fill property to do nothing (**not recommended**).
- `logOut`: URL or Callback or empty.
  - **URL**: When the component should make a log out process, redirects to that URL. It can also be a function that returns an `string`.
  - **Callback**: Calls the function to do the log out. Can be a simple API call or the *end session* flow of OIDC.
  - **Empty**: Does nothing (**not recommended**).
- `refreshSession`: Callback or empty.
  - **Callback**: Calls the function to do a session refresh. That is: to remove all information and access tokens of the current user and ask for new ones without doing a log out/*end session*.
  - **Empty**: Does what `logOut` is configured to do (**not recommended**).

oidc-client

To implement `logOut` and `refreshSession` using `oidc-client`, you can do the following:

```
const logOut = async () => {
  // this redirects the webpage to the *end session* endpoint
  await userManager.signoutRedirect();
};
const refreshSession = async () => {
  // removes all access tokens and user information
  await userManager.removeUser();
  // log in process
  await userManager.clearStaleState(); // optional
  await userManager.signinRedirect();
};
```

For improved debug experience, the prop `loggingEnabled` can be set to true to log in console a lot of debug information around the internal process. If there is an issue in the component, it is better to include a trace of those logs.

Theming is out of scope of this document, please read the README for more detailed information.

ANGULAR.JS

Integration using Angular.JS can be done with a simple directive, but depends on your implementation of the services which may affect the final implementation.

```
import { render } from 'react-dom';
import { UserSessionAvatar } from '@sequel/sequel.web.usersessionavatar';
import type OidcService from '.../oidc.service'; // your authentication service

const UserSessionAvatarDirective = ($q: ng.IQService, oidcService: OidcService): ng.IDirective => ({
  restrict: 'E',
  scope: {},
  template: '<div class="user-session-avatar" id="user-session-avatar"></div>',
```

```
  link(_, element) {
    $q.all([
      // this gets information about the user (can be the JWT parsed directly or whatever)
      // and also gets the URL to the Authority (which means to Sequel Authentication)
      oidcService.getAuthority(),
      oidcService.getUser(),
    ]).then(([authentication, user]) => {
      // do not render anything if there is no user, for example, while loggin in
      if (!user) return;

      render(
        (
          <UserSessionAvatar
            user={{
              userName: user.profile.sub,
              firstName: user.profile.given_name,
              lastName: user.profile.family_name,
              email: user.profile.email,
            }}
            urls={{
              authentication,
              async logOut() {
                await oidcService.logOut();
              },
              async renewSsoCookie() {
                await oidcService.renewSsoCookie();
              },
              async refreshSession() {
                await oidcService.refreshSession();
              },
            }}
            loggingEnabled={process.env.NODE_ENV === 'development'}
          />
        ), element[0],
      );
    });
  },
});

UserSessionAvatarDirective.$inject = ['$q', 'oidcService'];

export default UserSessionAvatarDirective;
```

Register the directive in the module and use it:

```
ngModule.directive('userSessionAvatar', UserSessionAvatar)
```

```
<div class="super-navbar">
    <!-- ... -->
    <user-session-avatar></user-session-avatar>
</div>
```

ANGULAR

Integration with angular is similar to angular.js but it requires some steps in order to bridge between both.

First ensure to have configured `"jsx": "react-jsx"` in the `tsconfig.json` file. And do not forget to install `@types/react` and `@types/react-dom`.

Then create a new component, but rename the `.component.ts` to `.component.tsx`.

Using the `@ViewChild` annotation will bridge the react component into angular:

```
// user-session-avatar.component.tsx
import {
  Component,
  ViewContainerRef,
  AfterViewInit,
  ViewChild
} from "@angular/core";
import { render } from "react-dom";
import { forkJoin } from 'rxjs';
import type OidcService from '.../oidc.service'; // your authentication service

@Component({
  selector: "app-user-session-avatar",
  templateUrl: "./user-session-avatar.component.html",
  styleUrls: []
})
export class UserSessionAvatarComponent implements AfterViewInit {
  @ViewChild("container")
  container!: { nativeElement: HTMLDivElement };

  constructor(private readonly oidcService: OidcService) {}

  ngAfterViewInit(): void {
    this.renderComponent();
```

```
  }

  private renderComponent(): void {
    forkJoin(
      // this gets information about the user (can be the JWT parsed directly or whatever)
      // and also gets the URL to the Authority (which means to Sequel Authentication)
      this.oidcService.getAuthority(),
      this.oidcService.getUser(),
    ).subscribe(([authentication, user]) => {
      // do not render anything if there is no user, for example, while loggin in
      if (!user) return;

      render(
        (
          <UserSessionAvatar
            user={{
              userName: user.profile.sub,
              firstName: user.profile.given_name,
              lastName: user.profile.family_name,
              email: user.profile.email,
            }}
            urls={{
              authentication,
              logOut: async () => {
                await this.oidcService.logOut();
              },
              renewSsoCookie: async () => {
                await this.oidcService.renewSsoCookie();
              },
              refreshSession: async() => {
                await this.oidcService.refreshSession();
              },
            }}
          />
        ),
        this.container.nativeElement,
      );
    })
  }
}
```

```
<!-- user-session-avatar.component.html -->
<div class="user-session-avatar" #container></div>
```

REACT

Just use the component wherever you want to call it. Recommended to wrap the User Session Avatar inside another component to keep together the mappings and callbacks.

If want an example, checkout Administration's code.

PLAIN JS

Component documentation talks about this integration, and there are even a bundle for it. But it is deprecated and some time in the future will be removed. Do not rely on this build!

Appendix: Configure Nexus Registry

Near the `package.json` write the following contents inside `.npmrc`:

```
@sequel:registry=https://nexus.office.sbs/repository/npm/
strict-ssl=false
```

3.7.8 How to request a token from a client application

How to request a token

This section describes how to obtain an token from Authentication server using Client Credentials flow from any `c#` application.

REQUIREMENTS

Software requirements

- Framework netstandard2.0 or net461
- IdentityModel NuGet package.

Security configuration requirements

It's necessary to now Security configuration values:

- *AuthenticationServerUrl*: URL address of Authentication server (including VirtualPath).
- *ClientId*: The client identifier that we're going to use to ask for the token.
- *ClientSecret*: Secret of the client in plain text.
- *Scope*: Scope or scopes the need about we need the token.

CODE EXAMPLE

Next code gets an Access Token using Client Credentials flow

```
using IdentityModel.Client;
using System;
using System.Net.Http;

namespace TokenProvider
{
    class Program
    {
        static async System.Threading.Tasks.Task Main(string[] args)
        {
            var authenticationServerUrl = "https://security.sequel.com/Authentication";
            var tokenEndpoint = $"{authenticationServerUrl}/connect/token";
            var clientId = "app.client";
            var clientSecret = "PasswordInPlainText";
            var scope = "app.scope1 app.scope2";

            var client = new HttpClient();

            // Request token using client credentials flow
            var tokenResponse = await client.RequestClientCredentialsTokenAsync(new ClientCredentialsTokenRequest
            {
                Address = tokenEndpoint,
                ClientId = clientId,
                ClientSecret = clientSecret,
                Scope = scope
            });

            if(tokenResponse.IsError)
                Console.Error.WriteLine(tokenResponse.Error);
            else
                Console.WriteLine(tokenResponse.AccessToken);
        }
    }
}
```

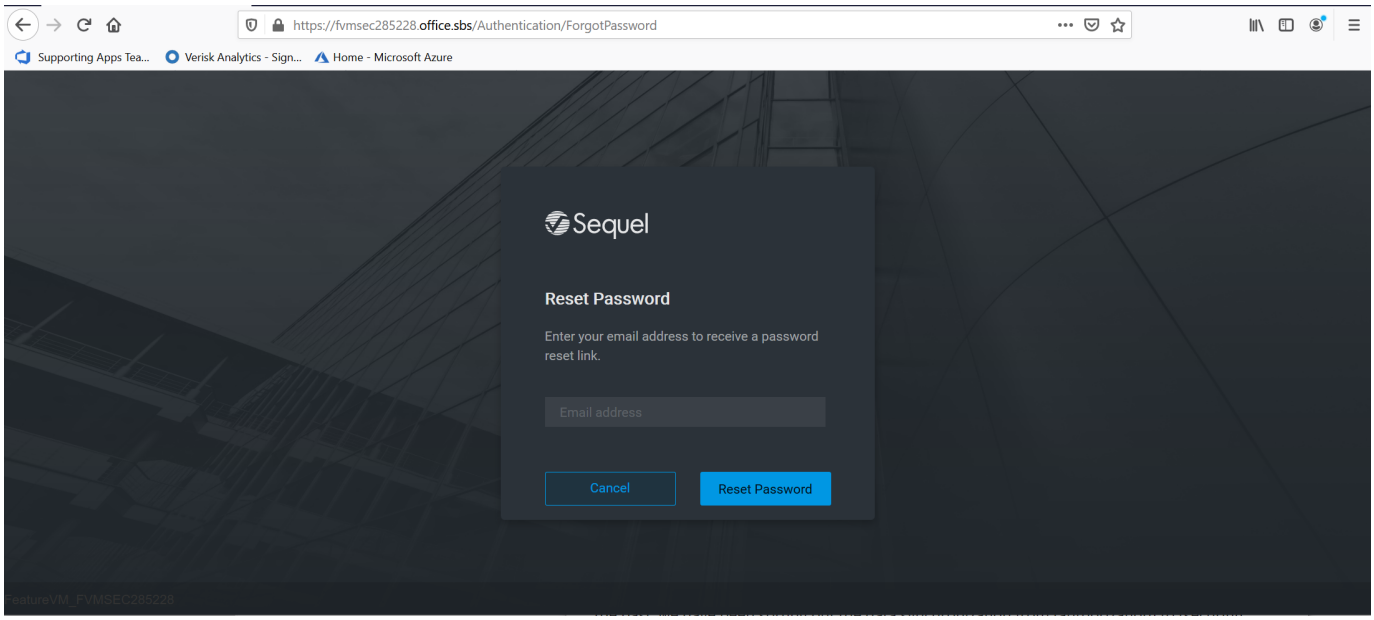
Performance improvement

All token's have a lifetime and we can take advantage of it reusing the token. It's possible to know directly the token's lifetime in seconds checking the `tokenResponse.ExpiresIn` property. It's recommended use this value to store the token until its expiration, reducing the requests to Authentication server and, this way, increasing the performance.

3.7.9 Reset password

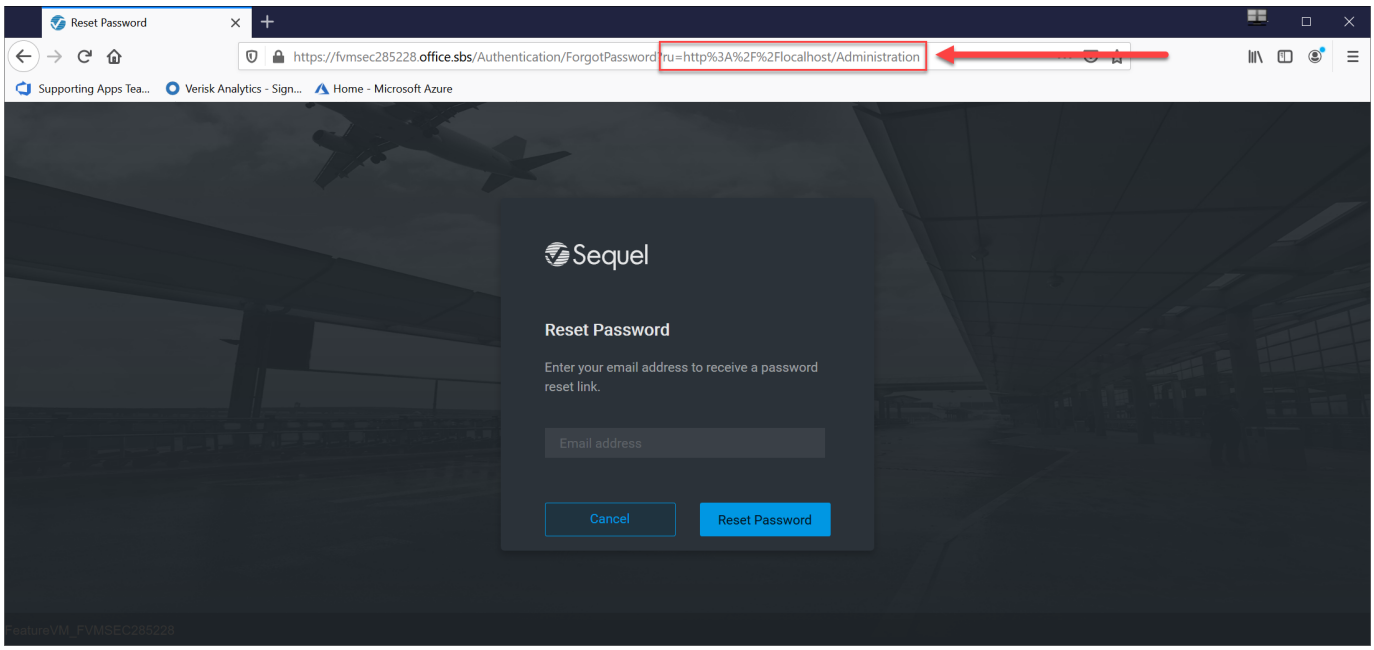
Reset password

Reset password flow allow to users change their passwords providing an existing email address:

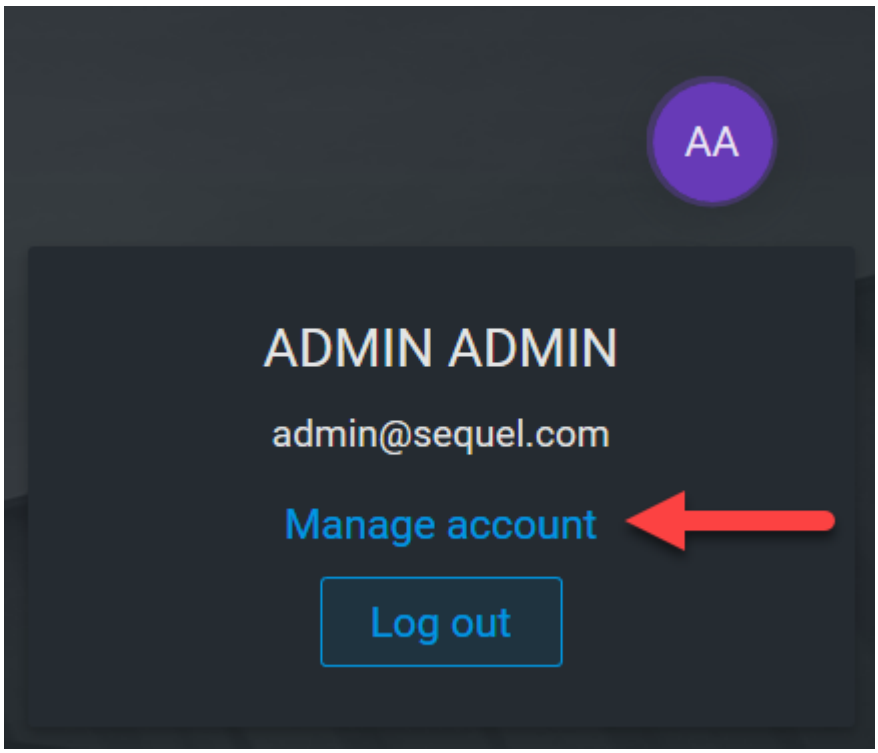


GOING BACK TO APPLICATION

It's possible return to the application after complete the reset password flow if *ru* parameter is provided in ForgotPassword's querystring:

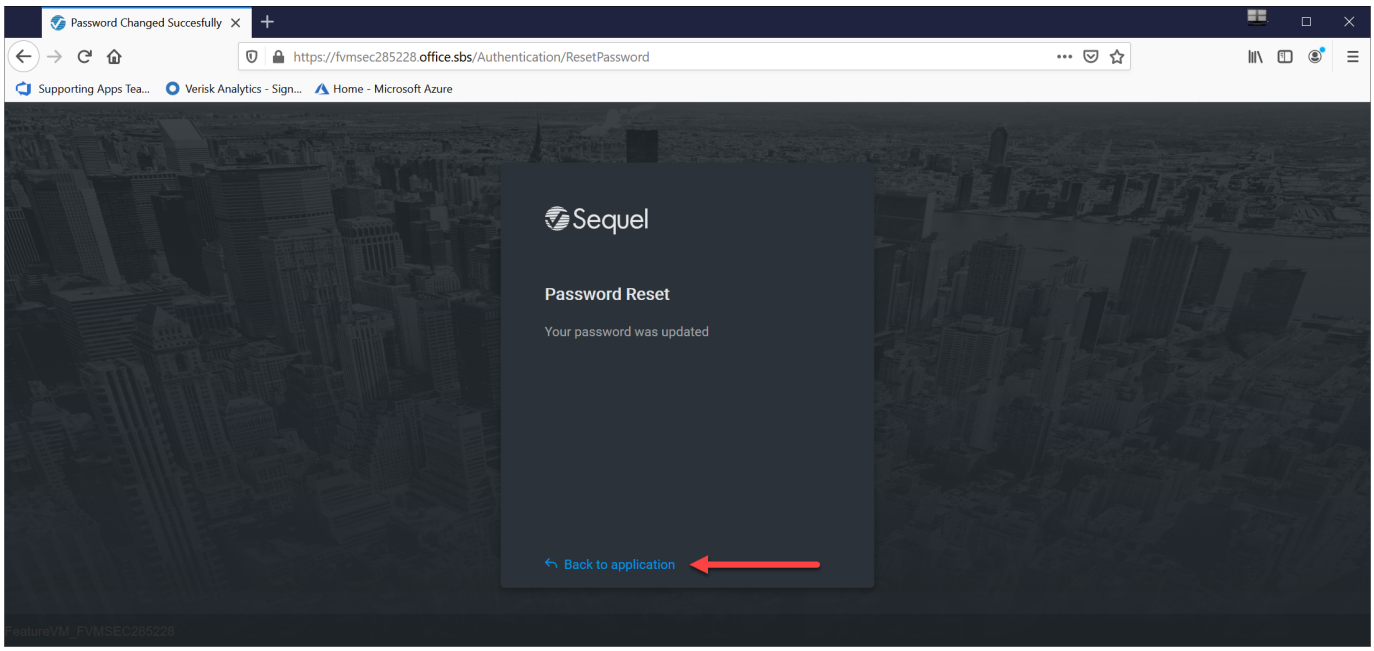


This **parameter** is **added automatically by web component** when click on Manage Account link from any application:



Return Url (*ru*) parameter **must be a valid URL** and this **URL must be in the same domain** of any RedirectUri configured for clients in ClientRedirectUri database table.

Adding this parameter when reset password process is completed **user will be redirected to application in 5s** or clicking in `Back to application` link:



3.8 OpenIdConnect

3.8.1 Client settings

Rules: Default(x) => in all mappings when value is null or empty (string) then force this value for storing it at the persistent layer.

Basics

Property	Category	Rules	Type	Description
Enabled	Basics	Default(true)	boolean	Specifies if client is enabled. Defaults to true.
ClientIdentifier	Basics	REQUIRED,UNIQUE	string	Unique ID of the client
ClientName	Basics	REQUIRED	string	Client display name (used for logging and consent screen)
Description	Basics	OPTIONAL	string	General description of the client, for helping administrators
ProtocolType	Basics	Default(oidc), Range(oidc)	string	Authentication protocol. Always oidc.
RequireClientSecret	Basics	Default(true)	boolean	Specifies whether this client needs a secret to request tokens from the token endpoint (defaults to true)
ClientSecrets	Basics/Secrets		ListOf(ClientSecret)	List of client secrets - credentials to access the token endpoint. Ignored by put/patch operations
ClientSecret.Description	Basics/Secrets	OPTIONAL	string	Description for management purposes of the secret
ClientSecret.Type	Basics/Secrets	Default(SharedSecret), Range(SharedSecret)	string	Type of secret. Default: SharedSecret
ClientSecret.Value	Basics/Secrets	Default(SharedSecret), Range(SharedSecret)	string	Secrets cannot be edited. If you need to change the secret, please create a new secret.
AllowedGrantTypes	Basics/ GrantTypes	Range(authorization_code, client_credentials, hybrid, implicit)	ListOf(string)	Specifies the grant types the client is allowed to use. Use the GrantTypes class for common combinations.
RequirePkce	Basics/ GrantTypes	Default(true)	boolean	Specifies whether clients using an authorization code based grant type must send a proof key (defaults to true).

Property	Category	Rules	Type	Description
AllowPlainTextPkce	Basics/ GrantTypes	Default(false), R/O, No UI	boolean	Specifies whether clients using PKCE can use a plain text code challenge (not recommended - and default to false). Not included in the UI and forced to false for new clients.
AllowAccessTokensViaBrowser	Basics/ GrantTypes	Default(true)	boolean	Specifies whether this client is allowed to receive access tokens via the browser. This is useful to harden flows that allow multiple response types (e.g. by disallowing a hybrid flow client that is supposed to use code id_token to add the token response type and thus leaking the token to the browser.
RedirectUris	Basics\RedirectUri		ListOf(string)	Specifies the allowed URIs to return tokens or authorization codes to
AllowedScopes	Basics\Scope	Range(Scopes)	ListOf(string)	By default a client has no access to any resources - specify the allowed resources by adding the corresponding scopes names (list of strings). List comes from existing scopes in ApiScopes table; however this information is just used as a typeahead, we will allow to add scopes that has not been yet defined in ApiScopes.
AllowOfflineAccess	Basics/Scope	Default(true)	boolean	Specifies whether this client can request refresh tokens (be requesting the offline_access scope)

Authorization and logout

Property	Category	Rules	Type	Description
PostLogoutRedirectUris	AuthLogout		ListOf(string)	Specifies allowed URIs to redirect to after logout. See the OIDC Connect Session Management spec for more details.
BackChannelLogoutUri	AuthLogout		string	Specifies logout URI at client for HTTP based back-channel logout. See the OIDC Back-Channel spec for more details.
BackChannelLogoutSessionRequired	AuthLogout	Default(true)	boolean	Specifies if the user's session id should be sent in the request to the BackChannelLogoutUri. Defaults to true.
FrontChannelLogoutUri	AuthLogout		string	Specifies logout URI at client for HTTP based front-channel logout. See the OIDC Front-Channel spec for more details.
FrontChannelLogoutSessionRequired	AuthLogout	Default(true)	boolean	Specifies if the user's session id should be sent to the FrontChannelLogoutUri. Defaults to true.
EnableLocalLogin	AuthLogout	Default(true)	boolean	Specifies if this client can use local accounts, or external IdPs only. Defaults to true.
IdentityProviderRestrictions	AuthLogout		ListOf(string)	Specifies which external IdPs can be used with this client (if list is empty all IdPs are allowed). Defaults to empty.

Token

Property	Category	Rules	Type	Description
IdentityTokenLifetime	Token	Default(300)	int	Lifetime to identity token in seconds (defaults to 300 seconds / 5 minutes)
AccessTokenLifetime	Token	Default(3600)	int	Lifetime of access token in seconds (defaults to 3600 seconds / 1 hour)
AuthorizationCodeLifetime	Token	Default(300)	int	Lifetime of authorization code in seconds (defaults to 300 seconds / 5 minutes)
AccessTokenType	Token	Default(Jwt), R/O, NoUI	enum	Specifies whether the access token is a reference token or a self contained JWT token (defaults to Jwt).
IncludeJwtId	Token	Default(true), R/O, NoUI	boolean	Specifies whether JWT access tokens should have an embedded unique ID (via the jti claim). Forced to true always.
PairWiseSubjectSalt	Token	NULL, R/O, NoUI	string	Salt value used in pair-wise subjectId generation for users of this client. NULL by

REFRESH TOKEN

Property	Category	Rules	Type	Description
RefreshTokenUsage	Token\Refresh	Default(1)	enum	*ReUse* the refresh token handle will stay the same when refreshing tokens (value=0). *OneTime* the refresh token handle will be updated when refreshing tokens (value=1). This is the default.
RefreshTokenExpiration	Token\Refresh	Default(1)	enum	*Absolute* the refresh token will expire on a fixed point in time (specified by the AbsoluteRefreshTokenLifetime). *Sliding* when refreshing the token, the lifetime of the refresh token will be renewed (by the amount specified in SlidingRefreshTokenLifetime). The lifetime will not exceed AbsoluteRefreshTokenLifetime.
AbsoluteRefreshTokenLifetime	Token\Refresh	Default(2592000)	int	Maximum lifetime of a refresh token in seconds. Defaults to 2592000 seconds / 30 days. (int, not null)
SlidingRefreshTokenLifetime	Token\Refresh	Default(1296000)	int	Sliding lifetime of a refresh token in seconds. Defaults to 1296000 seconds / 15 days
UpdateAccessTokenClaimsOnRefresh	Token\Refresh	Default(0)	boolean	Gets or sets a value indicating whether the access token (and its claims) should be updated on a refresh token request.

CORS

Property	Category	Rules	Type	Description
AllowedCorsOrigins	Token\CORS		ListOf(string)	If specified, will be used by the default CORS policy service implementations (In-Memory and EF) to build a CORS policy for JavaScript clients.

CLAIMS

Property	Category	Rules	Type	Description
Claims	Token\Claims		ListOf(Claim[Type+Value])	Allows settings claims for the client (will be included in the access token).
AlwaysSendClientClaims	Token\Claims	Default(false)	boolean	If set, the client claims will be sent for every flow. If not, only for client credentials flow (default is false)
AlwaysIncludeUserClaimsInIdToken	Token\Claims	Default(false)	boolean	When requesting both an id token and access token, should the user claims always be added to the id token instead of requiring the client to use the userinfo endpoint. Default is false.
ClientClaimsPrefix	Token\Claims	Default(client_)	string	If set, the prefix client claim types will be prefixed with. Defaults to client_. The intent is to make sure they don't accidentally collide with user claims.

3.8.2 Grant Types

Grant types are a way to specify how a client wants to interact with the Authentication service (based on Duende IdentityServer implementation). The OpenID Connect and OAuth 2 specs define many grant types; and we support the following grant types:

- Client credentials
- Implicit
- Hybrid
- Authorization code

Client credentials

This is the simplest grant type and is used for server to server communication - tokens are always requested on behalf of a client, not a user.

With this grant type you send a token request to the token endpoint, and get an access token back that represents the client. The client typically has to authenticate with the token endpoint using its client ID and secret.

Implicit

The implicit grant type is optimized for browser-based applications. Either for user authentication-only (both server-side and JavaScript applications), or authentication and access token requests (JavaScript applications).

In the implicit flow, all tokens are transmitted via the browser, and advanced features like refresh tokens are thus not allowed.

> **Note**
>
> For JavaScript-based applications, *Implicit* is not recommended anymore. Use *Authorization Code with PKCE* instead. More information on The state of the implicit flow in OAuth2

Authorization code

Authorization code flow was originally specified by OAuth 2, and provides a way to retrieve tokens on a back-channel as opposed to the browser front-channel. It also support client authentication.

While this grant type is supported on its own, it is generally recommended you combine that with identity tokens which turns it into the so called hybrid flow. Hybrid flow gives you important extra features like signed protocol responses.

This is our recommendation for JavaScript-based applications.

Hybrid

Hybrid flow is a combination of the implicit and authorization code flow - it uses combinations of multiple grant types, most typically `code id_token`.

In hybrid flow the identity token is transmitted via the browser channel and contains the signed protocol response along with signatures for other artifacts like the authorization code. This mitigates a number of attacks that apply to the browser channel. After successful validation of the response, the back-channel is used to retrieve the access and refresh token.

This is the recommended flow for native applications that want to retrieve access tokens (and possibly refresh tokens as well) and is used for server-side web applications (MVC) and native desktop/mobile applications.

3.9 Performance

3.9.1 Load Tests

This document provides the details about how to track SQL, CPU and memory usage of:

- The endpoints that we expect they will be more used in Security, providing the effective permissions for:
- A user in a specific application.
- A user in a specific application over an specific securable to perform an action.
- Navigating to login screen and completing a happy login.

Solution Sequel.Security.LoadTests is organized as:

- Web Tests without context parameters
- Web and Load Tests with context parameters

Web Tests without context parameters

Test parameters like authorization token and data like users, applications, groups and securables are load **automatically** to set then randomly in the requests.

1. EffectivePermissionsTest webtest

This web test gets all effective permissions for a user in a specific application. Test is run from Visual Studio (opened as administrator).
The following steps are executed **automatically** when we open the `EffectivePermissionsTest.webtest` file and we click the button `Run Test`:

1. Get the authorization token needed for the next 3 requests (Authorization\users, Authorization\applications and Authorization\groups) by 3 `WebTestPlugin`.
2. Request all *users* (a `WebTestRequestPlugin` set the authorization header request and the apiServer context parameter, and a `ExtractionRule` plugin extracts the result in a context parameter).
3. Request all *applications* (a `WebTestRequestPlugin` set the authorization header request and the apiServer context parameter, and a `ExtractionRule` plugin extracts the result in a context parameter).
4. Request all *groups* (a `WebTestRequestPlugin` set the authorization header request and the apiServer context parameter, and a `ExtractionRule` plugin extracts the result in a context parameter).
5. Request all effective permissions for an users in a specific application:
6. A `WebTestRequestPlugin` set the authorization token, the apiServer context parameter at first, and then, it selects randomly a user, application and/or group (group of the selected application) as context parameters too.
7. The request url is `{{apiServer}}/Authorization/EffectivePermissions/{{username}}/{{applicationKey}}?groupKeys={{groupKeys}}`.
8. This request is in a loop which repeats the request the number of iterations we want to test

2. HasPermissionsTest webtest

This web test checks if a specific user in an application has permissions over a specific securable to perform an action. Test is run from Visual Studio (opened as administrator).

The following steps are executed **automatically** when we open the `HasPermissionsTest.webtest` file and we click the button `Run Test`:

1. Get the authorization token needed for the next 4 requests (Authorization\users, Authorization\applications, Authorization\groups and Authorization\securables) by 4 `WebTestPlugin`.
2. Request all *users* (a `WebTestRequestPlugin` set the authorization header request and the apiServer context parameter, and a `ExtractionRule` plugin extracts the result in a context parameter).
3. Request all *applications* (a `WebTestRequestPlugin` set the authorization header request and the apiServer context parameter, and a `ExtractionRule` plugin extracts the result in a context parameter).
4. Request all *groups* (a `WebTestRequestPlugin` set the authorization header request and the apiServer context parameter, and a `ExtractionRule` plugin extracts the result in a context parameter).
5. Request all *securables* (a `WebTestRequestPlugin` set the authorization header request and the apiServer context parameter, and a `ExtractionRule` plugin extracts the result in a context parameter).
6. Request if a user in an application has permissions over a securable to perform an action:
7. A `WebTestRequestPlugin` set the authorization token, the apiServer context parameter at first, and then, it selects randomly a user, application, permission, securable (securable of the selected application) and/or a group (group of the selected application) as context parameters too.
8. The request url is `{{apiServer}}/Authorization/EffectivePermissions/{{username}}/{{applicationKey}}/hasPermission/{{permissionAction}}?securableKeys={{securableKeys}}&groupKeys={{groupKeys}}`.
9. This request is in a loop which repeats the request the number of iterations we want to test.

3. LoginTest webtest

The test simulates the session user navigating to login screen and completing a happy login (selected randomly, and repeating users). Test is run from Visual Studio (opened as administrator).
The following steps are executed **automatically** when we open the `LoginTest.webtest` file and we click the button `Run Test`:

1. Get the authorization token needed for the next request (Authorization\users) by a `WebTestPlugin`.
2. Request all *users* (a `WebTestRequestPlugin` set the authorization header request and the apiServer context parameter, and a `ExtractionRule` plugin extracts the result in a context parameter).
3. Request the login *GET* action `{{identityServer}}/Login)`, where a `WebTestRequestPlugin` set the identity Server context parameter .
4. Request the login *POST* action `{{identityServer}}/Login`, where other `WebTestRequestPlugin` selects randomly a user as the username post parameter (password is the same for all the users).
5. Request the logout action `{{identityServer}}/LogOut`.
6. The three last requests (login/logout) are in a loop which repeats the requests the number of iterations we want to test.

Web and Load Tests with context parameters

We do not want to load test the requests neither to all users, applications, groups or securables. We want just to load test the request to all effective permissions. To get this goal we have to set **manually** all data as context parameters even the authorization token using the `app.config` file with the following settings:

- *identity server*: url of authentication identity server to get the authorization token.
- *apiServer*: url of security Api to request the authorization actions.
- *contextPath*: directory which contains the json files with all the users, applications, groups and securables of the environment you are testing
- *authorization*: token to authorize.

Note 1: the context json files should be updated **manually** with the values of the environment your are testing.
Note 2: you can get the token when debugging any of the three web test above. Authorization class has a breakpoint to get the authorization token when debugging a web test. **Manually**, just copy and paste it as the value of authorization setting before running or debugging some of the following web or load tests below.
Note 3. have a look at the `Web Tests without context parameters` details before having a look at some `Web and Load Tests with context parameters`:

1. ContextEffectivePermissionsTest webtest

This web test requests all effective permissions for a user in a specific application. Test is run from Visual Studio (opened as administrator).
The following steps are executed when we open the `ContextEffectivePermissionsTest.webtest` file and we click the button `Run Test`:

- A `WebTestRequestPlugin` set the authorization token from the settings, the apiServer context parameter at first, and then, it selects randomly a user, application and/or group (group of the selected application) as context parameters too, from the json files in the context directory.
- The request url is **`{{apiServer}}/Authorization/EffectivePermissions/{{username}}/{{applicationKey}}?groupKeys={{groupKeys}}`**.

2. ContextEffectivePermissionsTest loadtest

This load test executes the web test `ContextEffectivePermissionsTest.webtest` before.

3. ContextHasPermissionsTest webtest

This web test checks if a specific user in an application has permissions over a specific securable to perform an action. Test is run from Visual Studio (opened as administrator).

The following steps are executed when we open the `ContextHasPermissionsTest.webtest` file and we click the button `Run Test`:

- A `WebTestRequestPlugin` set the authorization token from the settings, the apiServer context parameter at first, and then, it selects randomly a user, application, permission, securable (securable of the selected application) and/or a group (group of the selected application) as context parameters too, from the json files in the context directory.
- The request url is **`{{apiServer}}/Authorization/EffectivePermissions/{{username}}/{{applicationKey}}/hasPermission/{{permissionAction}}?securableKeys={{securableKeys}}&groupKeys={{groupKeys}}`**.

4. ContextHasPermissionsTest loadtest

This load test executes the web test `ContextHasPermissionsTest.webtest` before.

5. ContextLoginTest webtest

The test simulates the session user navigating to login screen and completing a happy login (selected randomly, and repeating users). Test is run from Visual Studio (opened as administrator).

The following steps are executed when we open the `ContextLoginTest.webtest` file and we click the button `Run Test`:

1. Request the login *GET* action ({{identityServer}}/Login), where a `WebTestRequestPlugin` set the identity Server context parameter.
2. Request the login *POST* action ({{identityServer}}/Login), where other `WebTestRequestPlugin` selects randomly a user, from the json files in the context directory, as the username post parameter (password is the same for all the users).
3. Request the logout action ({{identityServer}}/LogOut).

6. ContextLoginTest loadtest

This load test executes the web test `ContextLoginTest.webtest` before.

Some notes

Note 1: To compile the solution in Visual Studio without errors:

- you should add the feature Web performance and load testing tools
- update your local.testsettings file and add you Sequel.Security.LoadTest.dll as additional file to deploy. this assembly file is located in the /bin/debug directory.

Note 2: you can run web and load tests out of VS too with Developer Command Prompt for VS 2017:

- execute Developer Command Prompt for VS 2017
- execute the command mstest /testcontainer:X.webtest /testsettings:Y.testsettings, where X is the webtest filename and Y is the testsettings filename, for example:

```
mstest /testcontainer:"C:
\Source\Security\Source\LoadTests\Sequel.Security.LoadTest\Authorization\EffectivePermissionsTest.webtest" /testsettings:"C:
\Source\Security\Source\Solutions\Local.testsettings"
```

3.9.2 Database Performance

This document describes the behaviour of Security servers (SecurityAPI, Authorization) related to database interactions and SQL statements executed.

Database

Security servers interact with these SQL databases:

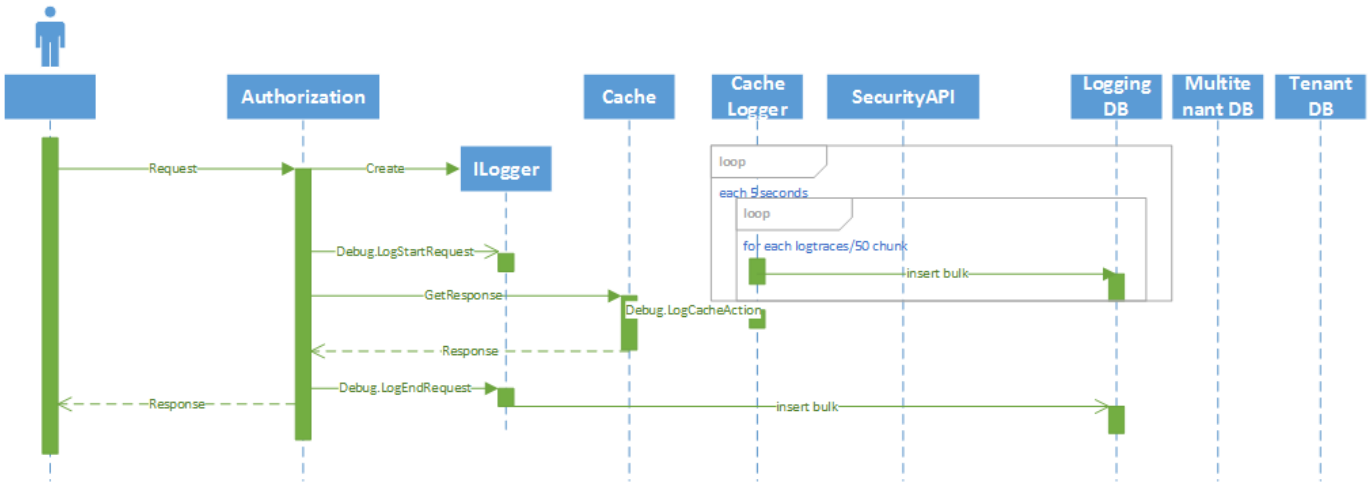
- Authentication: Logging DB.
- Authorization: Logging DB.
- SecurityAPI: Multi-tenant DB, Tenant DB, Logging DB.

As we see, only SecurityAPI interacts with business databases through two data context: authentication context and authorization context. Logging databases could be different for each server but in this document all servers share the same logging DB.

SQL statements sequence for Authorization server endpoints

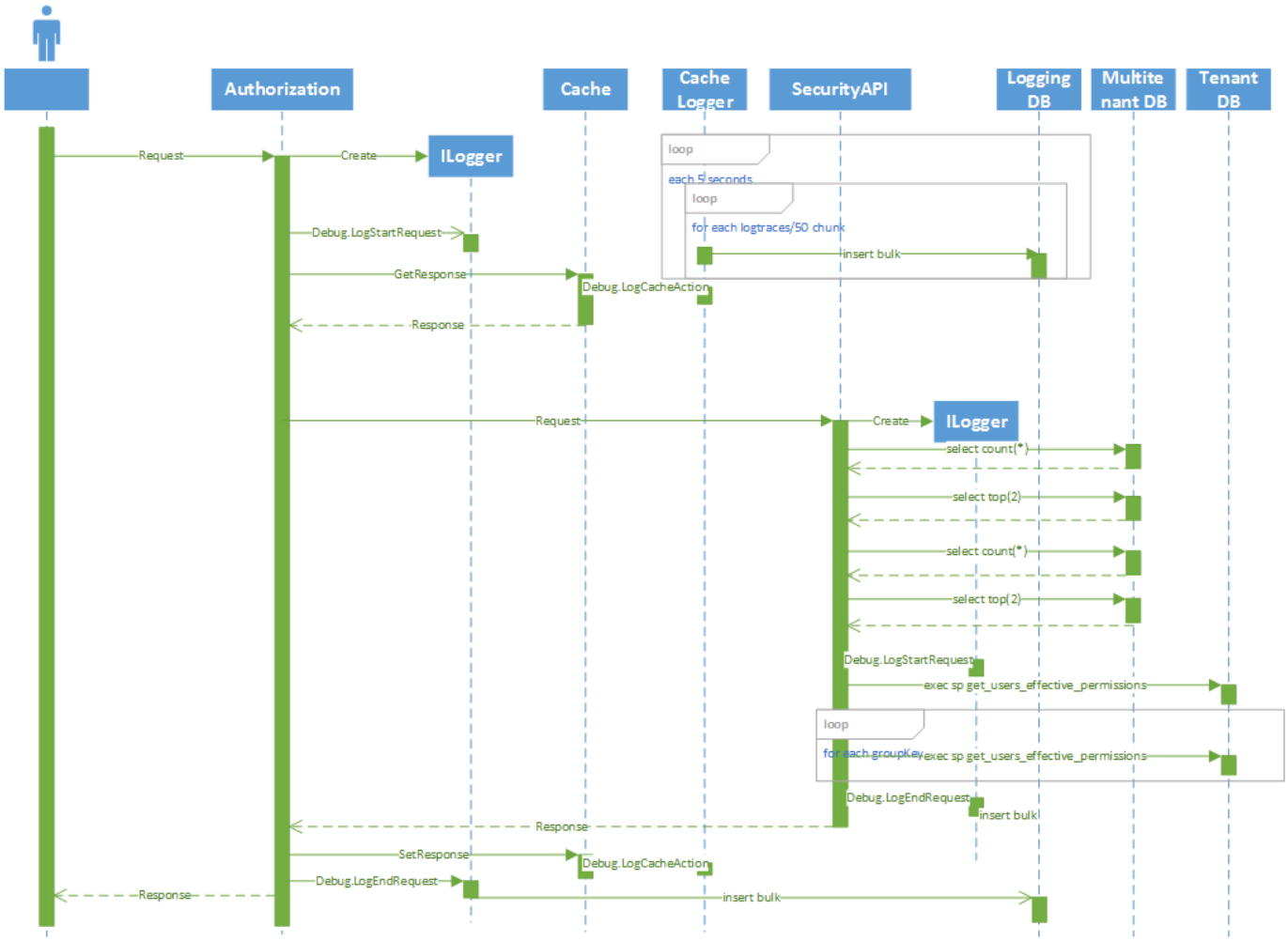
Authorization server has two GET endpoints HasPermissions y EffectivePermissions with same logging flow. We've considered only happy path scenario. There are two possible execution paths for each request depending on if request is cached or not:

1. Cache contains response



- Debug: **1** *insert bulk* per request. Each **5 seconds** several *insert bulk* operations depending on the number of cache interactions.
- Information: **0** *insert bulk* per request.

2. Cache not contains response



- Debug: **1** *insert bulk* per request and **4** *queries* required for database context creation and **1 to n** executions of *get_user_effective_permissions* store procedure depending on number of groupKeys supplied as requests' parameter. Each **5 seconds** several *insert bulk* operations depending on the number of cache interactions.
- Information: **0** *insert bulk* per request and **4** *queries* required for database context creation and **1 to n** executions of *get_user_effective_permissions* store procedure depending on number of groupKeys supplied as requests' parameter.

Load Tests Setup

We used the same configuration for testing the Authorization server endpoints:

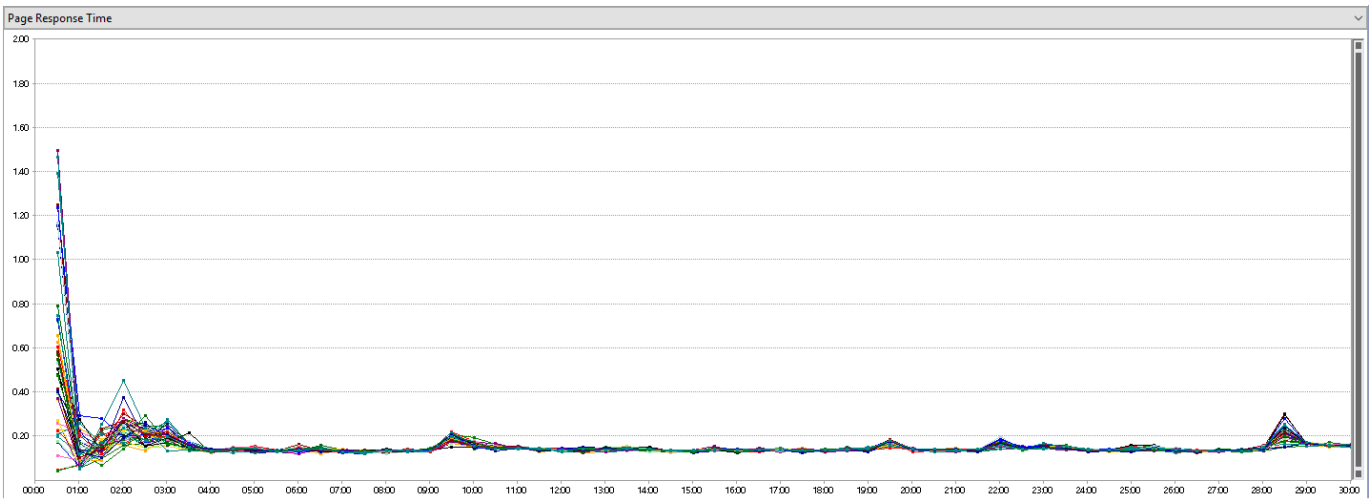
- 30 minutes running.
- Step goal of 100 users.
- Initial users are 10, adding 5 users each 10 seconds.
- Using Kangaroo environment.
- Data used: 2 applications, 5 groups, 5 users, 13 securables.

Results (Debug)

HASPERMISSIONS (DEBUG)

Results	
Max User Load	100
Tests/Sec	556
Tests Failed	0
Avg. Test Time (sec)	0.15
Transactions/Sec	0
Avg. Transaction Time (sec)	0
Pages/Sec	556
Avg. Page Time (sec)	0.15
Requests/Sec	556
Requests Failed	0
Requests Cached Percentage	0
Avg. Response Time (sec)	0.15
Avg. Content Length (bytes)	4.69
Total Tests	1,001,409
Failed Tests (% of total)	0 (0%)

Great performance in Request/sec (556), Avg. Response Time(0.15), Total Tests (≈1M) and Failed Test (0).



Cache usage		
Requests not in cache	808	0.08%
Requests in cache	1,000,601	99.92%
Total request	1,001,409	100%

High cache usage reduces in a great way the execution of sql statements.

SQL Statement	Times	Total Execution Time(s)	Average Execution Time (s)
insert bulk...	1,050,472	11,556.447392	0.011001
SELECT COUNT(*)...	1,616	0.053103	0.000033
SELECT TOP(2)...	1,616	0.053103	0.000033
exec [Authorization].get_user_effective_permissions...	96	0.033657	0.000351
exec [Authorization].get_user_effective_permissions...	96	0.038144	0.000397
exec [Authorization].get_user_effective_permissions...	96	0.027399	0.000260
exec [Authorization].get_user_effective_permissions...	96	0.024993	0.000260
exec [Authorization].get_user_effective_permissions...	96	0.029346	0.000306
exec [Authorization].get_user_effective_permissions...	49	0.017426	0.000355
exec [Authorization].get_user_effective_permissions...	49	0.013298	0.000271

`SELECT` operations are unnecessary repeated but them are executed fast (*0.033 ms*). Executions of SP are fast too, between *0.25 ms and 0.4ms*. Insert bulk operations are clearly slowest with *11ms*.

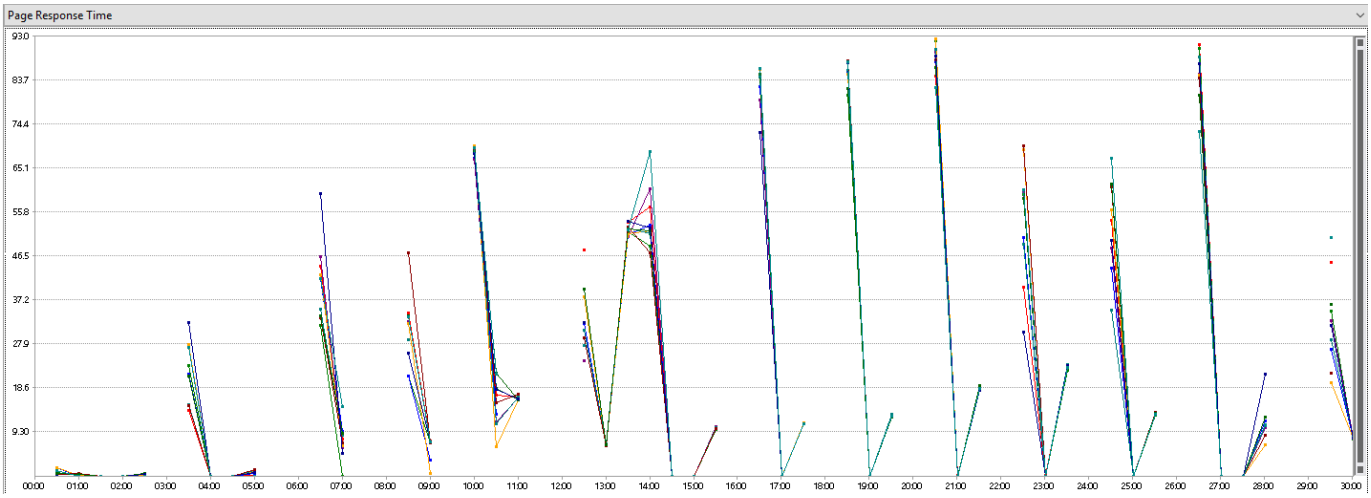
Top slowest statements	Execution Time(s)
insert bulk	4.746
insert bulk	3.925
insert bulk	2.989
insert bulk	2.792
insert bulk	1.996

This slowest statements *could* correspond to cache logger writings when concurrent users are 100. At this point there are 556+ *(CacheActionLogs/request*Request/sec*5s)*=(35565)/50≈167 concurrent insert bulk.

EFFECTIVEPERMISSIONS (DEBUG)

Results	
Max User Load	100
Tests/Sec	3.31
Tests Failed	0
Avg. Test Time (sec)	18.4
Transactions/Sec	0
Avg. Transaction Time (sec)	0
Pages/Sec	3.31
Avg. Page Time (sec)	18.4
Requests/Sec	2.88
Requests Failed	0
Requests Cached Percentage	0
Avg. Response Time (sec)	18.4
Avg. Content Length (bytes)	370
Total Tests	5,958
Failed Tests (% of total)	0 (0)

Low performance in Request/sec (2.88), Avg. Response Time(22.8s), Total Tests (≈5K).



Cache usage		
Requests not in cache	39	0.65%
Requests in cache	5,919	99.35%
Total request	5,958	100%

High cache usage reduces in a great way the execution of sql statements.

SQL Statement	Times	Total Execution Time(s)	Average Execution Time (s)
insert bulk...	7,759	46.311679	0.005969
SELECT COUNT(*)...	74	0.002629	0.000035
SELECT TOP(2)...	74	0.002629	0.000035
exec [Authorization].get_user_effective_permissions...	3	0.009615	0.003205
exec [Authorization].get_user_effective_permissions...	3	0.000750	0.000250
exec [Authorization].get_user_effective_permissions...	2	0.000731	0.000366
exec [Authorization].get_user_effective_permissions...	2	0.028336	0.014168
exec [Authorization].get_user_effective_permissions...	2	0.000455	0.000226
exec [Authorization].get_user_effective_permissions...	2	0.000448	0.000224
exec [Authorization].get_user_effective_permissions...	2	0.000627	0.000314

SELECT operations are unnecessary repeated but them are executed fast (*0.033 ms*). Executions of SP are fast mainly, between *0.25 ms and 0.36ms* but are two operations quite slow, between *1.4ms and 3.2ms*. Insert bulk operations are clearly slowest with *5.9ms*.

Top slowest statements	Execution Time(s)
insert bulk	16.996
insert bulk	10.011
insert bulk	0.499
insert bulk	0.073

Inexplicable results for the first two statements.

Commentaries

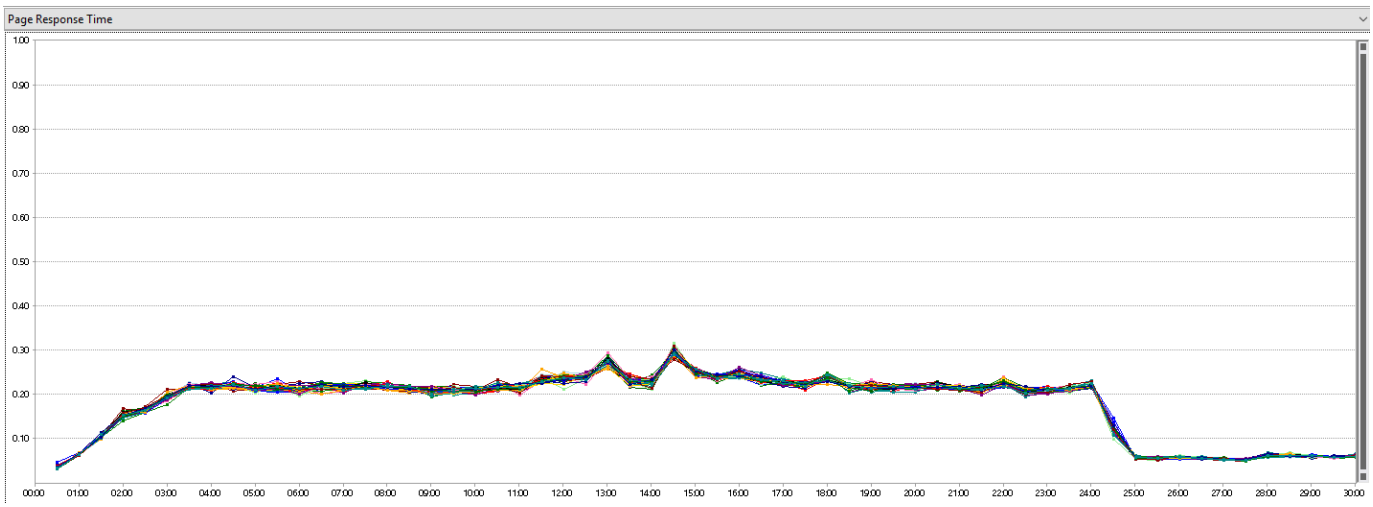
This results for EffectivePermissions are inexplicable because the execution path is pretty much similar to HasPermissions endpoint and it should be a slightly minor due to serialization of the response and response size. Many load test with empty logging database, different LoadTest database has been tried with no results.

Results (Information)

HASPERMISSIONS (INFORMATION)

Results	
Max User Load	100
Tests/Sec	448
Tests Failed	0
Avg. Test Time (sec)	0.17
Transactions/Sec	0
Avg. Transaction Time (sec)	0
Pages/Sec	468
Avg. Page Time (sec)	0.16
Requests/Sec	448
Requests Failed	0
Requests Cached Percentage	0
Avg. Response Time (sec)	0.15
Avg. Content Length (bytes)	4.71
Total Tests	843,249
Failed Tests (% of total)	0 (0%)

Great performance in Request/sec (448), Avg. Response Time(0.16s), Total Tests (≈850K) and Failed Test (0).



Cache usage		
Requests not in cache	≈1606	0.19%
Requests in cache	841,643	99.81%
Total request	843,249	100%

Cache has no impact on logging.

Generated logging traces		
Action logs	0	0%
Cache Action logs	0	0%
Total logs	0	0%

No Information logging traces.

SQL Statement	Times	Total Execution Time(s)	Average Execution Time (s)
SELECT COUNT(*)...	1,606	0.041983	0.002614
SELECT TOP(2)...	1,606	0.044243	0.002755
exec [Authorization].get_user_effective_permissions...	97	0.0.0237	0.000244
exec [Authorization].get_user_effective_permissions...	96	0.022077	0.000230
exec [Authorization].get_user_effective_permissions...	96	0.029830	0.000352
exec [Authorization].get_user_effective_permissions...	96	0.031195	0.000325
exec [Authorization].get_user_effective_permissions...	96	0.022982	0.002394
exec [Authorization].get_user_effective_permissions...	48	0.011293	0.000235
exec [Authorization].get_user_effective_permissions...	48	0.014901	0.000310
exec [Authorization].get_user_effective_permissions...	48	0.011135	0.000232

SELECT operations are unnecessary repeated but them are executed fast (*≈0.033 ms*). Executions of SP are fast, between *0.20 ms and 0.38ms*. No insert bulk operations due to cache configuration.

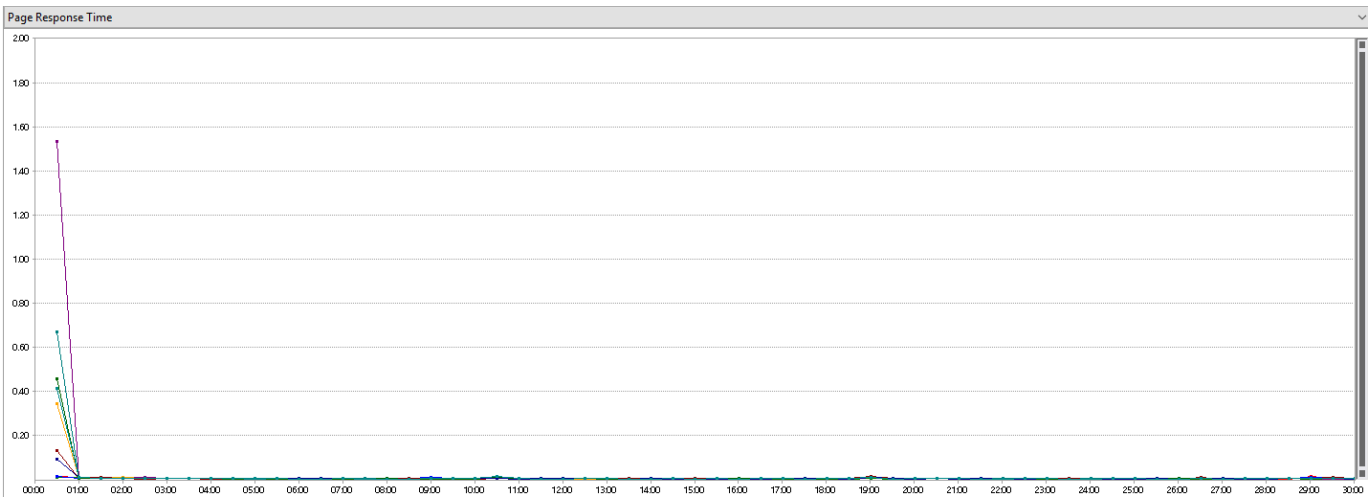
Top slowest statements	Execution Time(s)
exec [Authorization].get_user_effective_permissions...	0.000740
exec [Authorization].get_user_effective_permissions...	0.000515
exec [Authorization].get_user_effective_permissions...	0.000504
exec [Authorization].get_user_effective_permissions...	0.000497
exec [Authorization].get_user_effective_permissions...	0.000479

Good. Slower statement are executed between 0.48ms and 0.74ms.

EFFECTIVEPERMISSIONS (INFORMATION)

Results	
Max User Load	100
Tests/Sec	9.53
Tests Failed	0
Avg. Test Time (sec)	0.012
Transactions/Sec	0
Avg. Transaction Time (sec)	0
Pages/Sec	9.53
Avg. Page Time (sec)	0.01
Requests/Sec	9.53
Requests Failed	0
Requests Cached Percentage	0
Avg. Response Time (sec)	0.01
Avg. Content Length (bytes)	370
Total Tests	17,154
Failed Tests (% of total)	0 (0)

Moderate performance in Request/sec (9.53) and Total Tests (≈17K). Great performance in Avg. Response Time(0.01s).



Cache usage		
Requests not in cache	≈72	≈0.41%
Requests in cache	17,082	99.58%
Total request	17,154	100%

High cache usage reduces in a great way the execution of sql statements.

SQL Statement	Times	Total Execution Time(s)	Average Execution Time (s)
SELECT COUNT(*)...	72	0.002507	0.000035
SELECT TOP(2)...	72	0.002192	0.000030
exec [Authorization].get_user_effective_permissions...	3	0.000623	0.000208
exec

[Authorization].get_user_effective_permissions... | 3 | 0.000588 | 0.000294 | | exec [Authorization].get_user_effective_permissions... | 2 | 0.000704 | 0.000352 | | exec [Authorization].get_user_effective_permissions... | 2 | 0.000621 | 0.000311 | | exec [Authorization].get_user_effective_permissions... | 2 | 0.000680 | 0.000340 | | exec [Authorization].get_user_effective_permissions... | 2 | 0.000404 | 0.000202 | | exec [Authorization].get_user_effective_permissions... | 2 | 0.000749 | 0.000375 | | exec [Authorization].get_user_effective_permissions... | 2 | 0.000692 | 0.000346 |

SELECT operations are unnecessary repeated but them are executed fast (*≈0.033 ms*). Executions of SP are fast, between *0.20 ms and 0.38ms*. No insert bulk operations due to cache configuration.

Top slowest statements	Execution Time(s)
exec [Authorization].get_user_effective_permissions...	0.000421
exec [Authorization].get_user_effective_permissions...	0.000414
exec [Authorization].get_user_effective_permissions...	0.000404
exec [Authorization].get_user_effective_permissions...	0.000346
exec [Authorization].get_user_effective_permissions...	0.000344

Great. Slower statement are executed is less that 0.42ms

Commentaries

This results for EffectivePermissions are inexplicable because the execution path is pretty much similar to HasPermissions endpoint and it should be a slightly minor due to serialization of the response and response size. Many load test with empty logging database, different LoadTest database has been tried with no results.

Improvements

- Cache Multi-tenant DB operations could save unnecessary calls to this DB but saved time will be just a little.
- *get_user_effective_permissions* SP could be optimized and this way avoid some long execution peaks.

3.9.3 Logging Performance

This document describes the behaviour of logging in Security servers (SecurityAPI, Authentication and Authorization) and its impact on performance.

Logging Component

The Sequel.Core.Logging.Seriloger implementation of the Sequel.Core.Logging.ILogger interface is used in Security. Seriloger doesn't writes traces immediately into the database. Writings are executed in these cases:

- Each **5 seconds**.
- On dispose method. If logger is destroyed before reach 5 seconds from the last write then a flush is executed inside Dispose() method.

Before write in database, Seriloger can store an amount of logging traces only limited by system memory. All those traces are sent to database split in packets of **50 traces** an inserted in database using 'insert bulk' operations.

LOGGING USE IN SECURITY

Settings of Sequel.Core.Logging.Seriloger are stored in logging section of appsettings.json, and the values used by default in POTOROO and KANGAROO environments are:

appsettings.json "LoggingSettings": { "ConnectionString": "Server=SERVERNAME;Database=DATABASENAME;Trusted_Connection=True;MultipleActiveResultSets=true", "IgnoredLogTypes": [], "MinimumLogLevel": "Debug", "LoggingEnabled": "true" },

Each server Authorization, Authentication and SecurityAPI creates:

- A singleton instance of ILogger for registering application lifetime events (ApplicationStart, Application Stopping, ApplicationStopped)
- An instance for each request with a scoped lifetime.

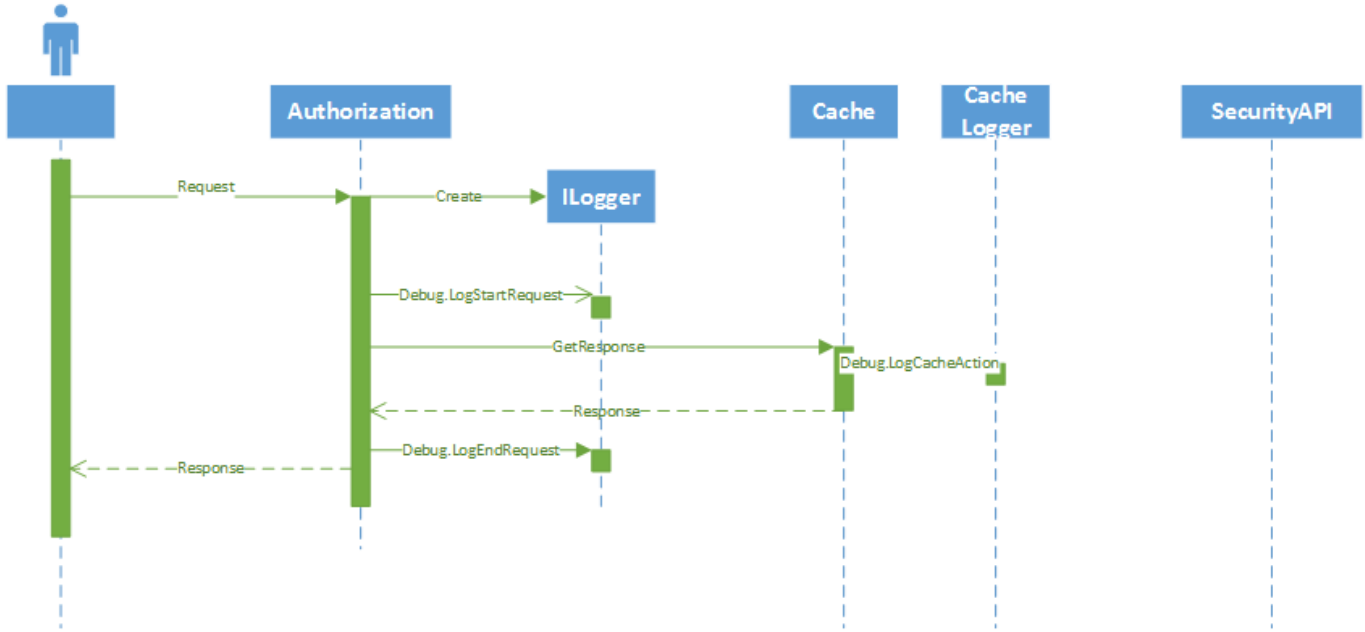
Additionally Authorization server:

- A singleton instance ILogger for request's cache.
- A singleton instance for Message Bus consumers.

Logging Sequence for Authorization server endpoints

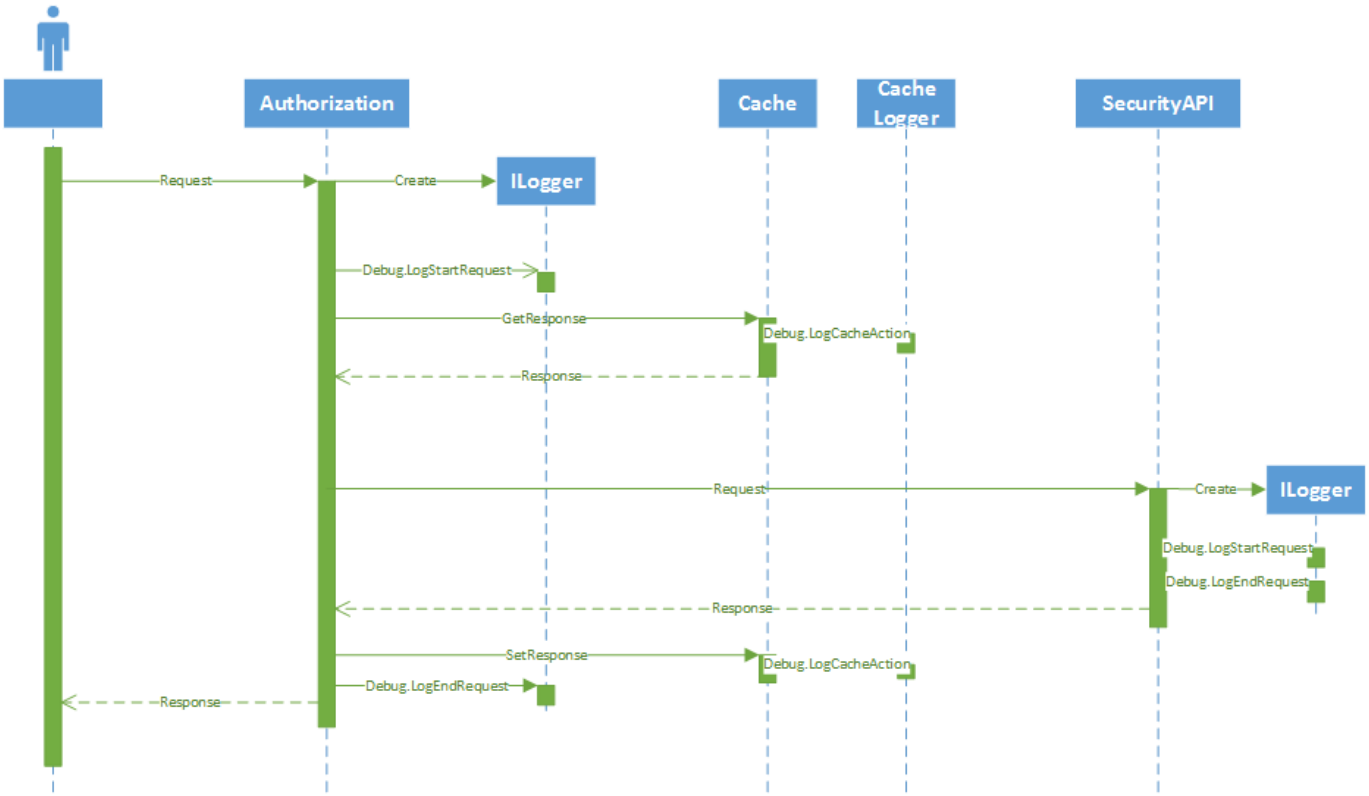
Authorization server has two GET endpoints HasPermissions y EffectivePermissions with same logging flow. We've considered only happy path scenario. There are two possible execution paths for each request depending on if request is cached or not:

1. Cache contains response



- Debug: **3 log** traces per request.
- Information: **0 log** traces per request.

2. Cache not contains response



- Debug: **6 log** traces per request.
- Information: **0 log** traces per request.

Load Tests Setup

We use the same configuration for testing the Authorization server endpoints:

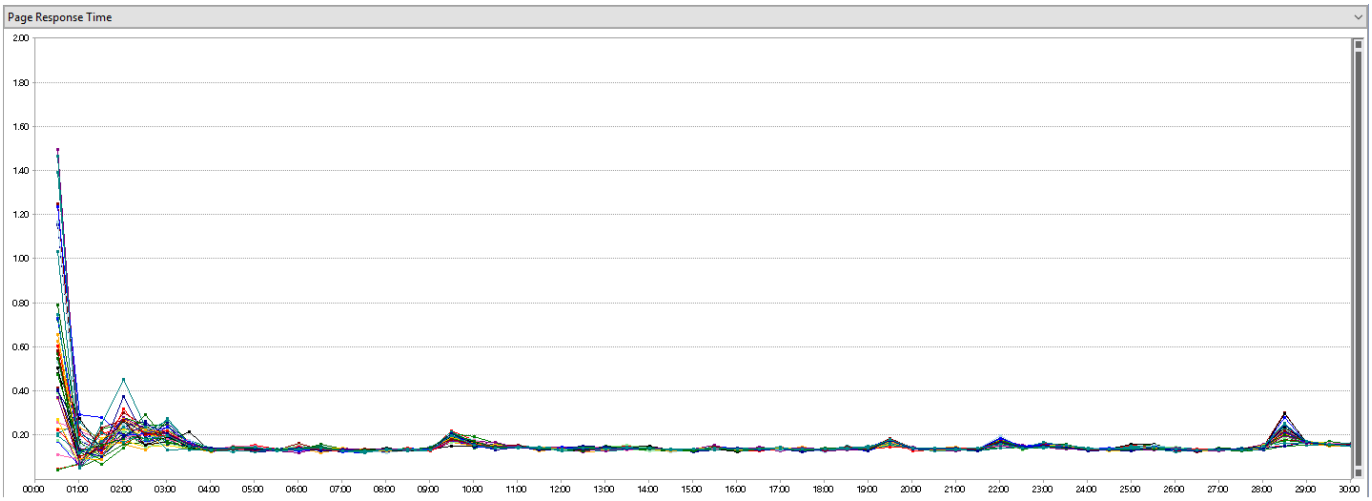
- 30 minutes running.
- Step goal of 100 users.
- Initial users are 10, adding 5 users each 10 seconds.
- Using Kangaroo environment.
- Data used: 2 applications, 5 groups, 5 users, 13 securables.

Results (Debug)

HASPERMISSIONS (DEBUG)

Results	
Max User Load	100
Tests/Sec	556
Tests Failed	0
Avg. Test Time (sec)	0.15
Transactions/Sec	0
Avg. Transaction Time (sec)	0
Pages/Sec	556
Avg. Page Time (sec)	0.15
Requests/Sec	556
Requests Failed	0
Requests Cached Percentage	0
Avg. Response Time (sec)	0.15
Avg. Content Length (bytes)	4.69
Total Tests	1,001,409
Failed Tests (% of total)	0 (0%)

Great performance in Request/sec (556), Avg. Response Time(0.15s), Total Tests (≈1M) and Failed Test (0).



Cache usage		
Requests not in cache	808	0.08%
Requests in cache	1,000,601	99.92%
Total request	1,001,409	100%

Most of request are cached reducing in a great way the page response time.

Generated logging traces		
Action logs	2,004,434	66.66%
Cache Action logs	1,002,217	33.33%
Total logs	3,006,651	100%

Most of request are cache hence each request write 3 logging traces as are describe in previous section

Sentence	Times	Total Execution Time(s)	Average Execution Time(s)
insert bulk	1,050,472	11,556.447	0.011

Numbers according with number of request and cache interactions.

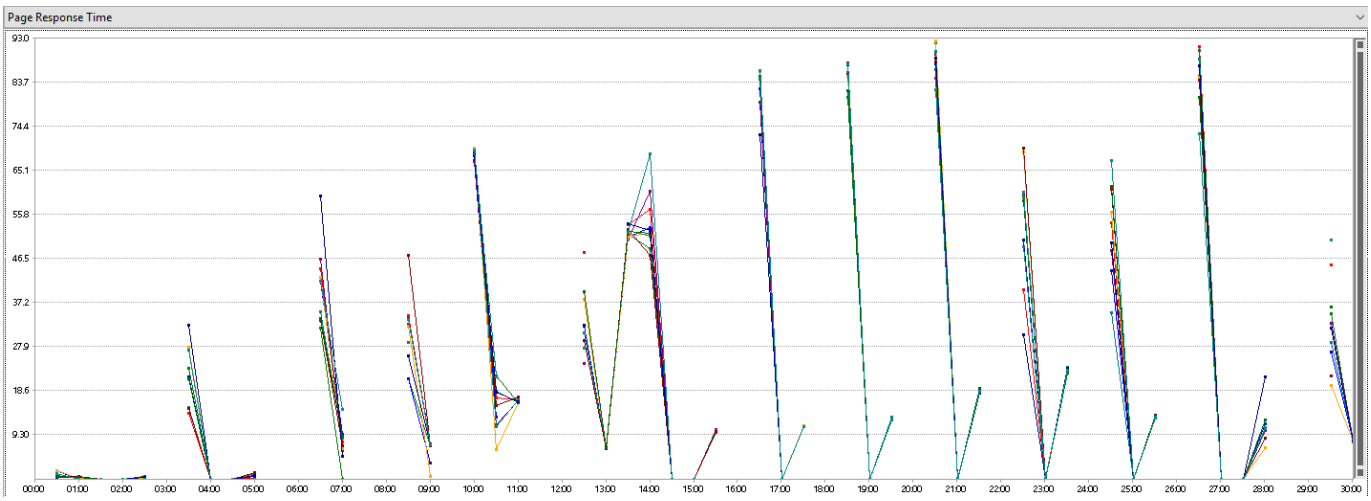
Top slowest sentences	Execution Time(s)
insert bulk	4.746
insert bulk	3.925
insert bulk	2.989
insert bulk	2.792
insert bulk	1.996

This slowest sentences *could* correspond to cache logger writings when concurrent users are 100. At this point there are 556+ *(CacheActionLogs/request*Request/sec*5s)*=(3*556*5)/50≈167 concurrent insert bulk.

EFFECTIVEPERMISSIONS (DEBUG)

Results	
Max User Load	100
Tests/Sec	3.31
Tests Failed	0
Avg. Test Time (sec)	18.4
Transactions/Sec	0
Avg. Transaction Time (sec)	0
Pages/Sec	3.31
Avg. Page Time (sec)	18.4
Requests/Sec	2.88
Requests Failed	0
Requests Cached Percentage	0
Avg. Response Time (sec)	18.4
Avg. Content Length (bytes)	370
Total Tests	5,958
Failed Tests (% of total)	0 (0)

Low performance in Request/sec (2.88), Avg. Response Time(22.8s), Total Tests (≈5K).



Cache usage		
Requests not in cache	39	0.65%
Requests in cache	5,919	99.35%
Total request	5,958	100%

High cache usage reduces in a great way the execution of sql statements.

Generated logging traces		
Action logs	12,120	66.44%
Cache Action logs	6,099	33.44%
Total logs	18,241	100%

Numbers according with number of request and cache interactions.

Sentence	Times	Total Execution Time(s)	Average Execution Time (s)
insert bulk	7,759	46.311	0.006

Number slightly oversized according with number of request and cache interactions.

Top slowest statements	Execution Time(s)
insert bulk	16.996
insert bulk	10.011
insert bulk	0.499
insert bulk	0.073
insert bulk	0.066

Inexplicable results for the first two statements.

Commentaries

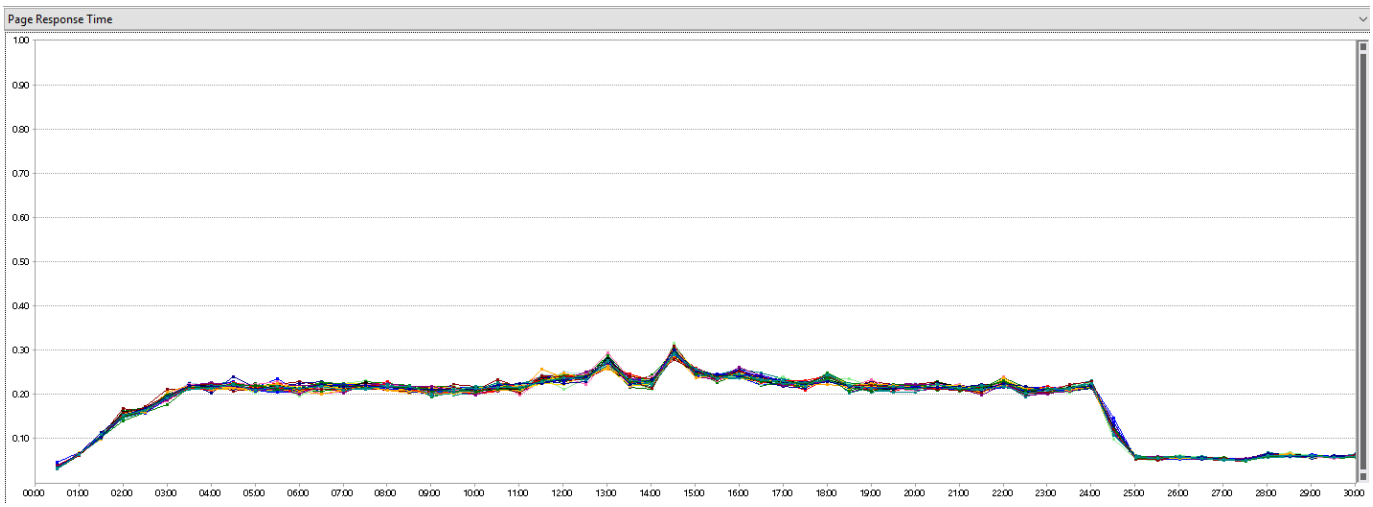
This results for EffectivePermissions are inexplicable because the execution path is pretty much similar to HasPermissions endpoint and it should be a slightly minor due to serialization of the response and response size. Many load test with empty logging database, different LoadTest database has been tried with no results.

Results (Information)

HASPERMISSIONS (INFORMATION)

Results	
Max User Load	100
Tests/Sec	448
Tests Failed	0
Avg. Test Time (sec)	0.17
Transactions/Sec	0
Avg. Transaction Time (sec)	0
Pages/Sec	468
Avg. Page Time (sec)	0.16
Requests/Sec	448
Requests Failed	0
Requests Cached Percentage	0
Avg. Response Time (sec)	0.15
Avg. Content Length (bytes)	4.71
Total Tests	843,249
Failed Tests (% of total)	0 (0%)

Great performance in Request/sec (448), Avg. Response Time(0.16s), Total Tests (≈850K) and Failed Test (0).



Cache usage		
Requests not in cache	≈1606	0.19%
Requests in cache	841,643	99.81%
Total request	843,249	100%

Cache has no impact on logging.

Generated logging traces		
Action logs	0	0%
Cache Action logs	0	0%
Total logs	0	0%

No Information logging traces.

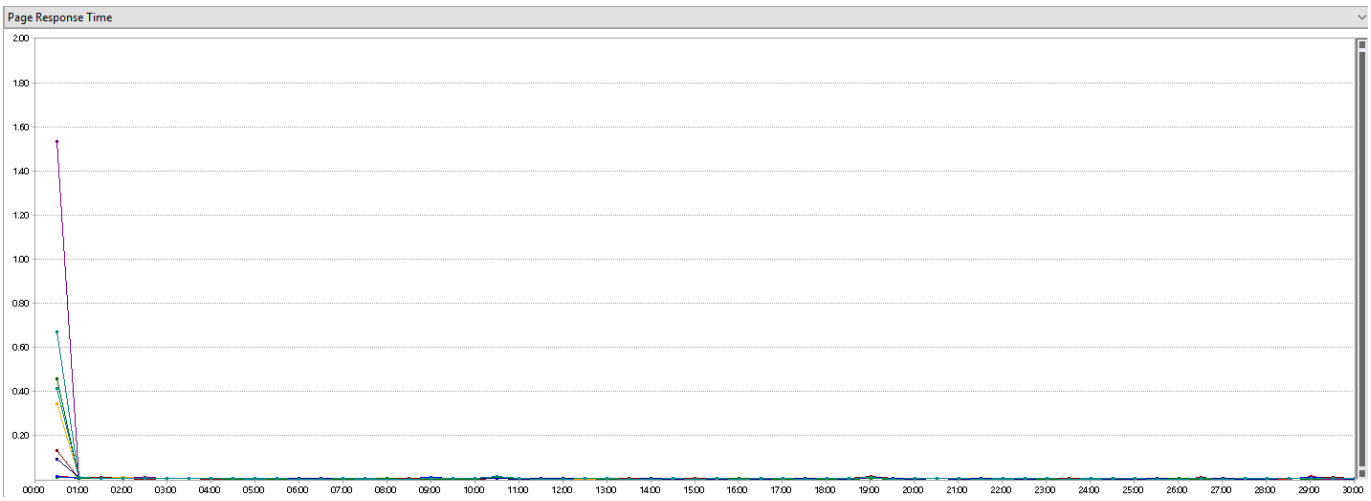
Sentence	Times	Total Execution Time(s)	Average Execution Time (s)
insert bulk	0	0	0

No insert bulk operations

EFFECTIVEPERMISSIONS (INFORMATION)

Results	
Max User Load	100
Tests/Sec	9.53
Tests Failed	0
Avg. Test Time (sec)	0.012
Transactions/Sec	0
Avg. Transaction Time (sec)	0
Pages/Sec	9.53
Avg. Page Time (sec)	0.01
Requests/Sec	9.53
Requests Failed	0
Requests Cached Percentage	0
Avg. Response Time (sec)	0.01
Avg. Content Length (bytes)	370
Total Tests	17,154
Failed Tests (% of total)	0 (0)

Moderate performance in Request/sec (9.53) and Total Tests (≈17K). Great performance in Avg. Response Time(0.01s).



Cache usage		
Requests not in cache	≈72	≈0.41%
Requests in cache	17,082	99.58%
Total request	17,154	100%

Cache has no impact on logging.

Generated logging traces		
Action logs	0	0%
Cache Action logs	0	0%
Total logs	0	0%

No Information logging traces.

Sentence	Times	Total Execution Time(s)	Average Execution Time (s)
insert bulk	0	0	0

No insert bulk operations

Commentaries

This results for EffectivePermissions are quite better that Debug inexplicable because the execution path is pretty much similar to HasPermissions endpoint and it should be a slightly minor due to serialization of the response and response size.

Improvements

- Include CacheAction in IgnoredLogTypes property in logging configuration section would reduce logging traces about a 33% and insert bulk operations depending on endpoint performance *(request/sec)*5s/50)*.
- Include Action in IgnoredLogTypes property in logging configuration section would reduce logging traces about a 66% and insert bulk operations depending on the number for request per second.

3.9.4 Setup Application Request Routing (ARR) Environment

This document provides information about setting up the ARR Environment using IIS; this is useful to testing scaling-out when load balancer are not available.

Procedure

1. Create 3 FVMs
2. Rdp into VM ARR_FVM
3. Install Microsoft Web Platform Installer
4. Run it and search for Application request routing.
5. Install Application request routing 3.0 or latest.
6. Open IIS Manager
7. Open Server Farms
8. Right click and click Create Server Farm
9. Enter a server farm name ARR_FVM and click next
10. Enter FVM 2 server address and click address
11. Enter FVM 3 server address and click address
12. Click yes to the following message **Rewrite Rules**: "There are URL rewrite rules on other server farms that can conflict with the rule that you are about to create for this server farm. Do you want to create this rule?"
13. For load balancing to work with Identity Server, the FVMs need to share Security Keys. To do this we need a shared windows folder. I put it on the ARR_FVM called AuthenticationSecurityKeys.
14. Set appsettings DataProtectionSettings for using Database to point to this as follows. a. Rdp into FVM 2 and FVM 3 b. Change in appsettings.json for projects so the urls point back to the ARR_FVM Authentication "DataProtectionSettings": { "Mode": "Database", "ExpirationInterval": "90.00:00:00" }, "SecurityApi": "http://ARR_FVM.office.sbs/SecurityApi" "SecurityApiConfigurationCacheEnabled": false

```
Authorization
    "AuthenticationServer": "http://ARR_FVM.office.sbs/Authentication",
    "SecurityApi": "http://ARR_FVM.office.sbs/SecurityApi"
    "SecurityApiAuthorizationCacheEnabled": false
    "MessageBusSettings": {
        "RabbitMqSettings": {
                "ServerUri": "rabbitmq://oxbus/ARR_FVM",
        },
        "Consumers": [
            {
                "QueueName": "EffectivePermissionChangedMessageName_FVM 1 or 2"
            }
        ]
    }

SecurityAPI
    "AuthenticationServer": "http://ARR_FVM.office.sbs/Authentication",
    "MultitenancyDatabase": "Data Source=ARR_FVM;Initial Catalog=MultiTenancyDatabase;Trusted_Connection=True;MultipleActiveResultSets=true"
    "SecurityApi": "http://ARR_FVM.office.sbs/SecurityApi"
    "ServerUri": "rabbitmq://oxbus/ARR_FVM",
```

You are then ready to send requests to the ARR_FVM machine such as login, logout, get access token, get effective permissions etc.

4. Operations handbook

4.1 Overview

> **Important**
>
> Sequel Security Service v3 is designed to be hosted only by Verisk's AWS Managed Service. For other hosting options, like on-premises, please contact with SuppApp Product Team.

Operations' Handbook contains information about:

- Platform specifications
- Installation guide
- General deployment topics: as internal and external traffic analysis, types of deployment, IIS configuration, Hangfire configuration, ...

4.1.1 Release notes

- Release Notes

4.2 Platform specs

> **Important**
>
> Sequel Security Service v3 is designed to be hosted only by Verisk's AWS Managed Service. For other hosting options, like on-premises, please contact with SuppApp Product Team.

This documentation refers to Security Services; outlines the server topology and specifies the pre-requisite software requirements to be used for the base platform of Sequel Security Service.

It covers all installation options and configuration of the operating systems and pre-requisite applications on all servers in the Sequel Security environment.

This specification requires that these servers are dedicated to the use of Sequel Security Service and are not shared with other services and do not have any additional software installed upon them.

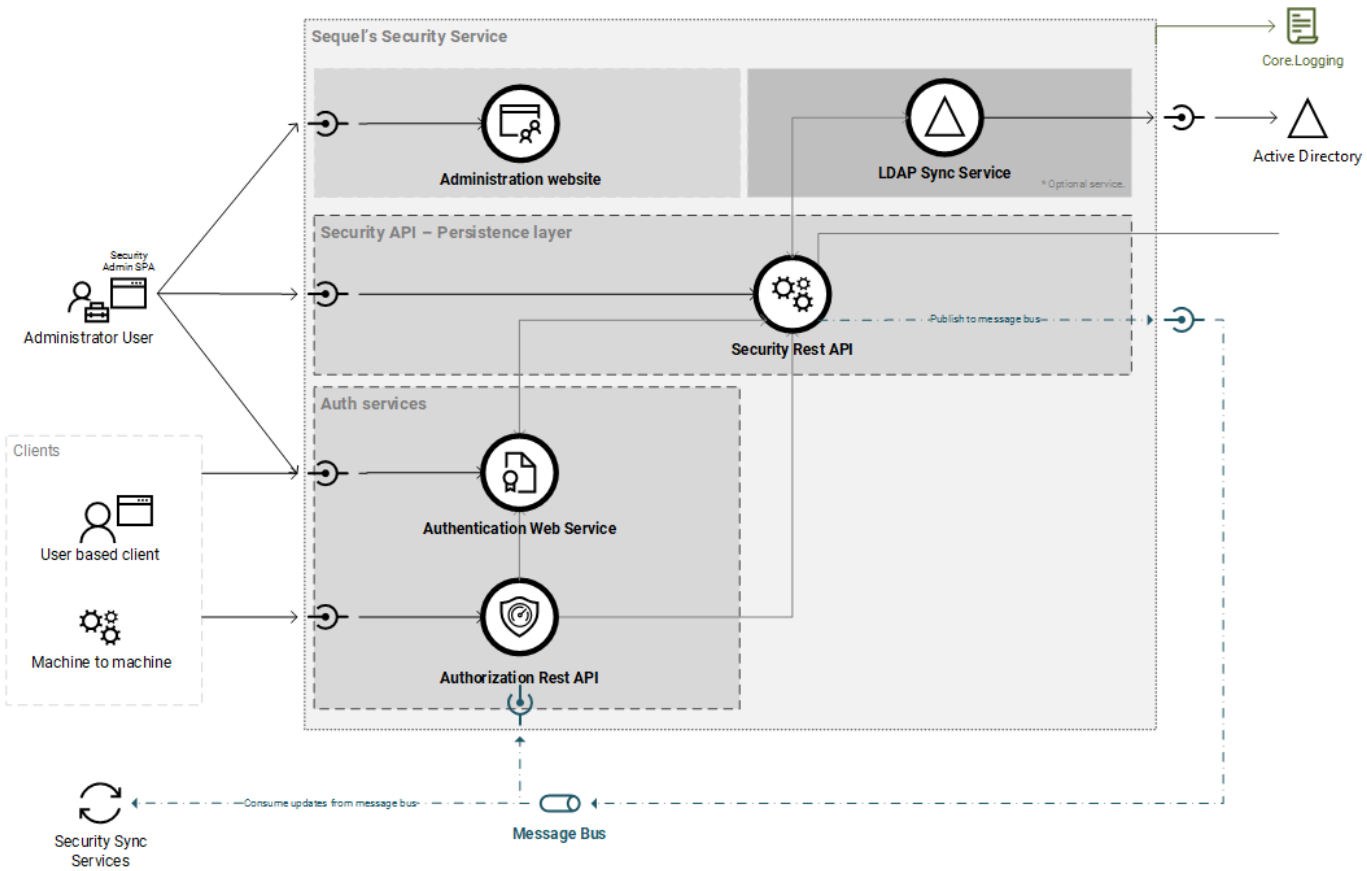
Virtualization of elements of the production environment is an option to consolidate server utilization.

Please note that we are keen to work in conjunction with you to ensure that the infrastructure is suitable and scalable for your requirements.

It is highly recommended to be familiar with the "Security Service Installation Guide" before installing the application.

4.2.1 Architecture overview

Sequel Security Application Services is based in a set of four web services/sites, developed in .NET Core.



Those services are:

AUTHENTICATION

OAuth2 and OpenID Connect service; provides interactive users and service-to-service authentication.

- Requires access to Sequel Security API Service.

AUTHORIZATION

Provides endpoints for verifying the permissions assigned to users and groups.

- Requires access to Sequel Security Authentication Service.
- Requires access to Sequel Security API Service.
- Requires access to an instance of RabbitMQ (consumer).

API

This Rest web API provides access to all information related to Authentication and Authorization; it's the bridge to the persistence layer.

- Requires access to Sequel Security Authentication Service.
- Requires access to a Microsoft SQL Server.
- Requires access to an instance of RabbitMQ (publisher).

ADMINISTRATION SITE

Single Page Application website for managing security configuration.

- Requires access to Sequel Security Authentication Service.
- Requires access to Sequel Security API Service.

OPTIONAL COMPONENTS

LDAP Sync Service

This is a windows service required to synchronize users and roles with a Windows Active Directory.

- Requires access to Sequel Security Authentication Service.
- Requires access to Sequel Security API Service.
- Requires access to an instance of RabbitMQ (consumer).

Azure AD Sync Service

This is a windows service required to synchronize users and roles with a Azure AD.

- Requires access to Sequel Security Authentication Service.
- Requires access to Sequel Security API Service.
- Requires access to an instance of RabbitMQ (consumer).

Security Database Sync Service

This windows service is responsible for syncing information from security services with Sequel's applications like Origin and Claims. This component is not included in the Security installation; it is included with the Origin and Claims installation. We recommend to install it in a Origin's or Claim' backend server.

Pre-requisites

Components required to install Security Services are:

- Database Server:
  - SQL Server 2014/2017/2019 or AWS SQL RDS
- Application Servers, can be installed on two different ways:
  - Hosted on IIS
    - Windows Server 2016 (recommended)
    - Windows Server 2012 supported.
    - *URL Rewrite* Module is required, https://www.iis.net/downloads/microsoft/url-rewrite
  - Hosted on AWS ECS
- RabbitMQ server 3.8.x (current test environments are pointed to 3.8.30)
  - Recommendation is to use the AmazonMQ RabbitMQ service

Topologies

Based on performance and security requirements, we will have different servers and topologies. The basics are described as:

- One dedicated database server (shared with other applications)
- One dedicated RabbitMQ server (single instance for Sequel Products)
- One or more application servers:
- All in one: all services in 1 server.
- Auth & App Servers. Recommended :
  - AuthServer: Authentication and Authorization Services.
  - AppServer: Administration and Security API service. None required for public/internet access. Just need to be accessible from AuthServer and other services. If LDAP Sync Service or Azure AD Sync Service are required, this is the recommended server to install them.
- Server per service. Each service is installed in a separate server:
  - Authentication Server
  - Authorization Server
  - AppServer (Admin UI + Security API + LDAP Sync Service + Azure AD Sync Service)
- All in one server shared with other services. While this is not our recommendation; this is possible if the expected load of work is low.

4.2.2 Database Server

This is the server specification for all database servers in Sequel Security Services environments. This applies just when SQL server is hosted on a Windows Server

Database Server: Operating system

- Microsoft Windows Server 2016 (recommended)
- Microsoft Windows Server 2012 R2

OPERATING SYSTEM CONFIGURATION

Configuration settings should be set to the default, except where referenced below.

- Windows Server 2012 R2 ‘Roles’ to install
  - Application Server
  - .NET Framework 4.5
- Regional Settings set to UK format (DD/MM/YYYY)
- Mixed Mode Authentication enabled

Database Server: Additional software

- Microsoft SQL Server 2014 SP2 (Supported)
  - Standard or Enterprise edition
- Microsoft SQL Server 2017 (supported and recommended)

Database Configuration

Database must be configured to work with:

- SQL Collation ‘SQL_Latin1_General_CP1_CI_AS
- READ_COMMITTED_SNAPSHOT enabled.

4.2.3 Application Server

This section applies when services are hosted on IIS.

Application Server: Operating system

- Microsoft Windows Server 2016 (recommended)
- Microsoft Windows Server 2012 R2 Standard 64-bit

OPERATING SYSTEM CONFIGURATION

Windows features (names for windows 2012 R2 version) settings should be set to the default, except where referenced below:

- File and Storage Services
  - File and iSCSI Services
  - File Server
  - Storage Services
- Web Server (IIS)
  - Web Server
    - Common HTTP Features
      - Default Document
      - Directory Browsing
      - HTTP Errors
      - Static Content
  - Health and Diagnostics
    - HTTP Logging
    - Request Monitor
  - Security
    - Request Filtering
    - Windows Authentication (optional - required to support SSO with Windows accounts)
  - Application Development
    - .NET Extensibility 4.6
    - ASP.NET 4.6
    - ISAPI Extensions
    - ISAPI Filters
  - WebSocket Protocol
  - Management Tools
    - IIS Management Console
- .NET Framework 4.6 Features
  - .NET Framework 4.6
  - ASP.NET 4.6
  - WCF Services
    - TCP Port Sharing
- Remote Differential Compression
- SMB 1.0/CIFS File Sharing Support
- Windows Defender Features
  - Windows Defender
  - GUI for Windows Defender
- Windows PowerShell
  - Windows PowerShell 5.1
  - Windows PowerShell ISE
- WoW64 Support

IIS application pool settings

Enable 32-bit Applications

Default settings of application pool works fine in most of the cases, and 32-bits is also supported. However, we recommend to use 64-bit mode (*Enable 32-bit Applications* set to false) for a better memory management.

Static compression

Static Content compression when enabled generates issues trying to access the application: 500 internal server errors.

Application Server: Additional Software

- Microsoft .NET Core 6.0 (we strongly recommend using minimum version 6.0.9, it contains important security patches from Microsoft).
  - Download from official site: https://dotnet.microsoft.com/download/dotnet/6.0.
- Install the .NET Core Hosting Bundle. Required to execute .NET Core web applications hosted in IIS.
  - More information: https://learn.microsoft.com/en-us/aspnet/core/host-and-deploy/iis/?view=aspnetcore-6.0
  - Direct download to current version: https://dotnet.microsoft.com/en-us/download/dotnet/thank-you/runtime-aspnetcore-6.0.9-windows-hosting-bundle-installer

It is quite important to install .NET Core Hosting Bundle aligned to the .NET Core version.

4.3 Installation guide

> **Important**
>
> Sequel Security Service v3 is designed to be hosted only by Verisk's AWS Managed Service. For other hosting options, like on-premises, please contact with SuppApp Product Team.

This document describes how to setup and install Sequel Security and setup connected services. The guide describes the typical installation of Sequel Security applications and is targeted mainly at IT professionals. Please accept defaults during installation unless they are not specified. Ensure no users are active in the environment before proceeding with installation, this includes batch users.

If there are any errors during installation, close Sequel.Deployment.Manager.exe and reopen it before retrying.

4.3.1 Pre-Requisite Information

Logging database

All Sequel Security applications have the ability to log information, errors, etc. to a logging database. This is important to have configured as it helps troubleshooting when we run into problems. The Deployment Manager does not create the logging database, it can only create the main database so this needs to be created beforehand and then referenced in the logging database settings during installation (*Global Settings* \ *Environment Settings* \ *Logging settings*).

> **Note**
>
> AWS Cloudwatch is also available as log store instead of logging database.

4.3.2 Installation Information

After reviewing the pre-requisite information and before starting installation with the Deployment Manager, we recommend becoming familiar with this guide and the design of the Security applications like the different types of authentication, certificates, components, scaling-out options, etc. All of these necessary topics can be found below in this section.

As a suggestion, before installing security, we recommend you to ensure the **pre-installation check-list** has been completed. All checks in this list are covered in this document during the next sections.

Pre-Installation Check-list	Done
Do you have a deployment diagram? (server, load balancers, interactions,...)	
Do you have the URL (public and private) of all involved servers and load balancers?	
Are the URL following the recommendations regarding domains?	
Do you have the required SSL certificates created with SAN as described?	
Do you have a valid certificate for signing the tokens?	
Are you going to use authentication with Azure AD? Do you have all required information?	
Are you going to use integration with Windows authentication?	
Do you have the email settings required for sending the reset/forgot password email?	
Are you going to require reCaptchas for resetting passwords? Do you have the information?	

Public vs. Private URLs

In the Sequel Security applications some communications are done using internal URLs and others are done using external ones. The internal URLs are private and are defined in the internal settings of the applications and can only be accessed between certain servers and applications in a private network, the external ones are public and can be accessed by external clients (like a web browser). Some examples of use are:

- Public URLs are for *interactive users*: logging in users, redirection that affect the browser, browser navigation, etc.
- Private URLs are for *service-to-service* communication.

The public and private URLs are always going to be the same and are so by default when using the Deployment Manager to install the Sequel Security applications for private networks (like configuration environments or system test environments, that does not require to be exposed to internet). For production environments, where the application is exposed to internet those URL are different. If for example we are deploying to an AWS, we have a private IPv4 and a public IPv4 address. We could then use these different addresses to maintain internal communication through our private AWS network to add that extra layer of security.

Another use case for different private and public URLs is for the use of load balancers. It can be either on premise load balancers or Amazon's Elastic Load Balancing (ELB). The load balancer will normally need a public facing address otherwise it cannot listen to external requests from the Internet. Then, the communication between the load balancer and the instances underneath will ideally use private addresses to keep the internal traffic in the private network.

The installation process will always ask us for the URL of the service (this is the private) and the public URL.

If some URL includes a *path* it's highly recommended to set this value using the PascalCase naming convention and follow this convention on all applications referring to this URL.

References:

- https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-instance-addressing.html#using-instance-addressing-common

Domains

It's important to keep in mind the use of domains and sub-domains when installing Security as they need to be correctly set for the cookies to be visible on the same domain by all the Sequel applications. This has to be done this way so that the single sign out and single sign in can function correctly, and also so that different Sequel clients can't see the cookies from another client. Below is an example of how the public URLs should look versus how they shouldn't so that we can better understand the differences:

- Bad configuration for 2 different clients:
  - securityclient1.sequel.com
  - workflowclient1.sequel.com
  - securityclient2.sequel.com
  - workflowclient2.sequel.com

As we can see here, this is a common mistake, as the domain here is actually "sequel.com", therefore both clients have all applications on the same domain so their cookies are visible between them, so when closing the session from one client it will close the sessions from the others as well. However, because of how the token are granted the user does not have access to other environments; but they will override the cookies causing strange behaviors in the navigation.

- Correct configuration for 2 different clients:
  - security.client1.sequel.com
  - workflow.client1.sequel.com
  - security.client2.sequel.com
  - workflow.client2.sequel.com

Here we have a correct configuration as the domain and the subdomain for each client are different "client1.sequel.com" for the first and "client2.sequel.com" for the second, so the cookies are only visible on their respective domains and SSI and SSO will work in an isolated manner as expected.

During the installation, the *security domain* and the *public security domain* are asked. The *public security domain* is defined with the **CookieDomain** attribute during installation process, this attribute specifies which hosts are allowed to receive the cookie. For allowing sub-domains, this value must be prefixed with a dot. As a sample, uat.sequel.com will be used for the domain, while .uat.sequel.com will be used for sub-domains too. In general, the dot prefixing the domain is preferred as we use sub-domains.

SSL Certificate

For a secure connection to be made between the web browser and the Sequel Security applications, or between different services inside the Sequel Security applications, it's necessary for the connection to be encrypted using HTTPS (default and recommended binding protocol configured in the Deployment Manager). For this, the web server must have a valid SSL certificate installed, issued by a Certificate Authority.

To obtain and install this certificate on IIS for a Windows Server please consult the following guide: https://www.digicert.com/kb/csr-creation-ssl-installation-iis-10.htm.

For an AWS instance please refer to Amazon's Certificate Manager (https://aws.amazon.com/certificate-manager/) for creating and managing SSL certificates. There is also an AWS Certificate Manager Private Certificate Authority (https://aws.amazon.com/certificate-manager/private-certificate-authority/) that extends the certificate manager's capabilities to both public and private certificates for use in EC2 instances.

With the certificate created and installed on the server we need to retrieve it's thumbprint as it is a required parameter during installation. We can find this value in the 'Details' tab of the certificate which will be in the servers certificate store. This value is for the parameter *Global Settings \ IIS Settings \ Https Certificate Thumbprint* of the Deployment Manager.

MANAGING CERTIFICATES FOR THE PUBLIC AND PRIVATE URLS

Previously, we have spoken about public and private URLs. We need to take these into account for the SSL certificates too, because we want to use the same certificate for both the internal and the external URLs (multi-domain certificates). To achieve this, we need to use Subject Alternative Names (SAN) which allows us to add a list of valid domains for one specific certificate. This can also be found in the 'Details' tab of the certificate and can be modified there. The main DNS Name it should contain will be the external URL, we need to make sure the internal URLs are added too.

References:

- https://www.digicert.com/ssl/
- https://www.digicert.com/multi-domain-ssl/
- https://www.digicert.com/csr-creation-ssl-installation-iis-10.htm
- https://www.digicert.com/subject-alternative-name.htm

The URL where the Sequel Authentication application is (or will be) available (protocol://subdomain.domain.tld/path) is case-sensitive and it's highly recommended to set this value using the PascalCase naming convention (e. g. protocol://subdomain.domain.tld/Authentication) and follow this convention on all applications referring to this URL. Same rule should be applied for all services. This is required for some cookies that are defined at path; a request-path matches a given cookie-path if at least one of the following conditions holds:

- The cookie-path and the request-path are identical.
- The cookie-path is a prefix of the request-path, and the last character of the cookie-path is %x2F ("/").
- The cookie-path is a prefix of the request-path, and the first character of the request-path that is not included in the cookie-path is a %x2F ("/") character.

Scaling-out

An important feature with the Sequel Security applications is the ability to scale-out each application, therefore being able to balance out the load across different servers when a lot of traffic is expected. Let's quickly go over each application and why this would be useful and what to take into account:

- **Administration**: the Security Administration web app is a simple SPA application developed using React capable of handling plenty of requests as not many users will be connected simultaneously plus it's content is mostly static, so the only reason to have several servers would be for failover reasons, if one server fails, we can redirect traffic to the other. Because of this, there are no special concerns of things to take into account if this component were scaled out.
- **Authorization**: the same applies for Authorization, main reasons for using several servers would be for failover and there isn't anything in particular to take into consideration. Authorization receives plenty of requests but has a cache system in place to maximize speed and stability.
- **Authentication**: this would probably be the most ideal one to scale-out as Authentication receives many requests regarding logins, log-outs, token verification, handling session timeouts, etc. If used, we must make sure that the all other applications are pointing to the Authentication's public load balancer URL. We must also keep in mind that when scaling out servers with Duende IdentityServer they each have to share the ASP.NET Core data protection keys. Without this they act independently and the applications will log in and out depending on the server that was hit (more information on this here: https://docs.duendesoftware.com/identityserver/v6/deployment/). The recommended setting for scaling out the authentication service is setting `DataProtectionSettings.Mode` to `Database`.
- **Security API**: scaling out the Security API can also be of great use as the API retrieves plenty of information from the database to respond to many requests from different applications. Just like Authentication, the only thing to consider is to make sure all the rest of the applications (especially Authorization) are pointing to the APIs public load balancer URL.

Security behind Load Balancers

When deploying Security to AWS behind a Load Balancer or behind any Reverse Proxy (like nginx, traefik, kong...) and HTTPS is required, Security must trust some headers sent by the Load Balancers/Reverse Proxies. Failing to do so, Security will not work properly, with errors like unable to log in in Administration or infinite redirection loop in Authentication. To enable the trust on these headers, set the `TrustForwardedHeaders` setting to `true` on Security API, Authentication and Authorization.

Certificate generation for signing tokens

In production, we need a constant self signed certificate that will be used by the Authentication server to sign tokens. To achieve this we need to use the **makecert** and **pvk2pfx** utilities to create a pair of certificates. These two executables can normally be used in a Visual Studio Developer Command Prompt, if not, then they can be found in the Windows Software Development Kit for the specific version of Windows being used (https://developer.microsoft.com/en-us/windows/downloads/sdk-archive). After installing the required version, the executables, by default, will be in 'C:\Program Files (x86)\Windows Kits\'.

We will need to generate a Certificate Authority and a Personal Information Exchange (that contains a Private Key and a Public Key).

- `_makecert -n "CN=NameOfCertificate" -a sha256 -sv signing-certificate.pvk -r signing-certificate.cer_`
  - This will create a new self-signed certificate with it's public key in 'signing-certificate.cer' which is exposed to clients so that they can validate tokens, and it's private key in 'signing-certificate.pvk' used to sign the tokens.
- `_pvk2pfx -pvk signing-certificate.pvk -pi PASSWORD -spc signing-certificate.cer -po PASSWORD -pfx signing-certificate.pfx_`
  - This will combine the PVK and CER files into a single PFX file containing both the public and private keys for the certificate.

This PFX file can now be used. In the installation process with the Deployment Manager when we come across the parameter 'Signing certificate: file' here we will need to introduce the path of the PFX file. It's recommended to simply copy and paste this file in '{DeploymentManagerRoot}\Packages\Authentication\Files', this way the certificate is in the root of where all the Authentication server files are and this parameter can be filled with just the name of the certificate and not worry about the path (e.g 'Signing certificate: file': signing-certificate.pfx).

The parameter 'Signing certificate: password' during installation is the 'PASSWORD' value used to create the PFX file.

Important: Certificate used in production environments must be unique. Do not reuse the same production certificate in other environments it will cause a high security breach.

References:

- http://amilspage.com/signing-certificates-idsv4/
- https://devblogs.microsoft.com/aspnet/asp-net-core-authentication-with-identityserver4/

Authentication providers

There are 3 different authentication providers available in the Security apps, so 3 different ways of signing in. We can enable each of them during the installation process to activate them on the Authentication page and then use whichever one we want. They can be found during the Deployment Manager installation in the section *Installation Parameters\Sequel Security\Authentication Server*.

SEQUEL IDENTITY

Sequel Identity or Password based. The most typical way to sign in to the Security apps is with user accounts created through the Security Admin website that are stored directly into the Security database. By default, we should create an initial ADMIN user with the Deployment Manager, this will allow us to use the Sequel identity method to sign in with the user name and password chosen for our ADMIN user and then start creating new users that will also be able to sign in with this method.

WINDOWS AUTHENTICATION

To make use of the Windows Authentication sign in method, a pre-requisite is to have this feature activated in Windows. To do this, we must tick the option to activate and install Windows Authentication on the server. It can be found under 'Server roles' if using a Windows Server or under 'Windows features' if using a normal Windows OS. The option is located:

- Windows Server: *Web Server (IIS) \ Web Server \ Security \ Windows Authentication*
- Windows: *Internet Information Services \ World Wide Web Services \ Security \ Windows Authentication*

Once we have this, we must configure the Authentication settings of the Internet Information Services (IIS) on the server where the Authentication server is installed (or will be). Both the 'Anonymous Authentication' and 'Windows Authentication' settings must be set to 'Enabled'. Lastly, when making use of this method to sign in we first have to make sure we have a Sequel user created with the same username as the one used to sign into the Windows session we are currently using. For example, if my username in Windows is 'ABC123', then there must be a user in the Sequel Security database with this same username or SsoUsername.

MICROSOFT AZURE ACTIVE DIRECTORY

This provider uses the Microsoft login service to authenticate users. Before using it, it's necessary to register the Sequel Authentication application in Microsoft's application registration portal by creating a new app (https://apps.dev.microsoft.com/). Once we have done this we will need to set or retrieve three values:

- Client secret: we must create a new client secret in Microsoft's application registration portal and save it.
- Application (client) ID: generated automatically when we have added our new app.
- Redirect URLs: we must set a new redirect URL which represents the URL Microsoft redirects us to after authenticating a user. It must match with the value of the setting 'LoginSettings:IdentityProvidersSettings:Microsoft:CallbackPath' in the appsettings.json file of Authentication. By default, this value is set to '/signing-microsoft', therefore this redirect URL should look something like this: 'https://{AuthenticationServer}/Authentication/signing-microsoft'.

ISO CLAIMSEARCH

This provider uses elements provided by ISO ClaimSearch to authenticate users. We need to set two values:

- Login Url: ISO ClaimSearch login page. Users will be redirected to this URL if they are not already authenticated in ISO ClaimSearch.
- Session Validation Endpoint: Endpoint to check if user have a valid ISO ClaimSearch's session.

OKTA

This provider uses a Okta to authenticate users. Before using it, it's necessary to register the Sequel Authentication application in the Okta domain. Once we have done this we will need to set or retrieve three values:

- Domain: Okta's domain (e.g. *mycompany.okta.com*)
- ClientId: ID generated by Okta for the application.
- Client secret: Secret generated by Okta for the application.
- AuthorizationServerId: ID of authorizations server. Used *default* if no value is provided.

JUMPCLOUD

This provider uses a JumpCloud to authenticate users. Before using it, it's necessary to register the Sequel Authentication application as SAML SSO application in JumpCloud console. Once we have done this we will need to set or retrieve three values:

- SPEntityId: Unique identifier used by configured Service Provider
- LoginUrl: JumpCloud URL for the login process (e.g. https://sso.jumpcloud.com/saml2/security-authentication).
- X509SigningCertificate: X509 certificate used JumpCloud SSO Application for signing its responses.

SMTP

If we are using the Sequel identity authentication provider then one required functionality is the ability to reset passwords. For this to work we will need a working SMTP server so that the Security applications can send the corresponding user a confidential email containing the link to reset their password.

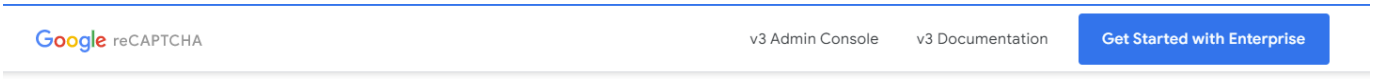
The settings to take into account and therefore the information you will need from the SMTP to be used are the following:

- *Installation Parameters* \ *Security API* \ *SMTP host name*: host name or IP address of the SMTP server.
- *Installation Parameters* \ *Security API* \ *SMTP host port*: port number to connect to the SMTP server.
- *Installation Parameters* \ *Security API* \ *SMTP host username*: user name of the email account that will be used on the SMTP server. Keep it empty if the SMTP server does not support authentication (seen in some AWS deployments).
- *Installation Parameters* \ *Security API* \ *SMTP host password*: password of the email account that will be used on the SMTP server.
- *Installation Parameters* \ *Security API* \ *Secure socket options*: allows us to configure SSL for sending email in different ways. Possible options:
  - None: No SSL or TLS encryption should be used. This setting is the most common in AWS deployments.
  - Auto: Allow to decide automatically which SSL or TLS options to use (default). If the server does not support SSL or TLS, then the connection will continue without any encryption.
  - SslOnConnect: The connection should use SSL or TLS encryption immediately.
  - StartTls: Elevates the connection to use TLS encryption immediately after reading the greeting and capabilities of the server. If the server does not support the STARTTLS extension, then the connection will fail and a NotSupportedException will be thrown.
  - StartTlsWhenAvailable: Elevates the connection to use TLS encryption immediately after reading the greeting and capabilities of the server, but only if the server supports the STARTTLS extension.
- *Installation Parameters* \ *Security API* \ *Validate server certificate*: determines whether or not the SMTPs server certificate should be validated. Defaults to true; if false, the server certificate will always be trusted and not be validated.
- *Installation Parameters* \ *Security API* \ *Enable send forgot password email*: determines whether or not the ability to reset passwords via email is possible.
- *Installation Parameters* \ *Security API* \ *Forgot password email address*: email address that will be used for sending the forgot password link.

reCAPTCHA

In the Security Authentication web application, when requesting a password reset because we have forgotten our password or because we are simply requesting an initial one, there is a captcha verification before being able to click on the 'Reset Password' button. We are using Google's reCAPTCHA v2 system (https://www.google.com/recaptcha) and it is necessary for every installation to have their own captcha created and configured from Google's reCAPTCHA admin console.

To be able to generate V2 captchas we need to select the "V3 Admin Console" option in the link above. As the "Get Started with Enterprise" option will lead us to the Google Cloud console which only generates Enterprise Captcha keys.



Here, we need to register a new web site filling in it's name, the type of reCAPTCHA (select v2) and a domain it is valid for. The domain is important as here we will need to enter in the URI where the Sequel Authentication Server is (or will be) available (subdomain.domain.tld). Lastly, we can configure how restrictive the check is, from simply showing a checkbox to having to classify photos, for example.

During the installation process of Authentication we will encounter 3 parameters related to the captcha. One to enable it or not, and then the data site key and the secret key, both can be found in the settings section of the previously configured web site in the Google admin console for reCAPTCHA.

Databases

Regarding the databases used in the Sequel Security applications, we have the following:

- multi-tenancy database: contains the list of valid tenants that can access the apps. Each tenant is represented by a row in the **multi-tenancy.Tenant** table and contains the connection string for that tenant's main database. Current version supports only a single tenant.
- Main database: the connection string for the main database does not appear in any of the applications settings, it is set in the multi-tenancy database as explained above. This is the most important database as it contains the roles, securables, users, etc. It is important to make sure the multi-tenancy database is set correctly during the installation process and that the main database is correctly set in its corresponding tenant row also.
- Logging database: used for saving logs and needs to be created prior to installation as explained in the 'Pre-Requisite Information' at the beginning of this guide.

It's important to note that the isolation level of all databases should be set to "read committed snapshot", this means the database engine uses row versioning and snapshot isolation as the default, instead of using locks to protect the data.

Also, if using high-availability and disaster-recovery SQL servers then the 'Always On availability groups' (AG) feature must be activated. For this the parameter *Global Settings \ Data Sources \ AG enabled* in the Deployment Manager must be set to true. This will set the 'MultiSubnetFailover' option of the connection strings to true.

References:

- https://docs.microsoft.com/en-GB/dotnet/framework/data/adonet/sql/snapshot-isolation-in-sql-server
- https://docs.microsoft.com/en-GB/sql/database-engine/availability-groups/windows/always-on-availability-groups-sql-server

Swagger

The Authorization and the Security API applications use Swagger (https://swagger.io/) for easy access to all the API calls and their documentation. This is especially useful for internal use when testing but **we don't want Swagger to be activated in production**. For this, both the *Installation Parameters \ Security API \ Swagger Enabled* and the *Installation Parameters \ Authorization Server \ Swagger Enabled* parameters should be set to false when installing the applications for a client.

4.3.3 Deployment Manager

The installation in deployment manager is structured in three modules: database, web components and sync services. Our recommendation is start installing the database first and then the web components. If the sync services are required, it will be installed after installing the other modules.

Databases

For installing the multi-tenancy and single tenant (aka security) databases with the vanilla configuration for authorization and the specific authentication configuration for the security services in this environment.

More information at database installation guide.

Security applications

For installing Sequel Security Web services:

- *Security API*
- *Authentication Server*
- *Authorization Server*
- *Administration Web*

More information at application installation guide.

Sync Services

There are two synchronization services related with security:

- The *Security Sync Service*, that is responsible of keeping up to date the `security` schema on *Sequel Workflow*, *Sequel Broking*, *Sequel UW*, and *Sequel Claims* databases with information from the Security service. Each target database requires a sync service configured; in some cases, like *Sequel Claims*, this service has been replaced by an specific version owned by Claims.
- The *LDAP Sync Service*, that is responsible of synchronization of users, roles and groups at Security from a Windows AD. This service is optional and depends on project requirements.
- The *Azure AD Sync Service*, that is responsible of synchronization of users, roles and groups at Security from a Azure AD. This service is optional and depends on project requirements.

More information at sync services installation guide.

OTHER SETTINGS

There are other settings included at the appsettings.json files that are not configurable during installation with the Deployment Manager:

Administration

Parameter	Description
`allowHttp`	Allows access to Security Admin through a HTTP connection. This must be true always in production.
`authenticationFlow`	Defines the flow type used for authentication. Default and recommended value is *authorizationCode*; *implicit* is also accepted and is the previous flow used. Setting this value back to implicit will require to ensure that there are not *AllowedGrantTypes* configured for `sec.app.admin` client of type *authorizationCode* and there is one of type *implicit*.

Authentication

Parameter	Description
`SingleSignOnSettings. RequestRulesForIgnoreCookieRenewal`	List of strings that are used to ignore certain HTTP calls , these HTTP calls cannot be avoided but generate traffic that should not be considered as activity therefore should not refresh the cookies.
`SameSiteCookiePolicyDisabled`	Disables cookie policy for right management of security cookies with *SameSite=none* attribute depending on browser. Default value is *false*.
`DataProtectionSettings`	Determines how the data protection will be handled. Use Database for scaled out installations and In Memory for single instance installations.
`SecurityApiSettings: AuthenticationApiKey`	Authentication key so that Authentication can communicate with the Security API. The Security API must have the exact same key saved as well so that they match.
`LoggingSettings: IgnoredLogTypes`	Log types to ignore and not store in the database.
`IdentityServerRaisedEventsSettings`	Determines which events are raised by the identity server.
`LoginSettings:ShowLogoutPrompt`	Enables whether or not a pop-up should appear prompting us to confirm if we want to logout or not.
`LoginSettings:AutomaticRedirectAfterSignOut`	Determines whether we are automatically redirected to the login page after signing out.
`LoginSettings:DetectActiveSessionAtLogin`	Detects if there is any active sessions open to do a single sign in (I think we always want this).
`LoginSettings:IdentityProvidersSettings: Sequel:RequestExpirationEnabled`	
`LoginSettings:IdentityProvidersSettings: Sequel:RequestExpirationTime`	
`LoginSettings:IdentityProvidersSettings: Sequel:InvalidCredentialsErrorMessage`	Error message that is shown when the credentials are incorrect when signing in.
`LoginSettings:IdentityProvidersSettings: Microsoft:CallbackPath`	Path where we are returned to on the Authentication server after signing in using a Microsoft account. It must match the one configured for that client as a Redirect URL in Microsoft's Application Registration Portal where the application is created. For example, if we have set a Redirect URL in the Microsoft application config to 'https://{AuthenticationServer}/Authentication/signing-microsoft' then the value for this parameter must be '/signing-microsoft'.
`LoginSettings:IdentityProvidersSettings:Microsoft:UserPolicies`	Contains the settings for the fields that will be used to match users from the Microsoft Azure Active Directory to its equivalent Sequel user. It also contains the settings for the fields that will be updated when anything is different.
`LoginSettings:IdentityProvidersSettings:Windows:IncludeGroups`	

Authorization

Parameter	Description
`SecurityApiSettings`	
`CacheSettings:SecurityApiAuthorizationCacheEnabled`	Enables the cache to avoid calling the Security API as often (why would we want to turn this off?).
`LoggingSettings:IgnoredLogTypes`	Log types to ignore and not store in the database.
`Consumers`	List of messages with their corresponding message type, consumer type, topic path and queue name, that will be published to the message bus.
`MessageBusSettings:Application`	Name of the Security application the message bus will be running for.
`MessageBusSettings:Instance`	Name of the server where the message bus will be running.
`MessageBusSettings:RabbitMqSettings:PurgeOnStartup`	If enabled deletes all messages in the queues when the application starts up.
`MessageBusSettings:RabbitMqSettings:Timeout`	
`MessageBusSettings:RabbitMqSettings:BindMessageExchanges`	Determines whether we should automatically bind the messages with their corresponding queues.

Security API

Parameter	Description
`PasswordPolicySettings`	Contains the settings for the different requirements the passwords have to comply with like length, ue of upper case, lower case, etc.
`AuthorizationApiKey`	Authorization key so that Authorization can communicate with the Security API.
`LoggingSettings:IgnoredLogTypes`	Log types to ignore and not store in the database.
`SwaggerSettings:AuthenticationFlow`	
`PaginationSettings`	Settings for the pagination of the GetUsers request, we can set the the number of pages returned, the number of users returned in one page and the maximum number of users we are allowed to ask for.
`SwaggerSettings.authenticationFlow`	Defines the flow type used for authentication. Default and recommended value is *authorizationCode*; *implicit* is also accepted and is the previous flow used. Setting this value back to implicit will require to ensure that there are not *AllowedGrantTypes* configured for `sec.api.swagger` client of type *authorizationCode* and there is one of type *implicit*.
`SyncWithAzureUserSettings`	Settings like the Application ID and password to connect the Security API with an Azure Active Directory and be able to sync users from there to our Sequel identity database.
`MessageBusSettings:RabbitMqSettings:PurgeOnStartup`	If enabled deletes all messages in the queues when the application starts up.
`MessageBusSettings:RabbitMqSettings:Timeout`	
`MessageBusSettings:RabbitMqSettings:BindMessageExchange`	Determines whether we should automatically bind the messages with their corresponding queues.
`MessageBusSettings:Topics`	List of messages with their corresponding message type, topic path and queue name, that will be published to the message bus.

Sync Services

Allows us to install services used to synchronize data related to security.

- *Security Sync Service*, synchronize the information stored in the security services with the information stored in Origin or Claims. The version included in the security package is basic version that synchronizes just some information. We highly recommend to install the versions provided by Origin and Claims products as those products have customized versions for this synchronization process. This service consumes information from Security, and never writes.
- *LDAP sync service*. This is only required for synchronizing users and AD groups (memberships) with Sequel Security Services. This service reads information from a LDAP server and pushes changes to the security service API.
- *Azure AD sync service*. This is only required for synchronizing users and AD groups (memberships) with Sequel Security Services. This service reads information from Azure AD and pushes changes to the security service API.

More information at sync services installation guide.

4.4 Database installation

Database installation for Sequel Security components

> **Documentation automatically generated from DatabaseMetadata.xml**

4.4.1 Products

This installation is organized around different product:

- **Sequel Security**. Deploys the security databases. Modules in this product are:
  - Sequel Security - MultiTenancy Database
  - Sequel Security - Security Database
- **Setup Configuration**. Deploys required configurations. Modules in this product are:
  - Setup Configuration - Create Admin User

4.4.2 Global settings

Global settings allow to define values that are reused across the different modules of the installation.

Data Sources

Data sources for the applications. Below table summarizes the available parameters under this category.

Parameter	Description
`MultiTenancyDatabaseServer` MultiTenancy Server Instance	The instance of SQL Server hosting the multitenancy database.
`MultiTenancyDatabaseName`	The name of the MultiTenancy database on the SQL Server. Default value: *Sequel.Security.Multitenancy*.
`DatabaseServer` Security Database Server Instance	The instance of SQL Server hosting the application.
`DatabaseName` Security Database Name	The name of the Security database on the SQL Server. Default value: `Security`.
`DatabaseUser`	The name of the user owner of the Database. Not required. If not provided, the Windows user domain is used.
`DatabasePassword` Database User Password	Not required. The password for the database user.
`DatabaseServerIsAGListener`	Database server is a AlwaysOn Availability Group listener. Default value: `False`.

Security Configuration

Security endpoints and further configuration for databases. Below table summarizes the available parameters under this category.

Parameter	Description
`SecurityAdminUrl`	Administration endpoint. Default value: *https://{var:CurrentMachine}.{var:USERDNSDOMAIN}/Administration*.
`SkipImportSecuritySettings`	Skip the activity for importing security settings. Default value: False. Default value: `false`.
`SecurityAdminUrlExternal` Public Security Admin Url	Public Administration endpoint. Default value: *https://{var:CurrentMachine}.{var:USERDNSDOMAIN}/Administration*.
`SecurityApiUrl`	Security web api endpoint. Default value: *https://{var:CurrentMachine}.{var:USERDNSDOMAIN}/SecurityApi*.
`SecurityApiUrlExternal` Public Security Api Url	Public Security web api endpoint. Default value: *https://{var:CurrentMachine}.{var:USERDNSDOMAIN}/SecurityApi*.
`SecurityAuthorizationUrl`	Authorization endpoint. Default value: *https://{var:CurrentMachine}.{var:USERDNSDOMAIN}/Authorization*.
`SecurityAuthorizationUrlExternal` Public Security Authorization Url	Public Authorization endpoint. Default value: *https://{var:CurrentMachine}.{var:USERDNSDOMAIN}/Authorization*.

4.4.3 Modules

Sequel Security - MultiTenancy Database

Module Id: `MultiTenancyDatabase`

Deploys the multitenant database where all tenants are registered.

Parameter	Description
`MultitenancyConnectionString` Database connection string	Default value: *Data Source={gs:DataSources.MultiTenancyDatabaseServer};Initial Catalog={gs:DataSources.MultiTenancyDatabaseName};Trusted_Connection=True;MultipleActiveResultSets=true.* `Mandatory`
`ConnectionString` Database connection string	Default value: *Data Source={gs:DataSources.DatabaseServer};Initial Catalog={gs:DataSources.DatabaseName};Trusted_Connection=True;MultipleActiveResultSets=true;MultiSubnetFailover={gs:DataS* `Mandatory`
`scriptDatabaseOptions` Allow DB settings deployment	This is recommended to be set as 'True' for the first installation or when a known change in the DB settings is required. Set to 'Fals are: `True`, `False`. Default value: `True`. `Mandatory`

Sequel Security - Security Database

Module Id: `SecurityDatabase`

Deploys the security database for each tenant. For single tenant installation the multiTenancy database could be used also for the security database. Schemas deployed are: authorization and authentication.

Parameter	Description
`ConnectionString` Database connection string	Default value: *Data Source={gs:DataSources.DatabaseServer};Initial Catalog={gs:DataSources.DatabaseName};Integrated Security=True;MultipleActiveResultSets=True*. `Mandatory`
`scriptDatabaseOptions` Allow DB settings deployment	This is recommended to be set as 'True' for the first installation or when a known change in the DB settings is required. Set to 'False' when using a database using AAG. Valid choices are: `True`, `False`. Default value: `True`. `Mandatory`
`ConfigurationClientSecret` Security Api Client Secret	`Mandatory` `Encrypted` `Password`
`ConfigurationClientSecretConfirmation` Security Api Client Secret Confirmation	`Mandatory` `Encrypted` `Password`
`SecurityApiUrl`	Default value: *{gs:SecurityConfiguration.SecurityApiUrl}*. `Mandatory`
`SecurityApiUrlExternal` Public Security Api Url	Default value: *{gs:SecurityConfiguration.SecurityApiUrlExternal}*. `Mandatory`
`SecurityAdminUrl`	Default value: *{gs:SecurityConfiguration.SecurityAdminUrl}*. `Mandatory`
`SecurityAdminUrlExternal` Public Security Admin Url	Default value: *{gs:SecurityConfiguration.SecurityAdminUrlExternal}*. `Mandatory`
`SecurityAuthorizationUrl`	Default value: *{gs:SecurityConfiguration.SecurityAuthorizationUrl}*. `Mandatory`
`SecurityAuthorizationUrlExternal` Public Security Authorization Url	Default value: *{gs:SecurityConfiguration.SecurityAuthorizationUrlExternal}*. `Mandatory`

Setup Configuration - Create Admin User

Module Id: `CreateAdminUser`

Creates an admin user for the security database using the `add-admin-user` command from the `sequel-security` tool.

Parameter	Description
`Admin.Name` Admin User Name	Security administrator account user name. Default value: `admin`. `Mandatory`
`Admin.Email` Admin email	Security administrator account email address. `Mandatory`
`Admin.Password` Admin Password	Security administrator account password. `Mandatory` `Encrypted` `Password`
`Admin.PasswordConfirmation` Admin user password confirmation	Security administrator account password confirmation. `Mandatory` `Encrypted` `Password`
`ConnectionString` Security Database	Security database connection string. Default value: *Data Source={gs:DataSources.DatabaseServer};Initial Catalog={gs:DataSources.DatabaseName};Integrated Security=True;MultipleActiveResultSets=True;MultiSubnetFailover={gs:DataSources.DatabaseServerIsAGListener}*. `Mandatory`

4.4.4 Appendix

Global settings:

For accessing to previously defined global settings use the syntax: `{gs:GLOBAL_SETTING_NAME}`

Variables:

Deployment manager offer access to environment variable from the current process, like `USERDNSDOMAIN`. Also, other built-in variables are available like:

- `CurrentMachine`: returns the machine name where the installation is executed.
- `RootFolder`: returns the root folder where Deployment Manager is installed.

The syntax is `{var:VARIABLE_NAME}`. A sample of variable usage on attribute `defaultValue`:

```
<parameter xsi:type="Url"
    name="Url"
    defaultValue="https://{var:CurrentMachine}.{var:USERDNSDOMAIN}/"
    />
```

4.5 Security Apps Installation

Component installation for Sequel Security components.

> **Documentation automatically generated from ApplicationMetadata.xml**

4.5.1 Products

This installation is organized around different product:

- **Security web services**. Install the services that are part of Sequel Security Services. Modules in this product are:
  - Security web services - Security API
  - Security web services - Authentication service
  - Security web services - Authorization service
  - Security web services - Administration site

4.5.2 Global settings

Global settings allow to define values that are reused across the different modules of the installation.

Environment Settings

Server level settings. Below table summarizes the available parameters under this category.

Parameter	Description
`ServerName`	The name of the web server where the application will be installed. Default value: `{var:CurrentMachine}`.
`ServerNameExternal` Public Server Name	The public name of the web server where the application will be installed (subdomain.domain.tld). Default value: `{var:CurrentMachine}`.
`ServerUrl`	The URI of the web server where the application will be installed (subdomain.domain.tld). Default value: *{var:CurrentMachine}.{var:USERDNSDOMAIN}*.
`ServerUrlExternal` Public Server URL	The public URI of the web server where the application will be installed (subdomain.domain.tld). Default value: *{var:CurrentMachine}.{var:USERDNSDOMAIN}*.
`AuthenticationServer`	URI where the Sequel Authentication Server is (or will be) available (subdomain.domain.tld). Default value: *{var:CurrentMachine}.{var:USERDNSDOMAIN}*.
`AuthenticationServerExternal` Public Authentication Server	Public URI where the Sequel Authentication Server is (or will be) available (subdomain.domain.tld). Default value: *{var:CurrentMachine}.{var:USERDNSDOMAIN}*.
`SecurityApiServer`	URI where the Sequel Security API Server is (or will be) available (subdomain.domain.tld). Default value: *{var:CurrentMachine}.{var:USERDNSDOMAIN}*.
`SecurityApiServerExternal` Public Security API Server	Public URI where the Sequel Security API Server is (or will be) available (subdomain.domain.tld). Default value: *{var:CurrentMachine}.{var:USERDNSDOMAIN}*.
`AuthorizationServer`	URI where the Sequel Authorization Server is (or will be) available (subdomain.domain.tld). Default value: *{var:CurrentMachine}.{var:USERDNSDOMAIN}*.
`AuthorizationServerExternal` Public Authorization Server	Public URI where the Sequel Authorization Server is (or will be) available (subdomain.domain.tld). Default value: *{var:CurrentMachine}.{var:USERDNSDOMAIN}*.
`Domain` Internal Security Domain	Internal domain where Sequel Security will be installed. Default value: `{var:USERDNSDOMAIN}`.
`DomainExternal` External/Public Security Domain	Public domain where Sequel Suite application will be installed. This attribute (aka CookieDomain) specifies which hosts are allowed to receive the cookie. For allowing subdomains, this value must be prefixed with a dot. As a sample, uat.sequel.com will be used for the domain, while .uat.sequel.com will be used for subdomains too. In general, the dot prefixing the domain is preferred as we use subdomains. Default value: `{var:USERDNSDOMAIN}`.

Data Sources

Data sources for the applications. Below table summarizes the available parameters under this category.

Parameter	Description
`MultiTenancyDatabaseServer` MultiTenancy Server Instance	The instance of the SQL Server hosting the multitenancy database.
`MultiTenancyDatabaseName`	The name of the multitenancy database on the SQL Server.
`DatabaseServer` Security Database Server Instance	The instance of the SQL Server hosting the application's database.
`DatabaseName` Security Database Name	The name of the Security database on the SQL Server.
`DatabaseServerIsAGListener` AG enabled	Determines whether the database servers (MultiTenancy and Database) are AlwaysOn Availability Group listeners. Default value: `False`.

IIS Settings

Configuration for Internet Information Services. Below table summarizes the available parameters under this category.

Parameter	Description
`Site`	IIS Site Name. Default value: `Default Web Site`.
`Port`	IIS Host Port. Default value: `443`.
`Protocol`	IIS Binding Protocol (http/https). Default value: .
`CertificateThumbprint` HTTPS Certificate Thumbprint	Thumbprint of the PFX Certificate for IIS HTTPS binding.
`AppPoolUserDomain`	Domain for the app pool user.
`AppPoolUserName`	Name of the app pool user.
`AppPoolUserPassword`	Password of the app pool user. This setting handles a *password*. `Encrypted`
`AppPoolUserPasswordConfirmation`	Password confirmation of the app pool user. `Encrypted`

Logging settings

Logging settings for the applications. Below table summarizes the available parameters under this category.

Parameter	Description
`DatabaseServer` Logging Server Instance	The instance of the SQL Server hosting the logging database.
`DatabaseName` Logging Database Name	The name of the logging database on the SQL Server.
`DatabaseServerIsAGListener` AG enabled	Determines whether the database server is an AlwaysOn Availability Group listener. Default value: `False`.
`Level` Logging Level	Logging level {Debug, Information, Warning, Error}. Default value: `Debug`.
`Type` Logs Output	Select the output of logs: MsSql or Console. Default value: `MsSql`.

RabbitMQ Settings

Configuration for RabbitMQ message bus. Below table summarizes the available parameters under this category.

Parameter	Description
`ServerUrl` RabbitMQ Server URL	The URL of the RabbitMQ server (rabbitmq://subdomain.domain.tld/virtualhost).
`UserName` RabbitMQ User Name	RabbitMQ user name.
`Password` RabbitMQ Password	Password of the RabbitMQ server account. This setting handles a *password*. `Encrypted`
`PasswordConfirmation` RabbitMQ Password Confirmation	Password confirmation of the RabbitMQ server account. `Encrypted`

4.5.3 Modules

Security web services - Security API

Module Id: `SecurityAPI`

REST API for managing all the resources related to security (users, roles, etc.) and to query security data (list of users, list of roles, etc.). It is the unique component in the system with access to security databases.

Parameter	Description
`InitializeSSL`	Enables HTTPS. Valid choices are: `True`, `False`. Default value: `True`. `Mandatory`
`TrustForwardedHeaders`	Trusts Forwarded Headers from Reverse Proxies/Load Balancers to properly detect HTTPS. Valid choices ar
`IIS Web Application Name` IIS Web Application Name	Name of the web application hosted in IIS. Default value: *{gs:IIS.Site}/SecurityAPI*. `Mandatory`
`IIS Binding Protocol` IIS Binding Protocol (http/https)	Protocol of the web application hosted in IIS. Default value: `{gs:IIS.Protocol}`. `Mandatory`
`IIS Port` IIS Binding Port	Port of the web application hosted in IIS. Default value: `{gs:IIS.Port}`. `Mandatory`
`CertificateThumbprint`	Thumbprint of the PFX Certificate for IIS HTTPS binding. Default value: *{gs:IIS.CertificateThumbprint}*.
`ConnectionStrings.MultitenancyDatabase` Multitenancy Database Connection String	Connection string of the multitenancy database. Default value: *Data Source={gs:DataSources.MultiTenancyDa Catalog={gs:DataSources.MultiTenancyDatabaseName};Trusted_Connection=True;MultipleActiveResultSets=* `Mandatory`
`LoggingSettings.MsSql.ConnectionString` Logging Database Connection String	Connection string of the logging database. Default value: *Data Source={gs:Logging.DatabaseServer};Initial Catalog={gs:Logging.DatabaseName};Trusted_Connection=True;MultipleActiveResultSets=true;MultiSubnetFa*
`LoggingSettings.MsSql.MinimumLogLevel` Logging Level	Logging level {Debug, Info, Warning, Error} (MsSql). Valid choices are: `Debug`, `Information`, `Warning`, `Erro`
`LoggingSettings.Console.MinimumLogLevel` Logging Level	Logging level {Debug, Info, Warning, Error} (Console). Valid choices are: `Debug`, `Information`, `Warning`, `Er`
`LoggingSettings.Console.UseJsonFormatter` Use JSON Format (Console)	Output logs in JSON (Console). Valid choices are: `True`, `False`. Default value: `True`. `Mandatory`
`LoggingSettings.Type` Logs Output	Select the output of logs: MsSql or Console. Valid choices are: `MsSql`, `Console`. Default value: `{gs:Loggin`
`SwaggerSettings.Enabled` Swagger Enabled	Enables Swagger UI documentation for Security API. Valid choices are: `True`, `False`. Default value: `False`
`ServiceDiscoverySettings.Mode` Service Discovery Mode	Service Discovery Mode: Consul or PointToPoint. This feature is an experimental feature. `PointToPoint` is t `PointToPoint`. Default value: `PointToPoint`. `Mandatory`
`ServiceDiscoverySettings. RequiredServices. Authentication. InternalUrl` Authentication URL	URL where the Sequel Authentication application is (or will be) available (protocol://subdomain.domain.tld/pa *Authentication*. `Mandatory`
`ServiceDiscoverySettings. RequiredServices. Authentication. ExternalUrl` Public Authentication URL	Public URL where the Sequel Authentication application is (or will be) available (protocol://subdomain.domain *{gs:Environment.AuthenticationServerExternal}/Authentication*. `Mandatory`
`ServiceDiscoverySettings. PublishedServices. SecurityApi. InternalUrl` Security API URL	URL where the Sequel Security API is (or will be) available (protocol://subdomain.domain.tld/path). Default va `Mandatory`
`ServiceDiscoverySettings. PublishedServices. SecurityApi. ExternalUrl` Public Security API URL	URL where the Sequel Security API application is (or will be) available (protocol://subdomain.domain.tld/path *SecurityAPI*. `Mandatory`
`MessageBusSettings.RabbitMqSettings.ServerUri` RabbitMQ Server URL	The URL of the RabbitMQ server (rabbitmq://subdomain.domain.tld/virtualhost). Default value: *{gs:RabbitMq*
`MessageBusSettings.RabbitMqSettings.UserName` RabbitMQ User Name	RabbitMQ user name. Default value: *{gs:RabbitMq.UserName}*. `Mandatory`
`MessageBusSettings.RabbitMqSettings.Password` RabbitMQ User Password	Password of the RabbitMQ server account. Default value: *{gs:RabbitMq.Password}*. `Mandatory` `Encrypted`
`SendEmailSettings.Host` SMTP host name	Host name or IP address of the SMTP host.
`SendEmailSettings.Port` SMTP host port	Port number of the SMTP host. Default value: `25`. `Mandatory`
`SendEmailSettings.UserName` SMTP host username	User name of the SMTP host.
`SendEmailSettings.Password` SMTP host password	Password for the user of the SMTP host. `Encrypted` `Password`

Parameter	Description
`SendEmailSettings.SendForgotPasswordEmail` Enable send forgot password email	Enables the ability to send the forgot password email. Valid choices are: `True`, `False`. Default value: `True`.
`SendEmailSettings.ForgotPasswordFromEmail` Forgot password email address	Email address from which the forgot password email will be sent.
`SendEmailSettings.SecureSocketOptions` Secure SSL Option	Used to configure or disable ssl security, valid options - None, Auto, SslOnConnect, StartTls, StartTlsWhenAv `StartTlsWhenAvailable`. Default value: `Auto`. `Mandatory`
`HealthCheckSettings.ApiKey` Health check settings apikey	Health check settings apikey.

Security web services - Authentication service

Module Id: `AuthenticationWeb`

Provides authentication using the OAuth2 and OpenID Connect protocols for interactive clients and machine to machine clients. Apart of the authentication protocol endpoints, this service offers the web forms for the users to enter their credentials. Depends on Security API.

Parameter	Description
`InitializeSSL`	Enables HTTPS. Valid choices are: `True`, `False`. Default value: `True`. `Mandatory`
`TrustForwardedHeaders`	Trusts Forwarded Headers from Reverse Proxies/Load Balancers to properly detect HTTPS. Valid choices are: `Tr`
`IIS Web Application Name` IIS Web Application Name	Name of the web application hosted in IIS. Default value: *{gs:IIS.Site}/Authentication*. `Mandatory`
`IIS Binding Protocol` IIS Binding Protocol (http/https)	Protocol of the web application hosted in IIS. Default value: `{gs:IIS.Protocol}`. `Mandatory`
`IIS Port` IIS Binding Port	Port of the web application hosted in IIS. Default value: `{gs:IIS.Port}`. `Mandatory`
`CertificateThumbprint`	Thumbprint of the PFX Certificate for IIS HTTPS binding. Default value: *{gs:IIS.CertificateThumbprint}*.
`SingleSignOnSettings.CookieDomain` External/ Public Security Domain	Public domain where Sequel Suite application will be installed. The CookieDomain attribute specifies which hosts a subdomains, this value must be prefixed with a dot. As a sample, uat.sequel.com will be used for the domain, whil general, the dot prefixing the domain is preferred as we use subdomains. Default value: *{gs:Environment.DomainEx*
`SingleSignOnSettings.SsoCookieProtection` Single Sign On cookie protection mode	Mode used to protect the Single Sign On cookie. For backward compatibility use AES. In environments where all a Sequel.Security.Integration use v3.1 o higher select RS256. Valid choices are: `AES`, `RS256`. Default value: `RS256`
`DataProtectionSettings.Mode` Data protection mode	Mode used by Data Protection to store encryption keys. For stand-alone installations use InMemory. For on-prem together use Database. For Amazon AWS environments use AWS or Database as value. Valid choices are: `InMem` `Mandatory`
`DataProtectionSettings.ExpirationInterval` Expiration interval	Expiration interval for encryption keys used by Database and AWS modes. Value must be greater than 7 days. Def
`ServiceDiscoverySettings.Mode` Service Discovery Mode	Service Discovery Mode: Consul or PointToPoint. This feature is an experimental feature. `PointToPoint` is the ur choices are: `Consul`, `PointToPoint`. Default value: `PointToPoint`. `Mandatory`
`ServiceDiscoverySettings. PublishedServices. Authentication. InternalUrl` Authentication URL	URL where the Sequel Authentication application is (or will be) available (protocol://subdomain.domain.tld/path). *{gs:Environment.AuthenticationServer}/Authentication*. `Mandatory`
`ServiceDiscoverySettings. PublishedServices. Authentication. ExternalUrl` Public Authentication URL	Public URL where the Sequel Authentication application is (or will be) available (protocol://subdomain.domain.tld/ *{gs:Environment.AuthenticationServerExternal}/Authentication*. `Mandatory`
`ServiceDiscoverySettings. RequiredServices. SecurityApi. InternalUrl` Security API URL	URL where the Sequel Security API is (or will be) available (protocol://subdomain.domain.tld/path). Default value: *SecurityAPI*. `Mandatory`
`ServiceDiscoverySettings. RequiredServices. SecurityApi. ExternalUrl` Public Security API URL	URL where the Sequel Security API application is (or will be) available (protocol://subdomain.domain.tld/path). De *{gs:Environment.SecurityApiServerExternal}/SecurityAPI*. `Mandatory`
`LoggingSettings.MsSql.ConnectionString` Logging Database Connection String	Connection string of the logging database. Default value: *Data Source={gs:Logging.DatabaseServer};Initial Catalog={gs:Logging.DatabaseName};Trusted_Connection=True;MultipleActiveResultSets=true;MultiSubnetFailover* `Mandatory`
`LoggingSettings.MsSql.MinimumLogLevel` Logging Level	Logging level {Debug, Info, Warning, Error} (MsSql). Valid choices are: `Debug`, `Information`, `Warning`, `Error`. D
`LoggingSettings.Console.MinimumLogLevel` Logging Level	Logging level {Debug, Info, Warning, Error} (Console). Valid choices are: `Debug`, `Information`, `Warning`, `Error`
`LoggingSettings.Console.UseJsonFormatter` Use JSON Format (Console)	Output logs in JSON (Console). Valid choices are: `True`, `False`. Default value: `True`. `Mandatory`
`LoggingSettings.Type` Logs Output	Select the output of logs: MsSql or Console. Valid choices are: `MsSql`, `Console`. Default value: `{gs:Logging.Ty`
`SigningKeyCredentials.KeyFilePath` Signing certificate: file	Name of the file that contains the certificate that will be used to sign tokens. `Mandatory`

Parameter	Description
`SigningKeyCredentials.KeyFilePassword` Signing certificate: password	Password to access the private key of the certificate. `Mandatory` `Encrypted` `Password`
`SigningKeyCredentials.KeyFilePasswordConfirmation` Signing certificate: password confirmation	Password to access the private key of the certificate (confirmation). `Mandatory` `Encrypted` `Password`
`CaptchaSettings.Enabled` Captcha: enabled	Enables the captcha in Authentication when requesting a password reset. Valid choices are: `True`, `False`. Defaul
`CaptchaSettings.DataSiteKey` Captcha: data site key	Captcha data site key.
`CaptchaSettings.SecretKey` Captcha: secret key	Captcha secret key.
`LoginSettings.RememberLoginAllowed` 'Remember Me' enabled	Determines whether or not the 'Remember Me' option is available on the login page. Valid choices are: `True`, `Fal`
`LoginSettings.RememberMeLoginDuration` 'Remember Me' duration	Amount of time the users credentials will be saved in the browser, even if the browser is closed the user will still be credentials expiration'. Default value: `12:00:00`. `Mandatory`
`LoginSettings.IdentityProvidersSettings.Windows.Enabled` Windows Authentication	Enables the possiblity to use Windows Authentication to login. Valid choices are: `True`, `False`. Default value: `Tr`
`LoginSettings.IdentityProvidersSettings.Sequel.Enabled` Sequel identity enabled	Enables the possiblity to use our Sequel user account to login. Valid choices are: `True`, `False`. Default value: `Tr`
`LoginSettings.IdentityProvidersSettings.Microsoft.Enabled` Microsoft Azure Active Directory enabled	Enables the possiblity to use a Microsoft Azure Active Directory to login (previously called Microsoft Account). Va `Mandatory`
`LoginSettings.IdentityProvidersSettings.Microsoft.TenantId` Microsoft Azure Active Directory Tenant Id	The ID of the Azure Active Directory in which the application was created.
`LoginSettings.IdentityProvidersSettings.Microsoft.ClientId` Microsoft Azure Active Directory Client Id	The ID of the application created in the Azure Active Directory (also known as Application ID).
`LoginSettings.IdentityProvidersSettings.Microsoft.ClientSecret` Microsoft Azure Active Directory Client Secret	Authentication key string of the application created in the Azure Active Directory. `Encrypted` `Password`
`LoginSettings.IdentityProvidersSettings.Microsoft.ClientSecretConfirmation` Microsoft Azure Active Directory Client Secret Confirmation	Authentication key string of the application created in the Azure Active Directory (confirmation). `Encrypted` `Pass`
`LoginSettings.IdentityProvidersSettings.ClaimSearch.Enabled` ISO ClaimSearch Authentication enabled	Enables ISO ClaimSearch integrated authentication. Valid choices are: `True`, `False`. Default value: `False`. `Mand`
`LoginSettings.IdentityProvidersSettings.ClaimSearch.SessionValidationEndpoint` ISO ClaimSearch session validation endpoint	Endpoint used to validate ISO ClaimSearch's Session ID. `Mandatory`

Parameter	Description
`LoginSettings. IdentityProvidersSettings. ClaimSearch. LoginUrl` ISO ClaimSearch login URL	URL to login in ISO ClaimSearch. `Mandatory`
`LoginSettings. IdentityProvidersSettings. Okta. Enabled` Okta Authentication	Enables Okta authentication. Valid choices are: `True`, `False`. Default value: `False`. `Mandatory`
`LoginSettings. IdentityProvidersSettings. Okta. Domain` Okta Domain	Organization's Okta domain (e.g. mycompany.okta.com).
`LoginSettings. IdentityProvidersSettings. Okta. AuthorizationServerId` Okta Authorization Server ID	ID of authentication server in Okta domain. Default value: `default`
`LoginSettings. IdentityProvidersSettings. Okta. ClientId` Okta Application Client ID	Client ID of the application in Okta domain.
`LoginSettings. IdentityProvidersSettings. Okta. ClientSecret` Okta Application Client Secret	Client secret of the application in Okta domain. `Encrypted Password`
`LoginSettings. IdentityProvidersSettings. Okta. ClientSecretConfirmation` Okta Application Client Secret Confirmation	The Client secret of the application created in Okta domain (confirmation). `Encrypted Password`
`LoginSettings. IdentityProvidersSettings. JumpCloud. Enabled` JumpCloud Authentication enabled	Enables JumpCloud integrated authentication. Valid choices are: `True`, `False`. Default value: `False`. `Mandatory`
`LoginSettings. IdentityProvidersSettings. JumpCloud. SPEntityId` SP Entity ID	Service Provider identifier.
`LoginSettings. IdentityProvidersSettings. JumpCloud. X509SigningCertificate` X509 Signing Certificate	Signing certificated included in SAML SP Metadata file.
`LoginSettings. IdentityProvidersSettings. JumpCloud. LoginUrl` JumpCloud login URL	SingleSignOnService location included in SAML SP Metadata file.
`HealthCheckSettings.ApiKey` Health check settings apikey	Health check settings apikey.
`IdentityServerOptions. Authentication. CookieLifetime` User credentials expiration	Amount of time the users credentials will be valid, after this time the user must reinsert their credentials. Default va
`IdentityServerOptions.IssuerUri` Issuer Uri	Set the issuer name that will appear in the discovery document and the issued JWT tokens. When leaving it empty, `Authentication` endpoints. Requests might fail when using the old Security Integration nuget (`Sequel.Securit` external and internal URLs are not the same. Default value: *security{gs:Environment.DomainExternal}*.

Security web services - Authorization service

Module Id: `AuthorizationWeb`

This REST API offers a fast access to all queries related to authorization; caching the data retreived from the Security API.

Parameter	Description
`InitializeSSL`	Enables HTTPS. Valid choices are: `True`, `False`. Default value: `True`. `Mandatory`
`TrustForwardedHeaders`	Trusts Forwarded Headers from Reverse Proxies/Load Balancers to properly detect HTTPS. Valid choices ar
`IIS Web Application Name` IIS Web Application Name	Name of the web application hosted in IIS. Default value: *{gs:IIS.Site}/Authorization*. `Mandatory`
`IIS Binding Protocol` IIS Binding Protocol (http/https)	Protocol of the web application hosted in IIS. Default value: `{gs:IIS.Protocol}`. `Mandatory`
`IIS Port` IIS Binding Port	Port of the web application hosted in IIS. Default value: `{gs:IIS.Port}`. `Mandatory`
`CertificateThumbprint`	Thumbprint of the PFX Certificate for IIS HTTPS binding. Default value: *{gs:IIS.CertificateThumbprint}*.
`SwaggerSettings.Enabled` Swagger Enabled	Enables Swagger UI documentation for Authorization. Valid choices are: `True`, `False`. Default value: `False`
`SecurityApiSettings.BaseUrl` Security API Endpoint	URL of the Security API endpoint (protocol://subdomain.domain.tld/path). Default value: *{gs:IIS.Protocol}://{* `Mandatory`
`ServiceDiscoverySettings.Mode` Service Discovery Mode	Service Discovery Mode: Consul or PointToPoint. This feature is an experimental feature. `PointToPoint` is t choices are: `Consul`, `PointToPoint`. Default value: `PointToPoint`. `Mandatory`
`ServiceDiscoverySettings. RequiredServices. Authentication. InternalUrl` Authentication URL	URL where the Sequel Authentication application is (or will be) available (protocol://subdomain.domain.tld/pa *{gs:Environment.AuthenticationServer}/Authentication*. `Mandatory`
`ServiceDiscoverySettings. RequiredServices. Authentication. ExternalUrl` Public Authentication URL	Public URL where the Sequel Authentication application is (or will be) available (protocol://subdomain.domain *{gs:Environment.AuthenticationServerExternal}/Authentication*. `Mandatory`
`ServiceDiscoverySettings. RequiredServices. SecurityApi. InternalUrl` Security API URL	URL where the Sequel Security API is (or will be) available (protocol://subdomain.domain.tld/path). Default v *SecurityAPI*. `Mandatory`
`ServiceDiscoverySettings. RequiredServices. SecurityApi. ExternalUrl` Public Security API URL	URL where the Sequel Security API application is (or will be) available (protocol://subdomain.domain.tld/path *{gs:Environment.SecurityApiServerExternal}/SecurityAPI*. `Mandatory`
`ServiceDiscoverySettings. PublishedServices. Authorization. InternalUrl` Authorization URL	URL where the Sequel Authorization application is (or will be) available (protocol://subdomain.domain.tld/pat *{gs:Environment.AuthorizationServer}/Authorization*. `Mandatory`
`ServiceDiscoverySettings. PublishedServices. Authorization. ExternalUrl` Public Authorization URL	Public URL where the Sequel Authorization application is (or will be) available (protocol://subdomain.domain. *{gs:Environment.AuthorizationServerExternal}/Authorization*. `Mandatory`
`MessageBusSettings.RabbitMqSettings.ServerUri` RabbitMQ Server URL	The URL of the RabbitMQ server (rabbitmq://subdomain.domain.tld/virtualhost). Default value: *{gs:RabbitMq*
`MessageBusSettings.RabbitMqSettings.UserName` RabbitMQ User Name	RabbitMQ user name. Default value: *{gs:RabbitMq.UserName}*. `Mandatory`
`MessageBusSettings.RabbitMqSettings.Password` RabbitMQ Password	Password of the RabbitMQ server account. Default value: *{gs:RabbitMq.Password}*. `Mandatory` `Encrypted`
`LoggingSettings.MsSql.ConnectionString` Logging Database Connection String	Connection string of the logging database. Default value: *Data Source={gs:Logging.DatabaseServer};Initial Catalog={gs:Logging.DatabaseName};Trusted_Connection=True;MultipleActiveResultSets=true;MultiSubnetFa* `Mandatory`
`LoggingSettings.MsSql.MinimumLogLevel` Logging Level	Logging level {Debug, Info, Warning, Error} (MsSql). Valid choices are: `Debug`, `Information`, `Warning`, `Erro`
`LoggingSettings.Console.MinimumLogLevel` Logging Level	Logging level {Debug, Info, Warning, Error} (Console). Valid choices are: `Debug`, `Information`, `Warning`, `Er`
`LoggingSettings.Console.UseJsonFormatter` Use JSON Format (Console)	Output logs in JSON (Console). Valid choices are: `True`, `False`. Default value: `True`. `Mandatory`
`LoggingSettings.Type` Logs Output	Select the output of logs: MsSql or Console. Valid choices are: `MsSql`, `Console`. Default value: `{gs:Loggin`

Parameter	Description
`HealthCheckSettings.ApiKey` Health check settings apikey	Health check settings apikey.

Security web services - Administration site

Module Id: `SecurityAdministrationWeb`

Static website application (SPA) for managing security data, depends on Security Rest API and Authentication Service.

Parameter	Description
`IIS Web Application Name` IIS Web Application Name	Name of the web application hosted in IIS. Default value: *{gs:IIS.Site}/Administration*. `Mandatory`
`IIS Binding Protocol` IIS Binding Protocol (http/https)	Protocol of the web application hosted in IIS. Default value: `{gs:IIS.Protocol}`. `Mandatory`
`IIS Port` IIS Binding Port	Port of the web application hosted in IIS. Default value: `{gs:IIS.Port}`. `Mandatory`
`CertificateThumbprint`	Thumbprint of the PFX Certificate for IIS HTTPS binding. Default value: *{gs:IIS.CertificateThumbprint}*.
`auth` Authentication URL	URL where the Sequel Authentication application is (or will be) available (protocol://subdomain.domain.tld/path). Default value: *{p:IIS Binding Protocol}://{gs:Environment.AuthenticationServerExternal}/Authentication*. `Mandatory`
`api` Security API URL	URL where the Sequel Security API application is (or will be) available (protocol://subdomain.domain.tld/path). Default value: *{p:IIS Binding Protocol}://{gs:Environment.SecurityApiServerExternal}/SecurityApi*. `Mandatory`

4.5.4 Appendix

Global settings:

For accessing to previously defined global settings use the syntax: `{gs:GLOBAL_SETTING_NAME}`

Variables:

Deployment manager offer access to environment variable from the current process, like `USERDNSDOMAIN`. Also, other built-in variables are available like:

- `CurrentMachine`: returns the machine name where the installation is executed.
- `RootFolder`: returns the root folder where Deployment Manager is installed.

The syntax is `{var:VARIABLE_NAME}`. A sample of variable usage on attribute `defaultValue`:

```
<parameter xsi:type="Url"
    name="Url"
    defaultValue="https://{var:CurrentMachine}.{var:USERDNSDOMAIN}/"
    />
```

4.6 Synchronization Services Installation

Installation of synchronization services (security sync and LDAP sync).

> **Documentation automatically generated from SyncMetadata.xml**

4.6.1 Products

This installation is organized around different product:

- **Security Database Sync Service**. Install synchronization windows service that consumes changes on security and apply them to legacy read-only schema on applications like Claims and Origin. Modules in this product are:
  - Security Database Sync Service - Security Sync Service
- **Security Ldap Sync Service**. Install synchronization windows service that polls changes on a Windows AD using LDAP and apply them into Sequel Security Services using the SecurityAPI. Modules in this product are:
  - Security Ldap Sync Service - LDAP Sync Service
- **Security Azure AD Sync Service**. Install synchronization windows service that polls changes on Azure AD apply them into Sequel Security Services using the SecurityAPI. Modules in this product are:
  - Security Azure AD Sync Service - Azure AD Sync Service

4.6.2 Global settings

Global settings allow to define values that are reused across the different modules of the installation.

Environment Settings

Server level settings. Below table summarizes the available parameters under this category.

Parameter	Description
`AuthenticationServer`	URI where the Sequel Authentication Server is (or will be) available (subdomain.domain.tld). Default value: *{var:CurrentMachine}.{var:USERDNSDOMAIN}*.
`AuthenticationServerExternal` Public Authentication Server	Public URI where the Sequel Authentication Server is (or will be) available (subdomain.domain.tld). Default value: *{var:CurrentMachine}.{var:USERDNSDOMAIN}*.
`SecurityApiServer`	URI where the Sequel Security API Server is (or will be) available (subdomain.domain.tld). Default value: *{var:CurrentMachine}.{var:USERDNSDOMAIN}*.
`SecurityApiServerExternal` Public Security API Server	Public URI where the Sequel Security API Server is (or will be) available (subdomain.domain.tld). Default value: *{var:CurrentMachine}.{var:USERDNSDOMAIN}*.

IIS Settings

Configuration for Internet Information Services. Below table summarizes the available parameters under this category.

Parameter	Description
`AppPoolUserDomain`	Domain for the app pool user.
`AppPoolUserName`	Name of the app pool user.
`AppPoolUserPassword`	Password of the app pool user. This setting handles a *password*. `Encrypted`
`AppPoolUserPasswordConfirmation`	Password confirmation of the app pool user. `Encrypted`
`Protocol`	IIS Binding Protocol (http/https). Default value: .

Logging settings

Logging settings for the applications. Below table summarizes the available parameters under this category.

Parameter	Description
`DatabaseServer` Logging Server Instance	The instance of the SQL Server hosting the logging database.
`DatabaseName` Logging Database Name	The name of the logging database on the SQL Server.
`DatabaseServerIsAGListener` AG enabled	Determines whether the database server is an AlwaysOn Availability Group listener. Default value: `False`.
`Level` Logging Level	Logging level {Debug, Information, Warning, Error}. Default value: `Debug`.
`Type` Logs Output	Select the output of logs: MsSql or Console. Default value: `MsSql`.

RabbitMQ Settings

Configuration for RabbitMQ message bus. Below table summarizes the available parameters under this category.

Parameter	Description
`ServerUrl` RabbitMQ Server URL	The URL of the RabbitMQ server (rabbitmq://subdomain.domain.tld/virtualhost).
`UserName` RabbitMQ User Name	RabbitMQ user name.
`Password` RabbitMQ Password	Password of the RabbitMQ server account. This setting handles a *password*. `Encrypted`
`PasswordConfirmation` RabbitMQ Password Confirmation	Password confirmation of the RabbitMQ server account. `Encrypted`

4.6.3 Modules

Security Database Sync Service - Security Sync Service

Module Id: `SecuritySyncService`

Security Sync Service; depends on connectivity to a Claims/Origin database, a logging database and message bus where Security services are publishing the changes.

Parameter	Description
`ServicePath` Service destination folder	Security sync service destination folder in current machine. Default value: *C:\Security\sequel-security-sync*.
`ServiceUser` Service User account	Security sync service user account. Default value: *{gs:IIS.AppPoolUserDomain}{gs:IIS.AppPoolUserName}*. M
`ServicePassword` Service User password	Security sync service user password. Default value: *{gs:IIS.AppPoolUserPassword}*. `Encrypted` `Password`
`ConnectionStrings.LegacySecurityDatabase` Legacy Security connection string	Connection string to access Workflow and Product Builder database which contains security info. E. g. : Data Catalog=DatabaseName;Integrated Security=True;MultipleActiveResultSets=True. `Mandatory`
`LoggingSettings. MsSqlLoggingSettings. ConnectionString` Logging Database Connection String	Connection string to the logging database shared by all services in the same environment. Default value: *Data Catalog={gs:Logging.DatabaseName};Trusted_Connection=True;MultipleActiveResultSets=true;MultiSubnetFa* `Mandatory`
`LoggingSettings. MsSqlLoggingSettings. MinimumLogLevel` Logging sql level	Recommended value for production is `Information`. Valid choices are: `Debug`, `Information`, `Warning`,
`LoggingSettings. RollingFileLoggingSettings. PathFormat` Logging file path	Logs during start-up are logged to file, instead of using the logging in database. Use double back slash e. g: C:\ *C:\Security\sequel-security-sync\log-{Date}.txt*. `Mandatory`
`LoggingSettings. RollingFileLoggingSettings. MinimumLogLevel` Logging file level	Recommended value for production is `Information`. Valid choices are: `Debug`, `Information`, `Warning`,
`LoggingSettings. ConsoleLoggingSettings. MinimumLogLevel` Logging Level	Logging level {Debug, Info, Warning, Error} (Console). Valid choices are: `Debug`, `Information`, `Warning`, `Er`
`LoggingSettings. ConsoleLoggingSettings. UseJsonFormatter` Use JSON Format (Console)	Output logs in JSON (Console). Valid choices are: `True`, `False`. Default value: `True`. `Mandatory`
`LoggingSettings.Type` Logs Output	Select the output of logs: MsSql or Console. Valid choices are: `MsSql`, `Console`. Default value: `{gs:Loggin`
`MessageBusSettings.RabbitMqSettings.ServerUri` RabbitMQ server Url	The URL of the RabbitMQ server. Default value: *{gs:RabbitMq.ServerUrl}*. `Mandatory`
`MessageBusSettings.RabbitMqSettings.UserName` RabbitMQ user name	RabbitMQ User name. Default value: *{gs:RabbitMq.UserName}*. `Mandatory`
`MessageBusSettings.RabbitMqSettings.Password` RabbitMQ user password	RabbitMQ User password. Default value: *{gs:RabbitMq.Password}*. `Mandatory` `Encrypted` `Password`
`SynchronizationPolicies.DeletePolicy` Deletion policy	Synchronized policies deletetion policies. Valid choices are: `Physical`, `Logical`, `PhysicalThenLogical`.

Security Ldap Sync Service - LDAP Sync Service

Module Id: `SecurityLdapSyncService`

LDAP Sync Service; depends on connectivity to a Windows AD, Security API, a logging database and message bus used by Security services.

Parameter	Description
`LdapConnection.Host` LDAP connection host	URL of the LDAP server where the user's data is hosted (subdomain.domain.tld). `Mandatory`
`LdapConnection.Port` LDAP connection port	Port of the LDAP host to connect through. Default value: `636`. `Mandatory`
`LdapConnection.SecureConnection` Enable secure LDAP connection	Determines whether a secure connection will be used to communicate with the LDAP host. Valid choices are:
`LdapConnection.DN` LDAP connection username	Username used to establish a connection with the LDAP host. `Mandatory`
`LdapConnection.Password` LDAP connection password	Password used to establish a connection with the LDAP host. `Mandatory` `Encrypted` `Password`
`AuthenticationSettings.ClientId` LDAP authentication client ID	Client ID for authentication when communicating with the Security API. Default value: `sec.app.ldapsync`. M
`AuthenticationSettings.ClientSecret` LDAP authentication client secret	Client secret for authentication when communicating with the Security API. Default value: *EAAAAIzZCcYg3W* `Encrypted` `Password`
`ServiceDiscoverySettings.Mode` Service Discovery Mode	Service Discovery Mode: Consul or PointToPoint. This feature is an experimental feature. `PointToPoint` is t choices are: `Consul`, `PointToPoint`. Default value: `PointToPoint`. `Mandatory`
`ServiceDiscoverySettings. RequiredServices. Authentication. InternalUrl` Authentication URL	URL where the Sequel Authentication application is (or will be) available (protocol://subdomain.domain.tld/pa *{gs:Environment.AuthenticationServer}/Authentication*. `Mandatory`
`ServiceDiscoverySettings. RequiredServices. Authentication. ExternalUrl` Public Authentication URL	Public URL where the Sequel Authentication application is (or will be) available (protocol://subdomain.domain *{gs:Environment.AuthenticationServerExternal}/Authentication*. `Mandatory`
`ServiceDiscoverySettings. RequiredServices. SecurityApi. InternalUrl` Security API URL	URL where the Sequel Security API is (or will be) available (protocol://subdomain.domain.tld/path). Default va *SecurityAPI*. `Mandatory`
`ServiceDiscoverySettings. RequiredServices. SecurityApi. ExternalUrl` Public Security API URL	URL where the Sequel Security API application is (or will be) available (protocol://subdomain.domain.tld/path *{gs:Environment.SecurityApiServerExternal}/SecurityAPI*. `Mandatory`
`ServicePath` Service destination folder	Security LDAP Sync service destination folder on the current machine. Default value: *C:\Security\sequel-secu*
`ServiceUser` Service user account	User account to run the Security LDAP Sync service. Default value: *{gs:IIS.AppPoolUserDomain}{gs:IIS.AppPo*
`ServicePassword` Service user password	Password of the account to run the Security LDAP Sync service. Default value: *{gs:IIS.AppPoolUserPassword*
`LoggingSettings. MsSqlLoggingSettings. ConnectionString` Logging Database Connection String	Connection string of the SQL Server logging database. Default value: *Data Source={gs:Logging.DatabaseServe Catalog={gs:Logging.DatabaseName};Trusted_Connection=True;MultipleActiveResultSets=true;MultiSubnetFa* `Mandatory`
`LoggingSettings. MsSqlLoggingSettings. MinimumLogLevel` Logging SQL level	Logging level {Debug, Info, Warning, Error}. Valid choices are: `Debug`, `Information`, `Warning`, `Error`. Defa
`LoggingSettings. RollingFileLoggingSettings. PathFormat` Logging file path	File path where the LDAP Sync service will log informationa and errors. Default value: *C:\Security\sequel-secu*
`LoggingSettings. RollingFileLoggingSettings. MinimumLogLevel` Logging file level	Logging level {Debug, Info, Warning, Error}. Valid choices are: `Debug`, `Information`, `Warning`, `Error`. Defa
`LoggingSettings. ConsoleLoggingSettings. MinimumLogLevel` Logging Level	Logging level {Debug, Info, Warning, Error} (Console). Valid choices are: `Debug`, `Information`, `Warning`, `Er`
`LoggingSettings. ConsoleLoggingSettings. UseJsonFormatter` Use JSON Format (Console)	Output logs in JSON (Console). Valid choices are: `True`, `False`. Default value: `True`. `Mandatory`
`LoggingSettings.Type` Logs Output	Select the output of logs: MsSql or Console. Valid choices are: `MsSql`, `Console`. Default value: `{gs:Loggin`
`MessageBusSettings.RabbitMqSettings.ServerUri` RabbitMQ server URL	The URL of the RabbitMQ server. Default value: *{gs:RabbitMq.ServerUrl}*. `Mandatory`
`MessageBusSettings.RabbitMqSettings.UserName` RabbitMQ user name	RabbitMQ User name. Default value: *{gs:RabbitMq.UserName}*. `Mandatory`

Parameter	Description
`MessageBusSettings.RabbitMqSettings.Password` RabbitMQ user password	RabbitMQ User password. Default value: *{gs:RabbitMq.Password}*. `Mandatory` `Encrypted` `Password`

Security Azure AD Sync Service - Azure AD Sync Service

Module Id: `SecuritySyncService`

Azure AD Sync Service; depends on connectivity to a Azure AD, Security API, a logging database and message bus used by Security services.

Parameter	Description
`SyncProcessScheduler.StartingMode` Process Scheduler Starting Mode	Scheduler Starting Mode: AtFirstMessage or AtServiceInit. Sets when sync process scheduler will start: when started. Valid choices are: `AtFirstMessage`, `AtServiceInit`. Default value: `AtServiceInit`. `Mandatory`
`SyncProcessScheduler.CheckingInterval` Sync process checking interval	Interval for determine if a new synchronization process must be executed. This value will determine the delay process. Value must be greater than 5 seconds. Default value: `00:00:05`. `Mandatory`
`SyncProcessScheduler.MaxProcessExecution` Sync process max execution time	Maximum execution time for each sync process before be cancelled. Value must be greater than 60 seconds
`SyncDataSources.AzureAD.TenantId` Microsoft Azure Active Directory Tenant Id	The ID of the Azure Active Directory in which the application was created. `Mandatory`
`SyncDataSources.AzureAD.ClientId` Microsoft Azure Active Directory Client Id	The ID of the application created in the Azure Active Directory (also known as Application ID). `Mandatory`
`SyncDataSources.AzureAD.ClientSecret` Microsoft Azure Active Directory Client Secret	Authentication key string of the application created in the Azure Active Directory. `Mandatory` `Encrypted` `Pa`
`AuthenticationSettings.ClientId` Web Sync authentication client ID	Client ID for authentication when communicating with the Security API. Default value: `sec.app.websync`. `Ma`
`AuthenticationSettings.ClientSecret` Web Sync authentication client secret	Client secret for authentication when communicating with the Security API. Default value: *EAAAAIzZCcYg3W(* `Encrypted` `Password`
`ServiceDiscoverySettings.Mode` Service Discovery Mode	Service Discovery Mode: Consul or PointToPoint. This feature is an experimental feature. `PointToPoint` is t choices are: `Consul`, `PointToPoint`. Default value: `PointToPoint`. `Mandatory`
`ServiceDiscoverySettings. RequiredServices. Authentication. InternalUrl` Authentication URL	URL where the Sequel Authentication application is (or will be) available (protocol://subdomain.domain.tld/pa *{gs:Environment.AuthenticationServer}/Authentication*. `Mandatory`
`ServiceDiscoverySettings. RequiredServices. Authentication. ExternalUrl` Public Authentication URL	Public URL where the Sequel Authentication application is (or will be) available (protocol://subdomain.domain *{gs:Environment.AuthenticationServerExternal}/Authentication*. `Mandatory`
`ServiceDiscoverySettings. RequiredServices. SecurityApi. InternalUrl` Security API URL	URL where the Sequel Security API is (or will be) available (protocol://subdomain.domain.tld/path). Default va *SecurityAPI*. `Mandatory`
`ServiceDiscoverySettings. RequiredServices. SecurityApi. ExternalUrl` Public Security API URL	URL where the Sequel Security API application is (or will be) available (protocol://subdomain.domain.tld/path *{gs:Environment.SecurityApiServerExternal}/SecurityAPI*. `Mandatory`
`ServicePath` Service destination folder	Security Sync service destination folder on the current machine. Default value: *C:\Security\Sequel.Security.Sy*
`ServiceUser` Service user account	User account to run the Security Sync service. Default value: *{gs:IIS.AppPoolUserDomain}{gs:IIS.AppPoolUse*
`ServicePassword` Service user password	Password of the account to run the Security Sync service. Default value: *{gs:IIS.AppPoolUserPassword}*. `Enc`
`LoggingSettings. MsSqlLoggingSettings. ConnectionString` Logging Database Connection String	Connection string of the SQL Server logging database. Default value: *Data Source={gs:Logging.DatabaseServe Catalog={gs:Logging.DatabaseName};Trusted_Connection=True;MultipleActiveResultSets=true;MultiSubnetFa* `Mandatory`
`LoggingSettings. MsSqlLoggingSettings. MinimumLogLevel` Logging SQL level	Logging level {Debug, Info, Warning, Error}. Valid choices are: `Debug`, `Information`, `Warning`, `Error`. Defa
`LoggingSettings. RollingFileLoggingSettings. PathFormat` Logging file path	File path where the Sync service will log informationa and errors. Default value: *C:\Security\Sequel.Security.Sy*
`LoggingSettings. RollingFileLoggingSettings. MinimumLogLevel` Logging file level	Logging level {Debug, Info, Warning, Error}. Valid choices are: `Debug`, `Information`, `Warning`, `Error`. Defa
`LoggingSettings. ConsoleLoggingSettings. MinimumLogLevel` Logging Level	Logging level {Debug, Info, Warning, Error} (Console). Valid choices are: `Debug`, `Information`, `Warning`, `Er`
`LoggingSettings. ConsoleLoggingSettings. UseJsonFormatter` Use JSON Format (Console)	Output logs in JSON (Console). Valid choices are: `True`, `False`. Default value: `True`. `Mandatory`
`LoggingSettings.Type` Logs Output	Select the output of logs: MsSql or Console. Valid choices are: `MsSql`, `Console`. Default value: `{gs:Loggin`

Parameter	Description
`MessageBusSettings.RabbitMqSettings.ServerUri` RabbitMQ server URL	The URL of the RabbitMQ server. Default value: *{gs:RabbitMq.ServerUrl}*. `Mandatory`
`MessageBusSettings.RabbitMqSettings.UserName` RabbitMQ user name	RabbitMQ User name. Default value: *{gs:RabbitMq.UserName}*. `Mandatory`
`MessageBusSettings.RabbitMqSettings.Password` RabbitMQ user password	RabbitMQ User password. Default value: *{gs:RabbitMq.Password}*. `Mandatory` `Encrypted` `Password`

4.6.4 Appendix

Global settings:

For accessing to previously defined global settings use the syntax: `{gs:GLOBAL_SETTING_NAME}`

Variables:

Deployment manager offer access to environment variable from the current process, like `USERDNSDOMAIN`. Also, other built-in variables are available like:

- `CurrentMachine`: returns the machine name where the installation is executed.
- `RootFolder`: returns the root folder where Deployment Manager is installed.

The syntax is `{var:VARIABLE_NAME}`. A sample of variable usage on attribute `defaultValue`:

```
<parameter xsi:type="Url"
    name="Url"
    defaultValue="https://{var:CurrentMachine}.{var:USERDNSDOMAIN}/"
    />
```

4.7 Housekeeping & maintenance

4.7.1 Introduction

This document is written with the intent of providing guidelines for the **housekeeping tasks** (periodic maintenance) that should be performed to keep the Sequel Security services performing optimally.

4.7.2 General housekeeping

Like any sizeable software solution, Sequel Security benefits from periodic maintenance, both of application data and of the platform on which the solution components are installed. This includes maintenance of the application servers and SQL servers.

In addition to maintenance of application data, it is important to monitor the overall health of the platform. It is also important to monitor overall capacity to ensure service levels are met and service continues uninterrupted.

There are a number of best practice procedures that benefit any environment.

Security Audit logs

Security audits all update and delete actions made during the day-to-day use of the application. Security does not consider this information as business data and is not used in the downstream; so is possible to remove it without affecting the system. The audit information is stored at `[AuditLog]` table in each schema `Authentication` and `Authorization`.

The `[AuditLog]` tables grow constantly. These tables tend to be quite large holding hundreds of millions of records.

The client can query the AuditLog tables from the database to find out updates and deletions to records.

We suggest clients to archive the Audit Log records into a different database after a while, removing data from the Security AuditLog tables into an archive database. That will release space on the database while logs would still be saved in another archived AuditLog database, in case needed in the future.

It is down to each client to decide how much data to keep in the AuditLog table and how much to move into another database. For example, Sequel recommends clients to keep the last 6 month of AuditLogs database and move everything that is older than 6 months into the archive database.

Sequel Security Services does not provide any scripts or tool to move data from the Security database into an archive AuditLog database.

Security clients table

Clients defined at `authentication` schema stores URLs, Origin and secrets in different tables. It is possible to end up with duplicated entries on those tables that can be safety removed using below stored procedures:

- `[authentication].[cleanDuplicatedClientCorsOrigins]`
- `[authentication].[cleanDuplicatedClientPostLogoutRedirectUris]`
- `[authentication].[cleanDuplicatedClientRedirectUris]`
- `[authentication].[cleanDuplicatedClientSecrets]`

All these stored procedures also acts as ID scripts (shows what is going to do, but does nothing), and by default this is the behaviour of them. To actually remove entries, an argument with value 1 must be set when calling the SP.

Data Protection table

Using Database as Data Protection mode, all keys will be store in authentication.DataProtection table. When a key expires a new one row is created and from that moment all new cookies generated by Authentication server are encrypted only with the new key. Due to the minimum expiration time allowed for this keys (7 days) a maximum of 52 keys per year could be created. Older cookies can be decrypted using expired values but if those old keys are removed that will force users to enter the credentials again when they will be redirected to Authentication server during a login flow.

Security Logs

In this section, we cover the logs generated by the application due to different events or action, the information stored is not intended to be for auditing (as this is done by AuditLog), the aim is trace how the application behaves and help monitoring and diagnostics tasks if required. It contains. The main log system used by Security is the logging database or *Sequel.Core.Logging*; although, there are some scenarios where logs are persisted to file during the start-up of the applications; this is covered in the troubleshooting section as those files do not required housekeeping actions.

LOGGING DATABASE

Most of the Sequel's web based application implement a logging mechanism to log actions and exceptions happened in the application (aka *Sequel.Core.Logging*). A separated database is recommended to store all the data generated by the logging.

The **logging verbosity** is defined in levels:

- *Fatal*: Fatal represents truly catastrophic situations, as far as your application is concerned.
- *Error*: An error is a serious issue and represents the failure of something important going on in your application. Unlike *fatal*, the application itself isn't going down necessarily.
- *Warning*: Indicates that the application might has a problem and that an unusual situation has been detected.
- *Information*: Information messages correspond to normal application behaviour and milestones. Like, a service started or stopped.
- *Debug*: Include more granular, diagnostic information. This is "noisy" territory and furnishing more information than we would not want in normal production situations. This should just be used for diagnostic purposes and moved back to a lower log level.

Information level is preferred on a production environment. When selecting a log level this will include all lower levels (i.e. *Information* will log as well *Warning*, *Error* and *Fatal* logs).

During the installation process the Logging database connection string is required in order to setup the application. Logging data is stored in a table called `Logs` in the `dbo` schema. Once installed you can see the logging database configuration by looking at the `appsettings.json` on each Security service; check the parameter named `LoggingConnectionString`. The verbosity level is defined at `MinimumLogLevel`, where the high verbosity level include the lower levels.

```
"LoggingSettings": {
  "ConnectionString": "Server=...",
  "MinimumLogLevel": "Information",
},
```

Growth Monitoring:

Growth of this database will depend on the minimum logging level selected (see previous section), the lower the level selected, the higher the growth rate of that database. The higher levels (*Information*, *Warning*, *Error* and *Fatal*) are the most important and the less recurrent, so if the logging database grows very fast these should be saved before deleting.

It is down to each client to decide how much data to keep in the Logging database and how much to move into another database. For example, Sequel recommends clients to keep the last 6 month of the logging database and remove or move into an archive database everything that is older than 6 months. We do not recommend to keep *debug* entries if this is not in the context of an issue investigation under any circumstance.

In the next lines is described the mechanism to automatically delete all entries older than a date.

The associated logging database (aka Sequel.Core.Logging) for Security could grow heavily based on the log level configured and the usage. It is recommended to follow the housekeeping recommendations for the Sequel.Core.Logging project: take actions on proactive housekeeping of old data is important. When logging is configured to use the SQL repository, the housekeeping can be done with the stored procedure `dbo.sp_LogTableCleanup(@DaysToKeep INT, @DeleteBatchSize INT = 5000)`. This stored procedure removes old entries in batches: indicate the number of days to keep (and all previous days will be removed) and size of each batch (delete action). This deletion is done in batches until there are no more outstanding rows that meets the filter; reducing the impact of this action in the database.

Ideally, this stored procedure can be configured in a job every day to remove old data keeping just a few days. More information about creating SQL job steps at: https://docs.microsoft.com/en-us/sql/ssms/agent/create-a-job?view=sql-server-ver15.

A T-SQL script for scheduling a housekeeping job for removing older entries than 180 days (6 months), using batches of 500 records in `##DATABASE##` database, could look like:

```
USE msdb;
GO

DECLARE @jobName nvarchar(50);
DECLARE @scheduleName nvarchar(50);
DECLARE @jobStepCommand nvarchar(2000);

set @jobName = N'Logging_Database_Housekeeping_Job' ;
set @scheduleName = N'Logging_Database_Housekeeping_Schedule';
set @jobStepCommand = N'EXEC [##DATABASE##].[dbo].[sp_LogTableCleanup] 180, 500';

EXEC dbo.sp_add_job
   @job_name = @jobName;

EXEC sp_add_jobstep
   @job_name = @jobName,
   @step_name = N'Remove logs older than 180 days',
   @subsystem = N'TSQL',
   @command = @jobStepCommand,
   @retry_attempts = 5,
   @retry_interval = 5 ;

EXEC dbo.sp_add_schedule
   @schedule_name = @scheduleName,
   @freq_type = 4,  -- Daily
   @freq_interval = 1, -- Once a day
   @freq_subday_type = 0x1, -- At a given time
   @active_start_time = 010000; --at 01:00 AM. On a 24-hour clock, and must be entered using the form HHMMSS.

EXEC sp_attach_schedule
   @job_name = @jobName,
   @schedule_name = @scheduleName;

EXEC dbo.sp_add_jobserver
   @job_name = @jobName;
GO
```

While this job for housekeeping is not required, is highly recommended.

Event logs

Regular checks of the Windows Application, System and Security event logs are recommended. In particular, the system log should be inspected for any sign of impending service interruption (such as low disk space warnings, net log-on warnings, disk timeouts etc). Additionally, the Windows Application event log should be inspected and any unexpected application errors analysed to assess the potential impact.

It is suggested that the event logs be checked on a regular basis e.g. weekly or monthly (and configured to roll over more frequently than this so events are not lost before they can be analysed).

Internet Information Services Logs

Ensure that if IIS logs are enabled for diagnostics are later turned off, or configured to automatically delete old files. More information at Microsoft documentation: https://docs.microsoft.com/en-us/iis/manage/provisioning-and-managing-iis/managing-iis-log-file-storage.

System capacity

As with any system, regular capacity checks should be made of the platform hosting application environment. These checks should include:

- Checking that there is sufficient disk space for continued operation on all volumes used by the system (based on existing usage trends). Additionally, de-fragmentation of the file system requires a minimum amount of free disk space (typically 15%). De-fragmentation of large files (SQL database) requires significant free disk space.
- Checking that there is sufficient system processing resource to allow the system to continue to perform optimally. This typically involves ensuring that the system is not building processor queues, and that there are free CPU cycles available during normal use. Occasional peaks of 100% CPU utilisation may be acceptable but if servers are operating regularly at 80% or more during working hours further work should be carried out to analyse the cause and implement plans to mitigate the issue.
- Checking that there is sufficient system memory available to allow the system to perform optimally. This typically involves confirming that the system is not paging to disk excessively, and that ideally there is free memory available for the system to use as a disk cache and free page pool. A minimum of 5% of free memory should always be available on the application server.

System changes / software updates

Best practice dictates that all changes to the platform and application environment are change controlled. This provides a necessary degree of *due diligence* control over system alterations. All changes should be clearly planned with any possible rollback detailed and then peer reviewed before implementation. Additionally, change control logs can be reviewed when diagnosing issues with the system. These logs can also be shared with the vendor to assist in vendor diagnosis. It is also important to ensure that the environments always adhere to the platform specification provided as part of the release.

4.7.3 Database maintenance

The SQL databases used by Sequel Security Services will benefit from the majority of typical period database maintenance procedures. As with any maintenance – good backups and a roll-back plan are prerequisite to any activity (https://docs.microsoft.com/en-us/sql/relational-databases/performance/monitor-and-tune-for-performance?view=sql-server-2017). Maintenance operations like should cover:

Index and statistics maintenance

SQL Server statistics are important for optimizing query speeds, by default SQL Server automatically updates statistics but it also recommended that you perform manual updates periodically. It is recommend to schedule an overnight job to run the attached script against Security databases. This script will reindex all tables in this database, not just those under the Authentication and Authorization schemas.

```
/*

SCRIPT_NAME         : Reindex_Script_AllTablesInADb.sql
PURPOSE             : Re-indexing database - varies rebuild / reorganise based on fragmentation.
AUTHOR              : Venkat
CREATED DATE        : 02-Oct-2014

*/

DECLARE @IndexOption varchar(255),
@ObjectName varchar(255),
@sql nvarchar(500);

/*
     Deal with Indexes first because rebuilding an index also updates the the statistics.
     We want to include both tables and Indexed views. Ignore small tables that are
     contained within one Extent.
*/

DECLARE IndexCursor CURSOR LOCAL FAST_FORWARD
FOR
SELECT  '[' + OBJECT_SCHEMA_NAME(O.[object_id]) + '].[' + O.NAME + ']' ObjectName,
          CASE WHEN MAX(st.avg_fragmentation_in_percent) >= 30 THEN ' REBUILD;'
               WHEN MAX(ST.avg_fragmentation_in_percent) > 5 THEN ' REORGANIZE;'
          END AS IndexOption
FROM    sys.objects AS O
          INNER JOIN sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, 'DETAILED') AS ST ON O.[object_id] = ST.[object_id]
                                                                                                   AND ST.avg_fragmentation_in_percent > 5
                                                                                                   AND ST.page_count > 8
WHERE   O.[Type] = 'U'
          OR (
               O.[type] = 'V'
               AND EXISTS ( SELECT 1
                            FROM   sys.indexes AS I
                            WHERE  I.[object_id] = O.[object_id] )
               )
GROUP BY '[' + OBJECT_SCHEMA_NAME(O.[object_id]) + '].[' + O.NAME + ']'
ORDER BY ObjectName;

OPEN IndexCursor;

FETCH NEXT FROM IndexCursor INTO @ObjectName, @IndexOption;

WHILE @@FETCH_STATUS = 0
BEGIN
          PRINT 'Index:' + @ObjectName + ' ' + @IndexOption

     SET @sql = 'SET QUOTED_IDENTIFIER ON; ALTER INDEX ALL ON ' + @ObjectName + @IndexOption;

     EXEC (@sql);

     FETCH NEXT FROM IndexCursor INTO @ObjectName, @IndexOption;
END;

CLOSE IndexCursor;
DEALLOCATE IndexCursor;


/*
     Now sort out Statistics, if we rebuilt the indexes the statistics would have already been
     updated and will not be picked up by this script as we filter for tables and views that
     have had data modifications since the Statistics were last updated.
```

```
*/

DECLARE StatsCursor CURSOR LOCAL FAST_FORWARD
FOR
SELECT  'SET QUOTED_IDENTIFIER ON; UPDATE STATISTICS ' + D.TableName + ' ' + D.Statistic + ' WITH FULLSCAN;' AS SQLStatement, D.TableName + ' ' + D.Statistic AS
ObjectName
FROM    (
          SELECT QUOTENAME(SCHEMA_NAME(SCH.[schema_id]), '[') + '.' + QUOTENAME(O.name) AS TableName,
                    QUOTENAME(S.name) AS Statistic,
                    SP.last_updated AS StatsLastUpdated,
                    SP.[rows] AS RowsInTable,
                    SP.modification_counter AS RowModifications,
                    CAST(SP.modification_counter AS decimal(19, 2)) / NULLIF(SP.[rows], 0) * 100 AS PercentageChange
          FROM   sys.stats AS S
                    INNER JOIN sys.objects AS O ON S.[object_id] = O.[object_id]
                    INNER JOIN sys.schemas AS SCH ON O.[schema_id] = SCH.[schema_id]
                    CROSS APPLY sys.dm_db_stats_properties(O.[object_id], S.[stats_id]) AS SP
          WHERE  O.[type] IN ('U', 'V')
                    AND SP.modification_counter > 0
          ) AS D
WHERE   (
          PercentageChange > 1
          OR (
               RowModifications >= 1000
               AND RowsInTable >= 100000
               )
          )
ORDER BY SQLStatement;

OPEN StatsCursor;

FETCH NEXT FROM StatsCursor INTO @sql, @ObjectName;

WHILE @@FETCH_STATUS = 0
BEGIN

          PRINT 'Statistic:' + @ObjectName

     EXEC (@sql);

     FETCH NEXT FROM StatsCursor INTO @sql;
END;

CLOSE StatsCursor;
DEALLOCATE StatsCursor;

GO
```

Integrity checking

It is prudent to periodically check database integrity and again Sequel recommends a regular set of checks are implemented. This should first be benchmarked on a production like environment to gather information to allow for realistic schedule regarding content and timings to be implemented.

INTEGRITY CHECKING (LIGHT)

For example a good starting point would be to regularly run the following command:

```
     DBCC CHECKDB (<DatabaseName>, NOINDEX) WITH PHYSICAL_ONLY
```

It is recommended that the command is run with the 'PHYSICAL_ONLY' option to reduce the run time. Using the 'PHYSICAL_ONLY' option limits the consistency check to the physical structure of the database, b-tree structure and allocation consistency of the database. It can also detect torn pages, checksum failures and common hardware failure that compromise data. The `(NOINDEX)` option stops the `CHECKDB` command from checking clustered indexes, which also significantly reduces execution time. By adding the `NO_INFOMSGS` option only displays error messages and not additional information messages, therefore can be more useful for quickly analysing the output of the integrity check.

INTEGRITY CHECKING (FULL)

A more comprehensive set of checks should be implemented to run less frequently depending on time and service constraints. An example of a suitable command is:

```
     DBCC CHECKDB WITH NO_INFOMSGS
```

This is a full integrity check which by adding the `NO_INFOMSGS` option only displays error messages and not additional information messages, therefore can be more useful for quickly analysing the output of the integrity check.

Data Size / Space

It is very important to guarantee that your databases will not run out of space and is therefore sized appropriately. It is recommended to set the databases logical files to auto grow when necessary. However, it is important to note that the auto grow feature is only used as a safety net. You should not rely on auto grow to grow your databases in a production environment as the allocation of extra space will impact disk performance, increase fragmentation and therefore may also degrade overall user experience.

Ensure that the database is configured to auto grow,

```
ALTER DATABASE [database]
      NAME = [logical filename], FILEGROWTH = 10%
```

This command should be execute for each logical file in the database. The names of the logical files can be retrieved by running 'sp_helpfile' against the database. Best practice dictates periodic review of the free space in a database. Run the following command against the main db:

```
sp_spaceused
```

This will detail the free space in the database If free space is below a safe threshold (based on current trends), the database size should be increased (outside of working hours) by using the command:

```
alter database [DatabaseName]
modify file (name = [LogicalFileName] , size = [absolute new file size])
```

...for each file to be grown. The logical file names are those returned by the previous `sp_spaceused` command.

> The size is the total new size of the file, not the amount by which to grow.

It is also recommended to periodically delete rows that are no longer relevant from the audit log table in the Sequel Security databases as it can easily grow very large.

SQL Alerting

SQL Agent alerting can be used as a final layer of monitoring to trap error states relating to low free space in the database and the inability to grow database files.

Error numbers for typical errors are reproduced below for convenience. Additionally it is possible to retrieve a list of all error number & associated descriptions by running the query `select * from sys.messages`

Free space related errors:

Error ID	Error Description
1101	Could not allocate new page for database '%.*ls'. *There are no more pages available in file group* %.ls. Space can be created by dropping objects, adding additional files, or allowing file growth.
1105	Could not allocate space for object '%.*ls' *in database* '%.ls' because the '%.*ls' file group is full.
1703	Could not allocate disk space for a work table in database '%.*ls'. You may be able to free up space by using BACKUP LOG, or you may want to extend the size of the database by using ALTER DATABASE.
9002	The log file for database '%.*ls' is full. Back up the transaction log for the database to free up some log space.

4.8 Advanced

4.8.1 Resilience in Security

Something that is resilient is something that is *able to withstand or recover quickly from difficult conditions*.

In this document we are going describe whether or not all of the different Security services are resilient when other services they depend on are not available, offline or simply not responding.

Security Admin

AUTHENTICATION NOT AVAILABLE: RESILIENT (PAGE REFRESH REQUIRED)

Security Admin stays on the loading page "Authenticating, please wait..." indefinitely. When Authentication comes back online we have to refresh the page for Security Admin to redirect us correctly.

SECURITY API NOT AVAILABLE: RESILIENT (PAGE REFRESH REQUIRED)

Security Admin redirects us to the Authentication page with no problem, although trying to login gives un an error message at the top of the screen. If we are already logged in to Security Admin, navigating to certain sections of the application, when the Security API is offline, will give us constant loading animations. When the Security API is back online, we have to refresh the page for correct behavior.

Authorization

AUTHENTICATION NOT AVAILABLE: RESILIENT

If we are already authenticated and Authentication is down the Authorization server still works without any issues. If we have to authenticate then we get a "Service unavailable" error message returned. When it's back online we can authenticate with no problems immediately after.

SECURITY API NOT AVAILABLE: RESILIENT

Authorization simply returns the values from it's cache if the Security API is down. If Authorization gets notified that there has been a change in the permissions and it has to go back to the Security API to refresh it's cache then we are returned with an Internal Server Error, but when Security API is back online and we try the call again then we get an updated response immediately.

MESSAGE BUS NOT AVAILABLE: RESILIENT

Authorization is completely unavailable if the message bus is not available, but when it comes back online Authorization recovers immediately.

LOGGING DATABASE OFFLINE: RESILIENT

Authorization still returns the correct information even if the logging database is offline.

Authentication

SECURITY API NOT AVAILABLE: RESILIENT

If the Security API is down then the Authentication login page loads up fine but it's impossible to authenticate ourselves as we are given an unexpected error message at the top of the screen. As soon as the API is back the login works perfectly.

LOGGING DATABASE OFFLINE: RESILIENT

Authentication still returns the correct information even if the logging database is offline.

Security API

AUTHENTICATION NOT AVAILABLE: RESILIENT

If Authentication is down then the Security API will obviously not be able to authenticate, but as soon as it comes back online a reattempt is enough to continue as normal.

MESSAGE BUS NOT AVAILABLE: RESILIENT

When the message bus is not available, certain API call that don't publish anything to the bus still work fine but the ones that do publish messages get stuck with no response. When the message bus comes back online if we simply reattempt the API call it should work correctly and publish the expected messages without any kind of manual intervention.

LOGGING DATABASE OFFLINE: RESILIENT

Just like the rest of the applications when the logging database is offline the Security API works perfectly fine, it just doesn't write to the logging database. When it's back online it writes logs again without any intervention.

MULTITENANT DATABASE OFFLINE: RESILIENT

The database info is cached so the Security API runs normally when the multitenancy database is offline and is not affected at all by this change. If, though, the Security API service is restarted somehow, therefore clearing the cache, then the API doesn't respond until the multitenancy database is back online.

TENANT DATABASE OFFLINE: RESILIENT

If the main tenant database is offline then of course the Security API will not be able to authenticate calls, but if we are already authenticated the problem relies on not even being able to access any of the data in the database, making it unusable. When the database is back online data can be retrieved immediately.

SMTP SERVER NOT AVAILABLE: RESILIENT

With no available SMTP server then the "Reset Password" functionality will not be functional, but when the SMTP server recovers then the email sending from Security should work perfectly.

Security Sync Service

MESSAGE BUS NOT AVAILABLE: NOT RESILIENT

With no message bus available then the Sync service is unable to publish messages and therefore the sync process cannot be triggered. If the message bus recovers then we have to manually restart the Sync service for it to recover it's expected behavior.

LOGGING DATABASE OFFLINE: RESILIENT

Just like the rest of the applications when the logging database is offline the Sync service works perfectly fine, it just doesn't write to the logging database. When it's back online it writes logs again without any intervention.

LDAP Sync Service

MESSAGE BUS NOT AVAILABLE: RESILIENT

When the message bus is not available, if an LDAP sync process is attempted then nothing happens at all, no entry is created in the LDAP Monitoring tab, just an error in the database. When the message bus is up and running again if we click the button to manually synchronize, Security Admins UI indicates the request for a sync has been done but, and now the new entry for the sync attempt is successfully created with no issues.

LOGGING DATABASE OFFLINE: RESILIENT

Just like the rest of the applications when the logging database is offline the LDAP Sync service works perfectly fine, it just doesn't write to the logging database. When it's back online it writes logs again without any intervention.

SECURITY API NOT AVAILABLE: RESILIENT

If the Security API is down, even if the LDAP Sync Service automatically runs it cannot complete any kind of sync as it depends on the API to create the new entry in the database and to updated or create users. Once back online though, synchronization is immediately available again though.

Azure AD Sync Service

MESSAGE BUS NOT AVAILABLE: RESILIENT

When the message bus is not available, if any sync process has been already scheduled it will be executed. if a new Azure AD sync process is attempted from the UI then nothing happens at all, no entry is created in the Azure AD Monitoring tab, just an error in the database. When the message bus is up and running again if we click the button to manually synchronize, Security Admins UI indicates the request for a sync has been done but, and now the new entry for the sync attempt is successfully created with no issues.

LOGGING DATABASE OFFLINE: RESILIENT

Just like the rest of the applications when the logging database is offline the Azure AD Sync service works perfectly fine, it just doesn't write to the logging database. When it's back online it writes logs again without any intervention.

SECURITY API NOT AVAILABLE: RESILIENT

If the Security API is down, even if the Azure AD Sync Service automatically runs it cannot complete any kind of sync as it depends on the API to create the new entry in the database and to updated or create users. Once back online though, synchronization is immediately available again though.

4.8.2 Docker

Security Administration

The administration image runs an nginx server with the static files at the root path. The static files are generated at the same time as the image. It is also prepared to be cached, so some steps could be skipped if has not been changes from previous builds.

BUILDING THE IMAGE

The image must be built from the root of the `Sequel.Security.Admin` project, not from the source root. The `Dockerfile` is inside the `docker` folder. Building the image does not require any parameters, but the version on the `appsettings.json` is filled from the value of `package.json`.

```
docker image build -t security/admin -f docker/Dockerfile .
```

There is also a `SECURITY_VERSION` build argument that is useful for CD pipelines to specify which is the version that is building.

Showing the right version

As mentioned before, the version in the `appsettings.json` file (the one inside `public` folder) is filled from the `package.json` when the image is being built. If the version showing in the web is wrong, then `version` in `package.json` should be changed to point to the right version.

Use BuildKit to build images

It is recommended to use BuildKit instead the classic build system to improve build times of the image and take advantage of advanced cache system that brings this new system.

RUNNING THE IMAGE

To properly run the web page, the image requires to know where the API and Authentication are located and what is the path where is located publicly the admin web. The latter is usually done when building the static files, but to allow more dynamic paths, this setting has been delegated to this step. The parameters are set using environment variables:

var	default value	optional	description
`PUBLIC_URL`	`/administration/`	yes	Public URL or path prefix for the administration web. Must end with `/`!
`PUBLIC_API_URL`	`/securityapi`	yes	Public URL or path prefix for the Security API service. Must not end with `/`.
`PUBLIC_AUTHENTICATION_URL`	`/authentication`	yes	Public URL or path prefix for the Security Authentication service. Must not end with `/`.
`AUTHENTICATION_FLOW`	`authorizationCode`	yes	Tells which authentication flow to use. Valid values are `implicit` and others (which will use authorization code).
`ROUTER_TYPE`	`browser`	yes	Tells the UI which router to use. Valid values are `hash` and `browser`. Invalid values will be taken as `hash`.

An example of running a container with the image pointing to the API and Authentication deployed using Visual Studio and IIS Express:

```
docker container run \
    --rm \
    -it \
    -e 'PUBLIC_URL=/' \
    -e 'PUBLIC_API_URL=https://localhost:44367' \
    -e 'PUBLIC_AUTHENTICATION_URL=https://localhost:44343' \
    -p 3000:80 \
    security/admin
```

Security API

The API image runs the .NET Core app directly inside the `/app` folder. The same image takes care of compiling the code. It is also prepared to be cached, so some steps could be skipped if has not been changes from previous builds.

BUILDING THE IMAGE

The image must be built from the `/Source` folder. The `Dockerfile` is inside the `Projects/Sequel.Security.Api/docker` folder. Building the image does not require any parameters, but the `appsettings.json` file will not be included in the image, this must be included either by creating a new image or using a volume or secret.

```
docker image build -f Projects/Sequel.Security.Api/docker/Dockerfile -t security/api .
```

There is also a `SECURITY_VERSION` build argument that is useful for CD pipelines to specify which is the version that is building.

> **Use BuildKit to build images**
>
> It is recommended to use BuildKit instead the classic build system to improve build times of the image and take advantage of advanced cache system that brings this new system.

PREPARING THE APP SETTINGS

As mentioned before, the image does not include an `appsettings.json` file, so it will always fail to run if no one is provided. The are three ways to add the settings into the container:

1. Create a new image which adds the file inside `/app` (not recommended)
2. Create a volume that maps the file into the container (`-v` argument in docker)
3. Create a secret in Swarm or Kubernetes and put it in `/app/appsettings.json`

There are also some settings that must be set taking into account the deployment environment:

- The `InitializeSSL` must be always set to `false`, failing to do so, will cause some troubles in the API.
- The `TrustForwardedHeaders` should be set to `true` if the API will be behind a reverse proxy/load balancer.
- The `EnableContainerEndpoint` in `HealthCheckSettings` must be set to `true`, if not, the container will be marked as unhealthy.
- The `DoubleEncodingRevert` in `RewriterSettings` must be set to `true` (it is outside IIS, it must be set to true)
- The `MultitenancyDatabase` context must point to the right SQL Server instance, using the right.hostname, and must use SQL Server Authentication login method (provide `User Id` and `Password`). Do the same for the `Logging` context.
- Ensure internal and external URLs for the API and Authentication services are OK.
- Check the email host to ensure it is accessible (health check may help).
- Check the RabbitMQ host to ensure it is accessible too.
- Check the health check (example command `curl -i -H "HealthCheck-ApiKey: cdb327e2-4f20-4a55-b44a-d3b948b84356" localhost/securityapi/health`)

> **Access to hosts services from Docker**
>
> To properly access the host services from inside a container, you must use special domains or IPs to accomplish that, and depends on the OS.
>
> - **Docker Desktop for Windows**: use `docker.for.win.localhost` to access host services.
> - **Docker Desktop for macOS**: use `docker.for.mac.localhost` to access host services.
> - **Docker on Linux**: there are several ways to do it, but probably the easiest is to point to the external IP of the host. Another way is to use the gateway of the docker network (and ensure services are listening in this network interface too).

SQL Server on Windows

This will not work unless the SQL Server has several changes applied. First, as mentioned, the API will use SQL Server Authentication method for login, instead of Windows AD. This must be enabled to allow this kind of logins. To connect to the server, will use a TCP connection, and must be enabled too.

- Enable TCP connections in SQL Server.
- Enable SQL Server Authentication login in SQL Server Management Studio.
- If you cannot find the SQL Server Configuration Manager, see this link.
- Restart `MSSQL` service.
- Then create a login using SQL Server Authentication method, and give that user permission to the databases.

RUNNING THE IMAGE

Once the `appsettings.json` file is ready, you may start a container and check everything is fine. The API requires Authentication to be running too, but API must run before Authentication. Below there is an example running a container and using a bind mount to provide the `appsettings.json` file:

```
docker container run --rm -it -v /deployments/sec/appsettings.api.json:/app/appsettings.json -p 8000:80 security/api
```

Running behind Reverse Proxies

Usually, in a reverse proxy environment, the published services will have path prefixes to have several services run under the same domain. To avoid any issues with URLs in Security API, it is recommended to fill the `PATH_PREFIX` environment variable in the container and disable any path prefix strip in the reverse proxy, Security API will do the rest.

Security Authentication

The API image runs the .NET Core app directly inside the `/app` folder. The same image takes care of compiling the code. It is also prepared to be cached, so some steps could be skipped if has not been changes from previous builds.

BUILDING THE IMAGE

The image must be built from the `/Source` folder. The `Dockerfile` is inside the `Projects/Sequel.Security.Authentication/docker` folder. Building the image does not require any parameters, but the `appsettings.json` file will not be included in the image, this must be included either by creating a new image or using a volume or secret.

```
docker image build -f Projects/Sequel.Security.Authentication/docker/Dockerfile -t security/authentication .
```

There is also a `SECURITY_VERSION` build argument that is useful for CD pipelines to specify which is the version that is building.

> **Use BuildKit to build images**
>
> It is recommended to use BuildKit instead the classic build system to improve build times of the image and take advantage of advanced cache system that brings this new system.

PREPARING THE APP SETTINGS

As mentioned before, the image does not include an `appsettings.json` file, so it will always fail to run if no one is provided. The are three ways to add the settings into the container:

1. Create a new image which adds the file inside `/app` (not recommended)
2. Create a volume that maps the file into the container (`-v` argument in docker)
3. Create a secret in Swarm or Kubernetes and put it in `/app/appsettings.json`

There are also some settings that must be set taking into account the deployment environment:

- The `InitializeSSL` must be always set to `false`, failing to do so, will cause some troubles in Authentication.
- The `TrustForwardedHeaders` should be set to `true` if Authentication will be behind a reverse proxy/load balancer.
- The `EnableContainerEndpoint` in `HealthCheckSettings` must be set to `true`, if not, the container will be marked as unhealthy.
- The `Logging` context must point to the right SQL Server instance, using the right.hostname, and must use SQL Server Authentication login method (provide `User Id` and `Password`).
- Ensure internal and external URLs for the API and Authentication services are OK.
- Windows identity provider will not work, so disable it if enabled.
- Check the health check (example command `curl -i -H "HealthCheck-ApiKey: cdb327e2-4f20-4a55-b44a-d3b948b84356" localhost/securityapi/health`)

> **Access to hosts services from Docker**
>
> See the Security API section for more information about this.

> **SQL Server on Windows**
>
> See the Security API section for more information about this.

RUNNING THE IMAGE

Once the `appsettings.json` file is ready, you may start a container and check everything is fine. Authentication requires API to be running too. Below there is an example running a container and using a bind mount to provide the `appsettings.json` file:

```
docker container run --rm -it -v /deployments/sec/appsettings.authentication.json:/app/appsettings.json -p 8001:80 security/authentication
```

Running behind Reverse Proxies

Usually, in a reverse proxy environment, the published services will have path prefixes to have several services run under the same domain. To avoid any issues with URLs and redirections in Security Authentication, the `PATH_PREFIX` environment variable must be set in the container. In the reverse proxy settings, path prefix strip must be disabled.

Security Authorization

The API image runs the .NET Core app directly inside the `/app` folder. The same image takes care of compiling the code. It is also prepared to be cached, so some steps could be skipped if has not been changes from previous builds.

BUILDING THE IMAGE

The image must be built from the `/Source` folder. The `Dockerfile` is inside the `Projects/Sequel.Security.Authorization/docker` folder. Building the image does not require any parameters, but the `appsettings.json` file will not be included in the image, this must be included either by creating a new image or using a volume or secret.

```
docker image build -f Projects/Sequel.Security.Authorization/docker/Dockerfile -t security/authorization .
```

There is also a `SECURITY_VERSION` build argument that is useful for CD pipelines to specify which is the version that is building.

> **Use BuildKit to build images**
>
> It is recommended to use BuildKit instead the classic build system to improve build times of the image and take advantage of advanced cache system that brings this new system.

PREPARING THE APP SETTINGS

As mentioned before, the image does not include an `appsettings.json` file, so it will always fail to run if no one is provided. The are three ways to add the settings into the container:

1. Create a new image which adds the file inside `/app` (not recommended)
2. Create a volume that maps the file into the container (`-v` argument in docker)
3. Create a secret in Swarm or Kubernetes and put it in `/app/appsettings.json`

There are also some settings that must be set taking into account the deployment environment:

- The `InitializeSSL` must be always set to `false`, failing to do so, will cause some troubles in Authorization.
- The `TrustForwardedHeaders` should be set to `true` if Authorization will be behind a reverse proxy/load balancer.
- The `EnableContainerEndpoint` in `HealthCheckSettings` must be set to `true`, if not, the container will be marked as unhealthy.
- The `DoubleEncodingRevert` in `RewriterSettings` must be set to `true` (it is outside IIS, it must be set to true)
- The `Logging` context must point to the right SQL Server instance, using the right.hostname, and must use SQL Server Authentication login method (provide `User Id` and `Password`).
- Ensure internal and external URLs for the API, Authentication and Authorization services are OK.
- Check the RabbitMQ host to ensure it is accessible too.
- Check the health check (example command `curl -i -H "HealthCheck-ApiKey: cdb327e2-4f20-4a55-b44a-d3b948b84356" localhost/securityapi/health`)

> **Access to hosts services from Docker**
>
> See the Security API section for more information about this.

> **SQL Server on Windows**
>
> See the Security API section for more information about this.

RUNNING THE IMAGE

Once the `appsettings.json` file is ready, you may start a container and check everything is fine. Authentication requires API and Authentication to be running too. Below there is an example running a container and using a bind mount to provide the `appsettings.json` file:

```
docker container run --rm -it -v /deployments/sec/appsettings.authorization.json:/app/appsettings.json -p 8002:80 security/authorization
```

> **Running behind Reverse Proxies**
>
> Usually, in a reverse proxy environment, the published services will have path prefixes to have several services run under the same domain. To avoid any issues with URLs in Security Authorization, it is recommended to fill the `PATH_PREFIX` environment variable in the container and disable any path prefix strip in the reverse proxy, Security Authorization will do the rest.

Security SaaS API

The API image runs the .NET Core app directly inside the `/app` folder. The same image takes care of compiling the code. It is also prepared to be cached, so some steps could be skipped if has not been changes from previous builds.

BUILDING THE IMAGE

The image must be built from the `/Source` folder. The `Dockerfile` is inside the `Projects/Sequel.Security.SaaS` folder. Building the image does not require any parameters, but the `appsettings.json` file will not be included in the image, this must be included either by creating a new image or using a volume or secret.

```
docker image build -f Projects/Sequel.Security.SaaS/Dockerfile -t security/saas-api .
```

> **Use BuildKit to build images**
>
> It is recommended to use BuildKit instead the classic build system to improve build times of the image and take advantage of advanced cache system that brings this new system.

PREPARING THE APP SETTINGS

As mentioned before, the image does not include an `appsettings.json` file, so it will always fail to run if no one is provided. The are three ways to add the settings into the container:

1. Create a new image which adds the file inside `/app` (not recommended)
2. Create a volume that maps the file into the container (`-v` argument in docker)
3. Create a secret in Swarm or Kubernetes and put it in `/app/appsettings.json`

There are also some settings that must be set taking into account the deployment environment:

- The `TrustForwardedHeaders` should be set to `true` if the API will be behind a reverse proxy/load balancer.
- Ensure paths for Ansible are correct and point into container paths.
- Prepare Ansible variables for the playbooks (see below guide).
- The `hosts` Ansible file must contain a host that point to a server with access to the k8s cluster and has docker running (see below guide).
- The `Database` context must point to the right SQL Server instance, using the right hostname (including `.office.sbs`), and must use SQL Server Authentication login method (provide `User Id` and `Password`). Do the same for the `Logging` context.
- Ensure internal and external URLs for the API and Authentication services are OK.
- Check the email host to ensure it is accessible (health check may help).
- Check the RabbitMQ host to ensure it is accessible too.
- Check the health check (example command `curl -i localhost/health`)

> **Access to hosts services from Docker**
>
> To properly access the host services from inside a container, you must use special domains or IPs to accomplish that, and depends on the OS.
>
> - **Docker Desktop for Windows**: use `docker.for.win.localhost` to access host services.
> - **Docker Desktop for macOS**: use `docker.for.mac.localhost` to access host services.
> - **Docker on Linux**: there are several ways to do it, but probably the easiest is to point to the external IP of the host. Another way is to use the gateway of the docker network (and ensure services are listening in this network interface too).

SQL Server on Windows

This will not work unless the SQL Server has several changes applied. First, as mentioned, the API will use SQL Server Authentication method for login, instead of Windows AD. This must be enabled to allow this kind of logins. To connect to the server, will use a TCP connection, and must be enabled too.

- Enable TCP connections in SQL Server.
- Enable SQL Server Authentication login in SQL Server Management Studio.
- If you cannot find the SQL Server Configuration Manager, see this link.
- Restart `MSSQL` service.
- Then create a login using SQL Server Authentication method, and give that user permission to the databases.

RUNNING THE IMAGE

The best way to run the image is deploying it directly to a Kubernetes cluster using the Helm chart (see below).

Image run as non-privileged user

The image by default runs in a non-privileged user called `saas` (1000:1000). In Linux, if you encounter any issues when running the image, ensure the `appsettings.json` has read permissions or it is owned by that user. Under Windows or macOS, that cannot be changed and by default has `777` permissions (all permissions).

Running behind Reverse Proxies

Usually, in a reverse proxy environment, the published services will have path prefixes to have several services run under the same domain. To avoid any issues with URLs in Security SaaS API, it is recommended to fill the `PATH_PREFIX` environment variable in the container and disable any path prefix strip in the reverse proxy, Security SaaS API will do the rest.

PREPARE ENVIRONMENT

Before running the API, either for local development or production environments, some bits are required to properly work.

Development environment

There will be a guide for this soon...

The SaaS API runs the scripts through a SSH connection to a host. So, first the target host must be configured to run the scripts.

1. Ensure the server where the scripts are going to run has docker installed and working.
2. Create a new user or pick an already existing one. The user must be in the `docker` group (`sudo usermod -a -G docker $USER`).
3. Ensure to have installed `kubectl` CLI tool (see https://kubernetes.io/docs/tasks/tools/install-kubectl/).
4. Ensure to have installed `helm` v3 tool (see https://helm.sh/docs/intro/install/).
5. Ensure to have configured the target Kubernetes cluster - the file `$HOME/.kube/config` file configured to the cluster.
6. Log in in the registries inside that user (`docker login docker.sequel.com`). Do this with Helm too (`helm repo add --username $NEXUS_USER --password $NEXUS_PASS sequel https://nexus.sequel.com/repository/helm`).
7. The server must have installed python 3 installed:
   - On Debian/Ubuntu, they are installed by default.
   - On CentOS/RHEL install these packages `sudo yum install python3 libselinux-python3` .
8. Install `docker` python package:
   - On Debian/Ubuntu install `python3-docker` .
   - On CentOS/RHEL install `python36-docker` .
9. Create a SSH key pair and add the public one into `.ssh/authorized_keys` from the user's home (remember that this file must have `0600` permisions and owned by the user). Example command to create a SSH key pair: `ssh-keygen -t ed25519 -C "sec-saas"` (when asked where to place it, change the path).
10. (optional) Change `sshd` to disable password authentication.

After preparing the user to run the scripts, it is time to prepare the database and user in the SQL Server. For this, we need to create a user with password, and give `sysadmin` role (or equivalent role that can create users and databases). Then create a database which will hold the data for the API. This can be done using the MSSMS.

PREPARE THE DEPLOYMENT

Now it is time to prepare the Helm chart variables for the environment. This variables configures the deployment in Kubernetes as well as for the scripts. First create a `yaml` file and fill with following options (this file will be refered as `helm-vars.yaml` in this guide):

```yaml
image:
  # if using a different repository, change it here
  repository: docker.sequel.com/security/saas-api
  # tag: latest

# if ingress is going to be used, then enable this
ingress:
  enabled: true
  hosts:
    - host: identity.office.sbs
      paths:
        - /saas
  # if HTTPS is going to be used, fill this section with the secret and hosts
  tls:
    - secretName: security-saas-tls-cert # the secret must have this name, if shares the host with the security instances
      hosts:
        - identity.office.sbs

# this is the prefix it wil have under the reverse proxy/load balancer (should match the path in the ingress but with final slash /)
pathPrefix: '/saas/'

# change the hostname (host.office.sbs) and user to your needs
# this is the hosts inventory file contents
hostsInventory: |-
  [me]
  host.office.sbs ansible_user=saas

  [all:vars]
  ansible_python_interpreter=/usr/bin/python3

connectionString: Server=identitydb.office.sbs;Database=SecuritySaaS;User Id=sec-saas;Password=V3ryS()chP4ssw0rd;MultipleActiveResultSets=true

# put here several ansible variable files: the key is the name of the file and the contents are the contents of the file
# there are plenty of variables to change, they are all documented through the roles in the ansible playbooks
# below there is a basic configuration
ansibleExtraVariables:
  vars:
    # full path where to store some files for the instances (should be readable and writable by the user)
    security_installations_folder: /home/saas/installations
    # go connection string for the admin user
    security_admin_connection_string: 'sqlserver://sec-saas:V3ryS()chP4ssw0rd@identitydb.office.sbs?connection+timeout=10'
```

```
    # sql server address where the instances will be placed (must be the same as above variable)
    security_sql_server_address: identitydb.office.sbs
    # repository name for the Sequel Helm chart
    security_helm_chart_repository: sequel
    security_helm_chart_repository_type: Helm

    # repositories where the security images are stored
    security_admin_image: docker.sequel.com/security/admin
    security_api_image: docker.sequel.com/security/api
    security_authentication_image: docker.sequel.com/security/authentication
    security_authorization_image: docker.sequel.com/security/authorization
    security_tools_image: docker.sequel.com/security/tools

    # in local development, this is not needed
    security_authentication_settings:
      singleSignOn:
        cookieDomain: .office.sbs

    # template for the user and databases for instances
    security_database_template: 'saas-%s'
    # external URLs for the ingress and security configurations (but as templates)
    security_url_external_templates:
      admin: https://identity.office.sbs/%s/admin
      api: https://identity.office.sbs/%s/api
      authentication: https://identity.office.sbs/%s/authentication
      authorization: https://identity.office.sbs/%s/authorization

# put here the contents of the private SSH key file
sshPrivateKey: |-
  -----BEGIN OPENSSH PRIVATE KEY-----
  ...
  -----END OPENSSH PRIVATE KEY-----
```

With this configuration, we set up the environment for the ansible playbooks through the `hosts` file (aka inventory) and preconfigured variables. This setup is important because the operations depends on these variables to be filled properly. See the readme for the Security SaaS chart (`Source/Deployment/helm/sec-saas` in source code) to know more about these settings, including the readme for all roles (all folders inside `Source/Deployment/ansible-playbooks/roles` in source code) in the ansible playbooks folder.

The deployment is done using a Helm chart, you need to copy the Helm chart from the source (or use it directly) for the deployment.

DEPLOY TIME

It is time to deploy! With helm command, the deployment is done with a simple command:

```
helm install -f helm-vars.yaml salamandra /path/to/the/chart/sec-saas
```

Dry run

If you want to ensure everything is fine before deploying, run the previous command but with some modifications:

```
helm install --dry-run --debug -f helm-vars.yaml salamandra /path/to/the/chart/sec-saas
```

RBAC and Roles

If deploying the chart fails due to unknown API version (`rbac.authorization.k8s.io/v1beta1`) or something related to the `Role` and `RoleBinding` , then disable the RBAC from the chart (add this in the yaml file):

```
rbac:
  create: false
```

To check if your cluster has RBAC enabled, run the command `kubectl api-versions` and look for a line that looks like this `rbac.authorization.k8s.io/v1beta1` . If there is not such line, then RBAC is not enabled and above lines **must** be included.

Namespace

It is recommended to use a different namespace in production, so all instances are put in the same namespace but it is its own namespace. In our production environment, we are using `sec-saas` namespace. To create a namespace use `kubectl create namespace sec-saas` .

To check out if it is running, run `kubectl get pods` and look for the SaaS API Pod (something like `salamandra-saas-api-64f9c8f464-tcnng`) then run `kubectl describe pod salamandra-saas-api-64f9c8f464-tcnng` and you will see if it is working or not. If you get constants restarts due to bad health checks, add these two variables and redeploy using `helm upgrade` insted of `helm install`:

```
useSequelLogging: false
enableProbles: false
```

With these options, disables the health check probe from the deployment and puts all logs into the `stdout`, which can be read with `kubectl log salamandra-saas-api-64f9c8f464-tcnng`.

HTTPS AND CERTIFICATE

To properly use HTTPS, a certificate is required. The platform expects to have a secret called `security-saas-tls-cert` with the certificate and private key inside. The certificate and key must be in PEM format (text file that starts with `--- BEGIN CERTIFICATE ---`). There is a script that helps you create the secret manifest to be able to deploy it in the cluster.

SaaS certificate generator script

This script creates a file called `saas-cert.yaml` from a certificate and private key files. To run it, first create a file and fill it with the script contents. Then give the file execute permissions (`chmod +x ./cert.sh`) and run it (`./cert.sh identity.crt identity.key`).

```
#!/bin/bash

if [[ "$#" -lt 2 ]] || [[ "$1" = '--help' ]] || [[ "$1" = '-h' ]]; then
  echo "$0 <CERTIFICATE> <PRIVATE KEY>"
  exit 0
fi

if [[ ! -f "$1" ]]; then
  echo "Certificate file $1 must exist"
  exit 1
fi

if [[ ! -f "$2" ]]; then
  echo "Private key file $2 must exist"
  exit 1
fi

cat > saas-cert.yaml <<-EOM
apiVersion: v1
kind: Secret
type: kubernetes.io/tls
metadata:
  name: security-saas-tls-cert
data:
EOM

echo -n '  tls.crt: ' >> saas-cert.yaml
cat "$1" | base64 -w 0 >> saas-cert.yaml
echo >> saas-cert.yaml

echo -n '  tls.key: ' >> saas-cert.yaml
cat "$2" | base64 -w 0 >> saas-cert.yaml
echo >> saas-cert.yaml
```

4.8.3 Paas

Elastic beanstalk configuration

SUMMARY

This document describes how to configure and deploy Sequel Security using Elastic Beanstalk.

We are going to deploy each Security component individually in each instance. You will have to create an application in the Elastic Beanstalk console and multiples environments, each environment will have an application of Security deployed (Administrator, Authentication, Authorization and SecurityAPI).

You also will need an external database (AWS RDS database in this example) and a RabbitMQ server.

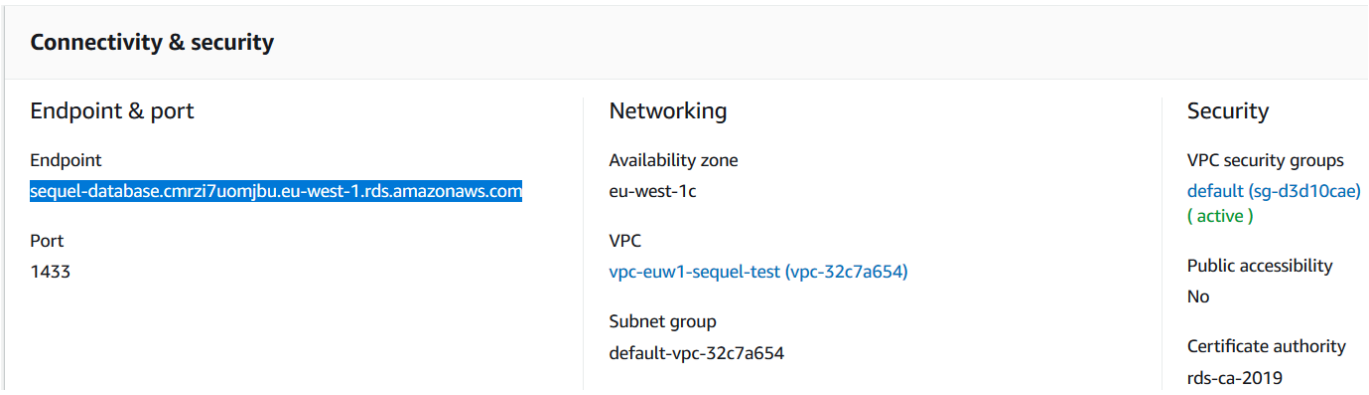
RDS DATABASE

To configure a database using RDS console, search for RDS in the service catalog https://eu-west-1.console.aws.amazon.com/rds/home?region=eu-west-1#databases

In this example we will create a database with the following options:

- Standard create
- Microsoft SQL Server
- SQL Server Express Edition (SQL Server 2017 14.00.3281.6.v1)
- Storage type: General Purpose (SSD), Allocated storage: 20, Storage autoscaling: Disabled
- VPC for Sequel internal: vpc-euw1-sequel-test (vpc-32c7a654)
- Publicly accessible: No
- VPC Security group: Choose existing or create a new one, up to you. In this example we use default.
- Availability zone: No preference
- Database port 1433
- Windows authentication disabled

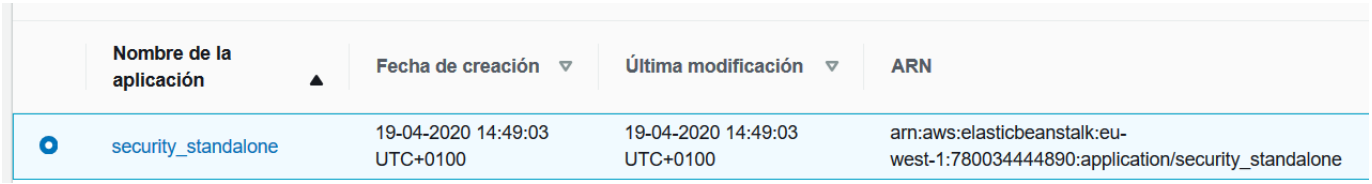
Once the database has been created, it will show you in the console an URL to connect with.



ELASTIC BEANSTALK APPLICATION AND ENVIRONMENTS CREATION

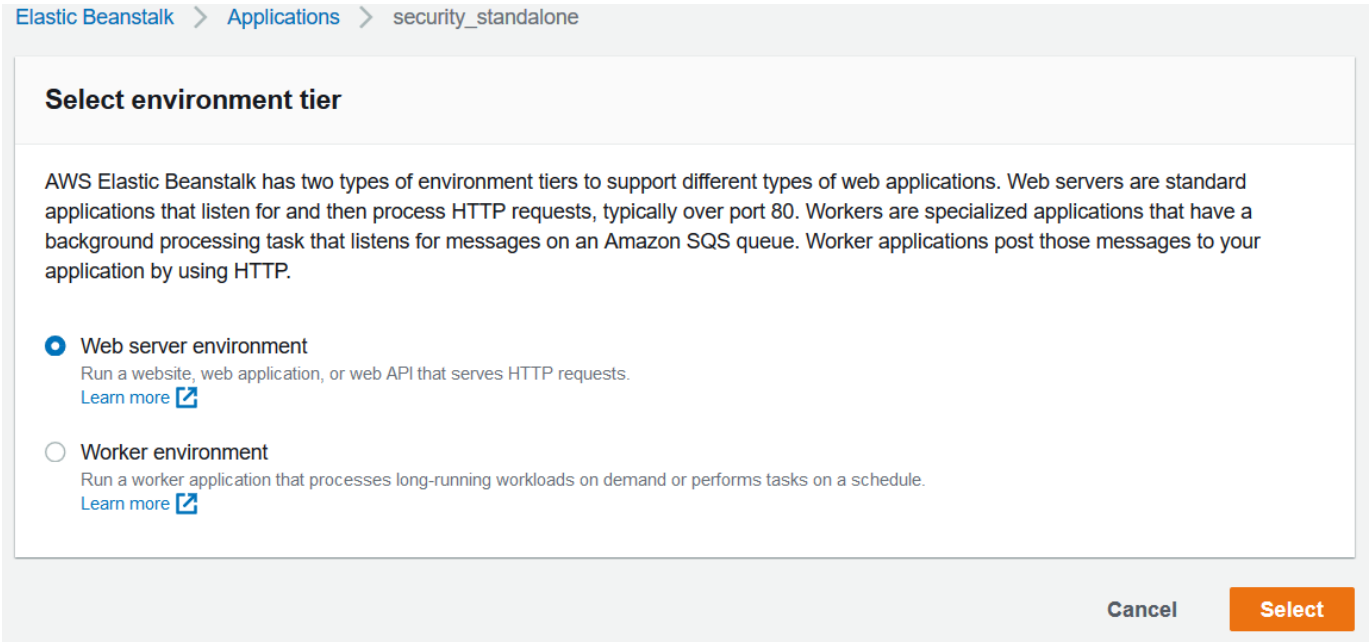
Search for Elastic Beanstalk in the Service Catalog.

In applications create a new application. In this example we've created "**security_standalone**" application.



	Nombre de la aplicación	Fecha de creación	Última modificación	ARN
◉	security_standalone	19-04-2020 14:49:03 UTC+0100	19-04-2020 14:49:03 UTC+0100	arn:aws:elasticbeanstalk:eu-west-1:780034444890:application/security_standalone

We will click in the application and now we will create an environment for SecurityAPI:



Environment information

Choose the name, subdomain, and description for your environment. These cannot be changed later.

Application name

security_standalone

Environment name

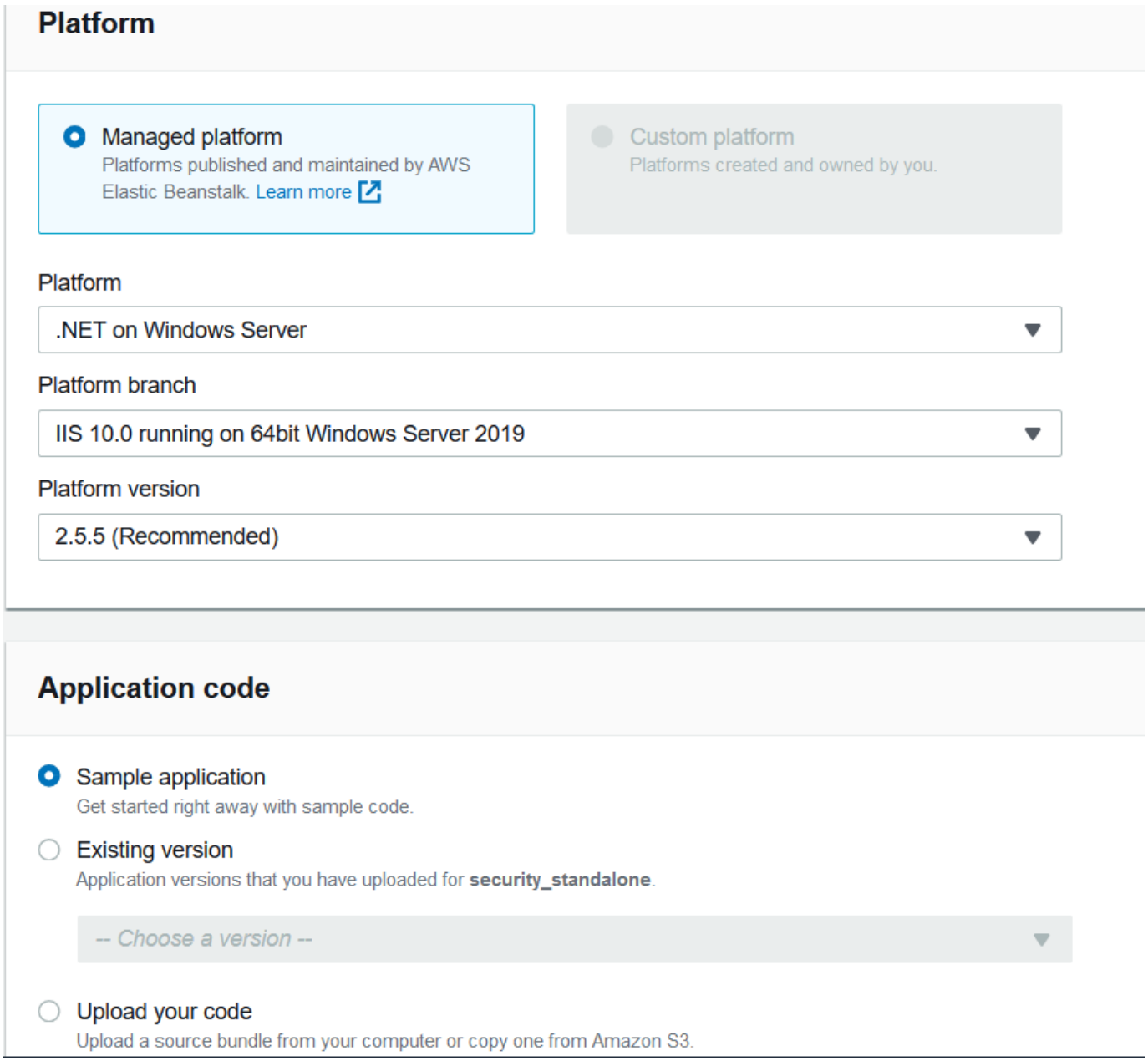
SecurityAPI

Domain

Leave blank for autogenerated value .eu-west-1.elasticbeanstalk.

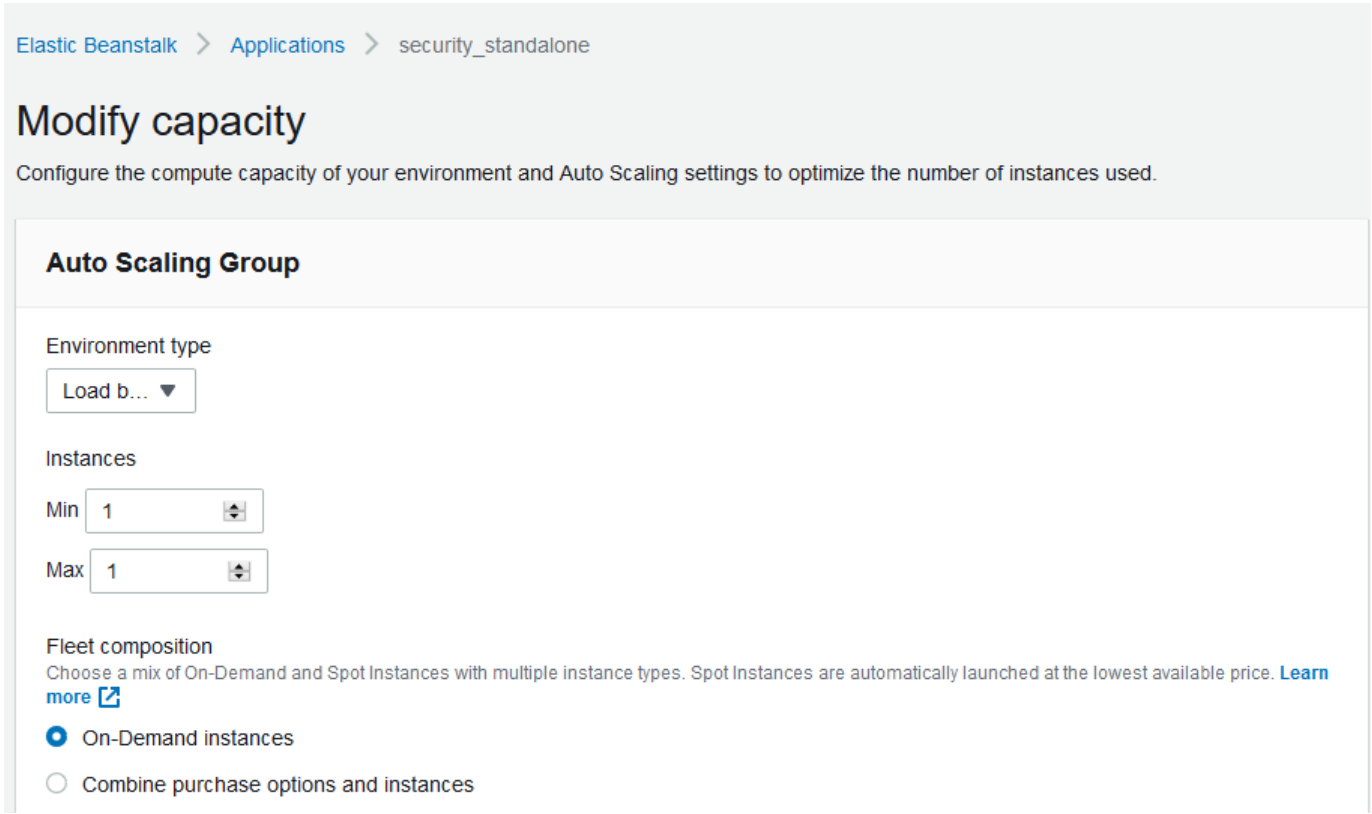
Check availability

Description

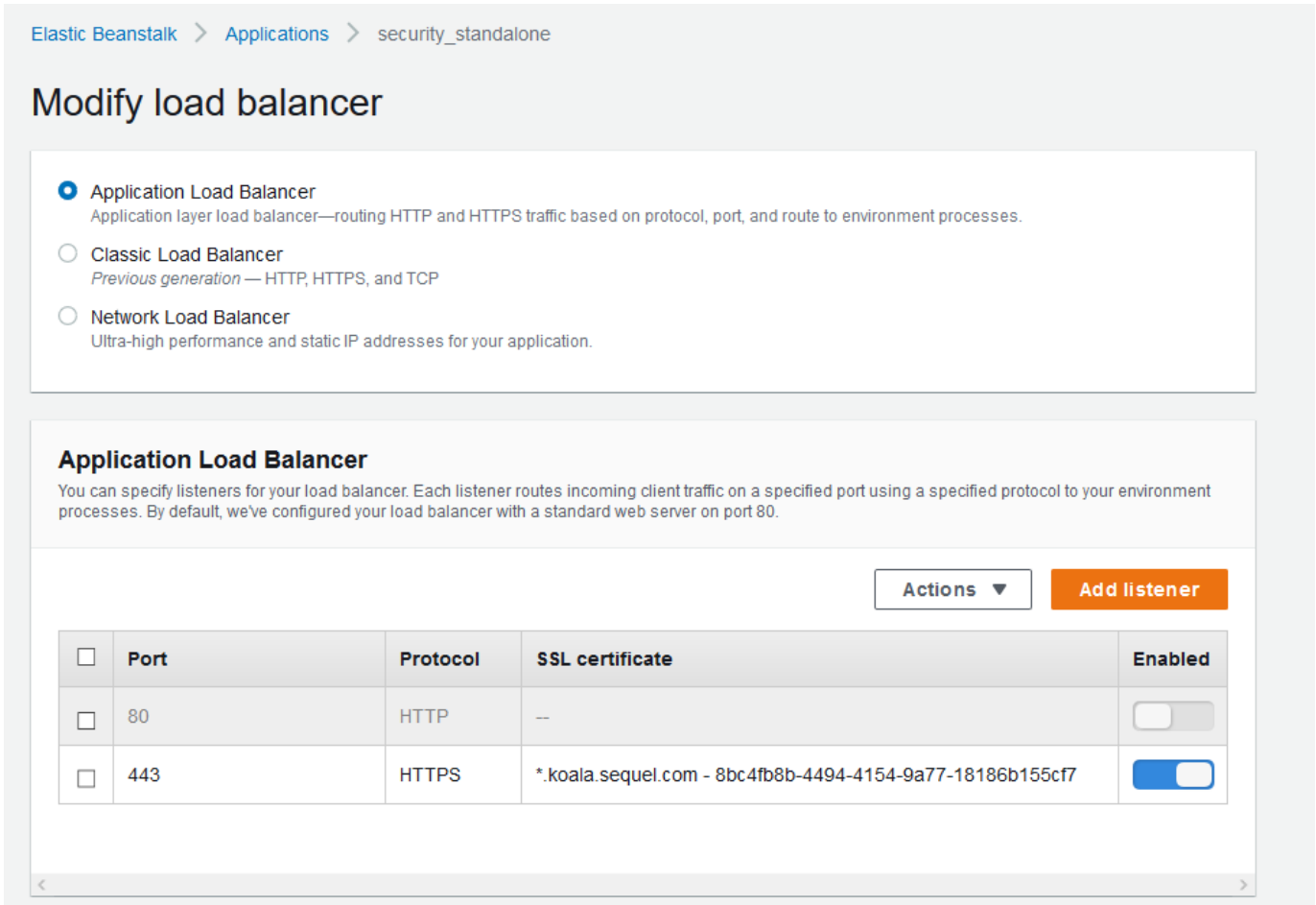


We would need to click in "**Configure more options**" in order to edit advanced options.

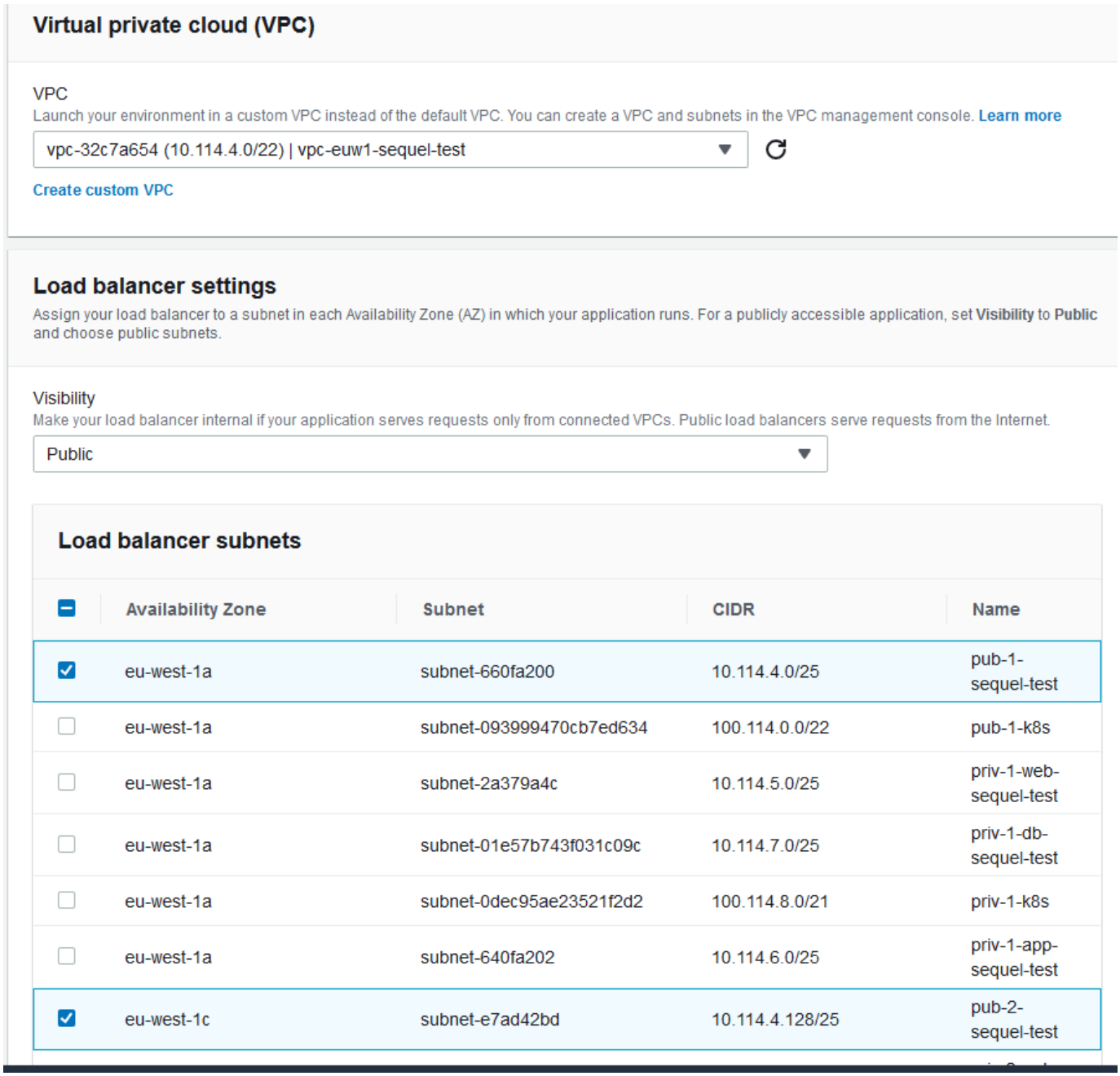
In order to configure a Load Balancer, we are going to edit the "**Capacity**" with Load Balanced Environment type with 1 Min 1 Max instance:

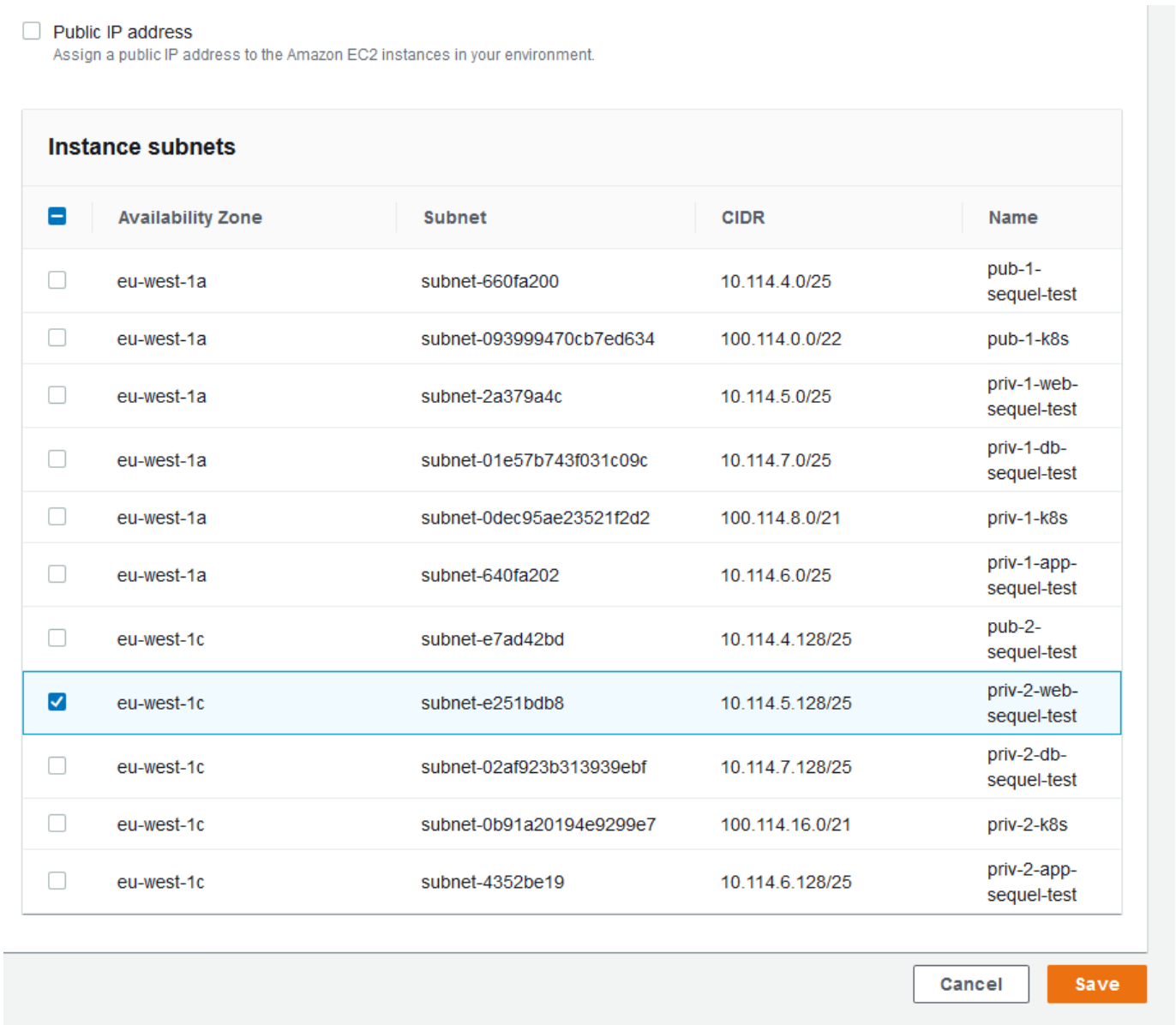


Then we will edit the APL (Application Load Balancer) to listening in the port 443:

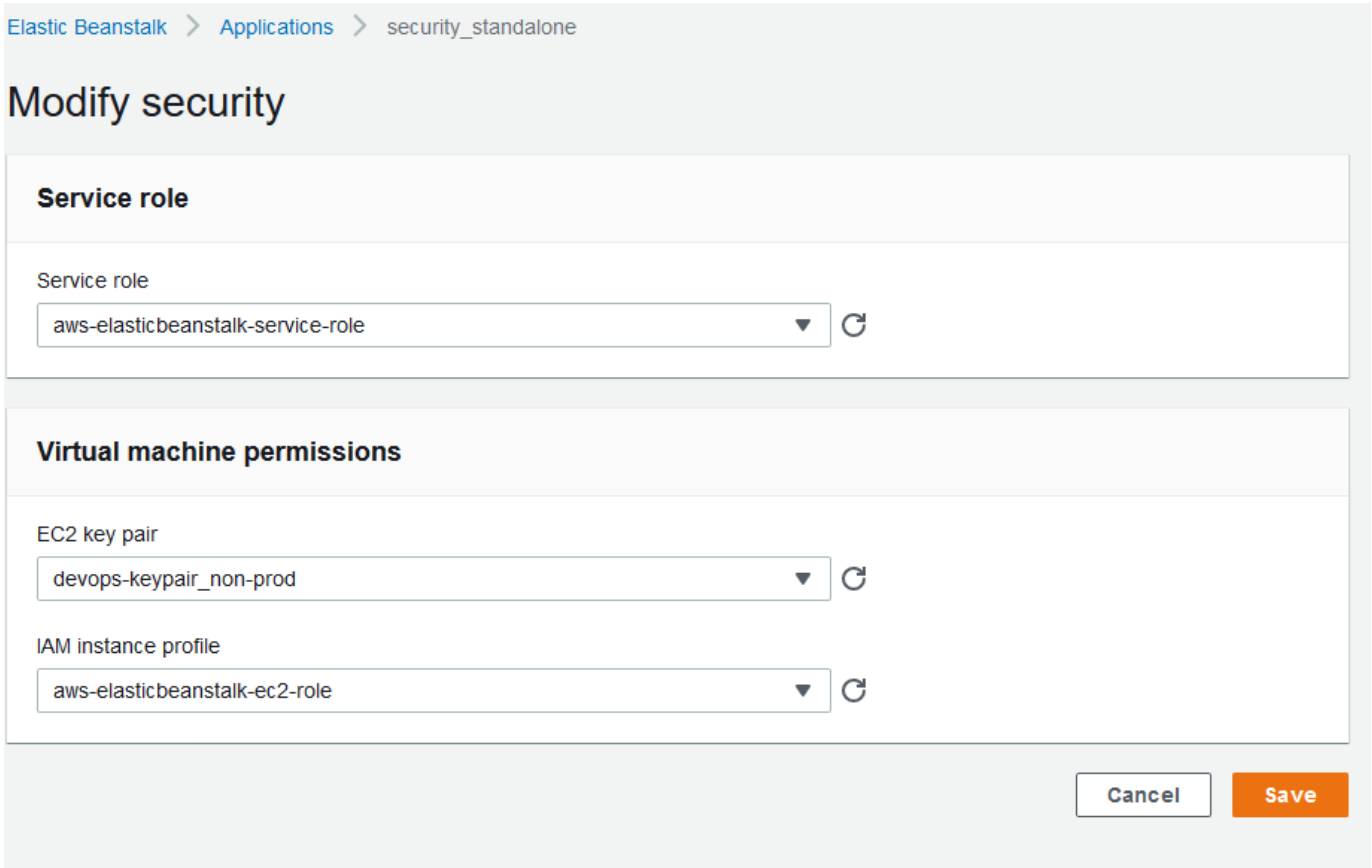


Edit the Network in order to select the correct VPC and the subnets for the ALB and the instance or intances:

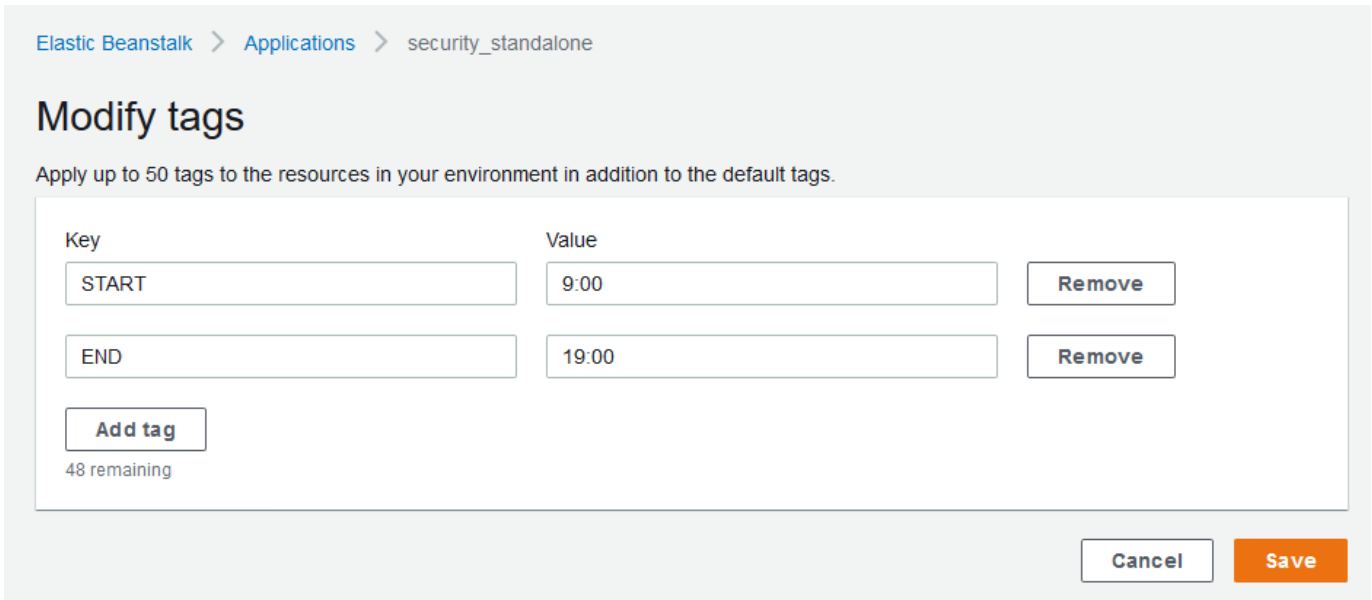




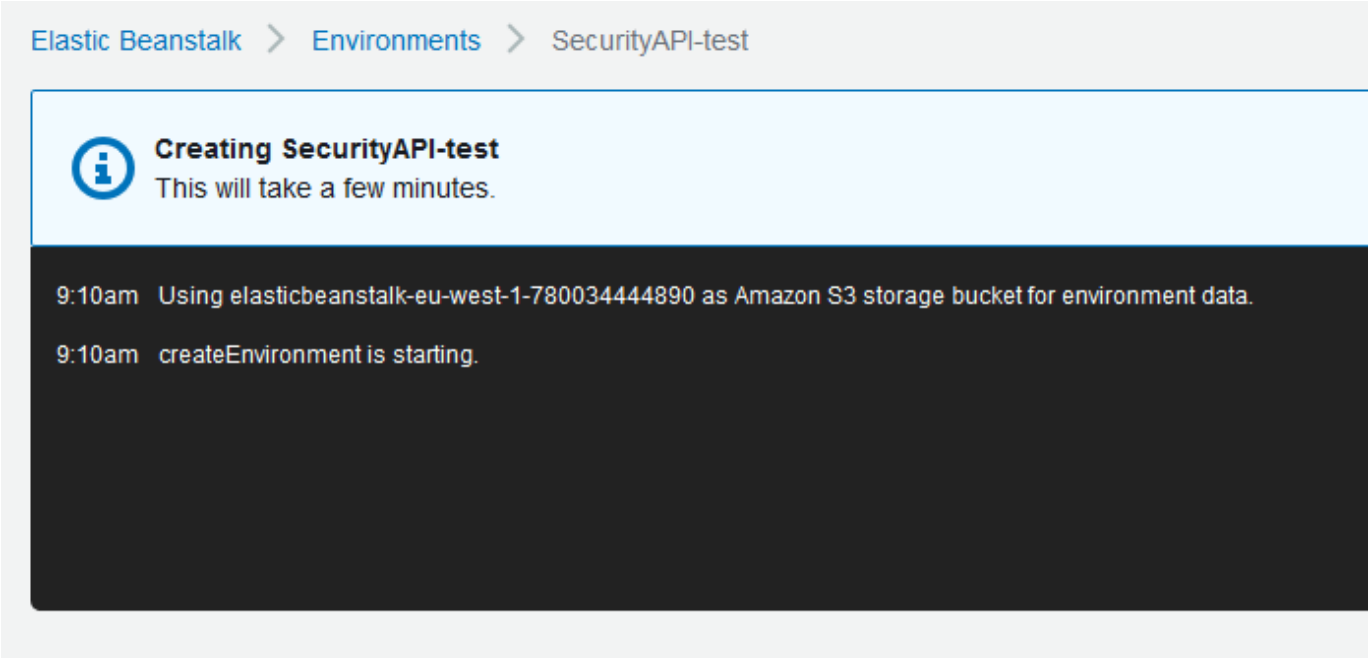
Now we need to edit the Security in order to select the key par we want to use for the instances:



And is important to add Tags START and END for the instances:



We can now create the environment.

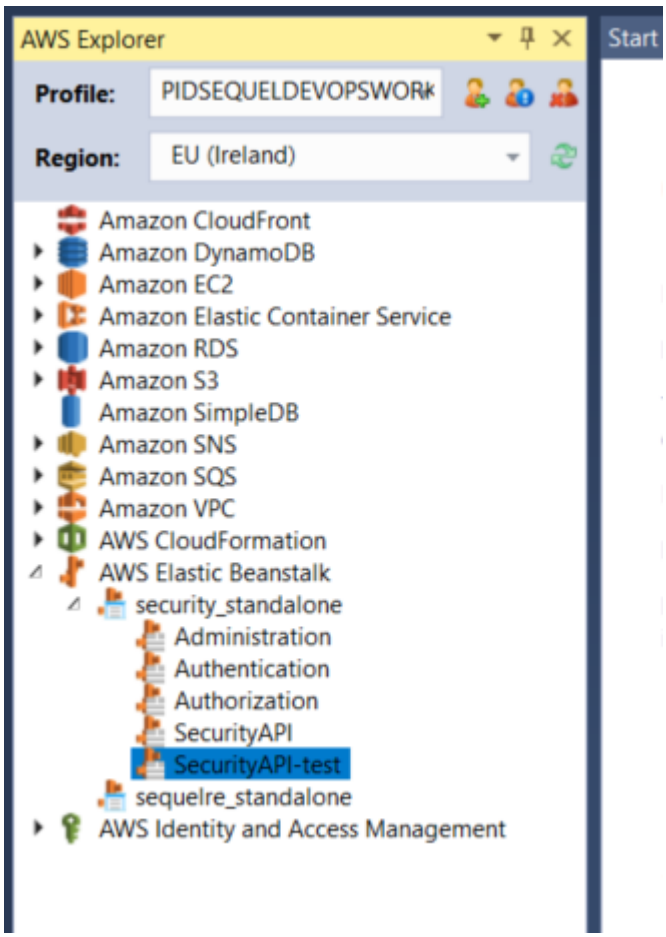


DEPLOYING .NET SOLUTIONS

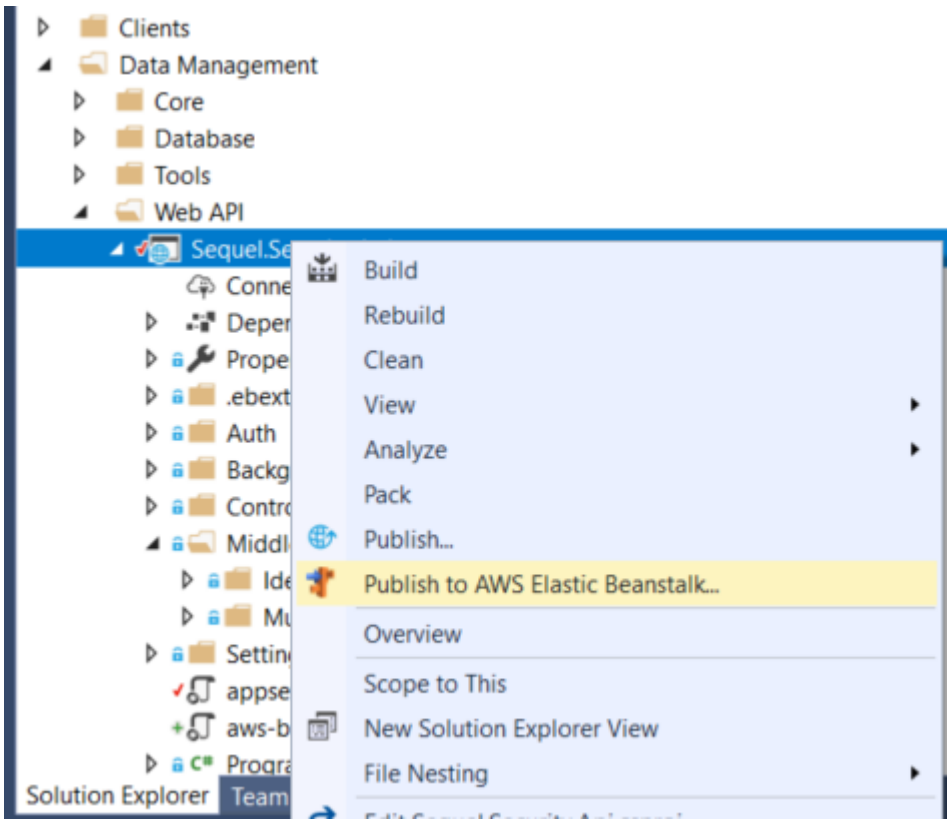
There are different ways to deploy the applications in "**Elastic Beanstalk**". In this example we will use "**Visual Studio 2017**" with the "**AWS Toolkit for Visual Studio 2017 and 2019**".

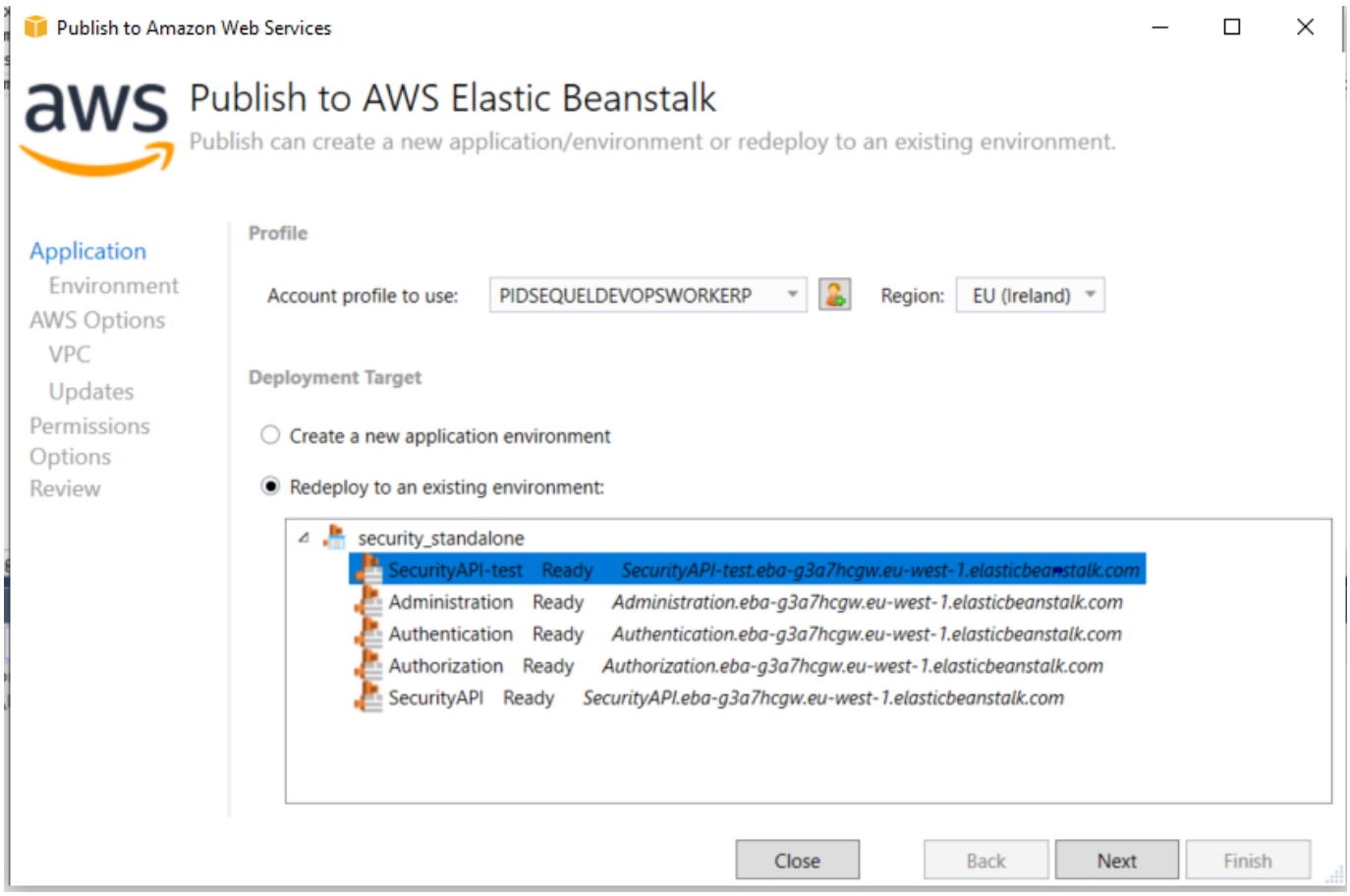
Please download and install AWS Toolkit for Visual Studio https://aws.amazon.com/es/visualstudio/

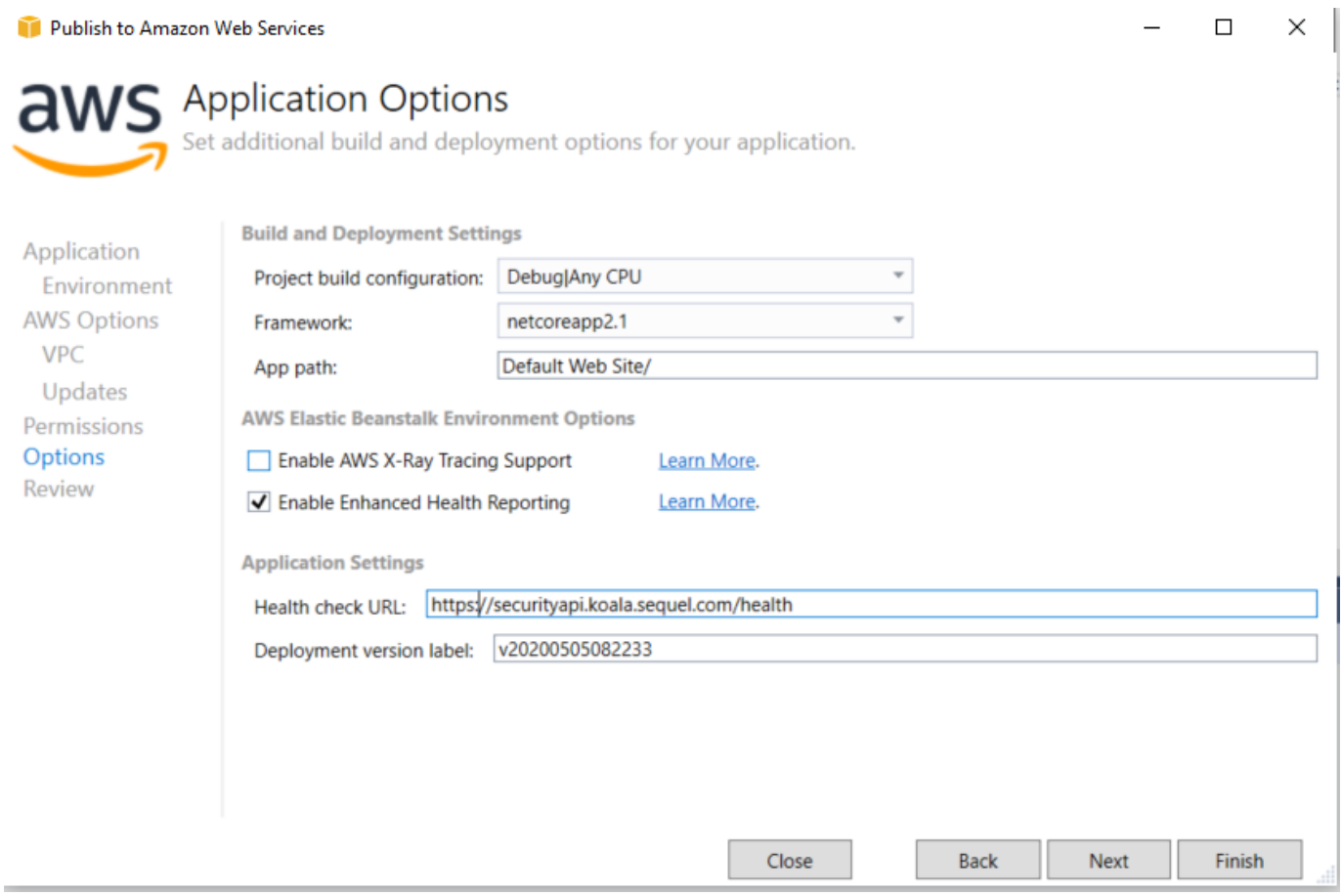
Once installed, you can log in to AWS using your credentials and will see the AWS Explorer:

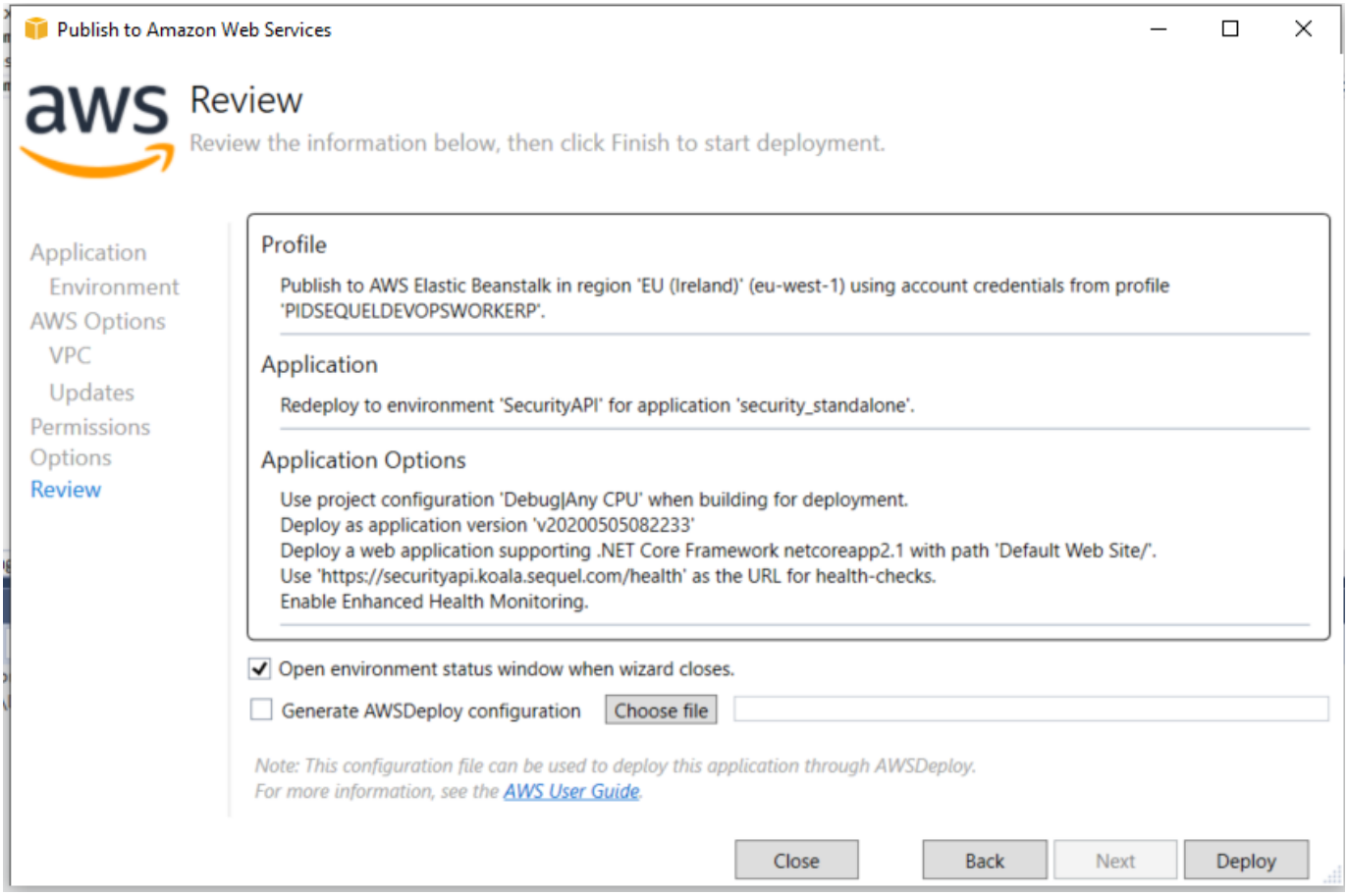


We can now open the solution for Security. We will have to edit the "**appsettings.json**" for each application using the right variables. You can right click on the Application project and select "**Publish to AWS Elastic Beanstalk**":





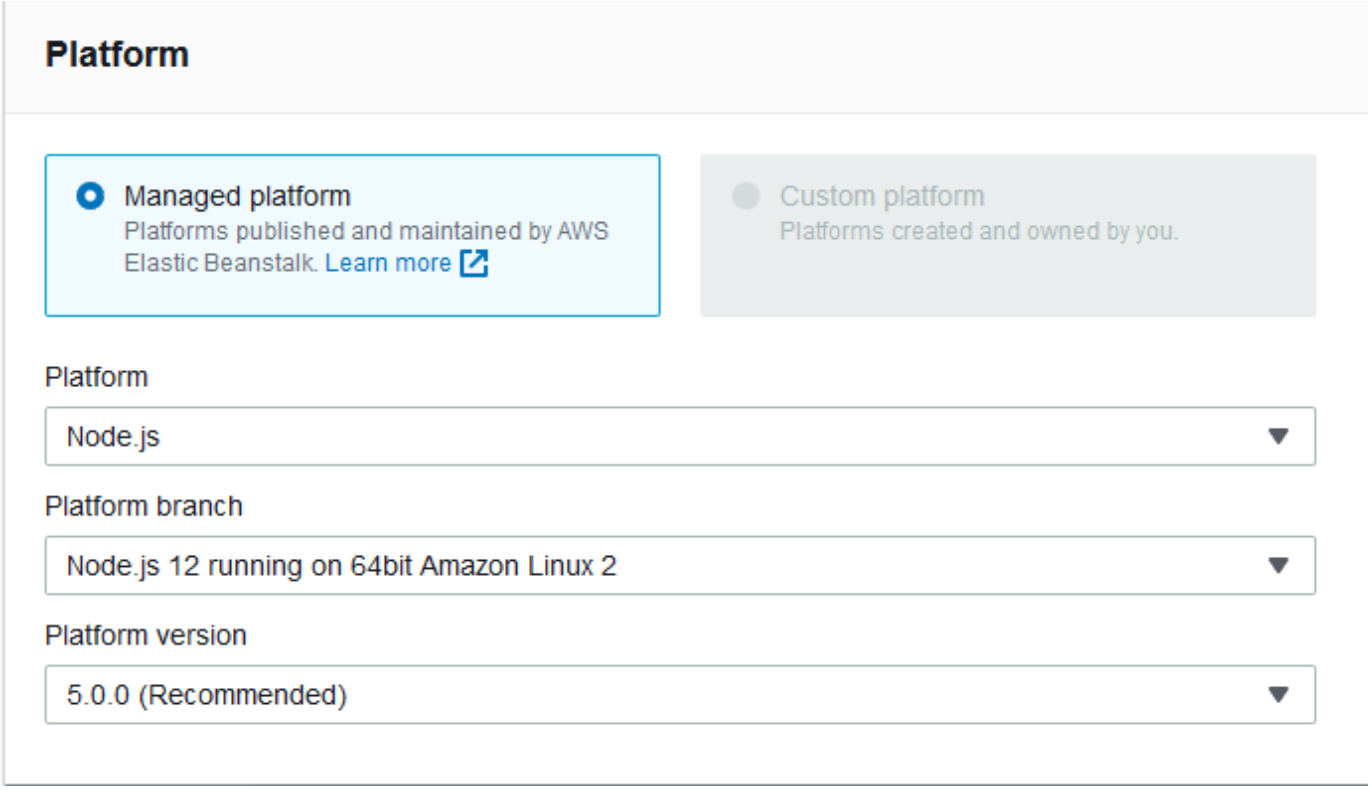




Now the application will be published in our Environment in Elastic Beanstalk.

DEPLOYING ADMINISTRATION

In order to publish this, we will need to create a Node.js Environment in Elastic Beanstalk (instead of .NET).



We will run the "**npm run build**" command manually from Powershell and publish the output. Once the project has been built, we will add the "**appsettings.json**" inside:

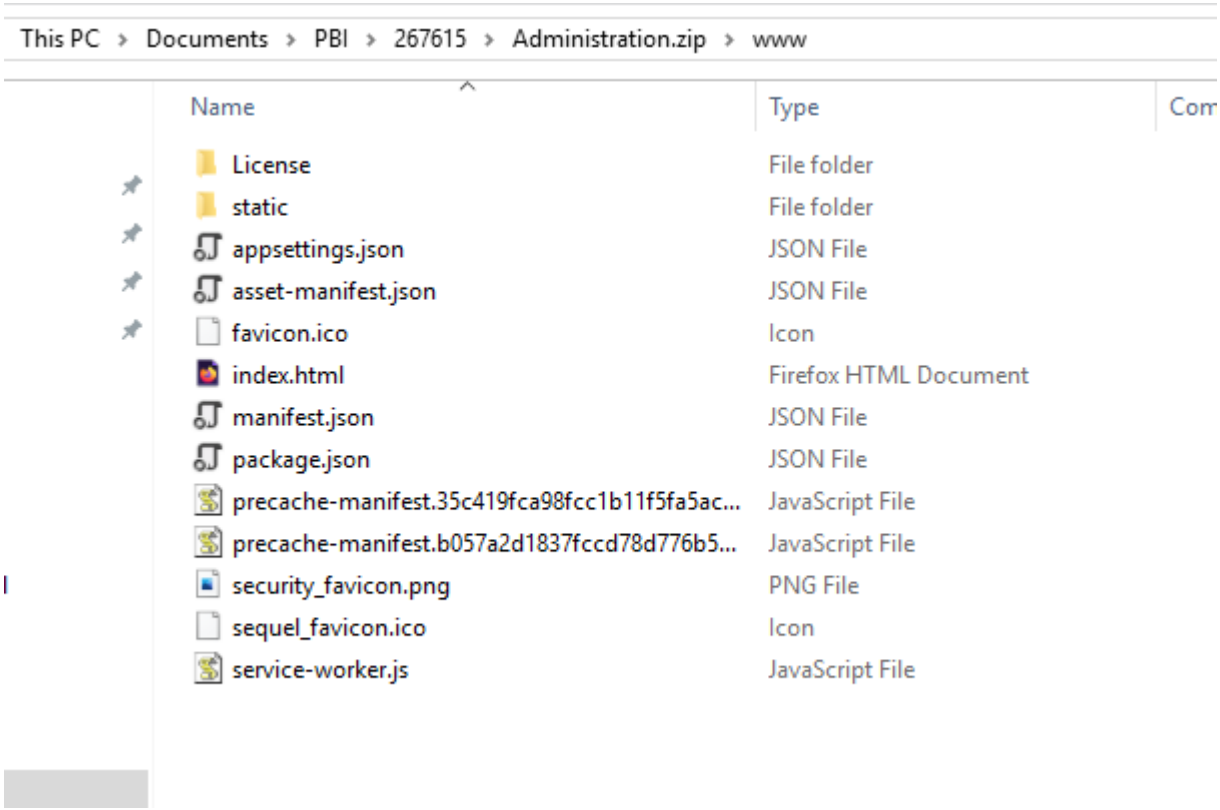
```
{
  "api": "https://securityapi.koala.sequel.com/",
  "auth": "https://authentication.koala.sequel.com/",
  "version": "elasticbeanstalk",
  "authenticationFlow": "authorizationCode",
  "router": "hash",
  "allowHttp": "True"
}
```

We will create a zip file with the following structure for Administration:

Administration.zip

```
- package.json
- server.js
-- www(folder)
```

Remove "**@sequel/sequel.web.usersessionavatar**" module from package.json Put all the build output inside the "**www**" folder:



Example of "**server.js**":

```
// server.js: This is a tiny NodeJS web server hosting static files from the /www folder in the Elastic Beanstalk deployment ZIP
// Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.
var express = require('express');
// Let's create an instance of an express web server
var app = express();
// By default, let's use port 80, unless we provide a different value as argument or system environment variable
var port = process.env.PORT || process.argv[2] || 80;
// Let's host all the static files in /www as root of our little web server
app.use('/', express.static(__dirname + '/www'));
// Start listening on the desired port for incoming traffic
var server = app.listen(port, function () {
    console.log('listening on port:', port);
});
```

NOTE: In the "**.env**" file you will have to change the path for "**PUBLIC_URL**" from "**/Administration**" to "**/**"

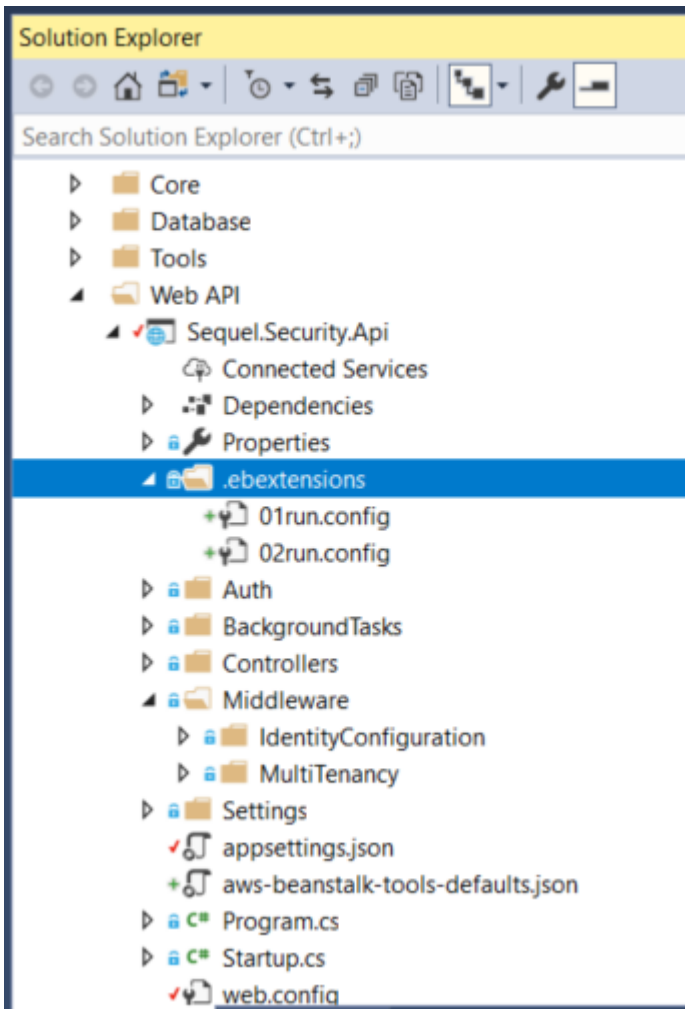
With the zip file created, just publish it from the Elastic Beanstalk console inside the Environment Menu.

DEPLOYING SECURITY-SYNC SERVICE

In order to deploy "**security-sync**", we will going to add some instructions in .config files using YAML or JSON formats (in this example we use YAML). We will add these instructions inside a folder called "**.ebextensions**".

See url for more details: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/ebextensions.html

In this deploy we added the "**.ebextensions**" folder inside Security API project:



First you will have to upload the "**SecurityInstaller**" with DeploymentManager inside to a S3 bucket named "**elasticbeanstalk-eu-west-**" like for example "**elasticbeanstalk-eu-west-1-780034444890**". This is import because the AIM role used for the instances has some policies limitations. The bucket is created by Elastic Beanstalk with a random number already.

The instance created by Elastic Beanstalk does not have the AWS tools installed, so you will have to install it manually inside the instance (Remote Desktop) or create a config file with the instructions to download and install the tools:

- 01run.config

```
files:
  "C:\\temp\\AWSCLI64PY3.msi":
    source: "https://s3.amazonaws.com/aws-cli/AWSCLI64PY3.msi"
commands:
  install_awscli:
    command: msiexec.exe /qn /i "C:\\temp\\AWSCLI64PY3.msi"
```

- 02run.config

Then we create the .config file with the instructions to download the SecurityInstaller for the S3 bucket and install the service inside the instance.

```
Resources:
  AWSEBAutoScalingGroup:
    Metadata:
      AWS::CloudFormation::Authentication:
        S3Access:
          type: S3
          roleName: aws-elasticbeanstalk-ec2-role
          buckets: elasticbeanstalk-eu-west-1-780034444890
sources:
  C:\\temp\\: "https://elasticbeanstalk-eu-west-1-780034444890.s3-eu-west-1.amazonaws.com/SecurityInstallerKoala.zip"
container_commands:
  00_fix_perms:
    command: cacls C:\\temp\\ /t /e /g Everyone:f
  01_install_sync_service:
    command: C:\\temp\\SecurityInstallerKoala\\Sequel.Deployment.Manager.exe -m SyncMetadata.xml -s environmentConfig.json -p SecuritySync
```

NOTE: Remember to create the Legacy database in the RDS database server before deploying security-sync service

4.8.4 Saas

Security SaaS

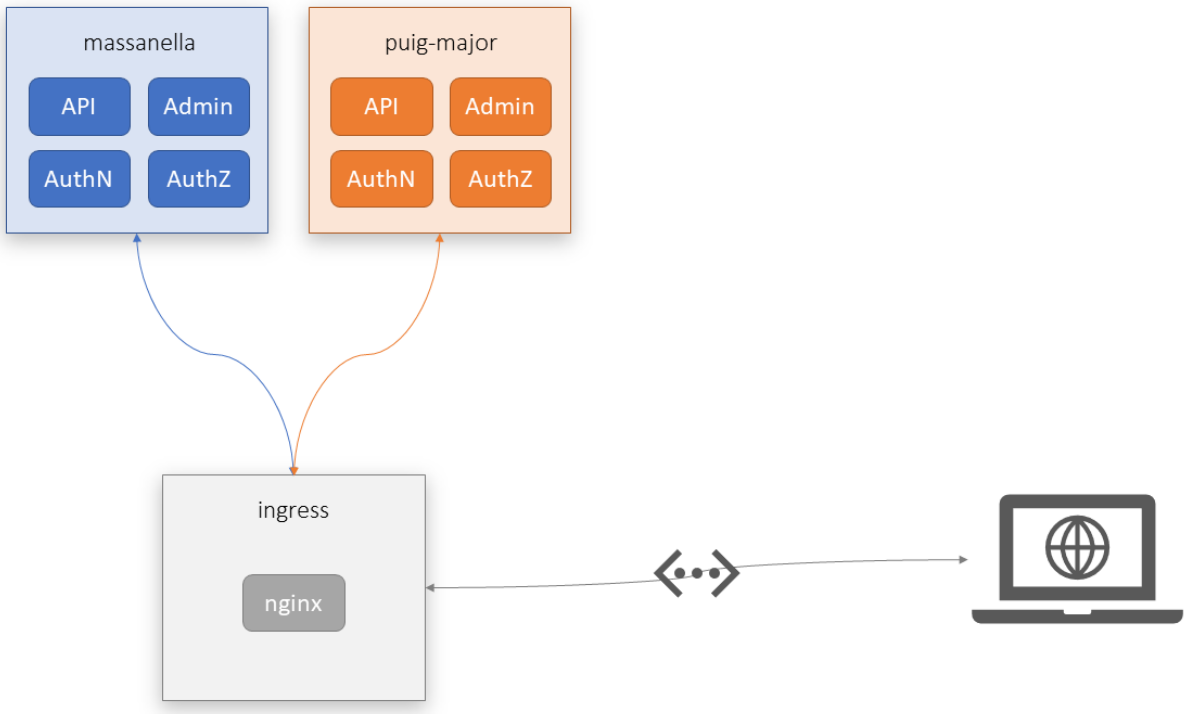
A new way to deploy Security FVMs, but faster and in containers

> **User Manual**
>
> If you are looking for the User Manual, there is a deditated page for this.

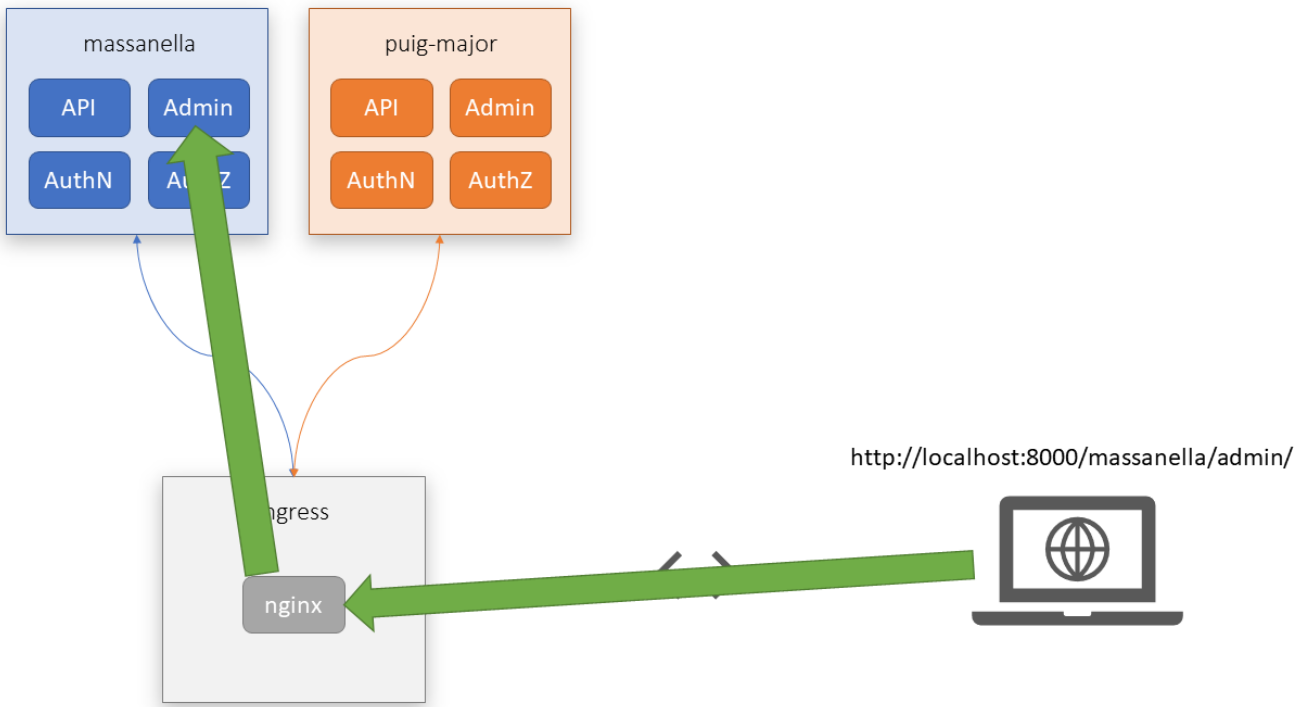
HOW IT WORKS

All instances run in a Kubernetes cluster and data is stored in an external SQL Server VM. Rabbit MQ, instead, is not managed by the platform, and one must be provided at creation. To interact with the platform, an API is available to run basic operations on it.

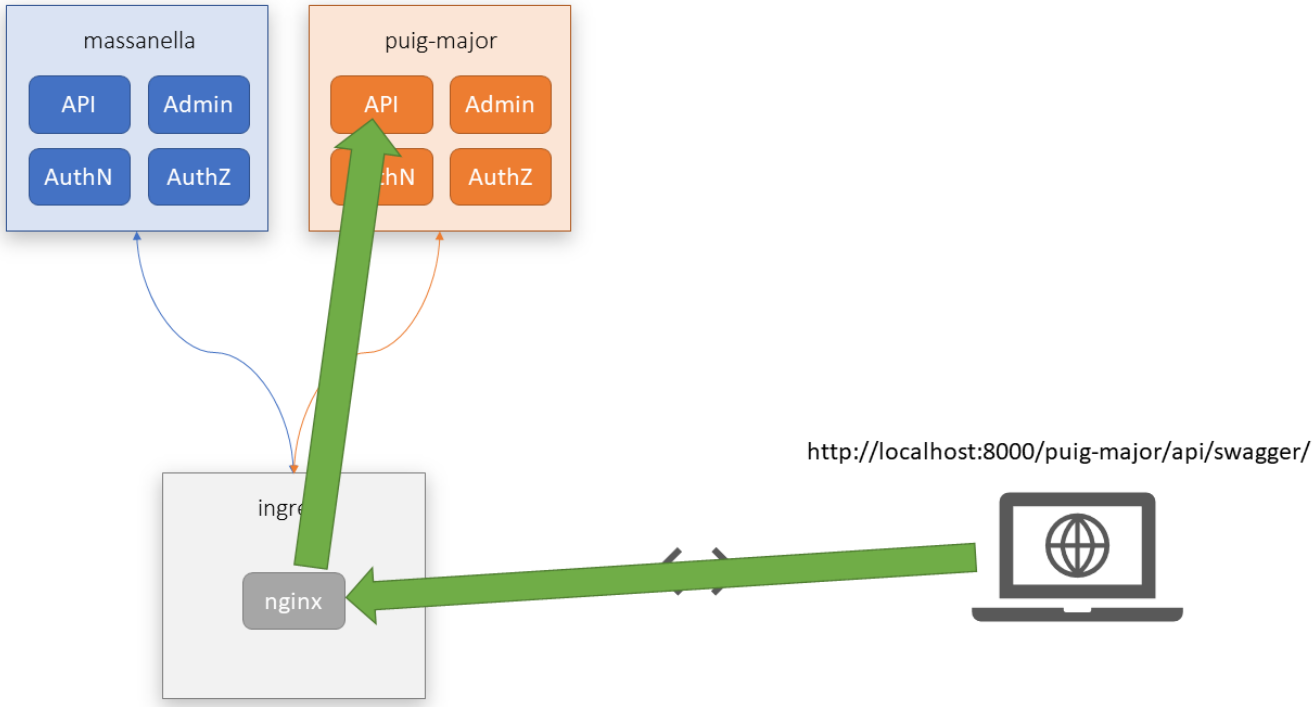
The API takes care of deploying and destroying the instances when a request is received, as well as to create and destroy the database for the instance. *Up*, *Down*, and *Upgrade* operations are managed through ansible playboks (kind-of scripts) and Helm charts (package manager for Kubernetes), and they runin the background when an operation is requested to run. Each operation is described in Operations section. Below there is a video that explains the *create new instance* operation:



A deployed instance will have 4 services that corresponds to **API**, **Authentication**, **Authorization** and **Administration** which can only be accessed from inside the cluster. As shown in the above diagram, in which two instances called *massanella* and *puig-major* have been deployed. A reverse proxy (represented above as *ingress*) is used in front of the cluster to carry the external requests to the right container for the requested URL.



For example, a user requests access to `https://identity.office.sbs/massanella/admin`. This request will be received by the *ingress controller*, then it will break down the URL to extract `massanella` as instance and `admin` as the service to connect to. After that, it will deliver the request to the container (in this case the **Adminstration** one). The container will produce a response and it will deliver it to the *ingress controller* container, which it will then redirect to the user that made the request. This works for other services and instances by only changing corresponding part of the URL.



In this other example, `https://identity.office.sbs/puig-major/api/swagger` will send the request to the instance `puig-major` and `api` (Security API) container (for that instance), and deliver the response to the user.

OPERATIONS

Up (creating an instance)

Behind the scenes, create a new instance consists basically in these steps:

1. Create the instance database in SQL Server
2. Create a user for that database
3. Publish DACPACs (database schemas) to the database
4. Create the default tenant in Multitenancy database
5. Fill Security tables with Security Vanilla configuration
6. Create the *admin* user
7. (*optional*) Fill Security tables with custom Security configurations (like Workflow, Claims, Origin...)
8. Install Helm chart with specific parameters for the new instance (aka deploy to the Kubernetes cluster)

Down (destroying an instance)

When a destroy request is received, it will do the following steps:

1. Uninstall Helm release (aka remove deployment from the Kubernetes cluster)
2. Drop instance database from SQL Server
3. Drop instance user from SQL Server

Upgrade

This operation can upgrade an instance to a new version, or update the configuration of it (or both). Whatever parameter is changed, the steps it will follow are:

1. Publish DACPACs (database schemas) to the database - this will update the existing database to the latest schema
2. Fill Security tables with (updated) Security Vanilla configuration
3. (*optional*) Fill Security tables with (updated) custom Security configurations
4. Upgrade the Helm chart release with the instance parameters (aka update deployment on the Kubernetes cluster)

Housekeeping: Upgrade All

This operation will upgrade a bunch of instances which has no fixed version (aka *use always latest version*). Internally calls the Upgrade operation for each instance to upgrade.

Housekeeping: Delete expired

Instances can have an expiration date. When this date is reached, this operation will destroy all these expired instances using the Down operation for each.

Get status

This operation allows a user to get information about an instance itself. When information is requested, it will try to read:

- The deployment status
- The public/external URLs for each Security service
- (*optional*) The logs for each container (`dbo.Logs` logs are not included here)
- (*optional*) The health checks for each service

Publish new version in platform

Just run the following SQL in the database:

```
insert into [sec-saas].SecurityVersion (Version, ImagesTag, HelmChartVersion, PublishedDate) values
('<version>', '<tag>', '<helm-version>', 'YYYY-MM-DD HH:MM:SS.0000000')
```

> **Our SaaS Env**
>
> The SQL Server is `SA-K8S-MSSQL\MSSQL2017` and the database is `saas`

HOW TO GET THE HELM CHART VERSION OF A RELEASE

Go to `Source/Deployment/helm/security` and open the file called Chart.yaml. The property `version` in the yaml file indicates the Helm Chart version.

HOW TO GET THE VERSION OF A RELEASE

Just grab the version from the *Master Build* but without the `.stable` part: `1.61.20275.01.stable` becomes `1.61.20275.01`.

HOW TO GET THE PUBLISHED DATE

Just check when the *Master Build* run and put something near that :)

HOW TO GET THE IMAGES TAG

In the *Master Build*, look for the **Push services** step and open the logs of it. In the first lines, should be one that starts with `[command]` (maybe around line 11). In that line, go to the end of it and you will see the tag of the image:

```
...
de689dcf-b051-4653-8605-04c5748d50d6 exists true
e36e269d-f3ae-4dcd-8ee5-c5cdec764318 exists true
[command]"C:\Program Files\Docker\docker.exe" -H tcp://tfsdockerl.office.sbs:2376 --tls --tlscacert='.dockercerts\ca.pem' --tlscert='.dockercerts\cert.pem' --
tlskey='.dockercerts\key.pem' push docker.sequel.com/security/admin:1.61.20273.01.stable
The push refers to repository [docker.sequel.com/security/admin]
d7d8b0a05014: Preparing
c6151278908b: Preparing
...
```

The tag from above example is `1.61.20273.01.stable` (the text after the `:`).

This can also be seen near the last lines of the log:

```
...
f9e2dd402711: Pushed
702be494be68: Pushed
1.61.20273.01.stable: digest: sha256:0566064d6c6644c5cc77c8bda4a4afb285bd5a46b0f41478adb8e98848c86357 size: 3463
##[section]Finishing: Push services
```

Troubleshooting

HOW TO ACCESS THE INSTANCE DATABASE

The data is stored at `SA-K8S-MSSQL.office.sbs\\MSSQL2017` database. Look for the database for your instance, which will look like `saas-instanceName`. The database contains the Multitenancy tables, Security Database tables and Logging tables.

HOW TO READ THE LOGS

The logs are stored alongside the Security tables, in the same database. They can be found at `dbo.Logs` table. See How to access the instance database.

HOW TO READ THE LOGS WHICH ARE OUTSIDE LOGGING

In general, logs are stored in the Logging table `dbo.Logs` which can be found in the same database as the instance Security. But there are logs that are important and they are not stored in that table (Authentication has some important errors logged outside Logging).

These logs can be retrieved using the `GET /api/instance/{instanceName}` endpoint with the parameter `includeLogs` set to `true`. With the response, it have included the more recent logs for each container running for the instance. The object `podLogs` in the response is a dictionary/map where the key is the name of the pod (which will look like `instanceName-api-dn1283jd-fd828a42`) and the value the logs. Logs are sorted by time ascending, so the last line is the more recent log.

> **Authentication errors**
>
> There are some errors related to Authentication like `invalid_client` or `invalid_redirect_uri` which the reason of failure can be found in these logs. It is recommended to grab those logs quickly and inspect them properly, because they usually appear between other kind of logs.

HOW TO SEE THE HEALTH CHECKS

The easiest way is to go to the API, send a request to `GET /api/instance/{instanceName}` with the parameters `includeHealthChecks` set to `true` and it will return them in the response. You may also issue the health check request directly to the Security services manually.

SOMETHING IN SECURITY IS NOT WORKING PROPERLY

First check the logs to see if there is something wrong. Also check out the health checks to see if there is something not working properly.

Currently, the only thing that may not work and can be fixed by users is the RabbitMQ connection.

For other kind of errors that you cannot fix, please contact Supporting Apps.

> **FQDN domains**
>
> It is recommended to use FQDN in the RabbitMQ URL to avoid any kind type of connection error.

I'M GETTING INVALID_CLIENT OR INVALID_REDIRECT_URI

Check out the Authentication logs using the method described in How to read the logs which are outside Logging. The cause of the error is usually on this logs.

A DEPLOYMENT IS FAILING

If you are using a custom configuration, check that is valid.

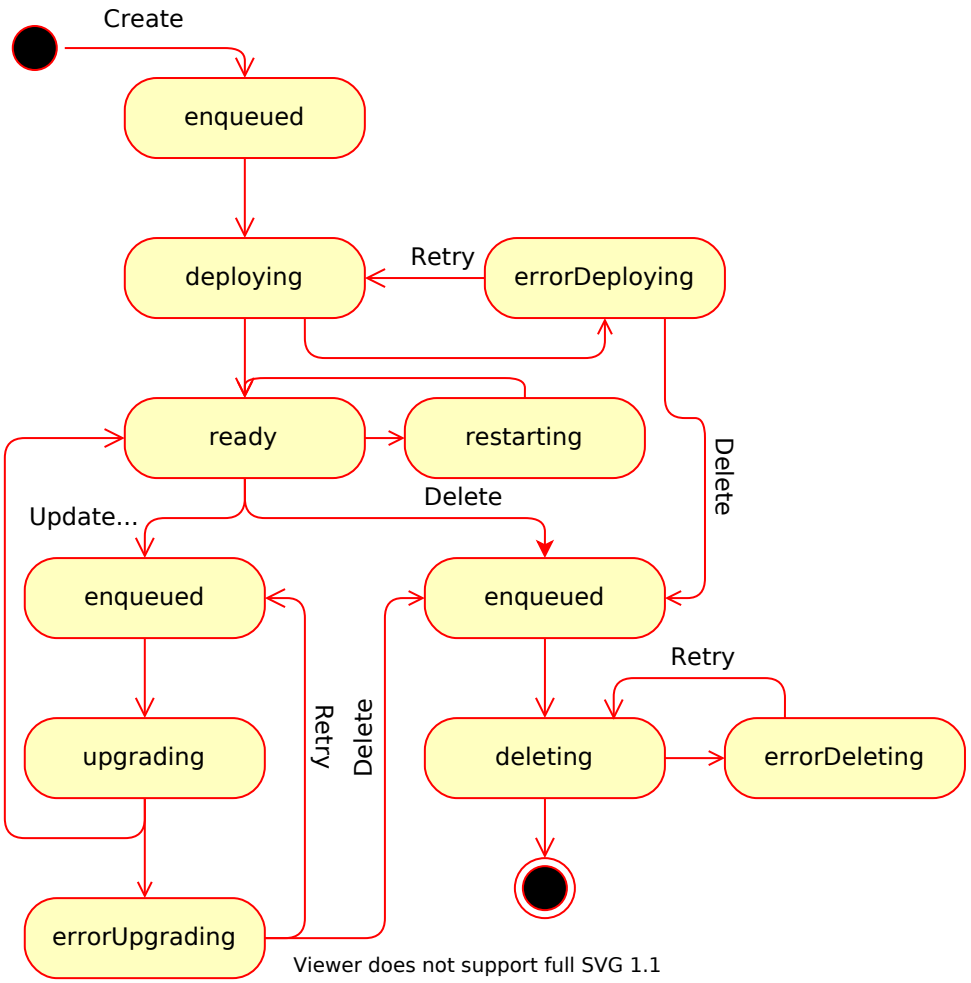
For any other cases, please contact Supporting Apps.

User Manual

The center for all operations in the Security SaaS platform is the SaaS API. You can find it at https://identity.office.sbs/saas/. The API has a Swagger open for everyone to use it easily, it can be found at https://identity.office.sbs/saas/swagger/. Below there is a list of the endpoints linked to sections of this manual:

- `POST /api/instance`
  - Creating an instance
  - Fill Security Database with custom configuration
- `GET /api/instance/{instanceName}`
  - Getting information about an instance
- `PUT /api/instance/{instanceName}`
  - Upgrading an instance
  - Updating settings for an instance
  - Fill Security Database with custom configuration
- `DELETE /api/instance/{instanceName}`
  - Deleting an instance
- `POST /api/instance/{instanceName}/restart`
  - Restarting an instance
- `GET /api/security-version`
  - Getting available Security versions
- `GET /api/security-version/latest`
  - Getting available Security versions
- `GET /api/security-version/{version}`
  - Getting available Security versions
- Housekeeping
  - Housekeeping: expired instances
  - Housekeeping: upgrade latest

INSTANCE STATES



The above image describes all possible states an instance can be, and their transitions. If a transition has a text, then means that is an action from an API call. The rest of transitions are internal of the API service.

CREATING AN INSTANCE

Creating an instance of Security is really simple and requires a few parameters to be filled. Each instance has a unique name which is assigned when creating it, and its up to you. As you may already know, the platform does not provide any RabbitMQ server, so this must be provided at creation. There are other parameters that can be filled if needed. Below there is an explanation of the parameters:

```
{
  "instanceName": "string",
  "daysToExpiration": 0,
  "version": "string",
  "rabbitMq": {
    "url": "rabbitmq://sbsmpormqsmt.office.sbs/Yeah",
    "user": "string",
    "password": "string"
  },
  "securityCustomConfigurations": [
    {
      "appKey": "ORI",
      "fileName": "vanilla-wf.zip",
      "base64Data": "string"
    }
  ],
  "securityVanillaConfiguration": {
    "adminEmail": "user@example.com",
    "adminPassword": "string",
    "e2EConfig": true
```

```
  }
}
```

- First, the `instanceName` parameter. Is the name that will receive the instance (and must be unique). This name will alsob be used for the database name and the public URLs. It is recommended to use a naming convention that clearly identify who is using them (like `imp29210` or `wf28571`). There are some requirements for the names that must be followed:
  - The name must be a DNS-compilant name (see this StackOverflow answer), that is:
    - Maximum 63 characters
    - Use only alphanumeric characters, but slashes can be used in the middle
    - Case insensitive (for that, the name **must be** lowercase)
    - It is recommended to use names with less than 40 characters to avoid errors after deployment
- `daysToExpiration` is optional, but if specified, then the instance will have an expiration date. When that day is reached, the instance will be deleted. The minimum value (if defined) is 1 day. See Housekeeping: expired instances to know more about this.
- `version` is also optional, but if specified, the instance will use this specficic version (see Getting available Security versions). By default uses the latest version available, and also marks the instance to be upgraded automatically when a new version is available (**recommended option**).
- `rabbitMq` options are **mandatory**, as mentioned before. It must have the URL to the server and virtual host `rabbitMq.url`, the user in `rabbitMq.user` and the password `rabbitMq.password` (which must be encrypted using `Sequel.Core.Cryptography` method). The API will validate the URL and password but won't check connectivity.
- `securityCustomConfigurations` defines custom Security configurations to deploy alongside the instance. It is not required to define configurations, but it can be useful to be able to use Security with other products. See Fill Security Database with custom configuration to know more about.
- `securityVanillaConfiguration` allows some customization around the Security vanilla configuration:
  - `securityVanillaConfiguration.adminEmail`: Defines a custom email address for the admin user. By default uses `{instanceName}@example.com`. Highly recommended to change that value.
  - `securityVanillaConfiguration.adminPassword`: Defines a custom password for the admin user. By default uses `Password123!`.
  - `e2EConfig`: If set to `true`, it will deploy the E2E Security configuration. It will not deploy it by default. This is not needed in most deployments.

When the request is done, and passes all validations, then the API will enqueue the request to be deployed as soon as possible. This will put the instance to `enqueued`. Use the `GET` instance endpoint to see the status of the deployment. If the deployment succeeds, it will change to `ready`. In case of error, it will go to `errorCreating`. The deployment will take some time (around 3 minutes), and while deploying, the state will be `deploying`.

If the deployment fails, it can be retried by sending the same request - including the `instanceName`, which must be the same. The output of the Ansible playbook (the "scripts" which deploys the instance) is not accessible through the API, but it is stored in database. See troubleshooting.

Below there are some examples of requests to create instances:

Simple

```
{
  "instanceName": "example1",
  "rabbitMq": {
    "url": "rabbitmq://rabbitmq.office.sbs/Example",
    "user": "example-user",
    "password": "EAAAAFqCRyzli+rtgXNGGv6xvxxkyh67Upr8xAR3e3SDaOFp"
  },
  "securityVanillaConfiguration": {
    "adminEmail": "email@sequel.com"
  }
}
```

With expiration

```
{
  "instanceName": "example2",
  "daysToExpiration": 1,
  "rabbitMq": {
    "url": "rabbitmq://rabbitmq.office.sbs/Example",
    "user": "example-user",
    "password": "EAAAAFqCRyzli+rtgXNGGv6xvxxkyh67Upr8xAR3e3SDaOFp"
  },
  "securityVanillaConfiguration": {
    "adminEmail": "email@sequel.com"
  }
}
```

With specific version

```
{
  "instanceName": "example3",
  "version": "1.47.20093.01",
  "rabbitMq": {
    "url": "rabbitmq://rabbitmq.office.sbs/Example",
    "user": "example-user",
    "password": "EAAAAFqCRyzli+rtgXNGGv6xvxxkyh67Upr8xAR3e3SDaOFp"
  },
  "securityVanillaConfiguration": {
    "adminEmail": "email@sequel.com"
  }
}
```

With custom configuration

```
{
  "instanceName": "example3",
  "version": "1.47.20093.01",
  "rabbitMq": {
    "url": "rabbitmq://rabbitmq.office.sbs/Example",
    "user": "example-user",
    "password": "EAAAAFqCRyzli+rtgXNGGv6xvxxkyh67Upr8xAR3e3SDaOFp"
  },
  "securityCustomConfigurations": [
    {
      "appKey": "WF",
      "fileName": "wf.zip",
      "base64Data": "..."
    },
    {
      "appKey": "PB",
      "fileName": "pb.zip",
      "base64Data": "..."
    },
    {
      "appKey": "CLM",
      "fileName": "claims.zip",
      "base64Data": "..."
    }
  ],
  "securityVanillaConfiguration": {
    "adminEmail": "email@sequel.com",
    "e2EConfig": true
  }
}
```

GETTING INFORMATION ABOUT AN INSTANCE

After the create request is completed successfully, the instance will be registered in the platform. Since that moment, the status and some information of the instance can be retrieved using the `GET` endpoint. If the instance is in `ready` state, it will show more information. Swagger has descriptions in the Schema that explains the fields. Below it there is some explainations for some options, to retrieve more information about the instance, for completness (only for `ready` state).

By default, when the instance is `ready` state, it will show a list of URLs for Security API, Authentication, Authorization and Administration, which are the public URLs.

If the option `includeHealthChecks` is set to `true`, then the response will include the contents of each health check response for the services. API, Authentication and Authorization will show the full report in json, but Administration will simply put `"health"`. If one of the healt checks cannot be obtained, a text with the description will be put instead.

As in Security there are some logs that are still written into `stdout`, this endpoint allows you to read these logs easily with the option `includeLogs`. This will list logs for all pods (containers - services and their replicas). By default will return up to 1000 lines for each pod, but this can be reduced (or extended) with the `tailLines` parameter. Lines are sorted by time ascending (the last line is the more recent log entry).

UPGRADING AN INSTANCE

> **Update settings or configuration**
>
> If what you look is to update instance settings (RabbitMQ settings) or Security configurations, go to the next section...

> **Warning**
>
> This section is not recommended for instances that is marked to use *always the latest version*. The instances will be upgraded as soon as a new verision is available.

To upgrade an instance, call the `PUT` endpoint with a new version in the body and wait until upgrade. The version must be valid (see Getting available Security versions). A call to this endpoint will schedule the upgrade request to run soon and move the instance to `enqueued` state.

To new specific version

```
{
  "version": "1.47.20093.01"
}
```

To latest version

```
{
  "version": "latest"
}
```

While upgrading an instance, the services could not work properly, the instance will be in `upgrading` state. Be patient and wait until the upgrade is finished. If the upgrade fails, it will go to `errorUpgrading` state. See Troubleshooting for more information. In error state, you can retry the upgrade using the same parameters.

The upgrade operation can be done with settings and/or configuration update.

> **Downgrade**
>
> Downgrade is not supported, once an instance is upgraded, cannot be downgraded to an old version.

UPDATING SETTINGS FOR AN INSTANCE

To update settings, or Security configurations (or both), the same `PUT` endpoint can be used to do this. Currently, RabbitMQ are the only settings that can be changed. Custom configurations can be updated or added using the instructions in Fill Security Database with custom configuration section. A call to this endpoint will schedule the update request to run as soon as possible and moves the instance to `enqueued` state.

Change RabbitMQ settings

```
{
  "rabbitMq": {
    "url": "rabbitmq://newsuperrabbitmq.office.sbs/2",
    "user": "the-new-user",
    "password": "EAAAAFqCRyzli+rtgXNGGv6xvxxkyh67Upr8xAR3e3SDaOFp"
  }
}
```

Update configurations

```
{
  "securityCustomConfigurations": [
    {
      "appKey": "ORI",
      "fileName": "vanilla-orig.zip",
      "base64Data": "..."
    }
  ]
}
```

Both

```
{
  "rabbitMq": {
    "url": "rabbitmq://newsuperrabbitmq.office.sbs/2",
    "user": "the-new-user",
    "password": "EAAAAFqCRyzli+rtgXNGGv6xvxxkyh67Upr8xAR3e3SDaOFp"
  },
  "securityCustomConfigurations": [
    {
      "appKey": "ORI",
      "fileName": "vanilla-orig.zip",
      "base64Data": "..."
    }
  ]
}
```

The instance will be in `upgrading` state while updating. If the update fails, it will go to `errorUpgrading` state. See Troubleshooting for more information. In error state, you can retry the update using the same parameters.

FILL SECURITY DATABASE WITH CUSTOM CONFIGURATION

> **Security Configurations**
>
> To clarify what it is talking about in this section, *Security custom configurations* refers to the configuration packages with Users, Securables, Roles, Clients, etc. for specific Applications (like Workflow, Origin, Claims...).

When creating a new instance, custom Security configuration can be uploaded in the same request and deployed alongside Security. After an instance has been created, at any time, new configuration can be uploaded to update already existing ones or add new ones. In both scenarios, the schema is the same, and all have the same limitations.

The configurations are uploaded by putting them in the same request as base64. The configuration must be stored in a zip archive with the same format as used in the `sequel-security` tool.

Configuration package structure

In case you are not sure about which is the structure refered in this documentation, see below example:

```
your-configuration.zip
  - Applications/
    - {appKey}/
      - Application.json
      - Authentication/
        - ApiResources.json
        - Clients.json
      - Authorization/
        - Groups.json
        - Roles.json
        - Securables.json
        - UserTypes.json
  - Users/
    - Users.json
```

Not all files are requred, only the `Application.json` is required. The rest will depend on your configuration. This structure is generated automatically by the Security Tool extract command.

Note that only one `appKey` per zip is supported currently. If you have one zip with two applications inside, you may want to upload the same zip twice with different `fileName` s and pointing to the right `appKey` .

In the create and upgrade, both have a dedication section to upload the configurations:

```
{
  "securityCustomConfigurations": [
    {
      "appKey": "WF",
      "fileName": "wf.zip",
      "base64Data": "..."
    },
    {
      "appKey": "PB",
      "fileName": "pb.zip",
      "base64Data": "..."
    },
    {
      "appKey": "CLM",
      "fileName": "claims.zip",
      "base64Data": "..."
    }
  ]
}
```

- The `base64Data` is where the zip in `base64` must be placed for each custom configuration.
- `fileName` is to specify a name for the file. It is mainly for troubleshooting purposes, but it is recommended to put something understandable. The name must be unique for the request.
- `appKey` is the application key for the configuration. Only one application can be uploaded by zip.

Quickly convert files to base64

Below there is a command to quickly convert a file into base64 string using a command:

Linux

```
cat /path/to/the/config.zip | base64 -w 0
```

PowerShell

```
[Convert]::ToBase64String([IO.File]::ReadAllBytes("C:\path\to\the\config.zip"))
```

macOS

```
cat /path/to/the/config.zip | base64
```

There are some limitations around the custom configurations:

- No *TFS Tokens* are accepted. `__Token__` are not supported at this moment. The values must be replaced before uploading the configuration.
- The request body cannot higher than 2MiB. It is difficult to upload almost 2MiB of `zip`s with `json`s inside, but take care of not reaching the limit, upload just the necessary files.
- If the application key is invalid, it won't fail.
- When using the upgrade endpoint to update configurations, you can reuse old file names, but only if they point to an updated configuration, not for completetly different ones. To ensure no issues raise when deploying the configurations, use different file names always.

DELETING AN INSTANCE

When an instance is not required anymore, and expiration date has not reached yet (or has no expiration date), the `DELETE` endpoint schedules the deletion of the instance (and moves the instance to `enqueued` state). This operation usually takes less than one minute to complete. When the deletion is in progress, it will change the state to `deleting`. If an error occurrs, it will go to state `errorDeleting`, and probably will be inaccessible.

> **Danger**
>
> After deleting the instance, nothing will be left about the instance, like it had never existed. If you want to keep some configurations or anything else, it is recommended to recover them before deletion.

RESTARTING AN INSTANCE

If there have been changes in database and some of the services requires a restart (mainly Authentication or Authorization), this endpoint will restart the requested services or all if `null` is put in the request. This operation usually takes around 10s. The supported services are `api`, `authentication` and `authorization` - `admin` can also be restarted but it is useless because it is a static page and an nginx.

GETTING AVAILABLE SECURITY VERSIONS

The endpoints related to Security versions allow you to inspect available Security versions in the platform. The `GET /api/security-version` endpoint will get all available versions sorted by published date (desc - first the latest published version). The `GET /api/security-version/latest` endpoint will show you which version is the latest, so you will know which version is going to deploy if `latest` is used. And the last `GET /api/security-version/{version}` endpoint is useful to check if a specific version exists.

HOUSEKEEPING: EXPIRED INSTANCES

An instance is expired when the expiration date is reached (today is the same date as in `expiration` field in the `GET /api/instance/{instanceName}` endpoint). All those expired instances will be removed at midnight using the same method as in Deleting an instance.

> **Time**
>
> Time is never taking into account in expiration, just the date.

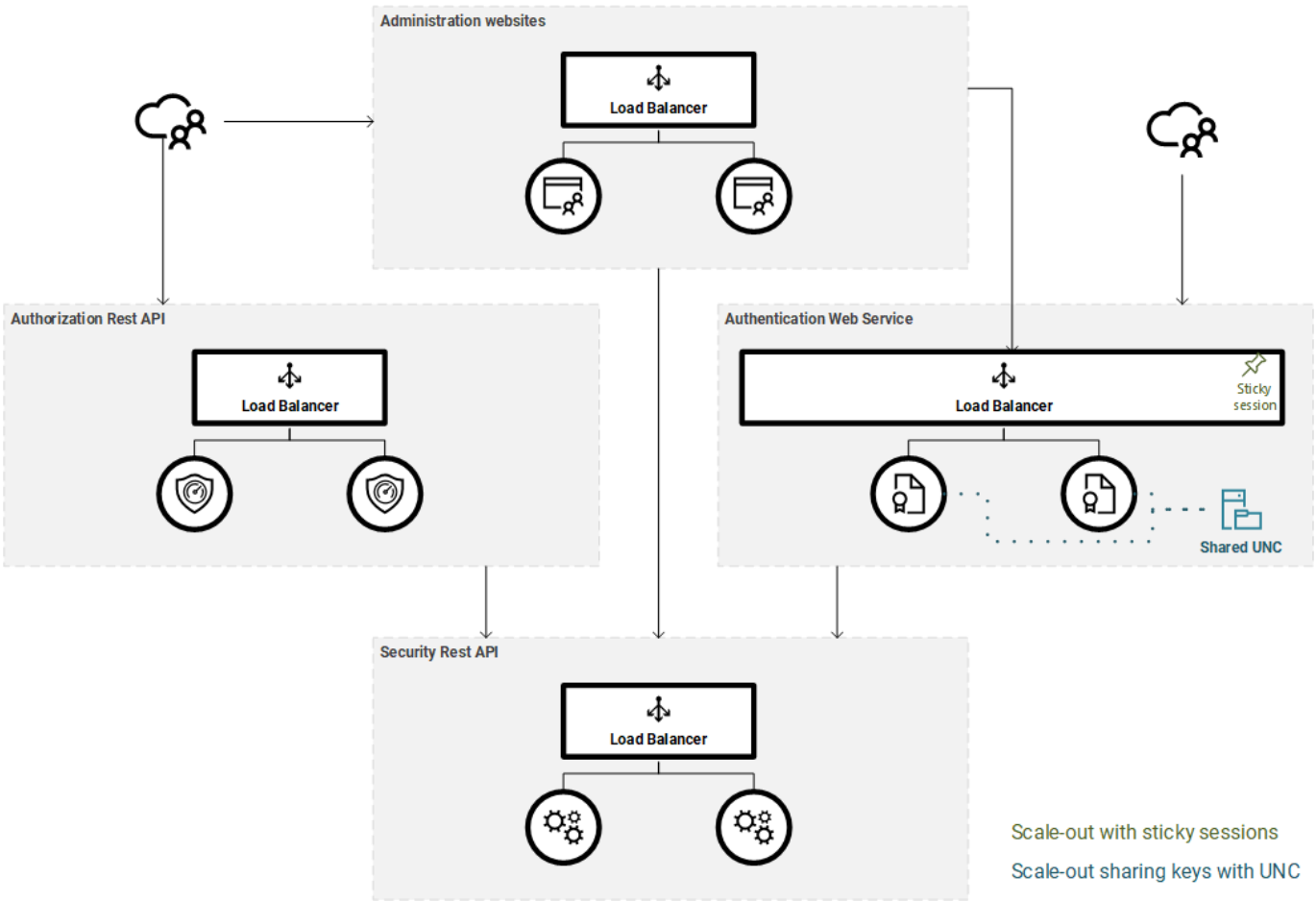
HOUSEKEEPING: UPGRADE LATEST

All instances created with no version (or `latest` version) will be marked for this upgrade housekeeping task. This task will upgrade each instance marked to use always the latest version when a new version is available in the platform. You don't need to do anything.

4.8.5 Scale out

Scale-out

Scaling-out security services is done by deploying multiple instances of each service behind a load balancer or any other similar solution, like an IIS Application Request Routing (AAR).



All services can be naturally scaled-out without any change by design, with the exception of the *Authentication Web Service*. *Authentication Web Service* as a needs to store some information for keeping state:

- **Duende IdentityServer operational data**: For certain operations, Duende IdentityServer needs a persistence store to keep state, this includes: issuing authorization codes, issuing reference and refresh tokens or storing consent. All this information is stored in the database and does not present scaling-out problems. More information at http://docs.identityserver.io/en/latest/topics/deployment.html.
- **ASP.NET Core data protection**: ASP.NET Core itself needs shared key material for protecting sensitive data like cookies, state strings etc. See the official docs at https://docs.microsoft.com/es-es/aspnet/core/security/data-protection. Currently, we offer two implementations: the *in-memory* store or *shared UNC path* resource.

AUTHENTICATION SCALE-OUT

Load balancer with sticky sessions

We have performed tests configuring the load balancer for Authentication service with sticky sessions, and the access token are refreshed properly.

Sharing Data Protection data

If there are no sticky sessions, then scaling-out authentication servers requires sharing *ASP.NET Core data protection* and this can be achieved sharing those keys with any of the below options:

Database

In Security database. This can be configured at `appsettings.json` file at `DataProtectionSettings.Mode` property introducing the value *Database*. This option is valid for on-premise and AWS deployments.

AWS System Manager

In cloud deployments in AWS, we can also use AWS System Manager Parameter Store to share the Data Protection data. The type of parameter used is *standard*, and the key is generated following the pattern: `/Sequel.Security/{AuthorityIdOfThisSecurityInstance}/DataProtection`, where `AuthorityIdOfThisSecurityInstance` is the authority id assigned in OpenIdConnect to this Security Service (IdP). This can be configured at `appsettings.json` file at `DataProtectionSettings.Mode` property introducing the value *AWS*.

LOAD BALANCERS AND HTTPS

When deploying Security to AWS behind a Load Balancer or behind any Reverse Proxy (like nginx, traefik, kong...), Security must trust some headers sent by the Load Balancers/Reverse Proxies. Failing to do so, Security will not work properly, with errors like unable to log in in Administration or infinite redirection loop in Authentication. To make Security trust these headers, set the `TrustForwardedHeaders` setting to `true` on Security API, Authentication and Authorization.

4.9 Registration forms

4.9.1 LDAP Sync Registration

> **Note**
>
> This registration form covers the cross-domain identity management with Windows AD using LDAP

Sequel Security Services offers the possibility of syncing users and permissions from Windows AD using LDAP. This document tries to identify all the basic information for configuring the connectivity between the LDAP Sync service and Windows AD.

Configuration of matching rules between Windows groups and Sequel Memberships are out of the scope of this document.

Connection settings

BASIC INFORMATION

Sequel LDAP Sync service must be installed and requires access to Windows AD and also to Security API. The LDAP connectivity requires below information that must be provided by the IT admin of the Windows AD:

Parameter	Type	Description
Host	string	A host name or a dotted string representing the IP address of a host running an LDAP server. It may also contain a list of host names, space-delimited. Each host name can include a trailing colon and port number
Port	integer	The TCP or UDP port number to connect to or contact. The default LDAP port is **389** and LDAPS is **636** (enabled if `SecureConnection` is true). The port parameter is ignored for any host name which includes a colon and port number.
SecureConnection	boolean	If the value is `true`, then uses TLS for the connection to LDAP (LDAPS). By default, is set to `false`.
DN	string	If non-null and non-empty, specifies that the connection and all operations through it should be authenticated with DN as the distinguished name. We highly recommend to use a service account or user with only read permissions on Windows AD.
Password	string	If non-null and non-empty, specifies that the connection and all operations through it should be authenticated with DN as the distinguished name and this argument as password.

PORTS

In terms of connectivity, we will potentially need access to below ports: *389, 636, 3268*. Depending if AD is configured with LDAPS or not and if the catalog is part of a global catalog.

Port 3268. This port is used for queries specifically targeted for the global catalog. LDAP requests sent to port 3268 can be used to search for objects in the entire forest. However, only the attributes marked for replication to the global catalog can be returned. For example, a user's department could not be returned using port 3268 since this attribute is not replicated to the global catalog. Normally, the most typical port to be used.

Port 636. Port for LDAPS.

Port 389. Although some background services may require access to this port for the LDAP sync process, we cannot explicitly use this port as it doesn't support the type of queries we are using.

Query settings

The users target of the sync are defined as those users that belongs to an specific Windows group. For performing those queries, it is required to provide below information:

Property	Type	Description
Domain	string	Defines the domain assigned to the sync'd users. This domain will be stored in the user record. This setting is quite important as the user will be authenticated using {domain}{ssoUserName}.
SearchBaseQuery	string	The base distinguished name to search from. *Empty* by default. The `SearchBaseQuery` indicates a base domain to be used in the queries. This value must be set if request are done against the global catalog. The value should contain `DC` entries: if your base domain is `office.local`, the `SearchBaseQuery` should be `DC=office,DC=local`.
GroupQueries	List of GroupQuery.	List of LDAP queries to retrieve the DN groups involved in the sync process. The Vanilla configuration contains a single entry with below LDAP query `(&(cn=Sequel Application Users))`. So, by default all users assigned to the group `Sequel Application Users` will be sync'd. If there are no groups queries, the process will be cancelled.
PageSize	int	Queries executed are paged using this setting as the page size. Default value is *1000*. This value should be lower or equal than the maximum number of objects returned by a single query in the AD server.

Inheritance of Windows groups is not supported.

Registering information for Sequel

Please provide below information back to Sequel in order to configure the application:

	Description	Value
Host	Host name of AD	
LDAP/LDAPS	True/False. Confirm standard ports	
DN	Service account with read-only permissions	
Password	Service account password	
Domain	ie. sbs	
SearchBaseQuery	ie. `DC=office,DC=local`	
GroupQueries or GroupName	Identify Windows Group where all user member will be synced	
PageSize	This value must match with AD server settings. Default 1000	

4.9.2 Azure AD Authentication Registration

> **Note**
>
> This registration form covers the federation of the authentication process to Azure; as an external identitu provider.

Sequel Security Services allows external providers for *authentication* like Azure AD. This registration process should be done for each client.

This document describes preliminary steps to be done by Sequel Clients to register a Sequel Application in Microsoft Azure.

Registering Sequel App at Azure

Registering an application in Azure is described at https://docs.microsoft.com/en-US/azure/app-service/configure-authentication-provider-aad.

This document tries to provide samples of the process. Please, keep in mind that Azure Portal UI could change since this document was released.

REQUIRED INFORMATION

For registering an application in Azure we need the following information:

Application name

This is a friendly name for the application; we will suggest to use *Sequel Authentication Service*; but this can be changed and also it could include references to production or UAT environments (ie. *Sequel Authentication Service - Production*, *Sequel Authentication Service - UAT*).

Redirection URI

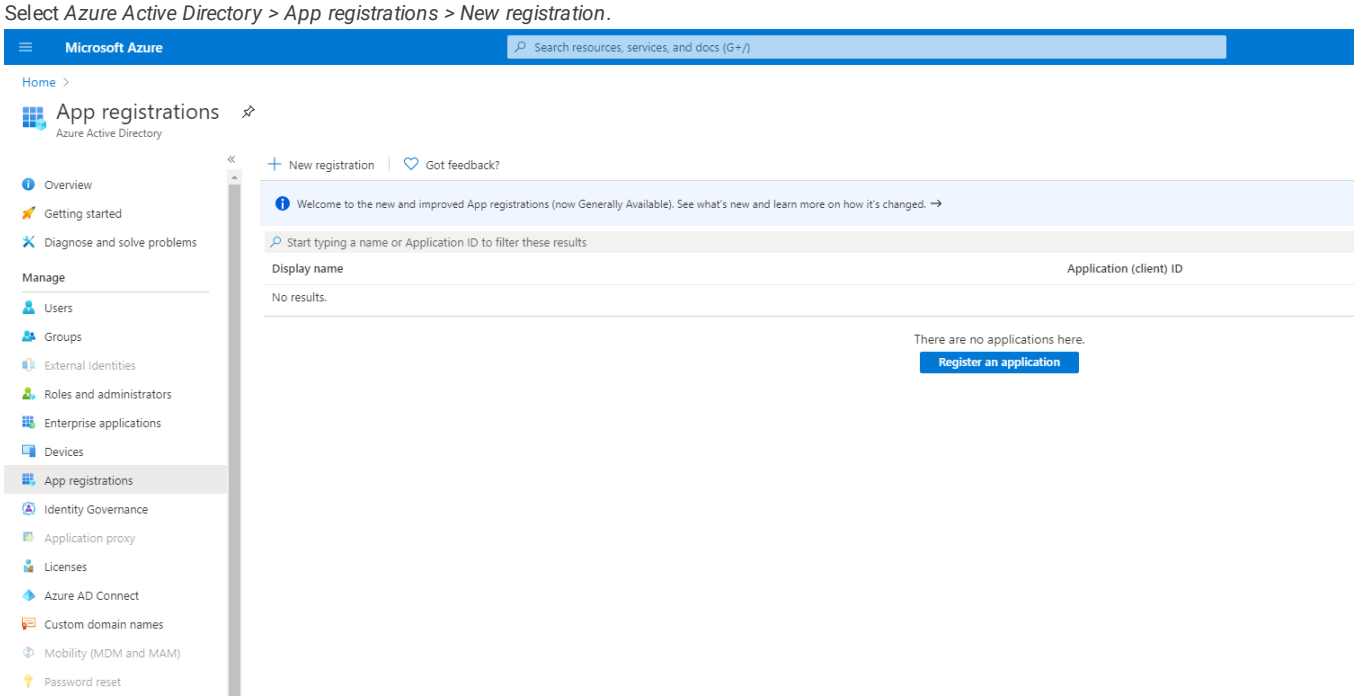
This is the public URI of the Sequel's authentication service. Usually, it looks like this: *Sequel-AuthN-Base-URI*/signing-microsoft (e.g. https://sequel_domain/Authentication/signing-microsoft). For this implementation the URI will be:

```
https://TO_BE_CONFIRMED/signing-microsoft
```

REGISTER AN APPLICATION

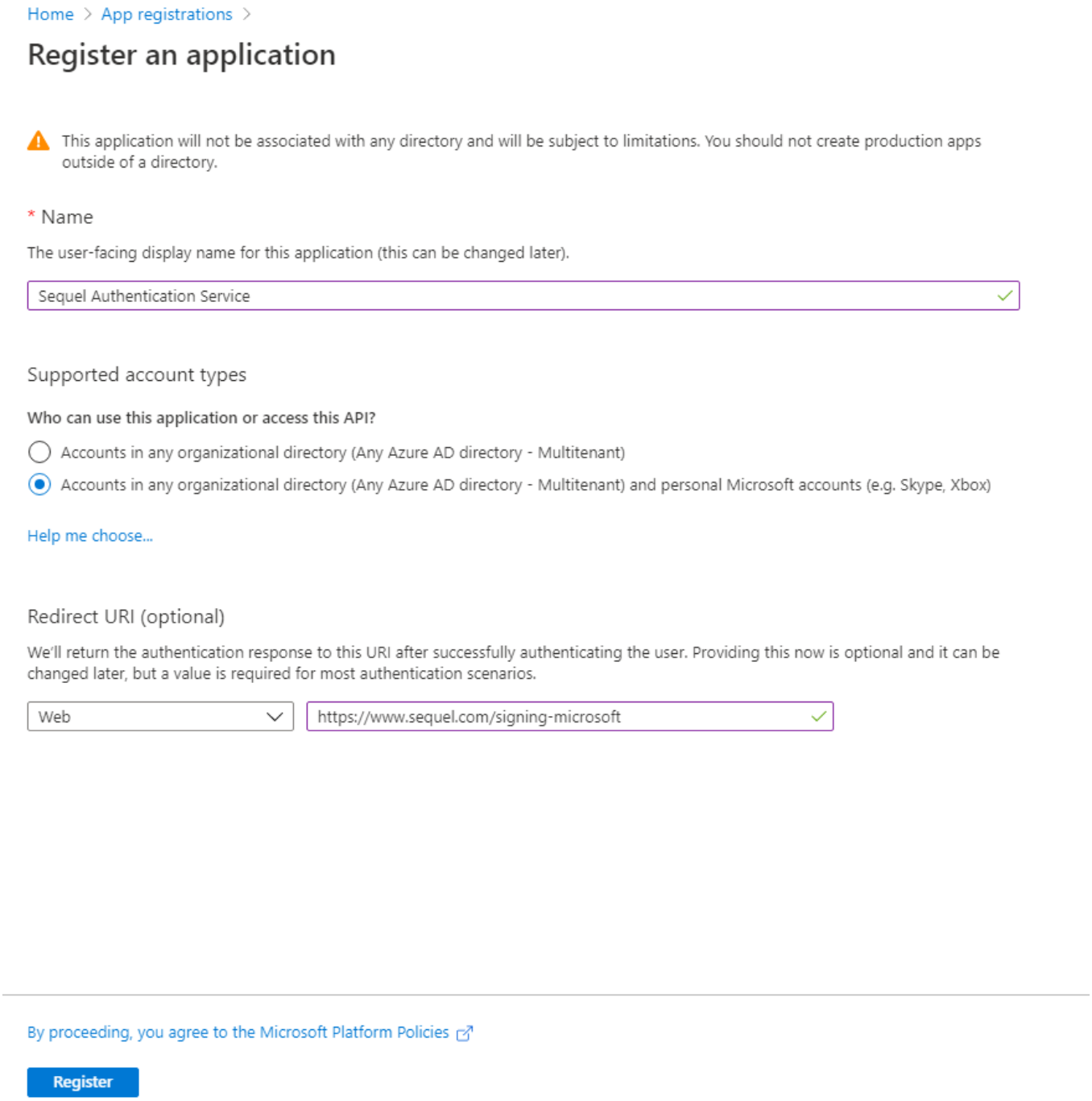
As a summary of this process, we will have to perform the following steps:

1. Sign in to the Azure portal.
2. Select *Azure Active Directory > App registrations > New registration*.

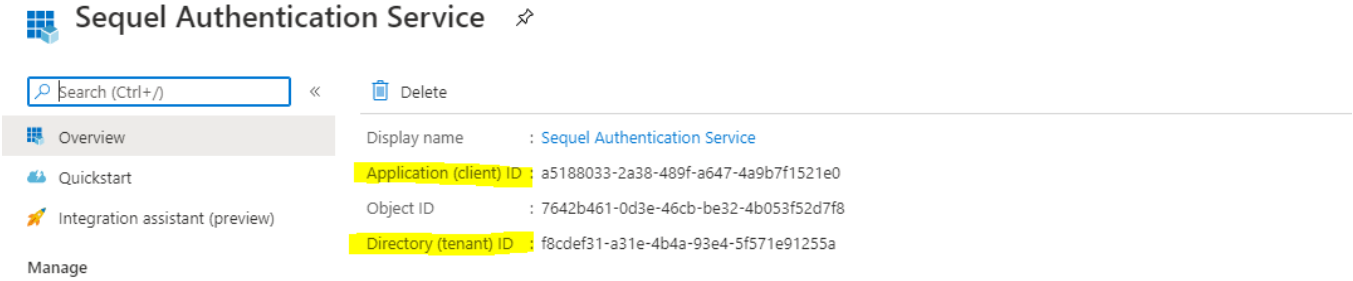


3. In the Register an application page, enter the *Name* for registering the app as described above.
4. In *Redirect URI*, select Web and type the redirect URI provided above.

5. Select Create.



6. After the app registration is created, copy the *Application (client) ID* and the *Directory (tenant) ID* for later. This information has to be provided to Sequel in order to register the SSO with Azure.

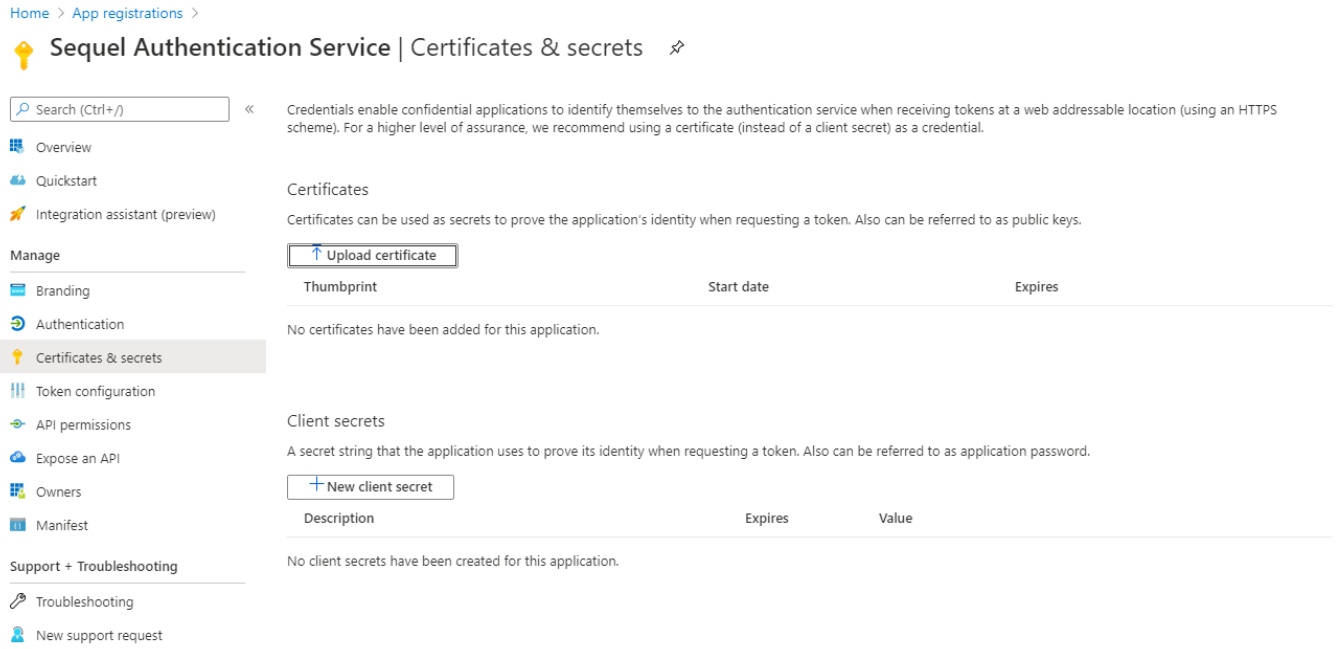


> **Note**
>
> It is possible to define multiple redirection Uri for the same application, this is covered in Microsoft's documentation.

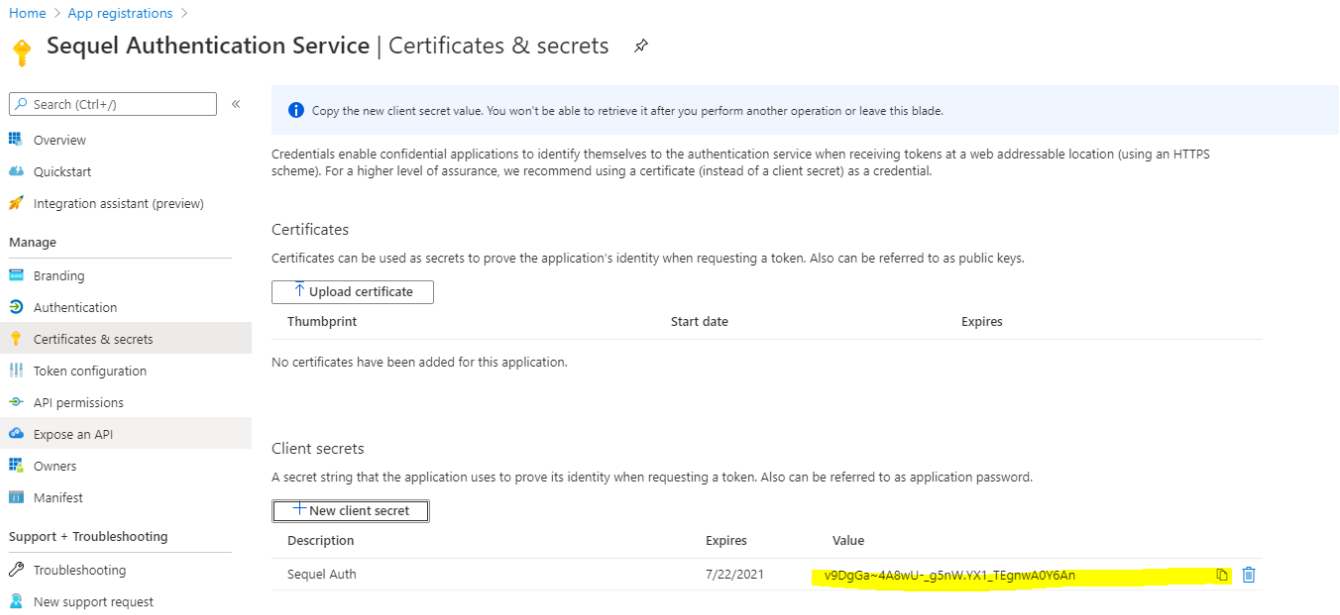
CREATE NEW SECRET

A Client secret needs to be created and shared with Sequel as part of the App registration. If you are in the App registration overview just select "Certificates & secrets"

1. To create a client secret, select *Certificates & secrets > New client secret*.

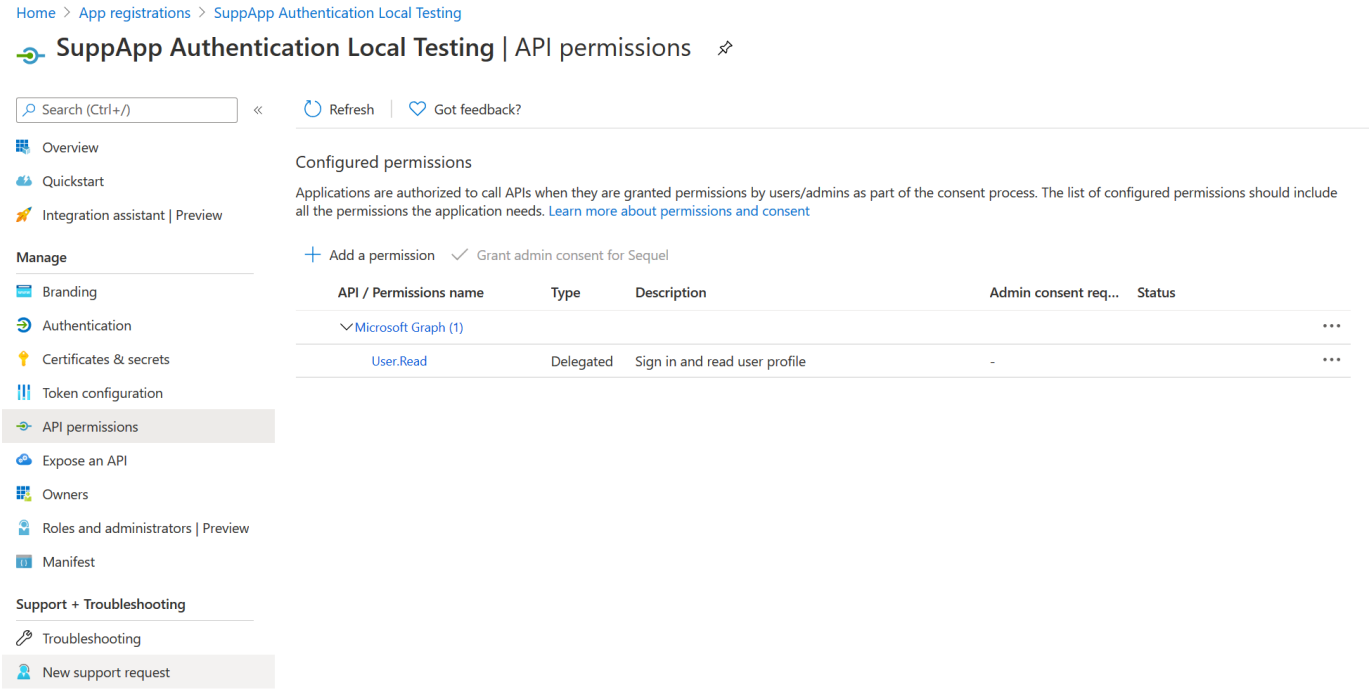


2. Copy the client secret value shown in the page. It won't be shown again.

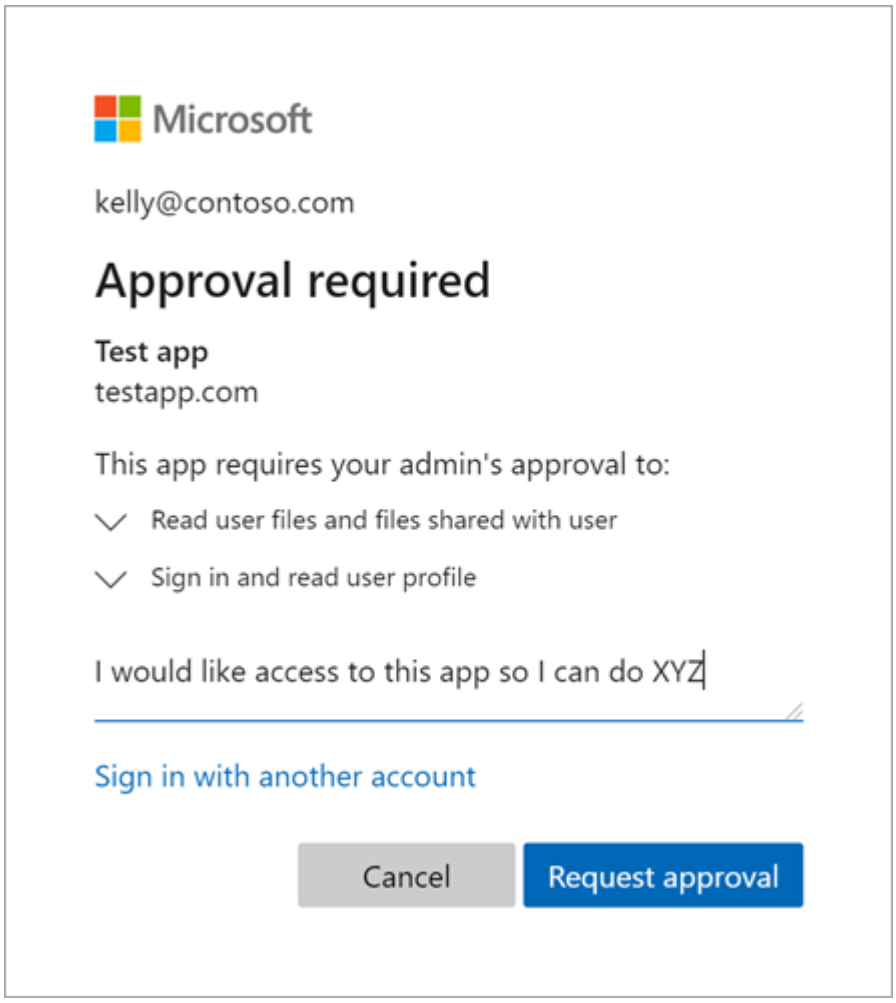


CONFIGURE API PERMISSIONS (OPTIONAL)

By default will be added the *User.Read* permission on Microsoft Graph for the application:

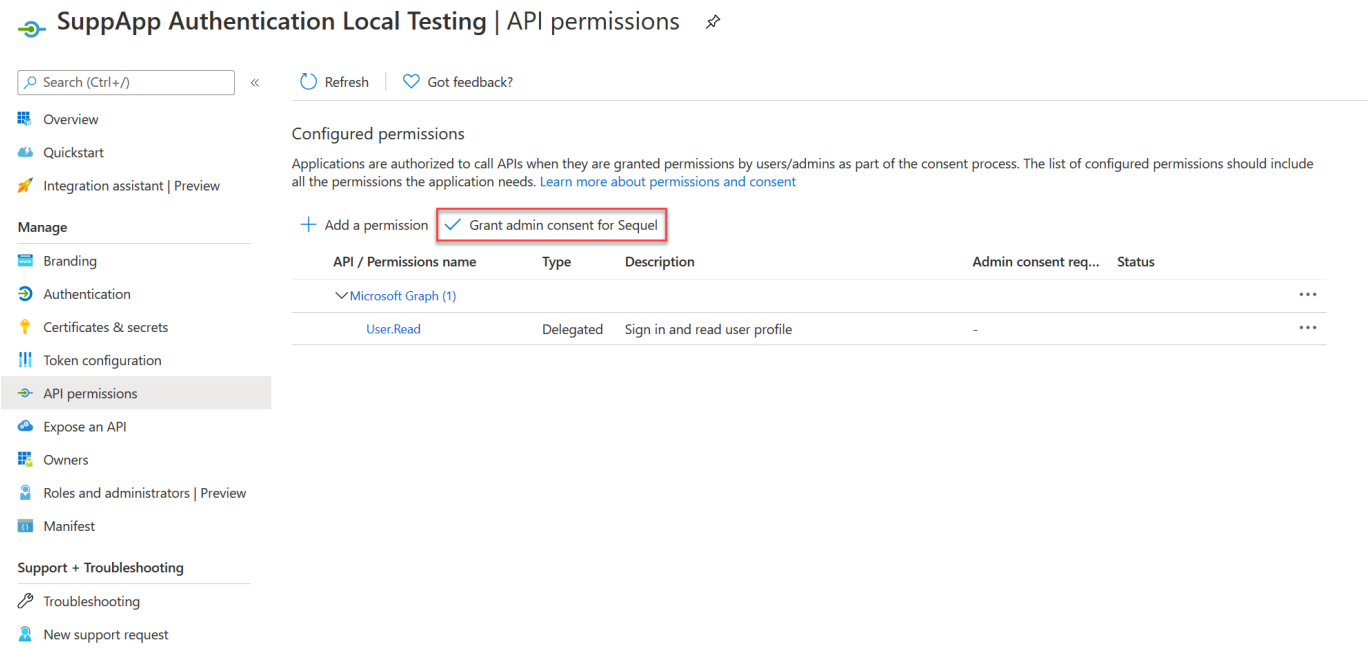


Depending on Admin Consent Workflow Configuration when a user uses for the first time the Azure AD authentication for the application the request must be review by the admin:

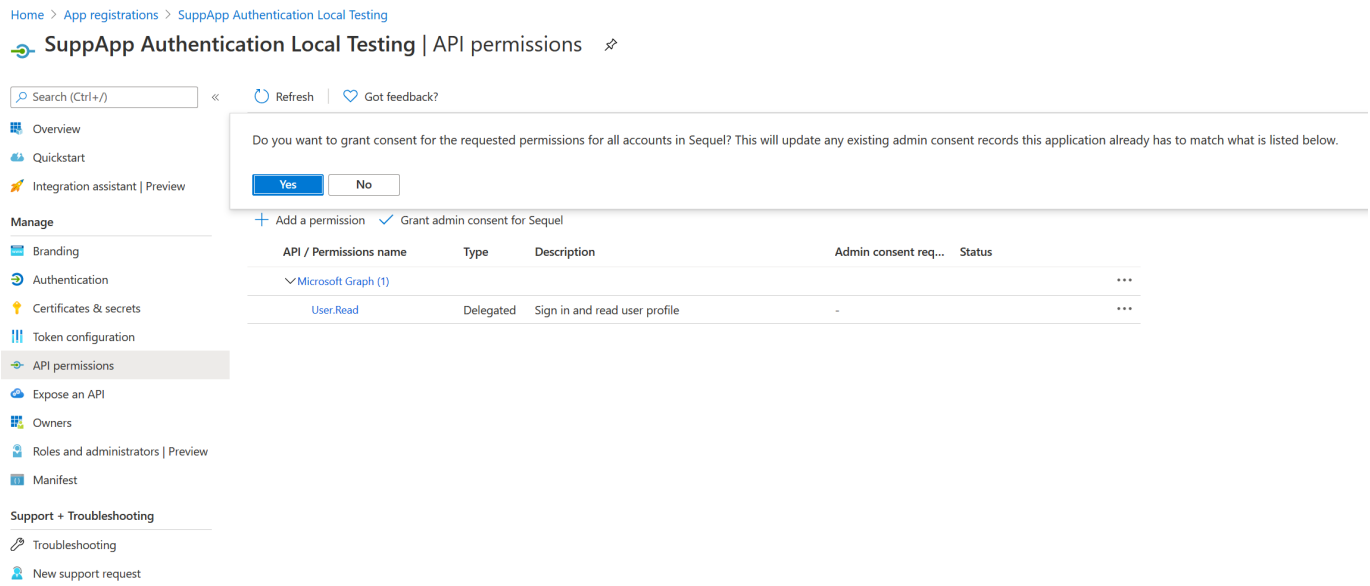


Its is possible avoid this dialog for this application and this way all requests from accounts of current Azure AD will be accepted by default like is explained in Grant admin consent in App registrations:

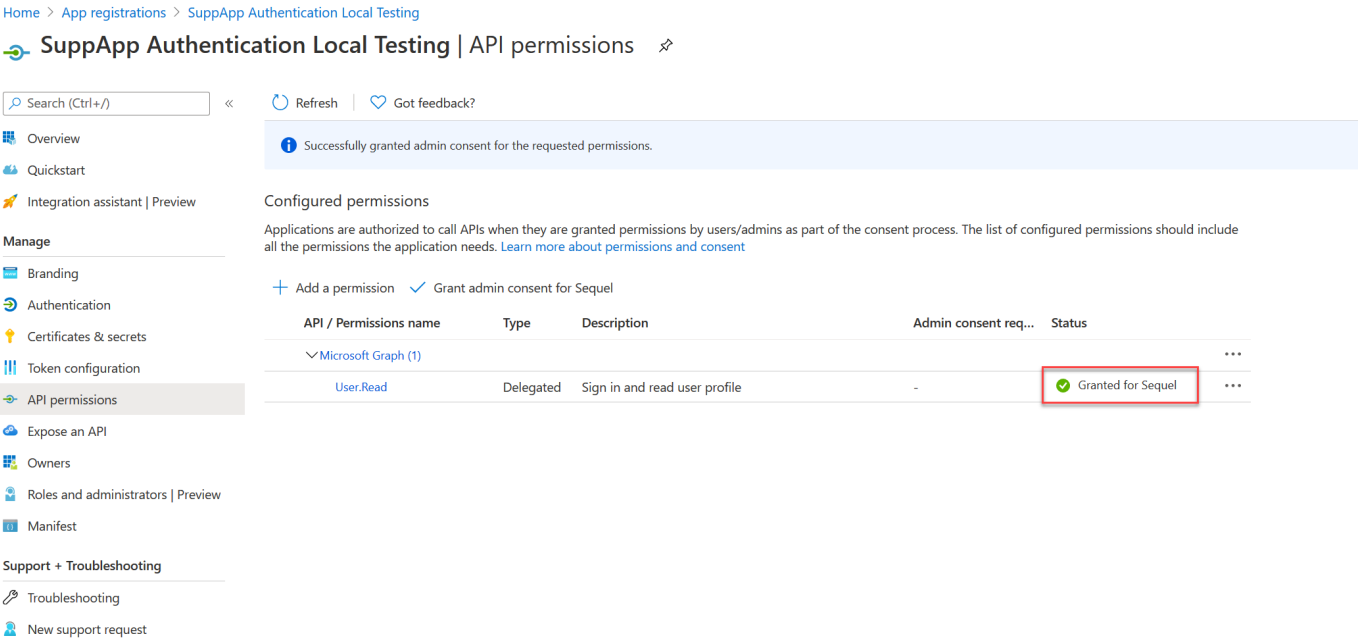
1. Click on *Grant Admin consent for* button:



1. Accept dialog box clicking on *Yes* button:



1. Permissions will be updated showing *Granted for* in Status column:



Registering users for SSO

Once Sequel App is registered in Azure ID, the next step is create the users in Sequel Security Services ensuring users are created using the same email provided by Azure in the claims. This email must be stored in the email or SsoUsername fields at the users records.

Registering information for Sequel

Once the registration process is completed below information must be provided back to Sequel in order to configure the application:

APPLICATION

	Description	Value
Client ID	Application (client) ID	
Tenant ID	Directory (tenant) ID	
Secret	Secret assigned by Azure	

USERS

List of users to be registered:

User name	Email

4.9.3 Azure AD Sync Registration

> **Note**
>
> This registration form covers the cross-domain identity management with Azure AD using Microsoft's Graph API (aka Azure AD Sync of users and groups)

Azure AD Sync service synchronize Azure AD users with Sequel Security Services.

This document describes preliminary steps to be done by Sequel Clients to register an Application in Microsoft Azure for being used by Azure AD Sync service.

Registering Azure AD App at Azure

Registering an application in Azure is described at https://learn.microsoft.com/en-us/azure/active-directory/develop/quickstart-register-app.

This document tries to provide samples of the process. Please, keep in mind that Azure Portal UI could change since this document was released.

REQUIRED INFORMATION

For registering an application in Azure we need the following information:

Application name

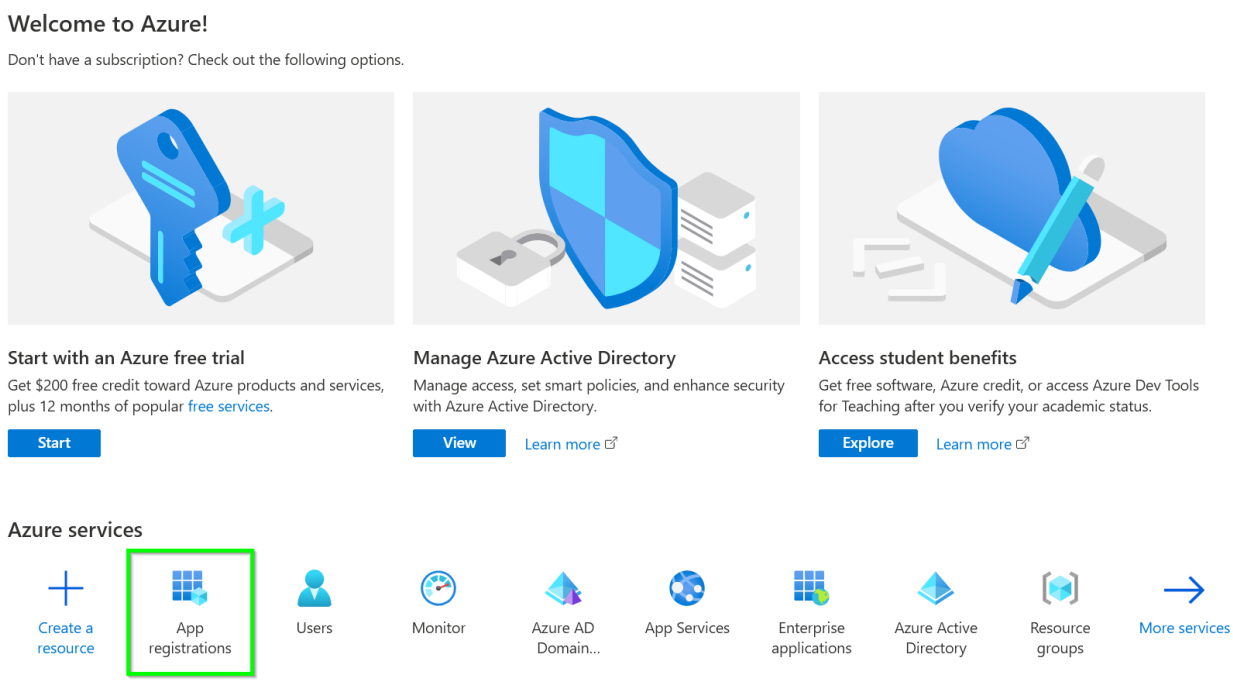
This is a friendly name for the application; we will suggest to use *Sequel Security Azure AD Sync*; but this can be changed and also it could include references to production or UAT environments (ie. *Sequel Security Azure AD Sync - Production*, *Sequel Security Azure AD Sync - UAT*).

Sequel Security Azure AD Sync

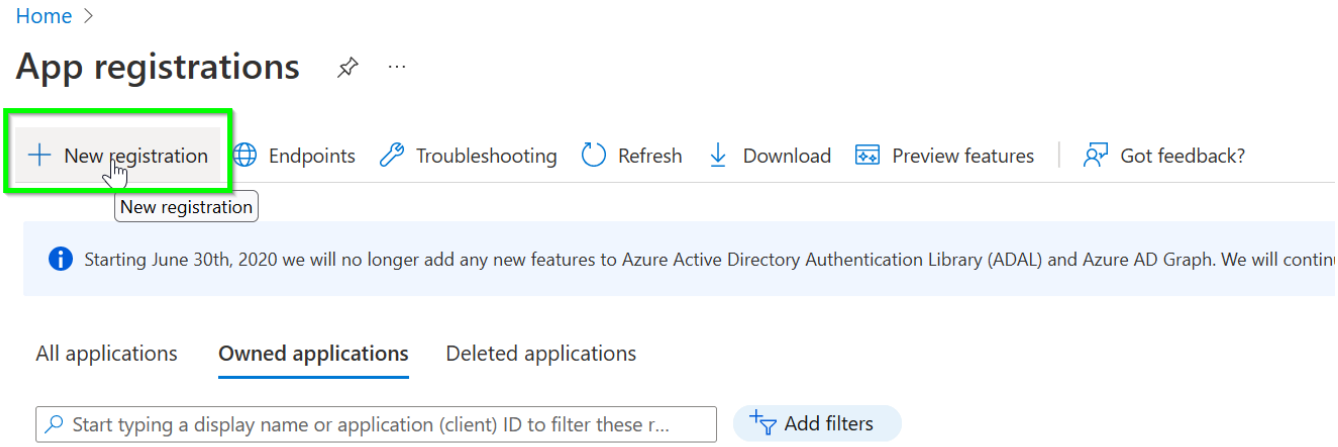
REGISTER AN APPLICATION

As a summary of this process, we will have to perform the following steps:

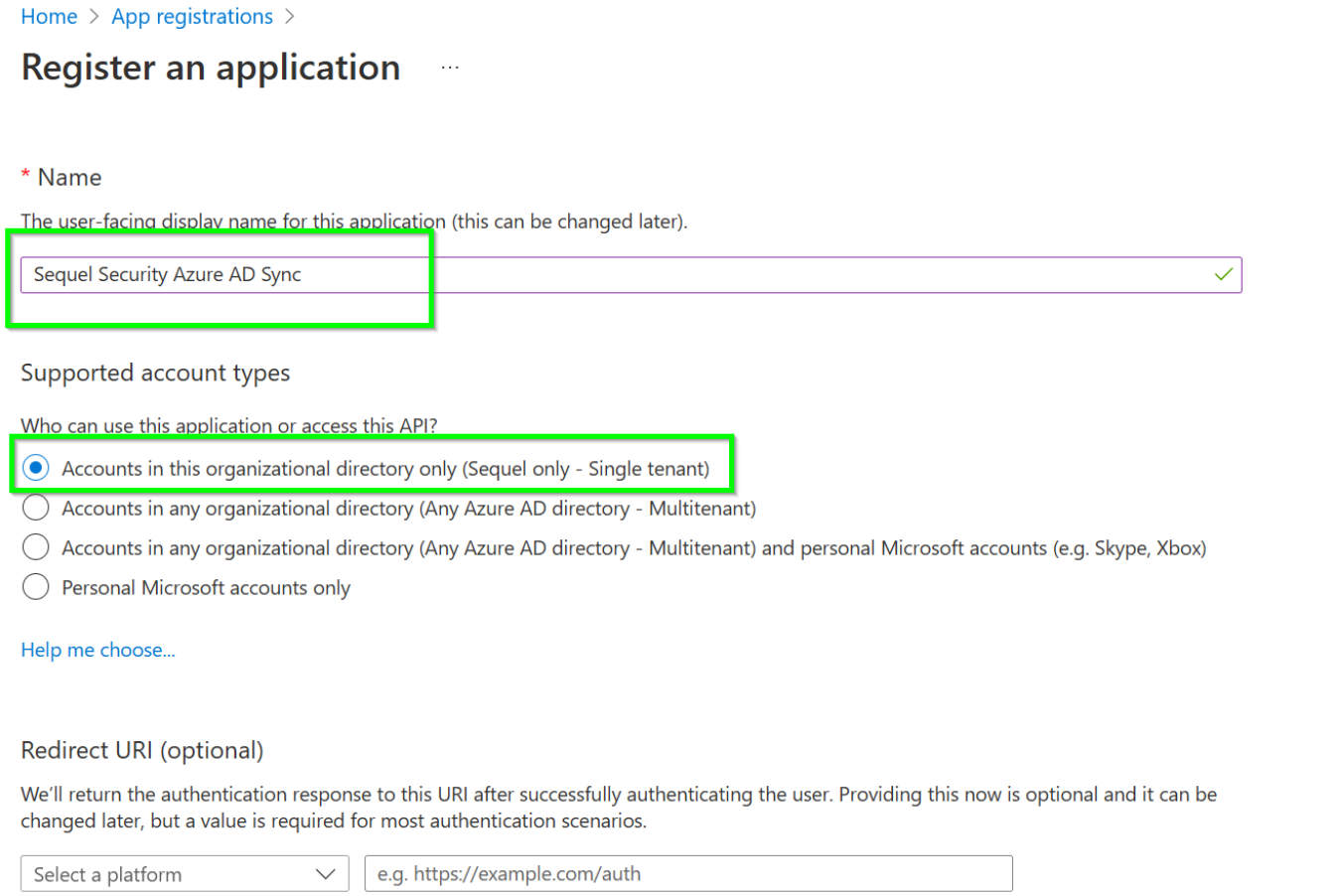
1. Sign in to the Azure portal.
2. Select *Azure Active Directory > App registrations*.



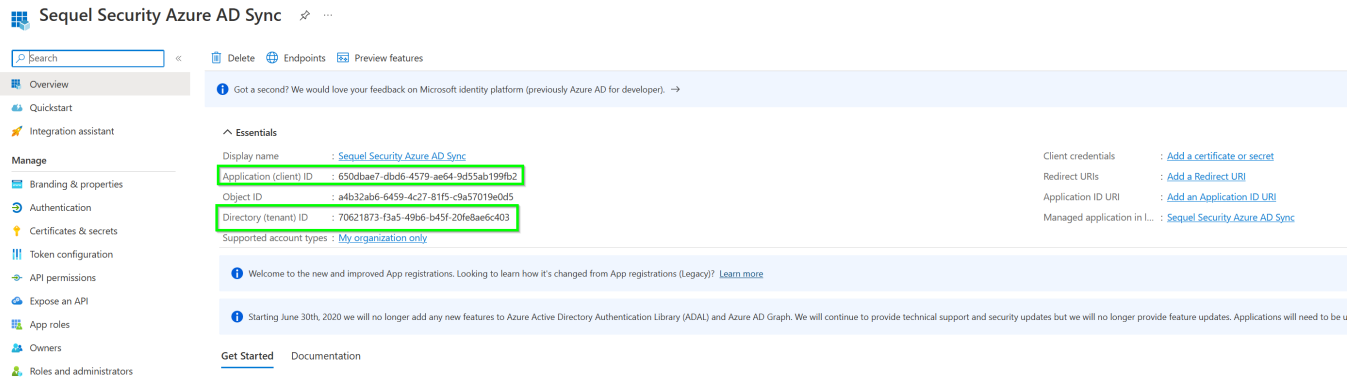
3. Click on *New registration*.



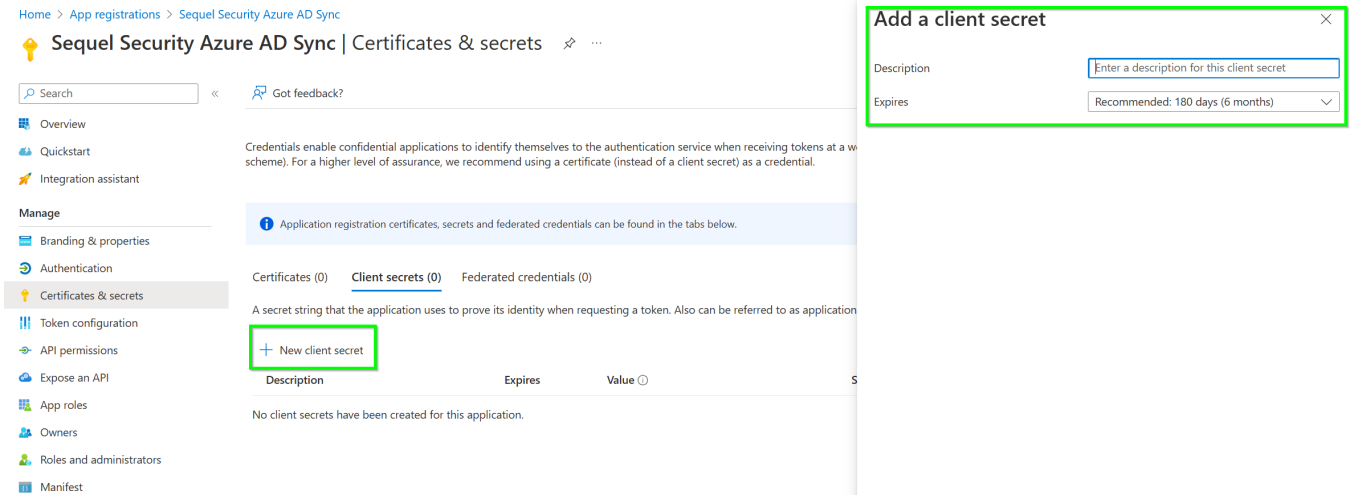
4. Introduce required properties for the application.



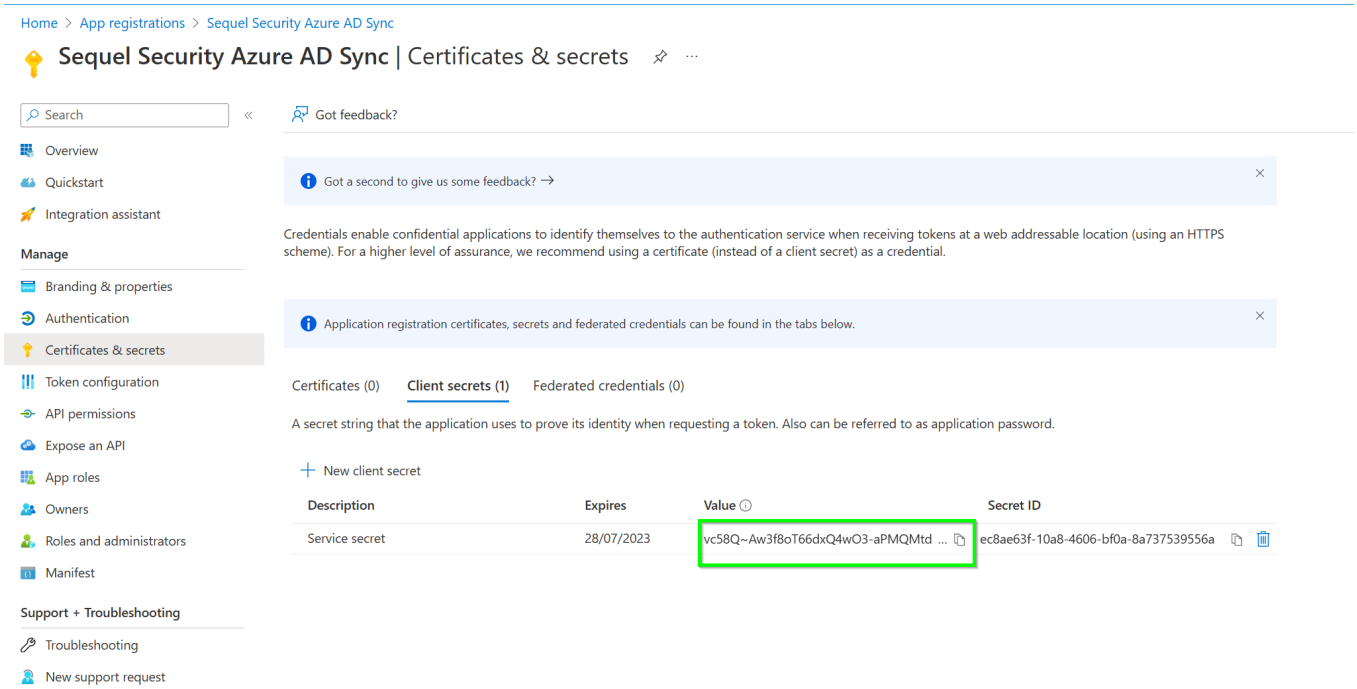
5. Once previous step is completed we'll have *TenantId* and *ClientId* parameters required for configure Azure AD Sync service.



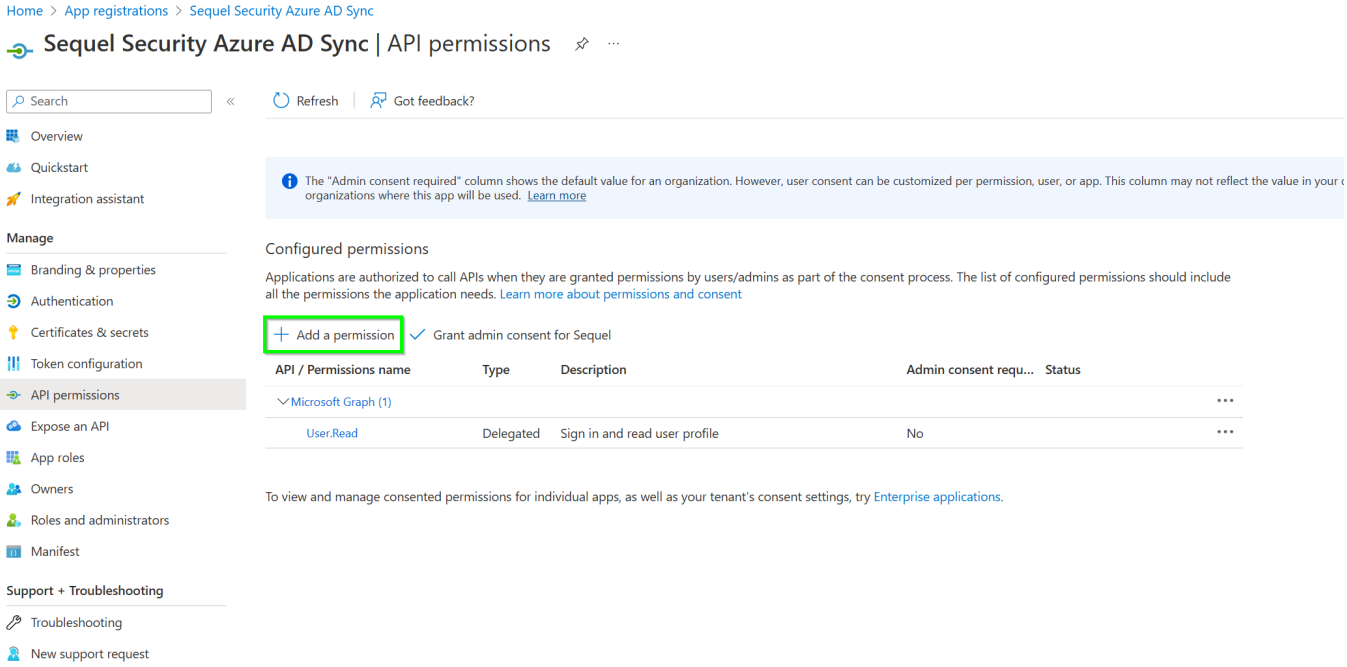
6. Now we're going to add a client secret in *Certificates & secrets > New client secret*



7. Then we'll have the *ClientSecret* parameter required for configure Azure AD Sync service.

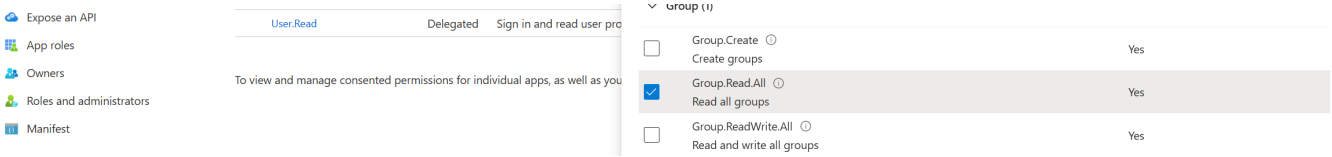


8. Finally is necessary add additional permissions for the application in *Api permissions > Add a permission*.

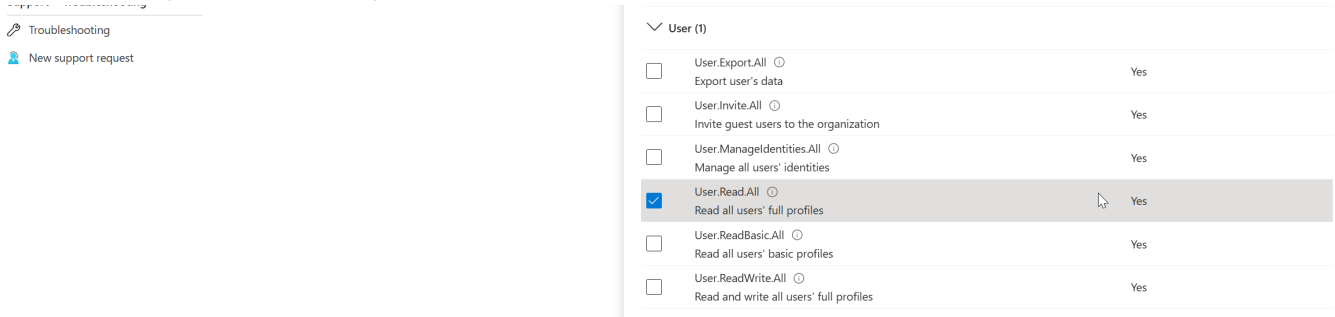


Next permissions are required for Azure AD Sync service:

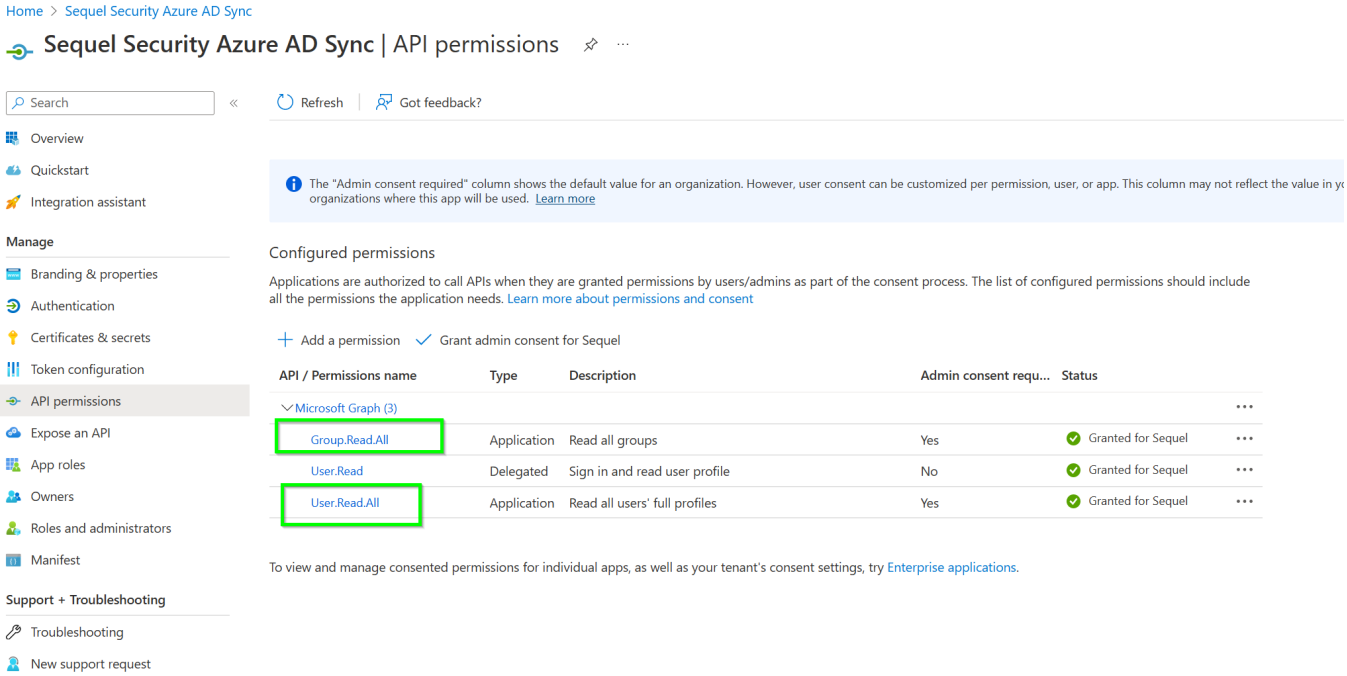
9. **Groups.Read.All**: this permission is required to be able to read *displayName* and *id* properties for the groups associated to the users. This allows to Azure AD Sync service the matching between Azure AD groups an Security MembershipSets.



10. **User.Read.All**: this permissions is required for reading all basic properties for the users (*id*, *mail*, *displayName*, *userPrincipalName*, *surname* and *accountEnabled*) and the user's groups. These properties will be used to find matches between Azure AD users an Security users, create/update them and create/update their memberships.



11. Once the application has all the permission is ready to be used by Azure AD Sync service.



4.9.4 JumpCloud Authentication Registration

Sequel Security Services allows external providers for *authentication* like JumpCloud. This registration process should be done for each client.

This document describes preliminary steps to be done by Sequel Clients to register a Sequel Application in JumpCloud.

Single Sign On with SAML 2.0 at JumpCloud

Registering an SSO in JumpCloud for be used for Security Authentication is described in https://support.jumpcloud.com/support/s/article/single-sign-on-sso-with-saml-20-connector1.

This document tries to provide samples of the process. Please, keep in mind that JumpCloud UI could change since this document was released.

REQUIRED INFORMATION

For registering an SSO in JumpCloud we need the following information:

Application name

This is a friendly name for the application; we will suggest to use *Sequel Authentication Service*; but this can be changed and also it could include references to production or UAT environments (ie. *Sequel Authentication Service - Production*, *Sequel Authentication Service - UAT*).

ACS URL

This is a public URI of the Sequel's authentication service. Usually, it looks like this: *Sequel-AuthN-Base-URI*/JumpCloud/Callback (e.g. `https://sequel_domain/Authentication/JumpCloud/Callback`). For this implementation the URI will be:

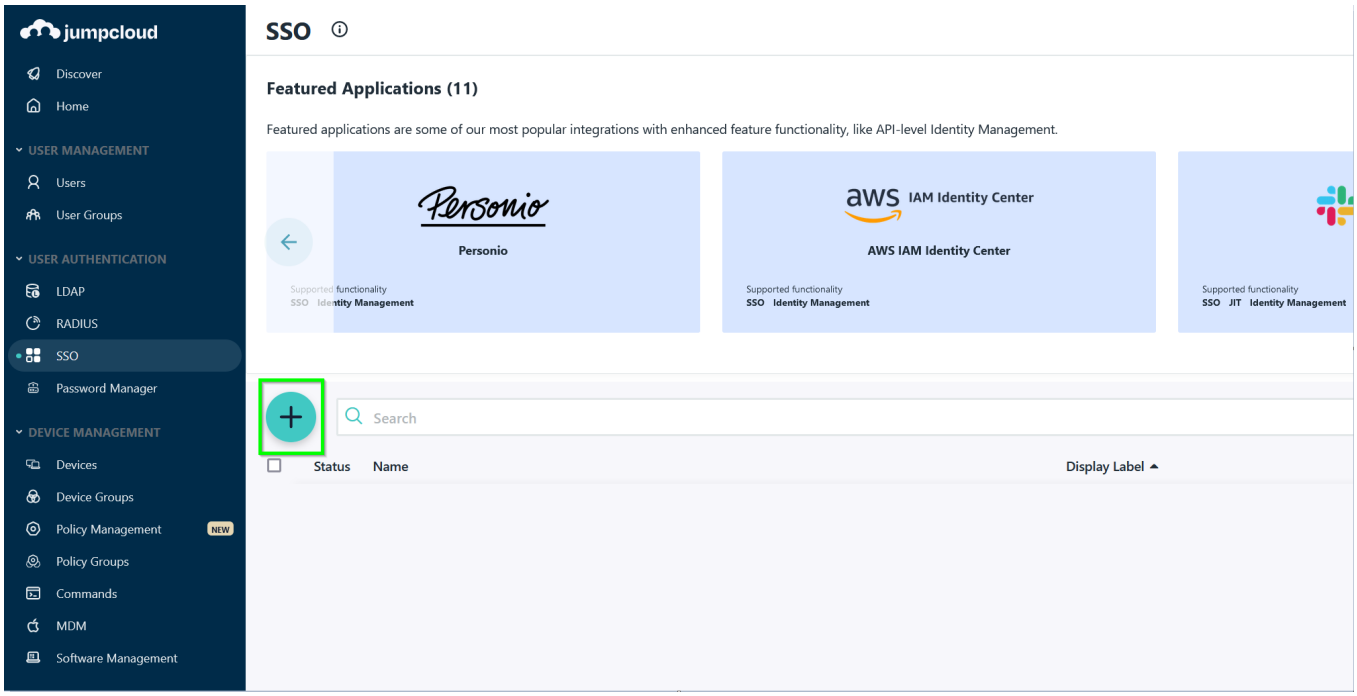
```
https://TO_BE_CONFIRMED/JumpCloud/Callback
```

Redirect URI's are case sensitive.

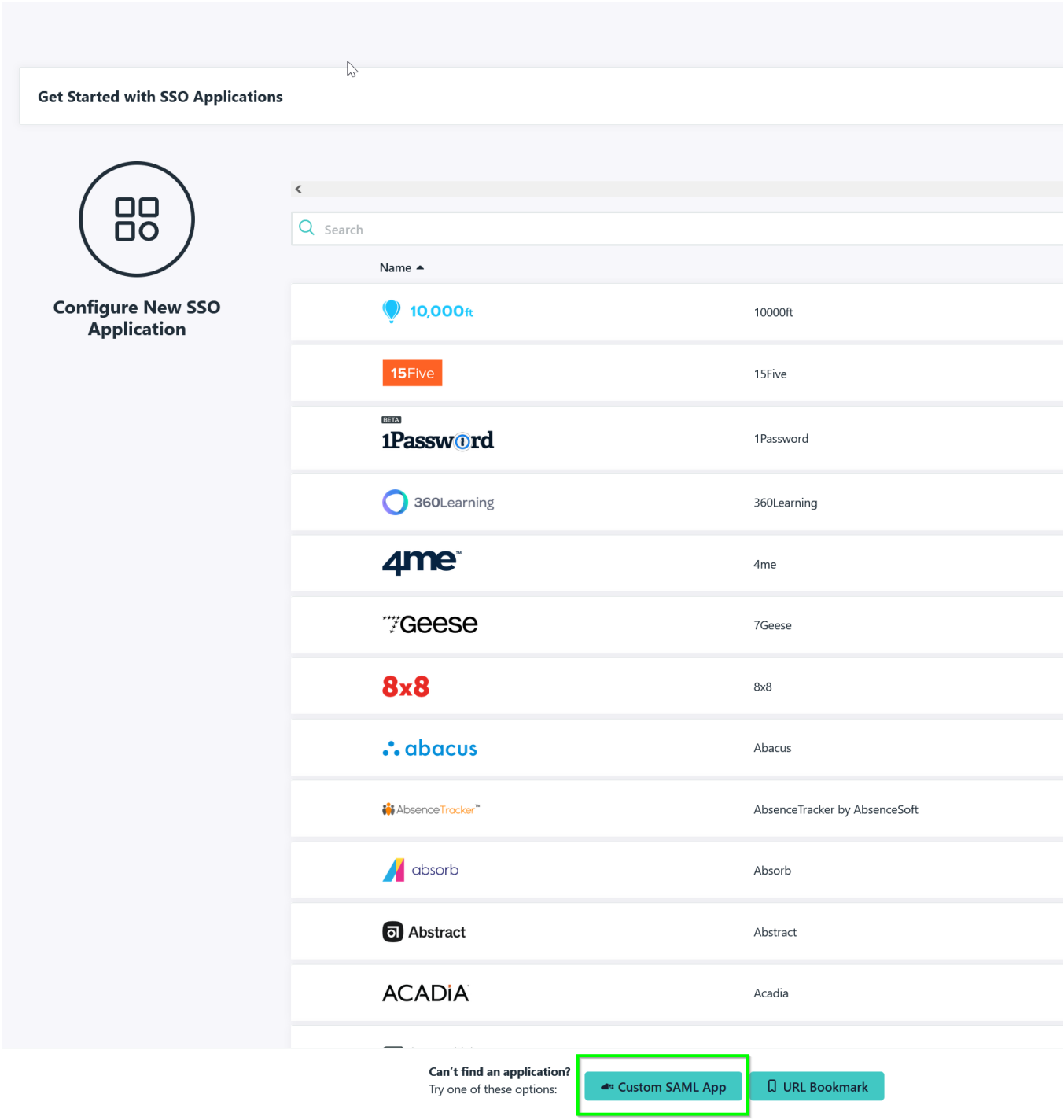
REGISTER SSO APPLICATION

As a summary of this process, we will have to perform the following steps:

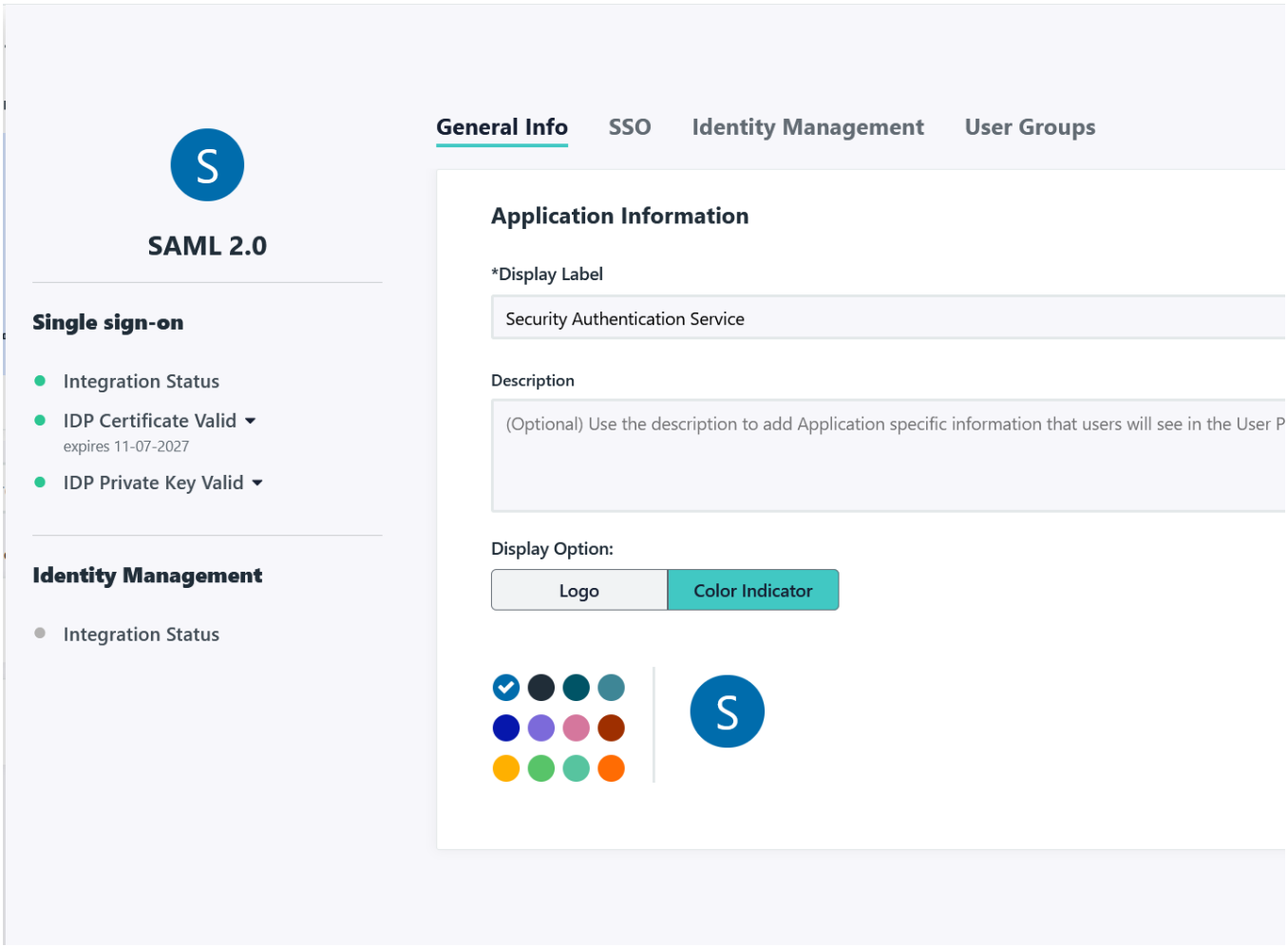
1. Sign in to the JumpCloud administration portal.
2. Select *SSO* > *+*.



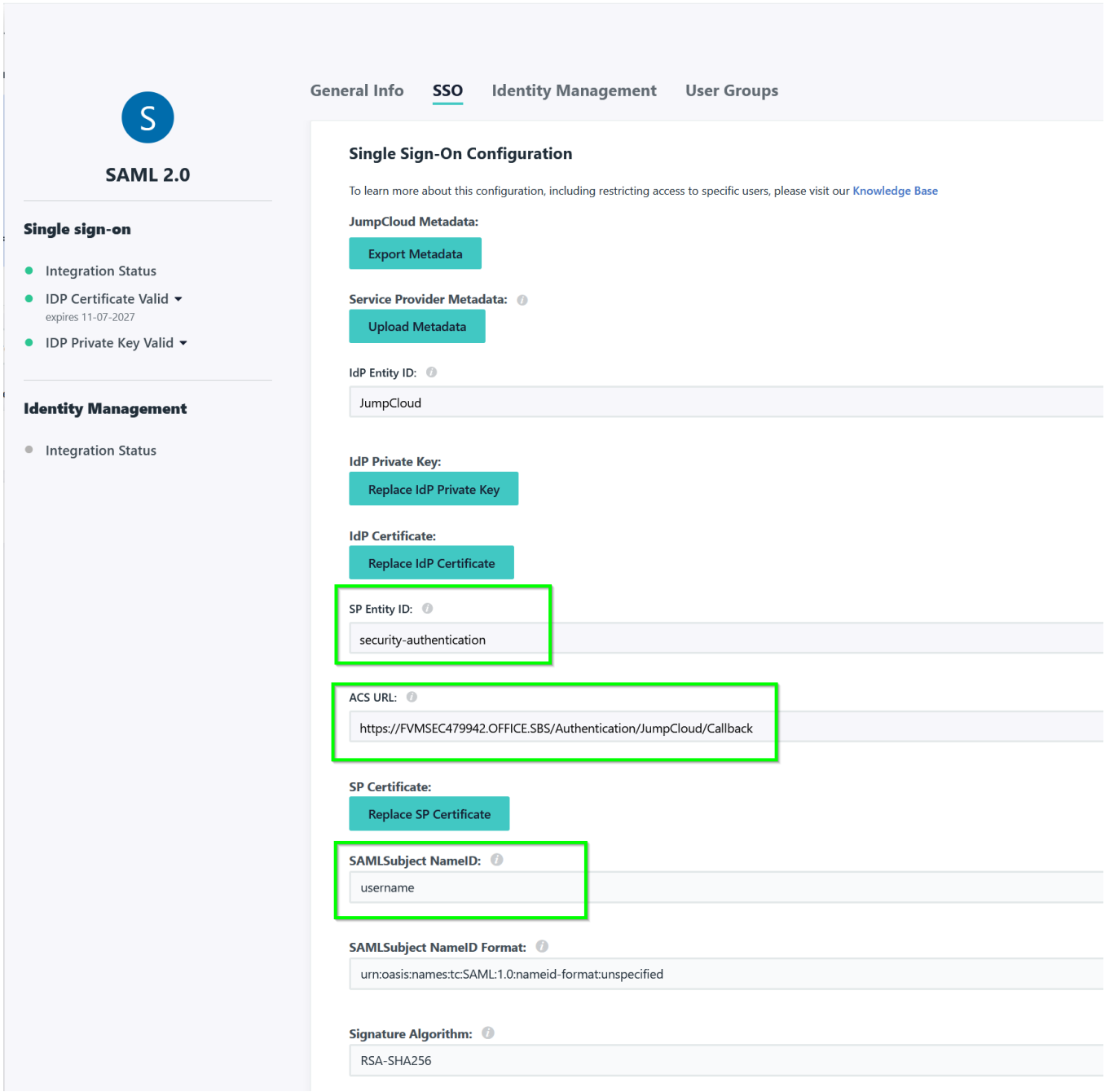
3. Click on *SSO > + > Custom SAML App*.



4. *General Info*



5. *SSO*.

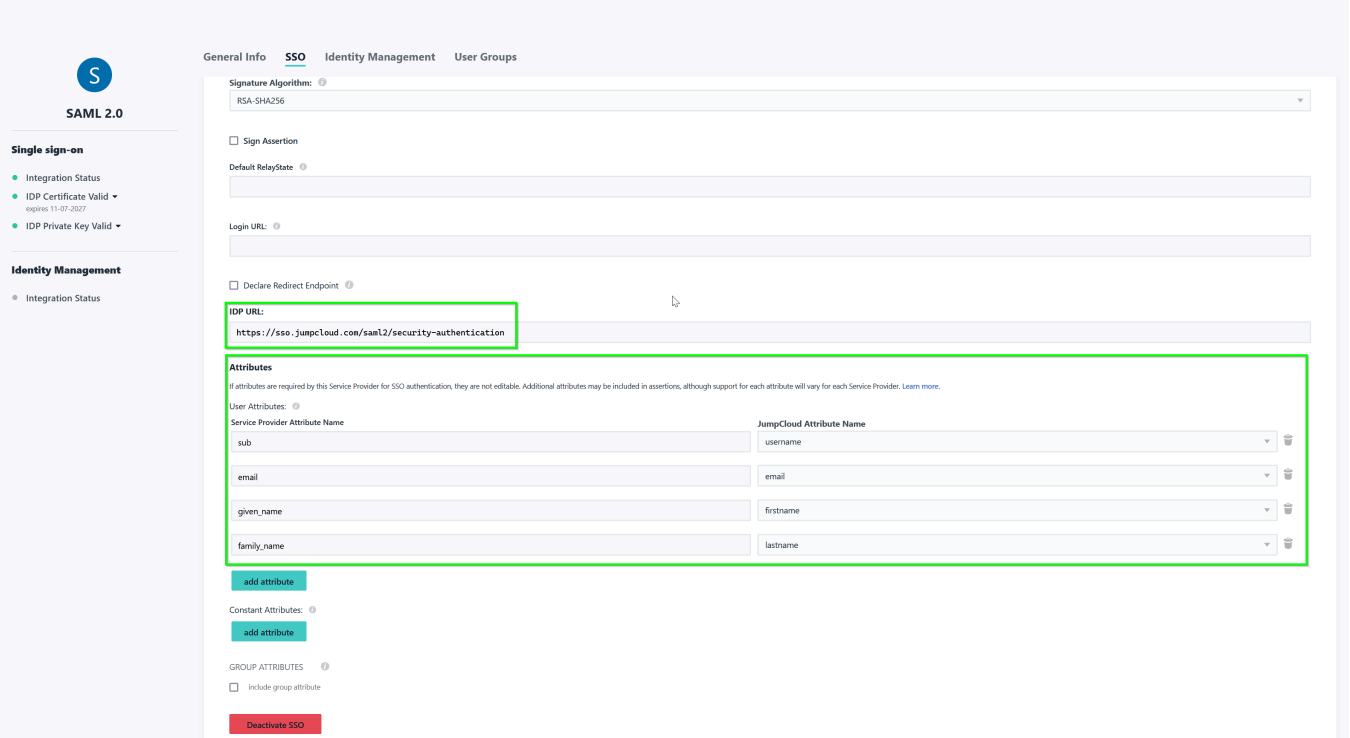


a. **SP Entity ID:** *SPEntityId* property in JumpCloud settings in Security Authentication

b. **ACS URL**: This value must be pointing to the Security Authentication service which will use this IdP.

c. **SAMLSubject NameID**: This property will be username as *username* to find the user in Security Authentication service.

6. *SSOO*

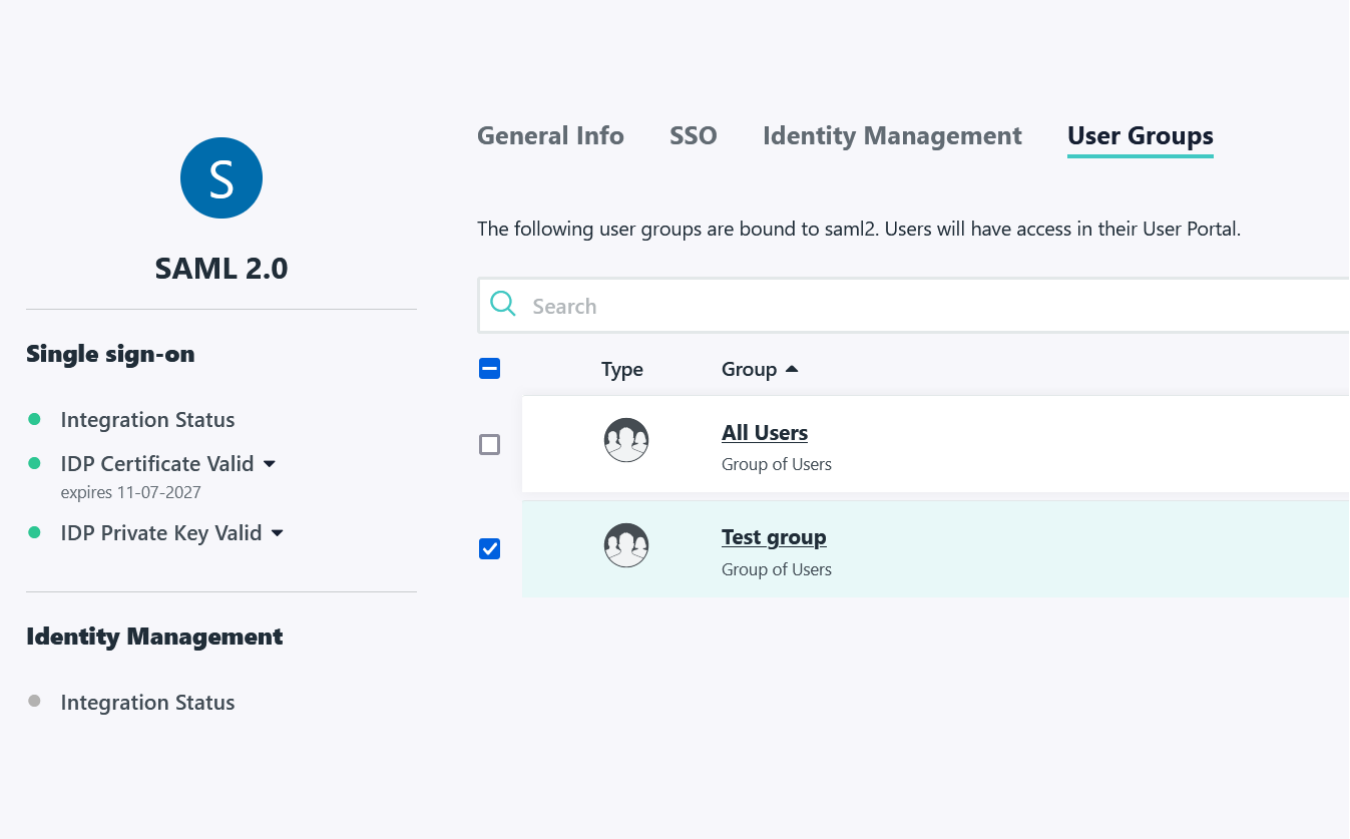


a. **IDP URL**: *LoginURL* property in JumpCloud settings in Security Authentication

b. **Attributes**: These mappings are necessary for matching between JumpCloud's users and Security Authentication's users.

Service Provider Attribute Name (Security Authentication)	JumpCloud Attribute Name
sub	username
email	email
given_name	firsname
family_name	lastname

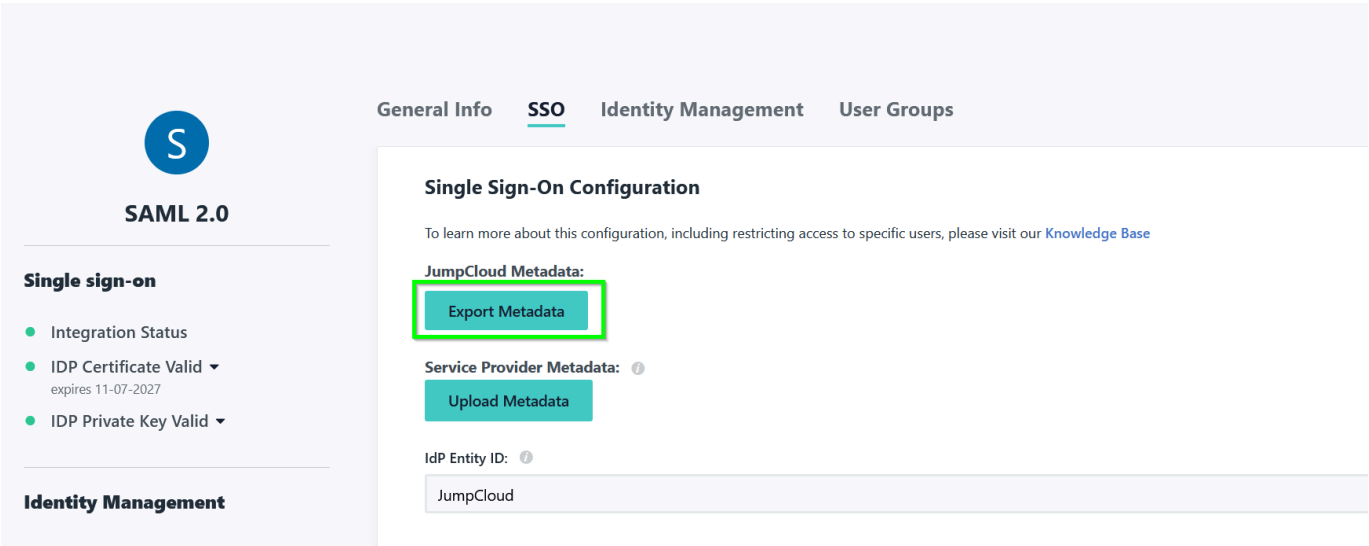
7. *User Groups*. Select which group or groups of JumpCloud users will be able to use this login provider



8. *Save* configuration

Get X509 Certificate info

After complete the SSO Application registering, configuration can be exported as XML:



From the XML file generated:

```
<?xml version="1.0" encoding="UTF-8"?>
<md:EntityDescriptor
    xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata" entityID="JumpCloud">
    <md:IDPSSODescriptor WantAuthnRequestsSigned="false" protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
        <md:KeyDescriptor use="signing">
            <ds:KeyInfo
                xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                <ds:X509Data>
                    <ds:X509Certificate>
                    MIIFfzCCA2egAwIBAgIVAIMEY6UL53rX4K0bwqI0/ybbO5ZfMA0GCSqGSIb3DQEBCwUAMHcxCzAJBgNVBAYTAlVTMQswCQYDVQQIEwJDTzEQMA4GA1UEBxMHQm91bGRlcjESMBAGA1
                    NTE1MDVaFw0yNzExMDcxNTE1MDVaMHcxCzAJBgNVBAYTAlVTMQswCQYDVQQIEwJDTzEQMA4GA1UEBxMHQm91bGRlcjESMBAGA1UEChMJSnVtcENsb3VkMRkwFwYDVQQLExBKdW1wQ2
                    MzjiptCl8WTbUXw9uCTi3gEIAm7xrBEmgS9Bs7WdG+ieQtPLWR3ttHCmFjBuovcBMbF2JOOwJtNW0SxQqGW4ih4TDI19IMcOkNfxeo2noYHEDQTyUhL1BcGutBYtODCTs21CAECiMT
                    uqU/Fw1i+5soI3zujOxSsSrpAzIbA4nWj3+FD1rmhsHS6nXlo+3UoqObBPygfoLC7z2BIYLlzK3cOSLLbYt/N3DRNcuAZ/I6npjIoJNxzN+5iwCXA+aaIyyanJoP6JmrtV3cNvoyYU
                    2aSmAgXc/ynd+WJRW5hzcS3AAxLbrfl4UnyrzcoOqodo4EU9UnChNuK2uBJs8mV8C7VTwg21Ltz+Vo/dB/rtWt8fNqwmmCbRNCQpJ6IeWZ+0oBwO4N1AYBvH22N5v8LH6PA32p2g22
                    whfCxiK7IBsQ7HZaScs6BOU6M0EpY/OfE9emX4Oj5sRJ8Qzuu7QLUM72eLh099q/BDZD0kSAZQtX056FQMRg+iksZmHTo5TACWezZQO/3WixF5v/tCVCYYDtZub6LM/4rQeVLD2PvY
                    ttLq+0dDFnzNUDbwueO18b8T/PUzpOd9dwT65mIREfZP111OWRas2Tb9vVD15ECGmPwI64wCizAAzqBLFxu2elsTsxAHLnX9bq1owTDy4awu9UXgMxqYKWiZSdgZjfDh+0YW/tNuMR
                    8lHga+gtPJ8kzQRjL8LSR6U2/23TBzhl+WHsUWkkSFqXAphKYwv2jdRyfvXlpt1QFZC/DRxgK9t/H++ObQ1AXVD+jkAulpgrQoDyTSWRHeio3QMJe76+ijEhZigfYhCJ6mvygsST0n
                    </ds:X509Certificate>
                </ds:X509Data>
            </ds:KeyInfo>
        </md:KeyDescriptor>
        <md:NameIDFormat>urn:oasis:names:tc:SAML:1.0:nameid-format:unspecified</md:NameIDFormat>
        <md:SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST" Location="https://sso.jumpcloud.com/saml2/security-authentication"/>
    </md:IDPSSODescriptor>
</md:EntityDescriptor>
```

- **X509Certificate**: This certificate must be placed in *X509SigningCertificate* property in JumpCloud settings in Security Authentication.

Registering users for SSO

Once SSO application is registered in JumpCloud, the next step is create the users in Sequel Security Services ensuring users are created using the same email or username provided by JumpCloud in the claims. This email must be stored in the email or SsoUsername fields at the users records.

4.9.5 Okta Authentication Registration

Sequel Security Services allows external providers for *authentication* like Okta. This registration process should be done for each client.

This document describes preliminary steps to be done by Sequel Clients to register a Sequel Application in Okta.

Registering Sequel App at Okta

Registering an application in Okta using OpenId Connect is described at https://help.okta.com/en-us/Content/Topics/Apps/Apps_App_Integration_Wizard_OIDC.htm.

This document tries to provide samples of the process. Please, keep in mind that Okta UI could change since this document was released.

REQUIRED INFORMATION

For registering an application in Okta we need the following information:

Application name

This is a friendly name for the application; we will suggest to use *Sequel Authentication Service*; but this can be changed and also it could include references to production or UAT environments (ie. *Sequel Authentication Service - Production*, *Sequel Authentication Service - UAT*).

Sign-in redirect URI

This is a public URI of the Sequel's authentication service. Usually, it looks like this: *Sequel-AuthN-Base-URI*/signing-okta-oidc (e.g. https://sequel_domain/Authentication/signing-okta-oidc). For this implementation the URI will be:

```
https://TO_BE_CONFIRMED/signing-okta-oidc
```

Redirect URI's are case sensitive.

Sign-out redirect URI

This is a public URI of the Sequel's authentication service. Usually, it looks like this: *Sequel-AuthN-Base-URI*/signout-okta-oidc (e.g.https://sequel_domain/Authentication/signout-okta-oidc). For this implementation the URI will be:

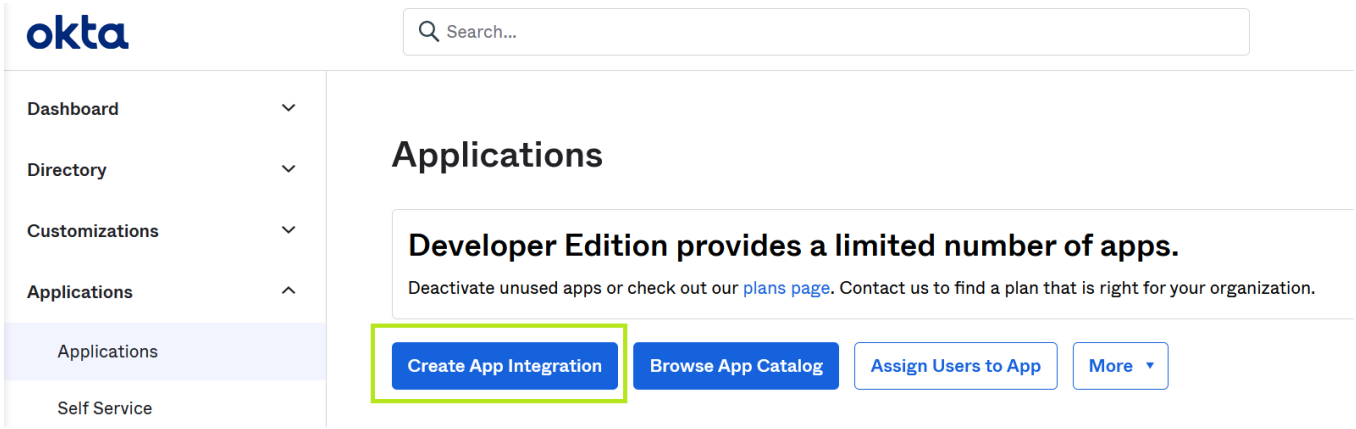
```
https://TO_BE_CONFIRMED/signout-okta-oidc
```

Redirect URI's are case sensitive.

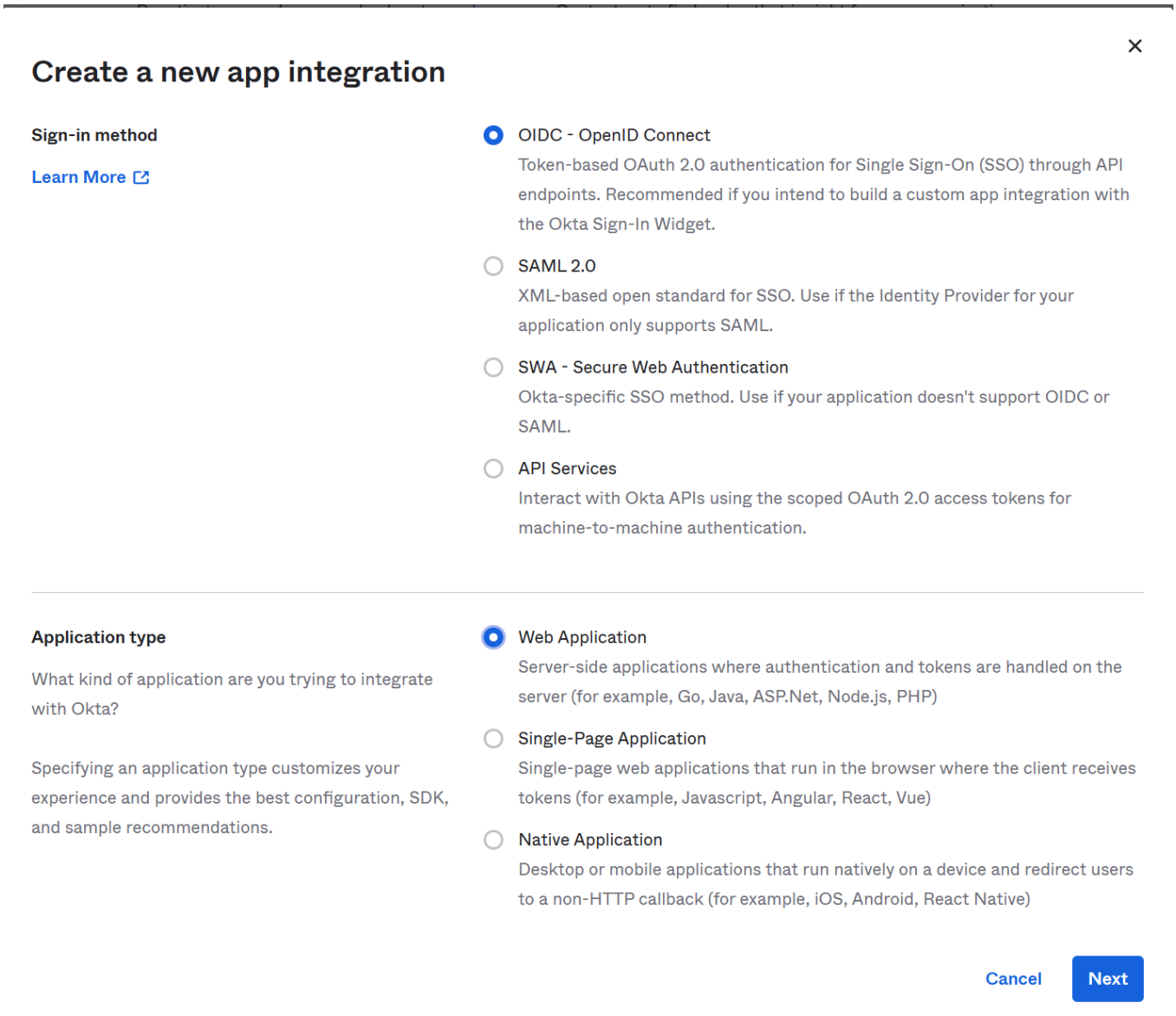
REGISTER AN APPLICATION

As a summary of this process, we will have to perform the following steps:

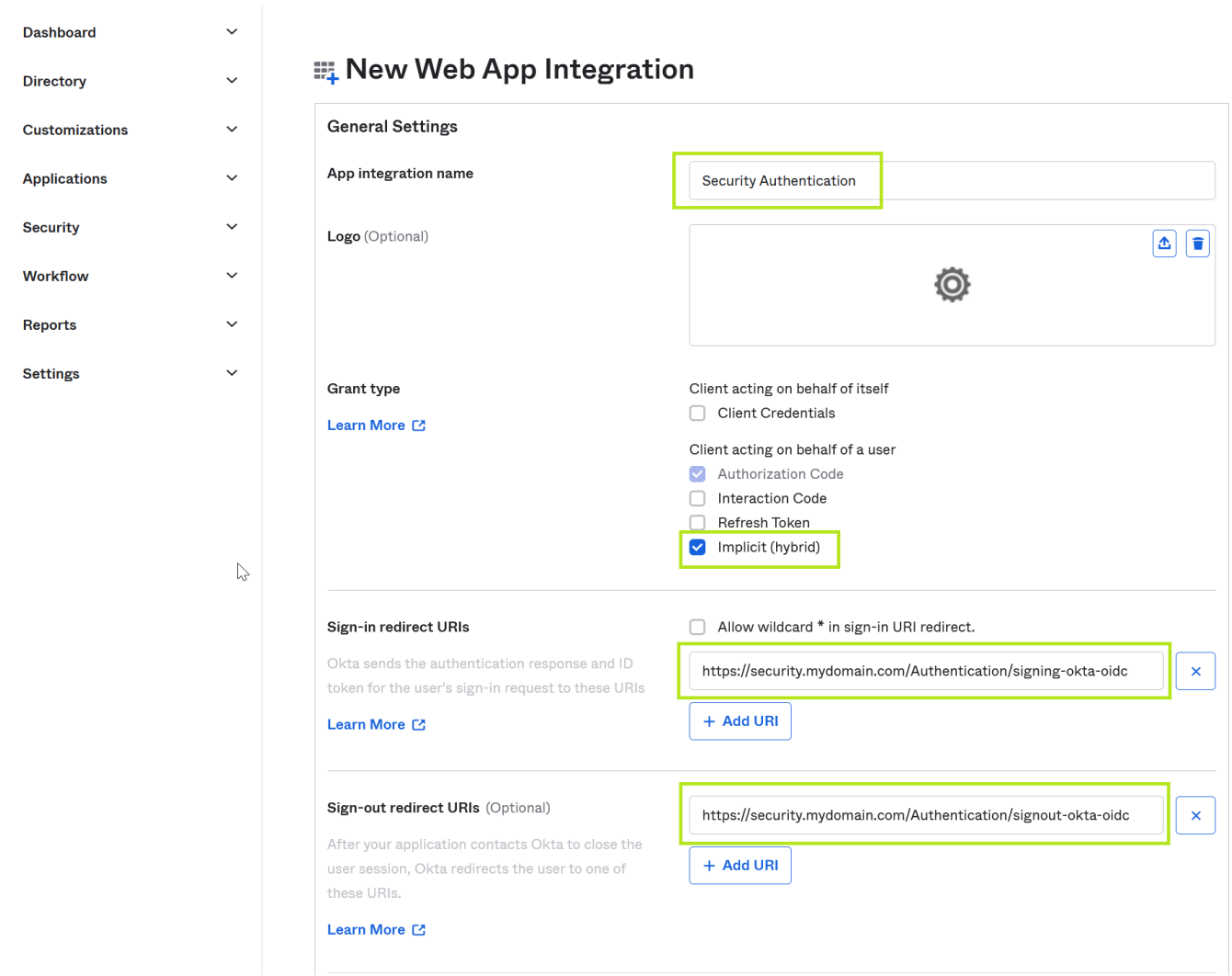
1. Sign in to the Okta domain portal.
2. Select *Applications* > *Applications* > *Create App Integration*.

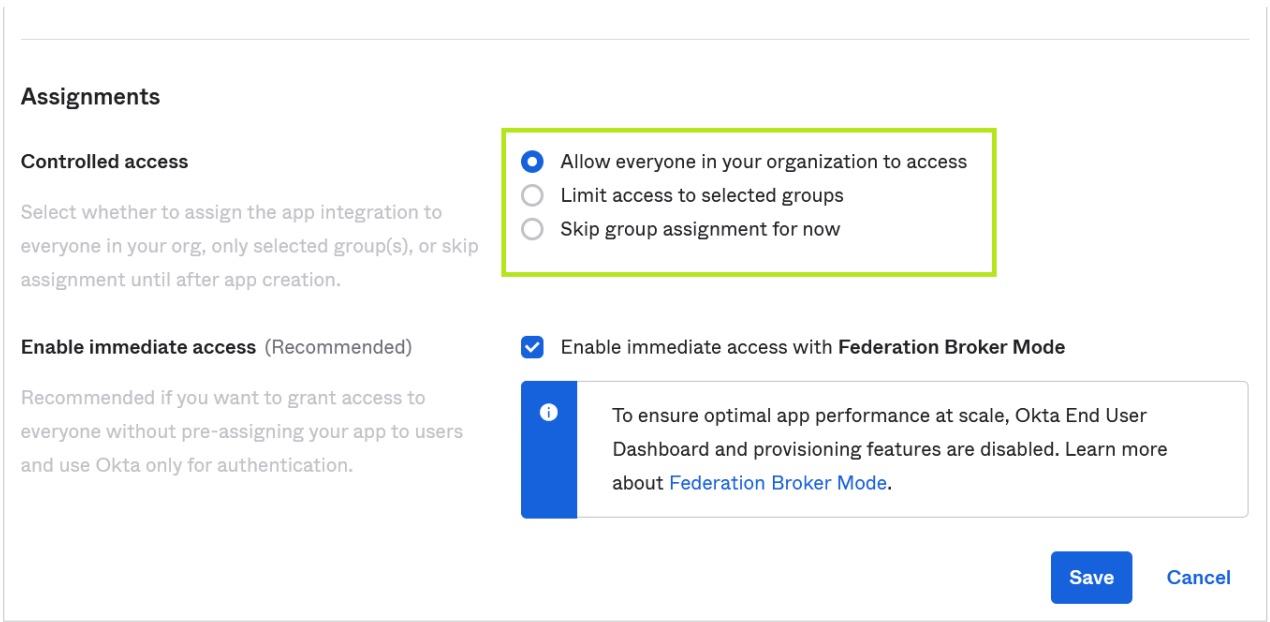


3. In the Create a new app integration select *OIDC - OpenID Connect* and *Web Application* as shown above.

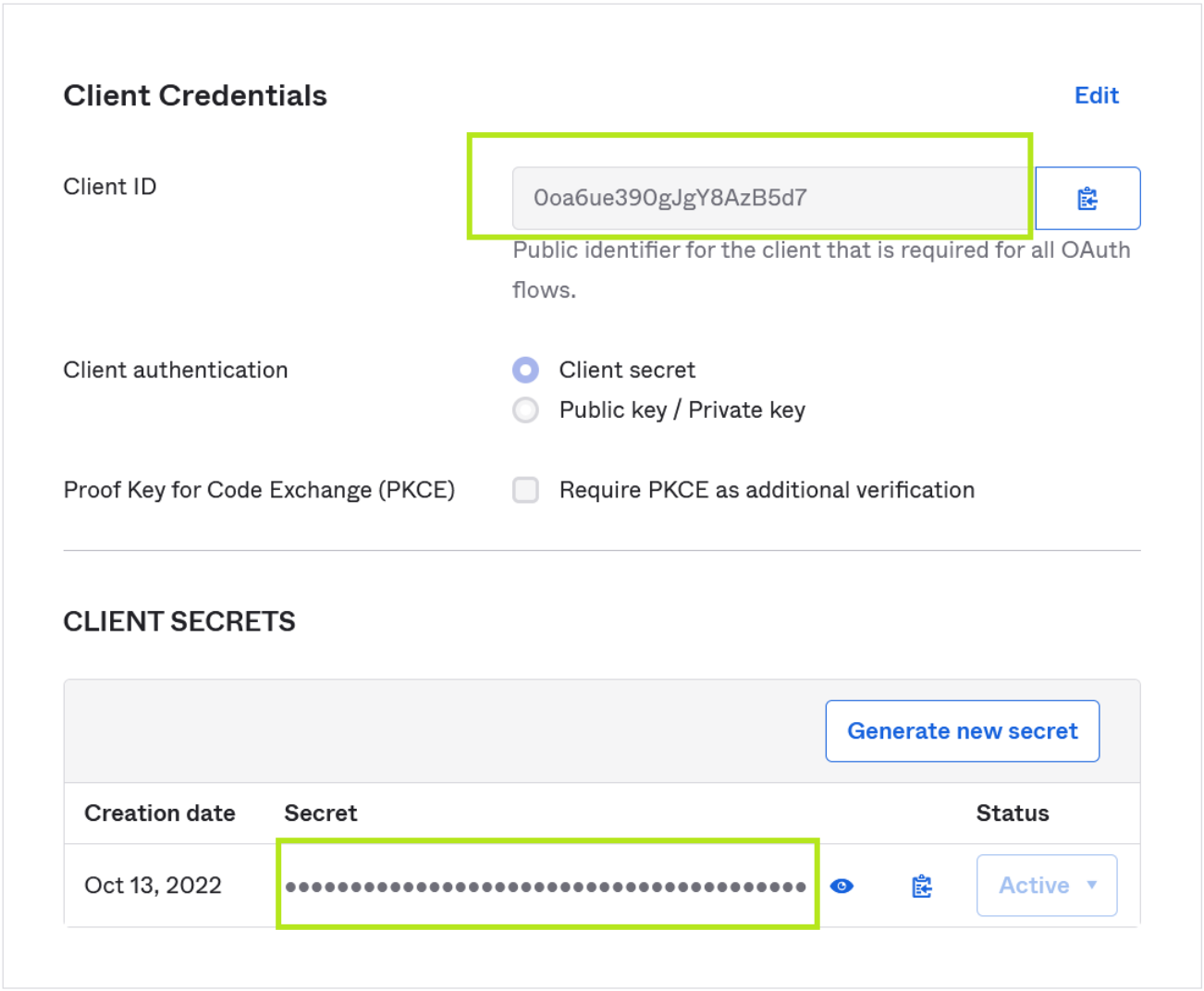


4. Add *App integration name*, check *Implicit grant type* and introduce *Sign-in redirect URI* and *Sign-out Redirect URI* and the desired *Assigment* policy as is shown above and press *Save* button.





1. *Client ID* and *Client Secret* will be needed to complete settings for Okta provider in Security Authentication



Registering users for SSO

Once Sequel App is registered in Okta, the next step is create the users in Sequel Security Services ensuring users are created using the same email provided by Okta in the claims. This email must be stored in the email or SsoUsername fields at the users records.

4.9.6 Verisk's Okta Authentication Registration

Sequel Security Services allows external providers for *authentication* like Okta. This registration process should be done for each client.

This document describes preliminary steps to be done by Sequel to register a Sequel Application in the Veriks's Okta tenant.

Registering Sequel App at Verisk's Okta

Registering an application in Verisk's Okta tenant using OpenId Connect must be requested at https://oneverisk.service-now.com/sp/. You can access from Okta home page https://verisk.okta.com/app/UserHome.

This document tries to provide samples of the process. Please, keep in mind that registration process could be changed, so always follow Verisk indications.

REQUIRED INFORMATION

For registering an application in Okta we need the following information:

Application description

The most important value here is the *application name*. This is a friendly name for the application; we will suggest to use *Sequel Authentication Service*; but this can be changed and also it could include references to production or UAT environments (ie. *Sequel Authentication Service - Production*, *Sequel Authentication Service - UAT*).

There are more information that we can provide to register the application, but these information is complementary and not key for the integration, like:

- *description*
- *logo image*
- *application owner's name*: We recommend to assign as owner to the technical lead resposible of the target environment we are protecting.
- *target users*: from SuppApp team we think that in most of the cases this authentication is going to be used to protect development environments; so the target users probably will be *Internal Employees*.

Integration type

The *integration type* must be **OIDC**; and this will require to provide below information:

- *application type*: **Web**
- *grant types*: we can select multiple options; for this purpose we need **Implicit (Hybrid)**

> **Important**

Redirect URIs are case sensitive.

Login redirect URIs

After Okta authenticates a user's sign-in request, Okta redirects the user to one of these URIs. This is a public URI of the Sequel's authentication service. Usually, it looks like this: *Sequel-AuthN-Base-URI*/signing-okta-oidc (e.g. https://sequel_domain/Authentication/signing-okta-oidc). For this implementation the URI will be:

```
https://TO_BE_CONFIRMED/signing-okta-oidc
```

Logout Redirect URIs

After your application contacts Okta to end the session, Okta then redirects the user to one of these URIs. This is a public URI of the Sequel's authentication service. Usually, it looks like this: *Sequel-AuthN-Base-URI*/signout-okta-oidc (e.g.https://sequel_domain/Authentication/signout-okta-oidc). For this implementation the URI will be:

```
https://TO_BE_CONFIRMED/signout-okta-oidc
```

Login initiated by

The login process can be started from the application itself, as we traditionally do; or be started from the Okta home page. This option allows to register our app on the Okta Hompage (**Either Okta or App**) or just allow start login from the application, that will redirect the flow to Okta (**App only**). Please, select here the option you prefer.

Initiate Login URI

Include a URI to have Okta initiate the sign-in flow. When Okta redirects to this endpoint, the client is triggered to send an authorize request. If you have selected login initiated by **either Okta or App**, you will need it. Our recommendation is to populate this value with you main URL for entering the application. This could be difficult to determine when multiple Sequel apps are deployed (Claims, UW, Broking,...)

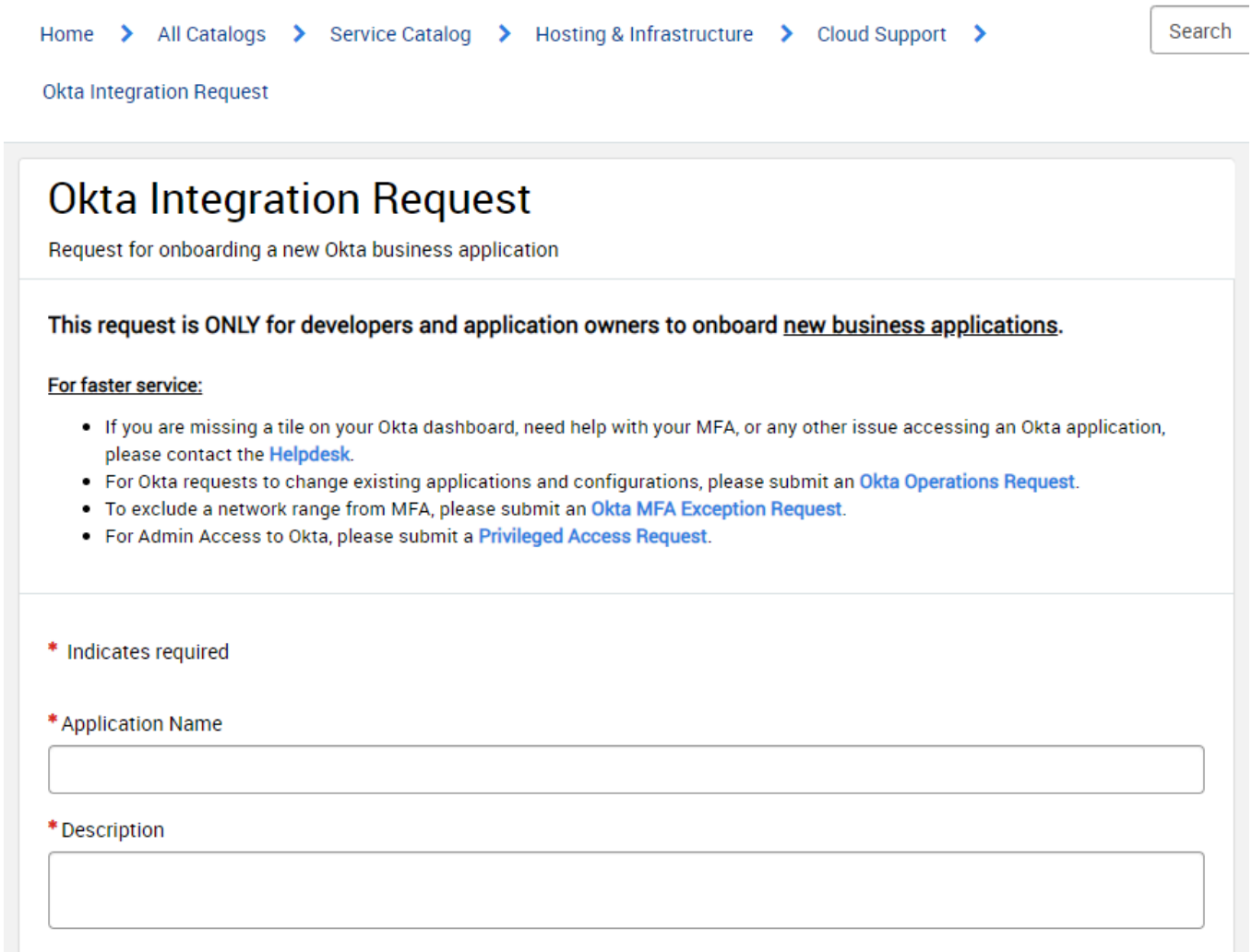
Initial Users to Provision

Provide a comma separated or line delimited list of users' iNumbers to receive access when the integration is established: `i00001, i00002, i00003`

REGISTER AN APPLICATION

As a summary of this process, we will have to perform the following steps:

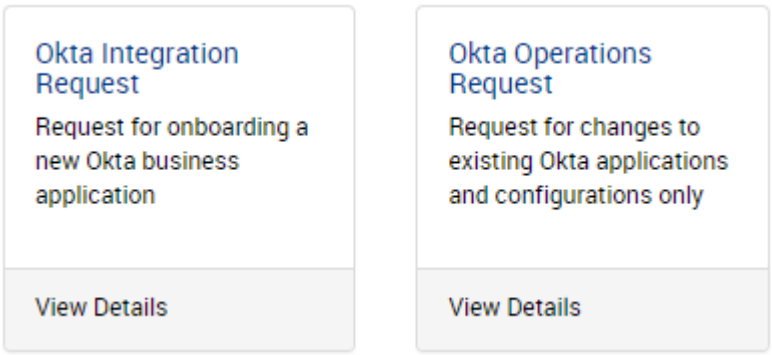
1. From **Verisk's Okta portal**: https://verisk.okta.com/app/UserHome.
2. Go to **servicenow** (https://oneverisk.service-now.com/sp/).
3. Open an **Okta Integration Request**. This could be found by navigating in the catalog to `Home > All Catalogs > Service Catalog > Hosting & Infrastructure > Cloud Support` or searching by *Okta Integration Request*.



4. Populate all information requested as described on Required information. Submit, including information on required approver(s), notes and watch list. At SuppApp Team we are keen to be included as approvers, so we can review settings are right.
5. Verisk will provide you the domain where is configured, the client Id (aka application Id) and the client secret. If they are not provifing you one of those values; please request them.

Verisk's Okta configuration request

In case you could need changes on your Okta configuration, there is a *Okta Operation Request* option on *servicenow* to request changes.



Registering users for SSO

Once Sequel App is registered in Okta, the next step is create the users in Sequel Security Services ensuring users are created using the same email provided by Okta in the claims. This email must be stored in the email or SsoUsername fields at the users records.

TFS configuration

With above information (clientId, client secret and domain), deployment can be configure. See screenshot from a configuration on TFS for automatic deployment:

Name	Value
loginSettings.okta.clientid	0oay4j7mk6j2FWnmf0x7
loginSettings.okta.clientsecret	[redacted]
loginSettings.okta.domain	verisk.okta.com
loginSettings.okta.enable	true
loginSettings.okta.userpolicies.fieldstoupdates...	\"FirstName\", \"LastName\"
loginSettings.okta.userpolicies.matchingfields	\"Email\"

4.10 Troubleshooting

4.10.1 Starting point

This document is written with the intent of suggesting tips for **troubleshooting** issues in production and record known issues.

Tips

Some general tips for troubleshooting issues in security services are:

Know the architecture, know the deployment

First of all, it is quite important to have a clear idea (an a diagram) about how security have been deployed in this environment (servers, load balancers, private and public domains, ...) and how different components interact between them. Please, become familiar with security architecture documentation.

Health checks

Authentication, Authorization and API services exposes a `/health` endpoint that can be used to troubleshoot issues. A complete report is provided when calling to `/health?diagnostic` endpoint with an authenticated users or passing a key in the header `healthcheck-apikey` that matches the one configured for this service in the `appsettings.json` file at `HealthCheckSettings.ApiKey`.

Use `/health` endpoint for checking if service is alive; for example from load balancers.

Logs

All logging traces at different levels from Authentication, Authorization and API services are collected at logging database. Please ensure this setting is properly done and the level of traces is the required; if this setting is wrongly configured this will be notified by the health endpoint and no traces will be created in the logging database.

Local log file

In some cases, when any of the services even starts, we can investigate the issue enabling the local log files. This can be done editing the `web.config` file; at `location>system.webServer>aspNetCore`, set `stdoutLogEnabled` to true and `stdoutLogFile` to the folder where the log file must be created. Restart the IIS pool to reload this settings.

```
<location path="." inheritInChildApplications="false">
   <system.webServer>
     <handlers>
       <add name="aspNetCore" path="*" verb="*"
            modules="AspNetCoreModule" resourceType="Unspecified" />
     </handlers>
     <aspNetCore processPath="dotnet" arguments=".\Sequel.Security.Api.dll"
                 stdoutLogEnabled="false"
                 stdoutLogFile=".\logs\stdout" />
   </system.webServer>
 </location>
```

4.10.2 General common issues

Application does not start and restarts on each request

Problem

Application does not start and the below error is in the event viewer:

There was an error during processing of the managed application service auto-start for configuration path: 'Machine/WebRoot/AppHost/Default Web Site/Authentication'. The error message returned is: ". The worker process will be marked unhealthy and be shutdown. The data field contains the error code.

Checking the installed program we can check that Host ASP.NET Core on Windows with IIS (Microsoft .Net Core xxxx - Windows Server Hosting) is not installed of does not match to the .net core version installed.

Solution

Review platform specification document and install ".NET Core Hosting Bundle".

All sessions are automatically logout after requesting the user to keep the inactive session open

Problem

Login to multiple applications (Origin, Workflow, Product Builder) and keep browser tabs open without activity; at some moment the inactivity popup is displayed to keep session open in any of the tabs: click on Stay log.

The session remains open in this application (even multiple tabs are open); however, in other applications the session is not refreshed and after a period of time this application will trigger a log out event that will close the session all tabs.

If checking the network in the browser we can detect that set-cookie action (for SQ.soo cookie) in the response headers contains a warning indicating "The Set-Cookie was blocked because its Domain attribute was invalid with regards to the current host url". In this case, the domain was configured with internal domain, instead of the external domain.

Solution

Review CookieDomain attributes in the affected applications to ensure matches with the public host URL.

4.10.3 Admin site issues

Reload page fails

Problem

Reload a page fails or reloads the home page. The URL contains a # character before the name of the page. This issue occurs just on Windows/IIS deployments.

Solution

The URL Rewrite module must be installed on the Windows Server.

Related information

This issue is not applicable when application is deployed in containers.

4.10.4 Security API issues

VIEW SERVER STATE permission was denied

Problem

The service account does not have **View Server State** permissions on the SQL Server. This error is related to the requirement on earlier versions of security of having this permission. The complete error message is:

- Microsoft.EntityFrameworkCore.DbUpdateException: An error occurred while updating the entries. See the inner exception for details. ---> System.Data.SqlClient.SqlException: VIEW SERVER STATE permission was denied on object 'server', database 'master'.

Solution

Provide the **View Server State** permission to the Active Roles service account.

From the SQL Server Management Studio via GUI:

1. Connect to the SQL instance
2. Navigate to **Security** | **Logins**
3. Right Click the ARS Service Account | Select **Properties**
4. Select the **Securables** page
5. In the bottom pane, scroll to the bottom and **Grant** - **View Server State**

From the SQL Server Management Studio via statement:

1. Connect to the SQL instance
2. Select **New Query**
3. Use the following statement: `GRANT VIEW SERVER STATE TO "AccountName"`

Related information

This issue is no longer applicable in newer versions as access to master database is not required.

Error sending email related to SSL or TLS

Problem

You get the following exception as in some AWS deployment: *MailKit.Security.SslHandshakeException: An error occurred while attempting to establish an SSL or TLS connection*

Solution

You should change "SecureSocketOptions" to None.

4.10.5 Authentication issues

Session automatically closed after login

Problem

After a successful login the session is closed in a few seconds.

Solution

This problem has been detected when the generated cookies are not trusted, this can be an issue in scaled-out installation where the data protection is not properly configured and the different instances are not sharing the keys to sign cookies, tokens and other signed items (aka data protection). To solve this issue the best solution is configure the Authentication server to use the Database for managing the data protection (`DataProtectionSettings`).

Related information

n/a

Entering valid credentials doesn't redirect back to application

Problem

Trying to login to an application A the user is redirected to the authentication server, entering valid credentials doesn't return to application A and authentication server ask for credentials again.

Solution

This problem is caused when `LoginSettings.AuthenticationTicketExpiration` property is not present in `appsettings.json` . This could be cause when it's being used an old version of authentication 's `appsettings.json` in a newer version of authentication. To solve this, use the right version or include the property in `appsettings.json` :

```
"LoginSettings": {
    "AuthenticationTicketExpiration": "7.00:00:00"
```

Related information

n/a

Unable to login: Invalid redirect_uri / unauthorized_client

Problem

Trying to login to an application A, the user is redirected to the authentication server and below error is displayed: `Invalid redirect_uri / unauthorized_client`

Solution

When a user is redirected to the login page, the application A includes two values in the query string of the redirection: `client_id` and `redirect_uri` . Those values have to match with the configuration in the Authentication service. The related information is stored in the tenant database in the `[authentication].[Client]` and `[authentication].[ClientRedirectUri]` . Please, ensure values are exactly the same; a client can contain multiple entries in the `[authentication].[ClientRedirectUri]` table.

Related information

n/a

Unable to login: Invalid client / invalid_client

Problem

Trying to login to an application A, the user is redirected to the authentication server and after completing the login is redirected to the caller application and application fails with in internal server errors. Accessing to the logs, below error appears:

`fail: Microsoft.AspNetCore.Authentication.OpenIdConnect.OpenIdConnectHandler. Message contains error: 'invalid_client', error_description: 'error_description is null', error_uri: 'error_uri is null', status code '400'.`

Solution

Ensure client is providing the same secret (usually stored in the appsettings file encrypted) than the one stored at `[authentication].[ClientSecret]` (as a hash).

Related information

n/a

Session automatically closed after login

Problem

After a successful login the session is closed in a few seconds.

Solution

This problem has been detected when the generated cookies are not trusted, this can be an issue in scaled-out installation where the data protection is not properly configured and the different instances are not sharing the keys to sign cookies, tokens and other signed items (aka data protection). To solve this issue the best solution is configure the Authentication server to use the Database for managing the data protection (`DataProtectionSettings`).

Related information

n/a

Unable to logout

Problem

The logout action from an application ends in *MyApplication* page with the user still logged in. Even if *LogOff* link is clicked, the user doesn't log out.

Solution

This problem is caused due to how some Authentication's cookies are managed by some browsers, specially by Google Chrome from its version 80. To solve it `SameSiteCookiePolicyDisabled` must be set to *false*.

Related information

n/a

Issuer validation failed

Problem

`IDX10205: Issuer validation failed. Issuer: '.awsveriskt.local'. Did not match: validationParameters.ValidIssuer: 'null' or validationParameters.ValidIssuers: '.office.sbs'.` in a secured application after performing a successful login.

Solution

We have seen this error on environment not configured properly:

- Stale cookies: when a first token was created using a value of `CookieDomain` that was changed later. In this case, we will suggest to delete all cookies stored in the browser.
- Scale-out wrongly configured: in scaled-out configurations where authentication instances are not configured with the same Issuer.

Related information

One value is coming from the discovery document in the Authentication service, and one value is coming from the token. The token must be generated by the same Authentication service. If they have not been generated by the same. Authentication service reads the issuer name from `CookieDomain` setting.

Nonce error

Problem

After enter credentials the return to application shows next error and login process can't be completed:

```
IDX10311: RequireNonce is 'true' (default) but validationContext.Nonce is null. A nonce cannot be validated. If you don't need to check the nonce, set OpenIdConnectProtocolValidator.RequireNonce to 'false'.
```

Solution

The reason why `IDX10311: RequireNonce is ‘true’ (default) but validationContext.Nonce is null` error occurs is that a cookie is missing or altered during the authentication process.

This issue might be perfectly expected for certain scenarios such as

- Using browser’s Back button
- Trying to login after a long time of inactivity (this case is solved by Authentication server)
- Using the site in multiple tabs

If the scenario one of these cases, the best way to proceed is to implement a code block to catch this exception and inform/redirect the user accordingly.

Related information

n/a

4.10.6 Authorization issues

Service returns HTTP 503 ServiceUnavailable

Problem

Authorization service returns HTTP 503 Service Unavailable error to all request. The response contains below message in the body:

Error calling Authorization server.Status code: ServiceUnavailable, Url: {ServerUrl} , Error: Error on startup, see health checks for details.

Solution

As described by the error, check the response from the health checks endpoint. Usually the error is related to connectivity problems with the message bus. Typical causes of the problem:

- Misconfiguration in the connection settings: URI, user name and password.
- Network errors: server is not reachable.
- RabbitMQ is not working properly, allows to start the connectivity but fails before accepting to complete the connection; causing an error in the caller. Check health on RabbitMQ.
- If security is stored in containers (ECS) message bus will notify RabbitMQ that the queues and exchanges are going to be temporal but RabbitMQ will not change his topology unless you delete the related queues and exchanges and re-create them again.

4.10.7 Security tool issues

Common issues using the `sequel-security` tool are covered on this section.

Invalid Tables exception using sequel-security

Problem

When a command is executed within `sequel-security` tool throws a Invalid Tables exception similar to:

```
Invalid option '--connection' value 'SECURITY DB CONNECTION: {ConnectionString} EXCEPTION: Invalid Tables: ApiClaim, ApiResource, ApiScope, ApiScopeClaim, ApiSecret,
Client, ClientClaim, ClientCorsOrigin, ClientGrantType, ClientIdPRestriction, ClientPostLogoutRedirectUri, ClientProperty, ClientRedirectUri, ClientScope,
ClientSecret, IdentityClaim, IdentityResource, PersistedGrant'
```

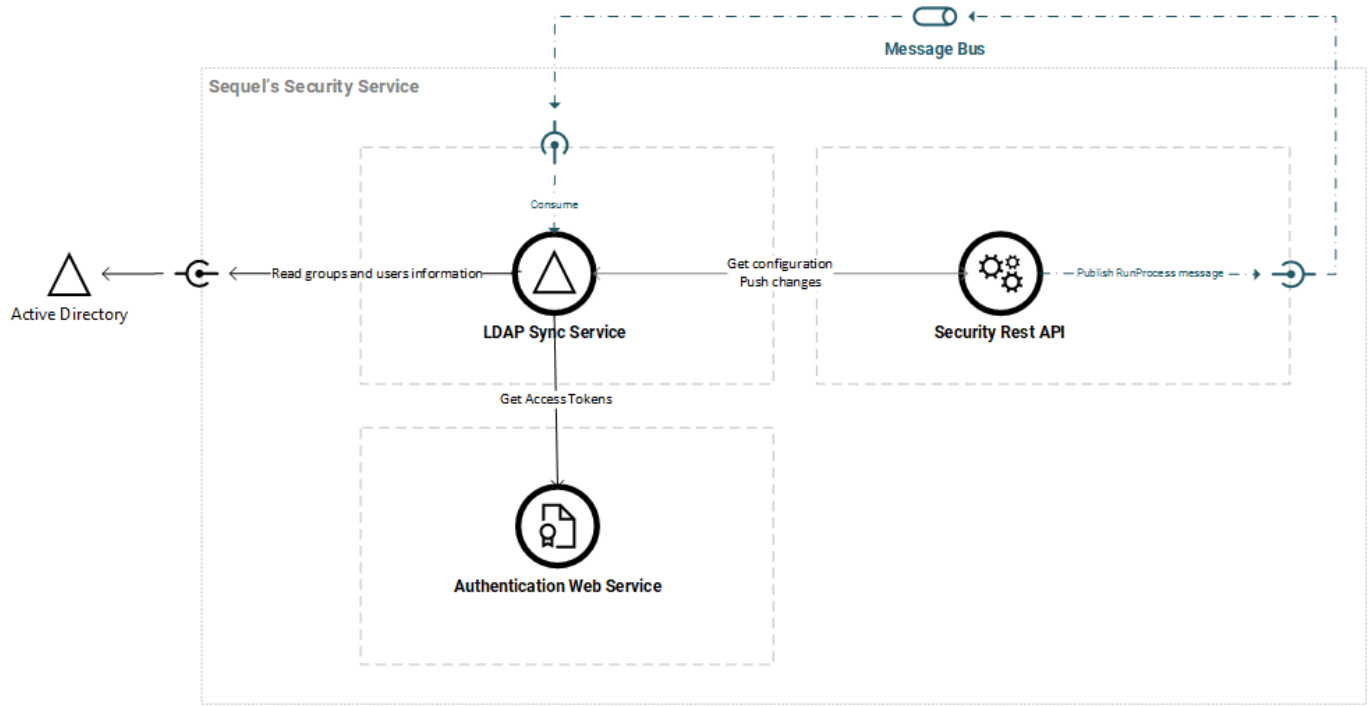
Solution

This error is produced when the `sequel-security` tool is obtained from Security v1.XX.YYYYY and database was created using a Security v2.XX.YYYYY (or vice versa). The `sequel-security` version used must be compatible with the Security database version.

Related information

n/a

4.10.8 LDAP Sync issues



How-to troubleshoot

LDAP Sync Service *is not synchronizing data from the Windows AD, even when manually requested from the Security Administration site.*

Troubleshooting issues on *LDAP Sync Service* when is not working should be approached with the architectural design in mind. Below, we present a sequence of steps to be followed in order to triage an issue.

1. Ensure LDAP Sync Windows Service **is installed**.
    - *Security Sync service* and *LDAP Sync Service* cannot be installed on the same server. *Security Sync service* should be installed on an Origin/Claims server.
2. Ensure LDAP Sync Windows Service **is running**.
3. Review potential **error logged**. Points 4, 5 and 6 covers the most probable source of those errors. Logs can be found at:
    - At filesystem, by default at *C:\TEMP\logs\sequel-security-ldap-sync\log-{Date}.txt* (check `RollingFileLoggingSettings` at appsettings.json file. Logs during start-up.
    - At the associated *Sequel.Core.Logging* database (check `MsSqlLoggingSettings` at appsettings.json file). Logs after start-up.
4. Review **RabbitMQ** configuration:
    - Check on LDAP service, the appsettings.json file: `MessageBusSettings` entry should be configured to use the same virtual host than the rest of the environment.
    - Access to RabbitMQ console and review **messages are published by Security API** to queue `LdapSyncRunProcess`, associated to message type `Sequel.Security.MessageBus.Contracts.LdapSync.v1.RunProcess`; when manual sync is forced from the Administration site.
    - Access to RabbitMQ console and review **messages are consumed by the LDAP Sync service** from queue `LdapSyncRunProcess`, associated to message type `Sequel.Security.MessageBus.Contracts.LdapSync.v1.RunProcess`.
5. **Security services configuration**:
    - Review `ServiceDiscoverySettings.RequiredServices[Authentication]` and `AuthenticationSettings` at appsettings.json file, for connectivity with Security Authentication Service.
    - Review `ServiceDiscoverySettings.RequiredServices[SecurityApi]` at appsettings.json file, for connectivity with Security API Service.
6. **Windows AD integration**:
    - Review `LdapConnection` at appsettings.json file.
        - Do we have network connectivity to the defined `Host` and `Port`?
        - Credentials issues? Did we check the `DN` and `Password` values?
        - Errors running LDAP queries? In this case, go to Security Administration to review the configuration

4.10.9 Legacy Sync Service

Legacy database is not synchronized

Problem

The security legacy database is not synchronized when changes are done in Security API

Solution

There are different potential issues that could cause this problem:

- Security API is not publishing messages to the service bus. Please review configuration, and state of message bus (up and running)
- Legacy Sync Windows Service fails during start-up. Check log file generated in the same folder where the service is deployed.
- Legacy Sync Windows Service logs: during start-up the log is done in file and as soon as connectivity to Core.Logging is ready the logging is moving there. Please, ensure logging connection string and settings are right
- Legacy Sync Windows Service is not consuming messages from the service bus. Ensure the connection to the bus is right and pointed to the same message bus and virtual host than the associated Security API.
- Legacy Sync Windows Service doesn't have access to the legacy database; ensure the connection string to the legacy security host application (Claims or Origin databases) are properly configured.

5. Product handbook

5.1 Product's handbook

Sequel's security services offers a central place to manage **authentication** and **authorization** concerns for different **applications**. The concept of application is key for the security services as most of the configuration is structured around this concept. This configuration can be done using the *Security Rest API*, the *Security Administration website* or the **sequel security tool**.

5.1.1 Authentication and Authorization

- **Authentication (AuthN)** is the process of verifying **who you are**. When you log on to a system with a user name and password you are authenticating.
- **Authorization (AuthZ)** is the process of verifying **what you can do**. Gaining access to a resource because the permissions configured on it allow you access is authorization.

Sequel's authentication implementation is based on the emerging leading standard for single sign-on and identity provision on the Internet: OpenIdConnect; this is explained with more detail at authentication article. The authorization implementation is a role based access control (RBAC) system designed by Sequel, it is wider described in authorization article.

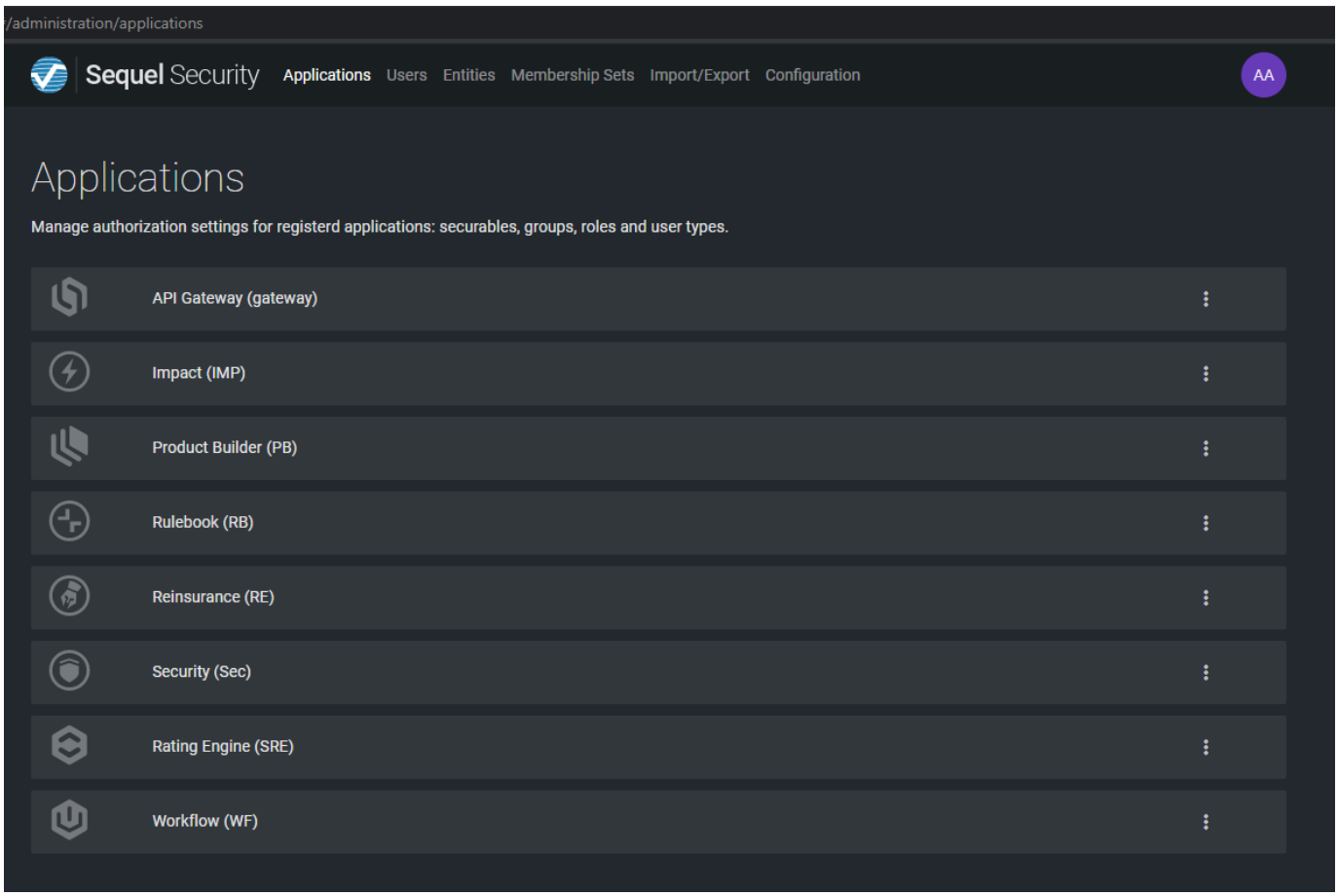
5.1.2 Definition of Application

Represents each application registered in the system for providing security, with all its configurations related to authentication and authorization. In terms of Identity Server terminology, an application could be considered a resource. An application is each principal module we can decompose Sequel's suit, e.g. API gateway, Claims, Origin, Rulebook, Impact, RE, Product Builder, Workflow, Rating Engine, Security Management (itself),... For us, an application is a product or a service part of a bigger system.

An application is defined by two properties:

Property	Description
Key	String. Unique Key to refer the Application. Required
Description	String. Friendly name to refer the Application. Required

Applications are not editable from API or user import process. However, the application is the central place for managing the configuration:



Application keys

All applications needs a `Key` that must be unique and short in the sequel ecosystem. In order to avoid collisions between different applications we will enforce a naming convention. The current application key assignation is displayed below (in bold are keys in use and in italic are reserved ones for future):

Application	Application Key
Broking	*BRK*
Claims	**CLM**
Document Service	*DOC*
API Gateway	**GWY**
Impact	**IMP**
Origin	**ORIGIN**
Product Builder	**PB**
Reinsurance	*RE*
RuleBook	**RB**
Security	**SEC**
Rating Engine	**SRE**
Underwriting	*UW*
Workflow	**WF**

Also, all objects managed by security that are defined in the boundaries of an application (groups, roles, clients, resources,...) must define the Key following this pattern:

```
ApplicationKey + "." + Code
```

Where `Code` is the identifier of each instance of this type of object.

5.1.3 Audit logs

Most of the models used in **authentication** and **authorization** schemas are auditable storing the information at:

- Last `UpdateUser` and `UpdateDate` in the affected record.
- Detailed track of changes in update and delete actions at `[authentication].[AuditLog]` and `[authorization].[AuditLog]`.

5.2 Security Rest API

The security service was designed to be **API first**:

- All operations are exposed from API
- All APIs are documented with OpenAPI (Swagger)

5.2.1 API organisation

The API is organised into two main URL and four specifications.

URLs

API resources are under `Authentication` or `Authorization` URLs. Those names are not accurate today as the API as evolving; originally, the idea were to use `Authentication` for internal usage for *Authentication Web Service* and `Authorization` for managing the authorization and users domains. Currently, this has change slightly and we should read them as:

- **Private API resources** are under `Authentication` URLs. Security is based on API-key.
- **Public API resources** are under `Authorization` URLs. Offers operations for managing all domains. Security is based on OAuth2 (authorization code or client credentials grant types).

OpenAPI Specifications

AUTHENTICATION

URLs defined under `Authentication`. As mentioned before, this is an internal set of resources for providing service to *Authentication Web Service*. Security is based on API-key.

AUTHORIZATION

URLs defined under `Authorization`. As mentioned before, this is the **public** set of resources for providing full access and management of all domains (authentication is just partially offered). Security is based on OAuth2 (authorization code or client credentials grant types).

DATAEXCHANGE

URLs defined under `Authorization`. This is the **public** set of resources for providing data exchange operations: export, import, trigger sync, entity bulk inserts and LDAP sync.

HEALTHCHECK

Resource defined at `/health`. It the public health check endpoint; it can be accessed without credentials and the current state without details will be returned. When called passing the *healthcheck-apikey* or with an *authenticated user* the response contains a detailed report of all check done and their state. This endpoint is developed with *Sequel.Core.HealthCheck*.

5.3 Authentication

5.3.1 Authentication Model

Overview

Under the authentication model we include all domain models required to manage the identification and authentication process based on OpenIdConnect. For this reason, the authentication model contains information highly related to deployment and environment; how, clients interacts with resources and where those clients and resources are deployed is described by this model. References to URIs and Origins are quite common here; and we are persisting these information in database (authentication schema).

There are four domain models: *api resources*, *identity resources*, *clients* and *persisted grants*.

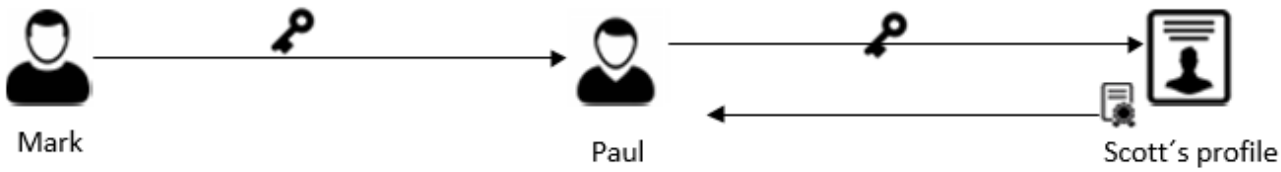
However, for continuing is important to define some terminology and ensure some technical background.

The oAuth 2.0 Authorization Framework

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf.

WHY NOT JUST OAUTH

OAuth has not notion of identity; actually, “oAuth is an access granting protocol, also known as delegation” . This is managed with an access token. Let’s seen an example:



“Mark grants access to Paul to access to Scott’s profile. That doesn’t mean that Mark is Paul, or Paul is Scott”. However, some sites does this mistakes and allows to impersonate users.

To fulfil this gap, OpenID Connect introduces the concept of ID Token.

OPENID CONNECT 1.0

OpenID Connect 1.0 is a **simple identity layer on top of the OAuth 2.0 protocol**. It allows Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User in an interoperable and REST-like manner.

OpenID Connect allows clients of all types, including web-based, mobile, and JavaScript clients, to request and receive information about authenticated sessions and end-users. The specification suite is extensible, allowing participants to use optional features such as encryption of identity data, discovery of OpenID Providers, and session management, when it makes sense for them.

DUENDE IDENTITYSERVER

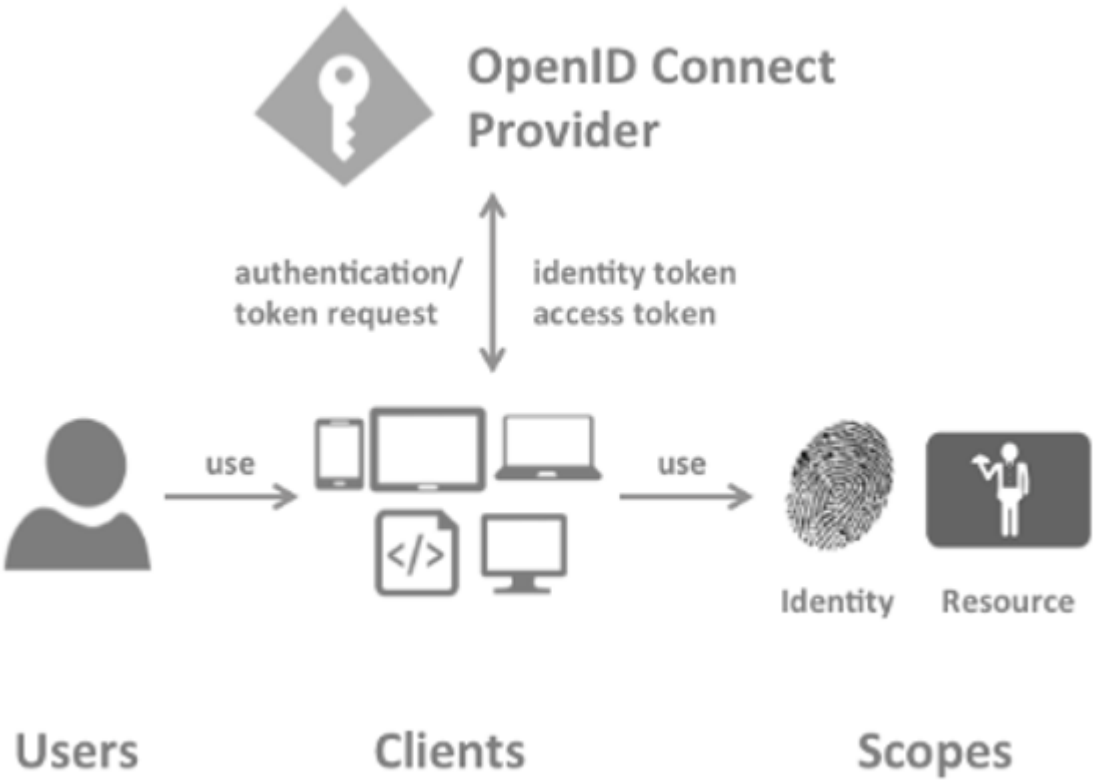
Duende IdentityServer v6 is an OpenID Connect and OAuth 2.0 framework for ASP.NET Core. This is the base open source project we have used to implement our “identity server” (authentication server). Different literature uses different terms for describing the same role that this service provides: *security token service*, *identity provider*, *authorization server*, *IP-STS* and more. But they are in a nutshell all the same: a piece of software that issues security tokens to clients.

It enables the following features in our applications:

- **Authentication as a Service**. Centralized login logic and workflow for all of your applications (web, native, mobile, services). Duende IdentityServer is an officially certified implementation of OpenID Connect.
- **Single Sign-on / Sign-out**. Single sign-on (and out) over multiple application types.
- **Access Control for APIs**. Issue access tokens for APIs for various types of clients, e.g. server to server, web applications, SPAs and native/mobile apps.
- **Federation Gateway**. Support for external identity providers like Azure Active Directory, Okta, JumpCloud, etc. This shields your applications from the details of how to connect to these external providers.
- **Focus on Customization**. The most important part - many aspects of Duende IdentityServer can be customized to fit your needs. Since Duende IdentityServer is a framework and not a boxed product or a SaaS, you can write code to adapt the system the way it makes sense for your scenarios.
- **Mature Open Source**. Duende IdentityServer uses the permissive Apache 2 license that allows building commercial products on top of it. It is also part of the .NET Foundation which provides governance and legal backing.

For .Net Framework applications we are using components from IdentityServer3; as this is the version targeting .net framework.

Terminology



This section is adapted from https://docs.duendesoftware.com/identityserver/v6/overview/terminology/.

USER

A user is a human that is using a registered client to access resources.

CLIENT

A client is a piece of software that requests tokens from authentication server - either for authenticating a user (requesting an identity token) or for accessing a resource (requesting an access token). A client must be first registered with authentication server before it can request tokens. Examples for clients are web applications, native mobile or desktop applications, SPAs, server processes etc.

RESOURCES

Resources are something you want to protect with Duende IdentityServer - either identity data of your users, or APIs. Every resource has a unique name - and clients use this name to specify to which resources they want to get access to.

Identity data: claims

Identity information (aka claims) about a user, e.g. name or email address.

Identity resources are data like user ID, name, or email address of a user. An identity resource has a unique name, and you can assign arbitrary claim types to it. These claims will then be included in the identity token for the user. The client will use the scope parameter to request access to an identity resource.

The OpenID Connect specification specifies a couple of standard identity resources. The minimum requirement is, that you provide support for emitting a unique ID for your users - also called the subject id. This is done by exposing the standard identity resource called *openid*. Supported scopes defined by specification are openid, email, profile, telephone, and address; however, we are just using *openid* and *profile*.

APIs

APIs resources represent functionality a client wants to invoke - typically modelled as Web APIs, but not necessarily. To allow clients to request access tokens for APIs, we need to define API resources; and for getting access tokens for APIs, we also need to register them as a scope. So, at least each resource needs to define one scope.

TOKENS

Identity Token

An identity token represents the outcome of an authentication process. It contains at a bare minimum an identifier for the user (called the sub aka subject claim) and information about how and when the user authenticated. It can contain additional identity data.

Access Token

An access token allows access to an API resource. Clients request access tokens and forward them to the API. Access tokens contain information about the client and the user (if present). APIs use that information to authorize access to their data.

GRANT TYPES

Grant types are a way to specify how a client wants to interact with the authentication server. The OpenID Connect and OAuth 2 specs define the following grant types: Implicit, Authorization code, Hybrid, Client credentials, Resource owner password, Refresh tokens and Extension grants.

In our current implementation we are using *client credentials*, *authentication code* and *hybrid*; in the first implementation we were using *implicit* as well, but we stopped using it due to security risks. So we will focus on those grant types:

Client credentials

This is the simplest grant type and is used for server to server communication - tokens are always requested on behalf of a client, not a user. With this grant type you send a token request to the token endpoint, and get an access token back that represents the client. The client typically has to authenticate with the token endpoint using its client ID and secret.

Authorization code

Authorization code flow was originally specified by OAuth 2, and provides a way to retrieve tokens on a back-channel as opposed to the browser front-channel. It also support client authentication. While this grant type is supported on its own, it is generally recommended you combine that with identity tokens which turns it into the so called hybrid flow. Hybrid flow gives you important extra features like signed protocol responses.

Hybrid

Hybrid flow is a combination of the implicit and authorization code flow - it uses combinations of multiple grant types, most typically code id_token. In hybrid flow the identity token is transmitted via the browser channel and contains the signed protocol response along with signatures for other artefacts like the authorization code. This mitigates a number of attacks that apply to the browser channel. After successful validation of the response, the back-channel is used to retrieve the access and refresh token. This is the recommended flow for native applications that want to retrieve access tokens (and possibly refresh tokens as well) and is used for server-side web applications (MVC) and native desktop/mobile applications.

Implicit

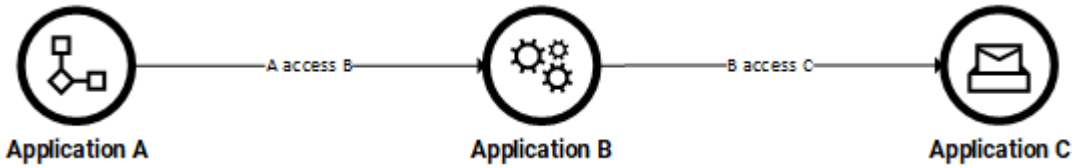
The implicit grant type is optimized for browser-based applications. Either for user authentication-only (both server-side and JavaScript applications - MVC WebApp), or authentication and access token requests (JavaScript applications - SPA). In the implicit flow, all tokens are transmitted via the browser, and advanced features like refresh tokens are thus not allowed.

The implicit grant type is not recommended due to the inherent risks of returning access tokens in an HTTP redirect without any confirmation that it has been received by the client.

Applications

In terms of Authentication configuration we will have to consider that an application has a dual behaviour:

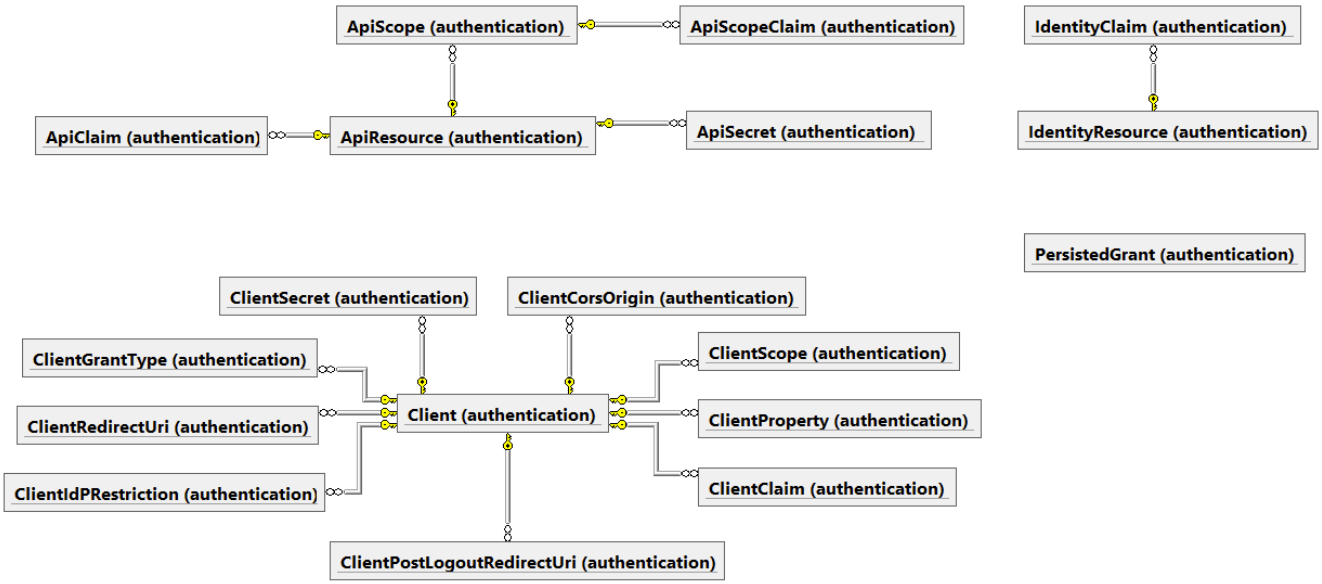
- As an *API resource* that needs protection
- As a *client* that requires access to other resources (user identities and API resources)



As described in the above sample, at the highest level, applications A and B are clients and B and C are API resources. Although, B is a client and also an API resource.

Persistence and database

All information related to authentication is stored in the security database in the `authentication` schema, and following the same design provided by IdentityServer.



> **Warning**
>
> Authentication Model contains information highly related to deployment and environment, so we do not recommend to restore backups from other environments. Instead, please use the deployment manager for configuring *authentication* and the `sequel-security` tool for moving authorization data.

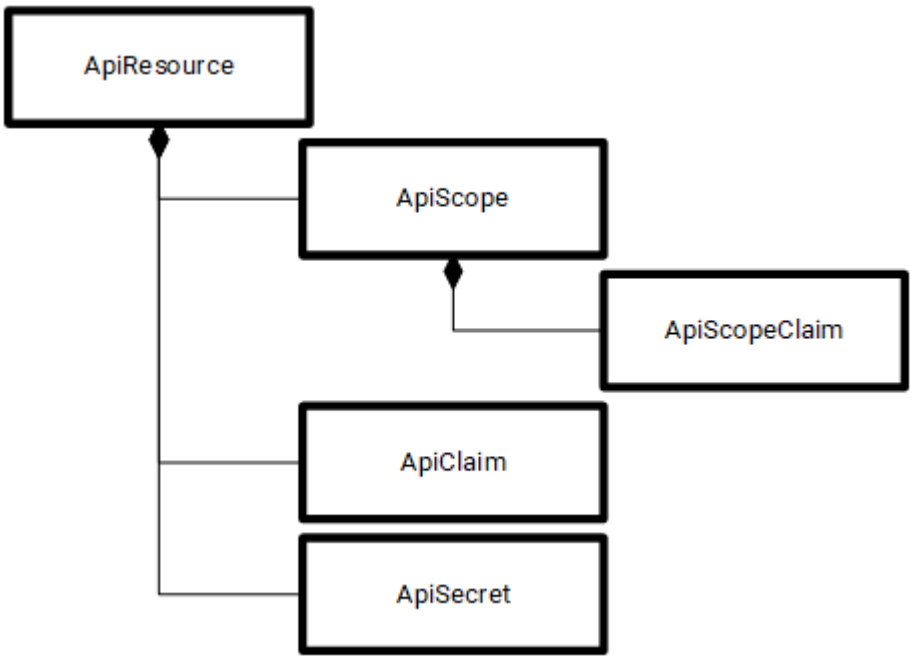
5.3.2 API resources

The fundamental concept in any RESTful API is the resource. A resource is an object with a type, associated data, relationships to other resources, and a set of methods that operate on it. API resources are something you want to protect and required all access to be authenticated and probably authorized.

Every resource has a unique name - and clients use this name to specify to which resources they want to get access to. APIs resources represent functionality a client wants to invoke - typically modelled as Web APIs, but not necessarily.

API Resources are defined and provided by each application as part of the *vanilla configuration*.

Models



More technical information at https://docs.duendesoftware.com/identityserver/v6/fundamentals/resources/api_resources/.

APIRESOURCE

This class model an API resource.

APICLAIMS

List of associated user claim types that should be included in the access token.

APISECRETS

The API secret is used for the introspection endpoint. The API can authenticate with introspection using the API name and secret. Internally in Sequel we are not using introspection, as we are using JWT validation.

APISCOPE

In the simple case an API has exactly one scope. But there are cases where you might want to sub-divide the functionality of an API, and give different clients access to different parts.

ApiScopeClaim

List of associated user claim types that should be included in the access token. The claims specified here will be added to the list of claims specified for the API.

Sample

At `sequel-security` tool section we will cover how to export and import configurations, in this section we will cover the data exchange formats. Following this format, we represent an API resource in JSON format as:

```
{
  "Enabled": true,
  "Name": "sec.api",
  "DisplayName": "Security API",
  "Description": "Security Rest API",
  "UserClaims": [],
  "ApiSecrets": [],
  "Scopes": [
    {
      "Name": "sec.api",
      "DisplayName": null,
      "Description": "sec.api",
      "Required": true,
      "Emphasize": true,
      "ShowInDiscoveryDocument": true,
      "UserClaims": []
    }
  ]
}
```

The above sample represents a traditional API resource definition where the most important properties are:

- *Name*: unique name of the API resource. Must follow the key naming convention, where all key starts with the application key followed by a dot and the code.
- *DisplayName*: friendly short name used in consent screens. Currently this is not used in our implementations.
- *Description*: text describing the API resource.
- *UserClaims*: Usually empty, as we are not requiring it.
- *ApiSecrets*: Usually empty, as we are not using the introspection endpoint.
- *Scopes*: At least, one scope is required. The name must be a valid key.

How to manage API resources

API resources are part of each application configuration and just can be managed using the sequel-security console tool.

5.3.3 Identity resource

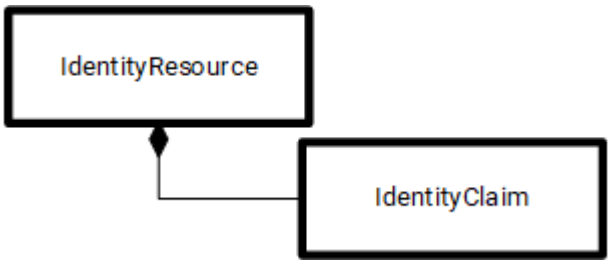
Identity resources are data like user ID, name, or email address of a user. An identity resource has a unique name, and you can assign arbitrary claim types to it. These claims will then be included in the identity token for the user. The client will use the scope parameter to request access to an identity resource.

The OpenID Connect specification specifies a couple of standard identity resources. The minimum requirement is, that you provide support for emitting a unique ID for your users - also called the subject id. This is done by exposing the standard identity resource called `openid`.

The IdentityResources class supports all scopes defined in the specification (openid, email, profile, telephone, and address). If you want to support them all, you can add them to your list of supported identity resources: `openid`, `profile`, `email`, `phone` and , `address`. In our implementation, this has been already configured for using `openid` and `profile`, so you do not have to be worried of them.

The request with `profile` scope access to the End-User's default profile Claims, which are: name, family_name, given_name, middle_name, nickname, preferred_username, profile, picture, website, gender, birthdate, zoneinfo, locale, and updated_at.

Models



More technical information at https://docs.duendesoftware.com/identityserver/v6/reference/models/identity_resource/.

IDENTITYRESOURCE

This class models an identity resource.

IDENTITYCLAIM

List of associated user claim types that should be included in the identity token.

How to manage Identity resources

There are no mechanism for managing; this is part of the security configuration and it is already configured.

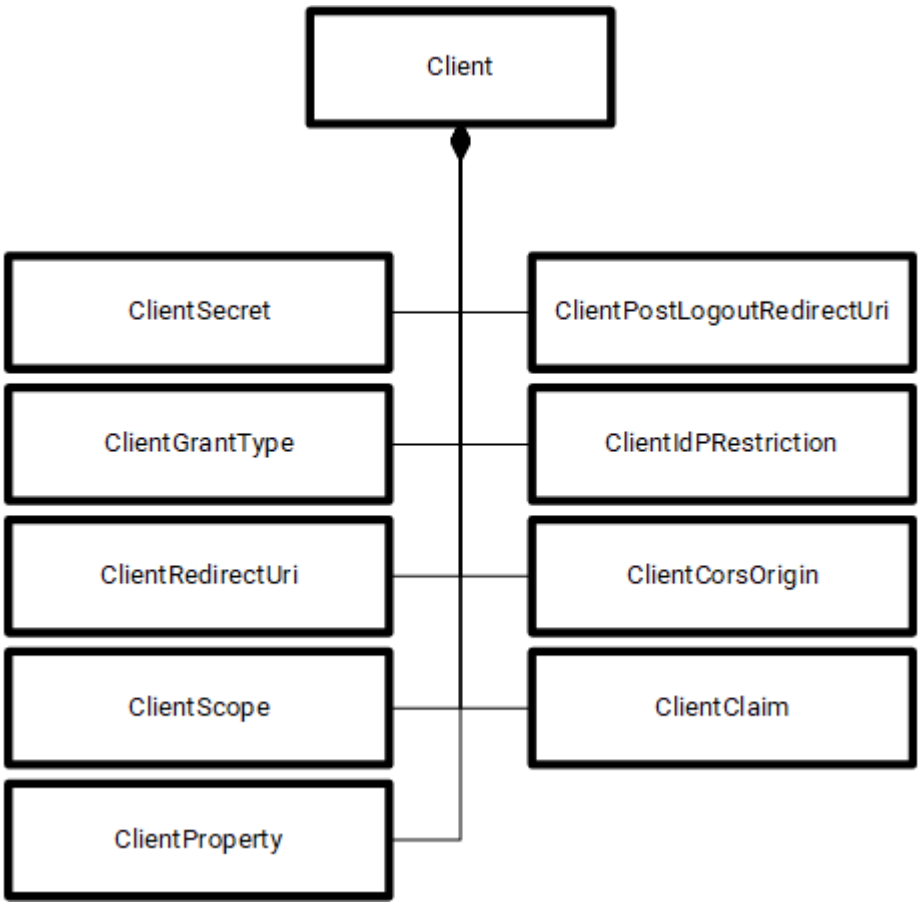
5.3.4 Client

A client is a piece of software that requests tokens from Duende IdentityServer - either for authenticating a user (requesting an identity token) or for accessing a resource (requesting an access token). A client must be first registered with Duende IdentityServer before it can request tokens.

Examples for clients are web applications, native mobile or desktop applications, SPAs, server processes etc.

Client configuration contains information related to the environment as some *URIs* and *Origin* are stored.

Models



A full description of all properties can be found at: https://docs.duendesoftware.com/identityserver/v6/reference/models/client/. Below we cover the most important objects that defines a client.

CLIENT

The Client class models an OpenID Connect or OAuth 2.0 client - e.g. a native application, a web application or a JS-based application. The most important property is the `ClientIdentifier`, a unique ID of the client that must follow the key naming convention.

CLIENTSECRETS

List of client secrets - credentials to access the token endpoint.

CLIENTGRANTTYPE

ClientGrantType or *AllowedGrantTypes* specifies the grant types the client is allowed to use; this determines the type of client. See Client Management section below for more information.

CLIENTREDIRECTURI

ClientRedirectUri specifies the allowed URIs to return tokens or authorization codes to. This is really important for interactive users grant types during the login process; the Uri of the API/website (the *client*) we are accessing is passed in the request to the authentication server and must match with an Uri defined for this client. A mismatch on those setting will end up in errors of type *invalid_client* during sign-in process.

CLIENTSCOPE

Or *AllowedScopes*, by default a client has no access to any resources - specify the allowed resources by adding the corresponding scopes names. Scopes defined here should exists for any resources (API or Identity).

CLIENTPROPERTY

Dictionary to hold any custom client-specific values as needed.

CLIENTPOSTLOGOUTREDIRECTURI

Specifies allowed URIs to redirect to after logout.

CLIENTIDPRESTRICTION

Specifies which external IdPs can be used with this client (if list is empty all IdPs are allowed). Defaults to empty.

CLIENTCORSORIGIN

If specified, will be used by *Cross-origin resource sharing* (CORS) policy service implementations to build a CORS policy for **JavaScript clients**. *CORS* is a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served. See the OIDC Connect Session Management specification for more details.

The proper definition of *Origin* can be found at w3.org and is expected to found a collection of origins (*scheme://host:port*). Previous implementations allowed URL format but it isn't longer supported so review *normalize-cors* command in `sequel-security` tool if you're planning upgrade an existing environment. In any case *import* command in `sequel-security` will normalize the ClientCorsOrigin by default to match the expected *Origin* pattern.

> **Any Origin**
>
> Special value `*` for Origins is not longer supported due to has been marked as insecure.

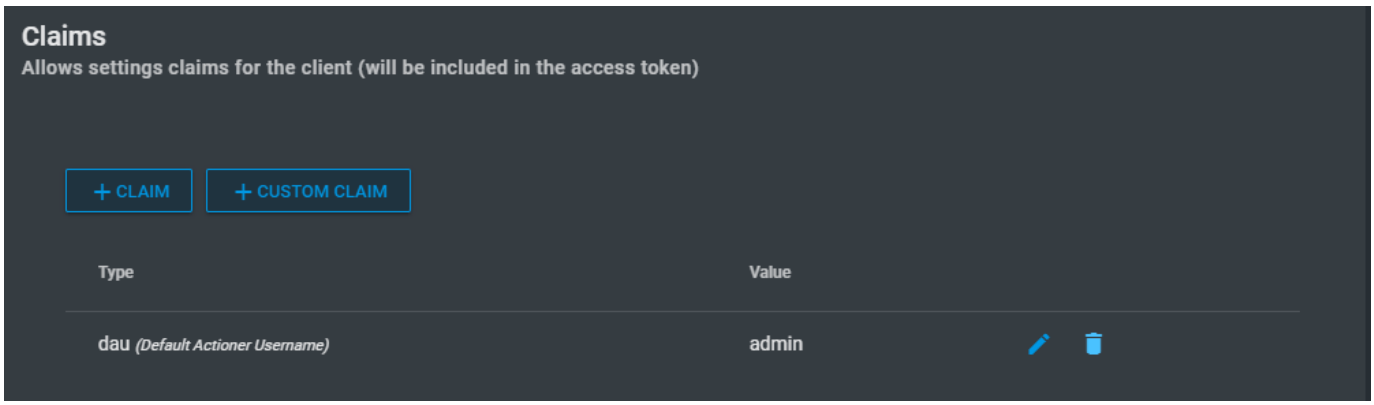
Restart Security API after changes in this field, to properly apply them.

CLIENT CLAIM

Allows settings additional Claims for the client. Those will be included in the access token. Each claim key will have the *Client claims prefix* as prefix and the *Type* defined in the table.

> **Claim prefix and additional claims**
>
> Imagine that the client has `client_` as *Client claim prefix* (which is also the default value). Now a new additional claim is added with *Type* `ex`. In the access token, that claim will appear as `client_ex`. This applies as well to the list of special Claims (see below), where they will list only the *Type* but the Claim will appear with the prefix appended.
>
> The *Client claim prefix* is configurable through Administration. The default value is `client_` but can be changed any time through the Clients page of an Application.

There are a few additional Claims that can be configured for a client and has a defined meaning. Those Claims can be added using Administration, by clicking on the *+ Custom Claim* button:



See below a list of special Claims:

- `dau` : Claim that indicates which Username should be used as *Default Actioner Username* in certain operations that require one.

Sample

Clients differ mainly on the grant types that they are allowed to use. Using our JSON specification for representing authentication objects.

As a sample, the security administration SPA website client looks like:

```
{
    "BackChannelLogoutSessionRequired": true,
    "AlwaysIncludeUserClaimsInIdToken": true,
    "IdentityTokenLifetime": 300,
    "AccessTokenLifetime": 3600,
    "AuthorizationCodeLifetime": 300,
    "AbsoluteRefreshTokenLifetime": 2592000,
    "SlidingRefreshTokenLifetime": 1296000,
    "ConsentLifetime": null,
    "RefreshTokenUsage": 1,
    "UpdateAccessTokenClaimsOnRefresh": false,
    "RefreshTokenExpiration": 1,
    "AccessTokenType": 0,
    "EnableLocalLogin": true,
    "IdentityProviderRestrictions": [],
    "IncludeJwtId": true,
    "Claims": [],
    "AlwaysSendClientClaims": true,
    "ClientClaimsPrefix": "client_",
    "PairWiseSubjectSalt": null,
    "AllowedScopes": [
      "sec.api",
      "sec.authorization",
      "openid",
      "profile"
    ],
    "AllowOfflineAccess": true,
    "Properties": {},
    "BackChannelLogoutUri": null,
    "Enabled": true,
    "ClientId": "sec.app.admin",
    "ProtocolType": "oidc",
    "ClientSecrets": [
      {
        "Description": "Authorization (Security Admin App)",
        "Value": "__ClientSecretHash__",
        "Expiration": null,
        "Type": "SharedSecret"
      }
    ],
    "RequireClientSecret": false,
    "ClientName": "Security Admin App",
    "ClientUri": null,
    "LogoUri": null,
    "AllowedCorsOrigins": [
      "__SecurityApiUrlExternal__"
    ],
    "RequireConsent": false,
    "AllowedGrantTypes": [
      "authorization_code"
    ],
    "RequirePkce": true,
    "AllowPlainTextPkce": false,
    "AllowAccessTokensViaBrowser": true,
    "RedirectUris": [
      "__SecurityAdminUrlExternal__/#/callback#",
      "__SecurityAdminUrlExternal__/#/silentrefresh#"
```

```
    ],
    "PostLogoutRedirectUris": [
      "__SecurityAdminUrlExternal__"
    ],
    "FrontChannelLogoutUri": null,
    "FrontChannelLogoutSessionRequired": true,
    "AllowRememberConsent": true
  }
```

How to manage clients

The configuration of clients is a big part of the Authentication server as it allows it to know which APIs, apps, etc. are allowed to ask for authentication and how. Clients can be managed:

- using `sequel-security` tool, importing and exporting packages.
- using the API,
- using the *Administration website*.

This resource is protected by `Sec.Client` securable.

In most of the cases, client configuration are provided by Vanilla configurations of each product and you do not have to manage them. However, there are scenarios like adding new clients for accessing API Gateway where it is required to create clients. Please, refer to each application about how create clients if required.

API CLIENT RESOURCE

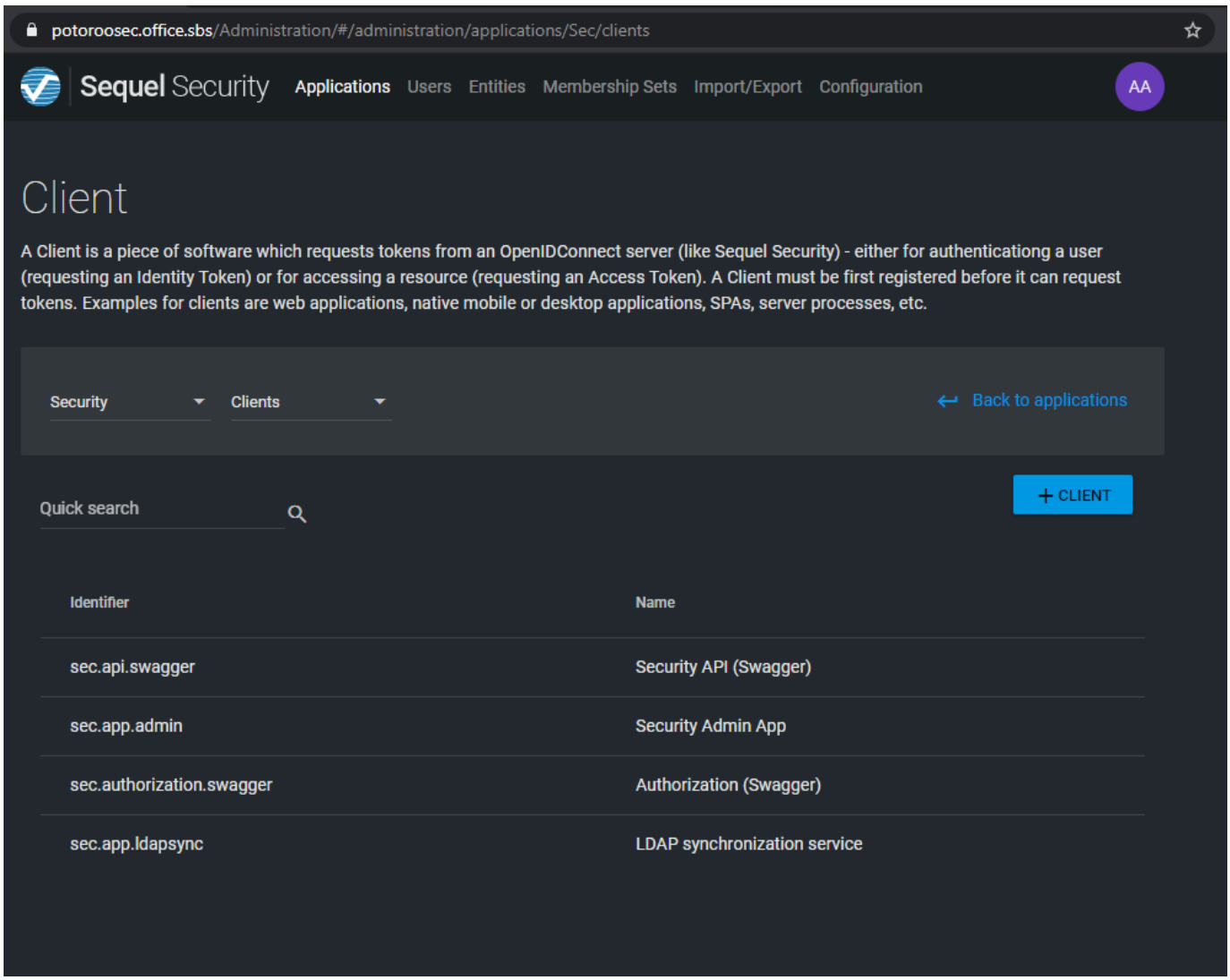
The `clients` resource is exposed at `/Authorization/Clients` (note this is at Authorization, see how Security Rest API is organized for more information). This is fully documented using Swagger.

CLIENT ADMINISTRATION PAGE

For managing clients from Administration website:

Go to applications page, an expand the contextual menu of the application you want to manage its clients. Select option Client.

Once at client's page you can create new clients or managing an existing one.



> **Warning**
>
> Wrong configurations can break applications Please, keep in mind that configuring clients is really a key part of the system, and a wrong configuration could break this client.

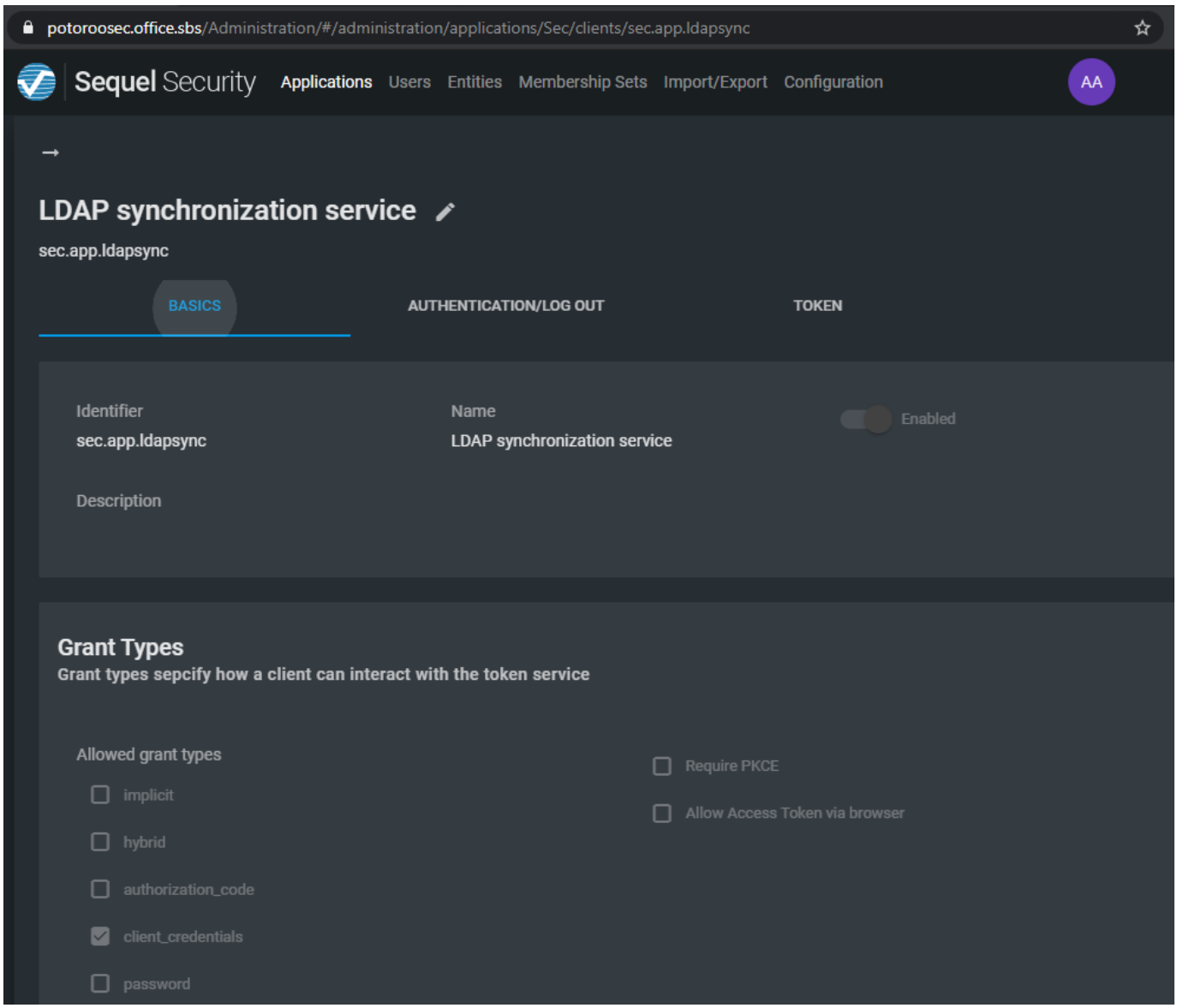
New clients

Click on `+ CLIENT` button. Fill in all required settings for the type of client you are creating and save.

For creating a new client is important to fully understand how OpenIdConnect works; with the aim of simplifying this action we have included some typical client configurations below.

Manage existing clients

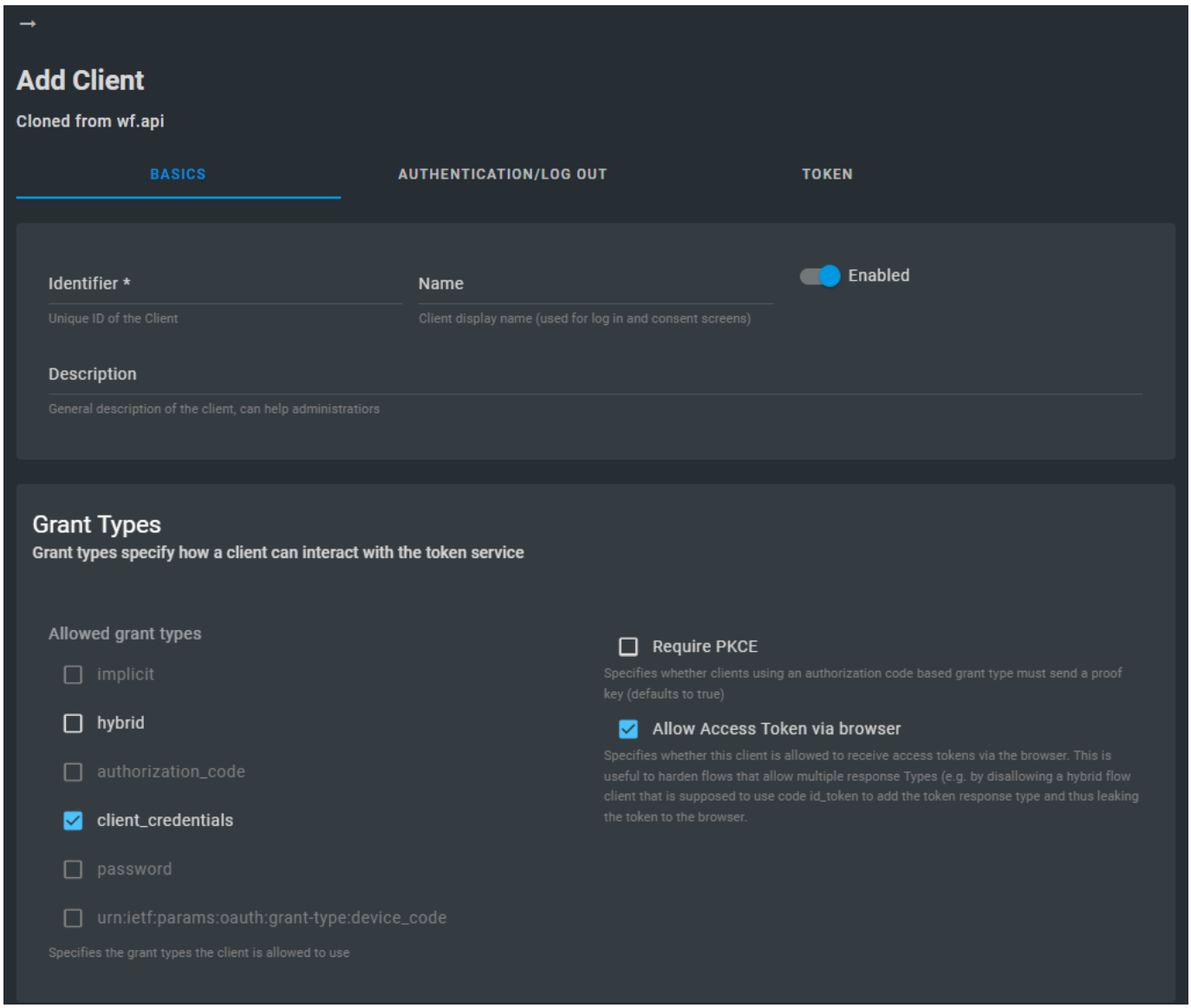
Click on the client you want to manage for opening in read-only mode. Click on edit icon to change or delete it.



Clone existing clients

You can also clone existing clients, you just have to click on the desired client to be cloned, go to the top right corner of the window and in the three dots menu select `Clone Client`.

An exact copy of the client will appear, except for the client ID, client name and client secrets.



TYPICAL CLIENT CONFIGURATIONS

In this section we are going to list the three most typical client configurations and list the minimum information that would need to be filled in the UI for them to work correctly, if any settings aren't mentioned it's because they are not mandatory or can be left as default..

Client credentials

The **client credentials** grant type is used by clients to obtain an access token outside of the context of a user. This is typically used by clients to access resources about themselves rather than to access a user's resources (more info: https://www.oauth.com/oauth2-servers/access-tokens/client-credentials/), for example, in Workflow it is used to call the Security Authorization server for a user's specific permissions. For this reason it is typically said it's recommended for **machine-to-machine communication** (or in this case, more specifically, API-to-API). A clear sample of creating this type of clients is for providing a *client* to our customers that calls to *Sequel Api Gateway*.

The minimum information for a client of this type is:

- Grant Types:
  - `client_credentials`
- Scopes:
  - {APP_KEY}.{SCOPE} (any specific scopes created for your application, e.g. 'wf.api')
- Secrets (one client secret is mandatory to be used for client credentials of type 'SharedSecret')

Authorization code

The **authorization code** grant type for authenticating clients is typically used for SPAs (Single Page Applications). It consists of exchanging an authorization code for an access token and the after that redirects the user via a redirect URL (more info: https://www.oauth.com/oauth2-servers/access-tokens/authorization-code-request/).

So for a client of this type and use, the minimum information needed to be filled in when configuring this client would be:

- Grant Types:
  - `authorization_code`
  - Require PKCE
  - Allow access token via browser
- Scopes:
  - `openid`
  - `profile`
  - {APP_KEY}.{SCOPE} (any specific scopes created for your application, e.g. 'sec.app.admin')
- Redirect URI (one redirect URI is needed so the application knows where to redirect back after requesting a token)
- Allowed CORS origins

Hybrid

The **hybrid** flow is recommended to be used by MVC applications where we have a backend that can securely store passwords and cookies and needs it own token, and then a front end that needs a separate one. Our Workflow MVC application uses this type of client authentication as it needs authentication for the user using the application but also for the calling it's own backend, as would applications like Product Builder, Claims, Origin, etc.

The minimum information for a client of this type would be:

- Grant Types:
  - `hybrid`
  - Allow access token via browser
- Scopes:
  - `openid`
  - `profile`
  - {APP_KEY}.{SCOPE} (any specific scopes created for your application, e.g. 'sec.api')
- Secrets (one client secret is mandatory to be used for the hybrid flow of type 'SharedSecret')

5.3.5 Persisted Grants

The OAuth 2.0 specification is a flexible authorization framework that describes a number of grants (“methods”) for a client application to acquire an access token (which represents a user’s permission for the client to access their data) which can be used to authenticate a request to an API endpoint. There are different types of tokens that must be temporally stored in order to support the authentication flows in subsequent requests.

- **Authorization Code** is required for hybrid flow when a user is being authenticated and has a lifetime of 300 seconds (by default) and only is used once per login process. Ideally it should be deleted when the login process is completed or it expires.
- **Refresh Token** is generated once per login process and updated each time a user ask for a new Access Token and it has a sliding lifetime of 30 days (by default). It is removed when a user sign-out.

All those tokens (or grants) are persisted in the **[authentication].[PersistedGrant]** table.

How to manage persisted grants

Information stored at persisted grants are not configuration, it is runtime data. There are no needs for managing this information as this is internally done by security security. However, it could be required to perform some housekeeping tasks.

Housekeeping

In some scenarios is possible that persisted grant are not removed; this can cause this table keep growing affecting the performance of the application due to storing out of date grants. The **housekeeping process** handles the removal of expired grants. This process can be triggered:

SCHEDULED BACKGROUND JOB

Automatically triggered in a scheduled background job. The schedule job is hosted by *Security Rest API*, and works like a daemon trying to impact as less as possible to the system.

The housekeeping execution removes the oldest out of date grants in batches (*batchSize*), during a period of time (from *fromUTC* to *toUTC*), each execution occurs with an interval ot time (*interval*). It can be configured setting table in database customising below settings: * *enabled*: Enables the scheduled housekeeping job (false by default) * *batchSize*: Maximum number of rows affected in each execution (Default value: 1000). * *fromUTC*: When execution period starts (timespan format, by default 00:05) * *toUTC*: When execution period ends (timespan format, by default 05:00) * *interval*: Interval when each execution is performed (By default 15 minutes)

Samples of configurations:

- "*fromUTC* and *toUTC* hours each *interval* minutes": ie from 00:05 to 03:00 each 15 minutes
- "When *fromUTC* > *toUTC*, use *toUTC* = 01.XX:XX": ie from 22:05 to 01.00:05

API REQUEST

Manually sending a request to the `PersistedGrants` resources, at endpoint `DELETE /Authentication/PersistedGrants`.

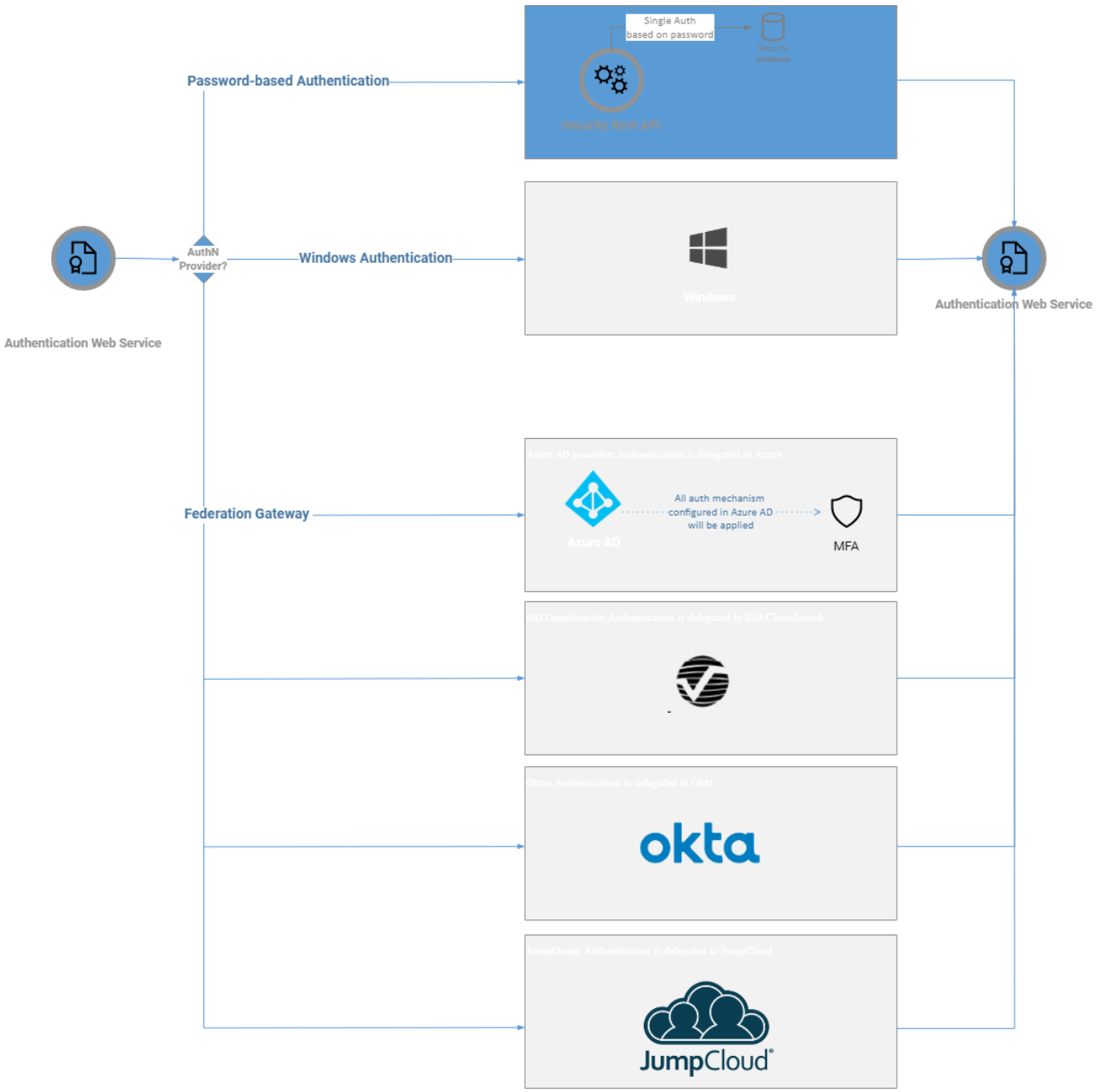
For manually deletions we remove in batches as for automatic deletions, the response indicates if there was row deleted or not. So caller will know if there are more rows to delete or not.

5.3.6 Authentication of users

Sequel security service offers different mechanism or authentication providers to authenticate a user:

- Built-in password based authentication.
- Windows Authentication.
- **Federation gateway**. One of the features provided by Duende IdentityServer is the **federation gateway**; allowing our security service to act as a gateway to other external identity providers:
  - Azure AD
  - ISO ClaimSearch
  - Okta

At least, one authentication provider must be configured; being possible to configure more than one at the same time.



Configuring authentication providers

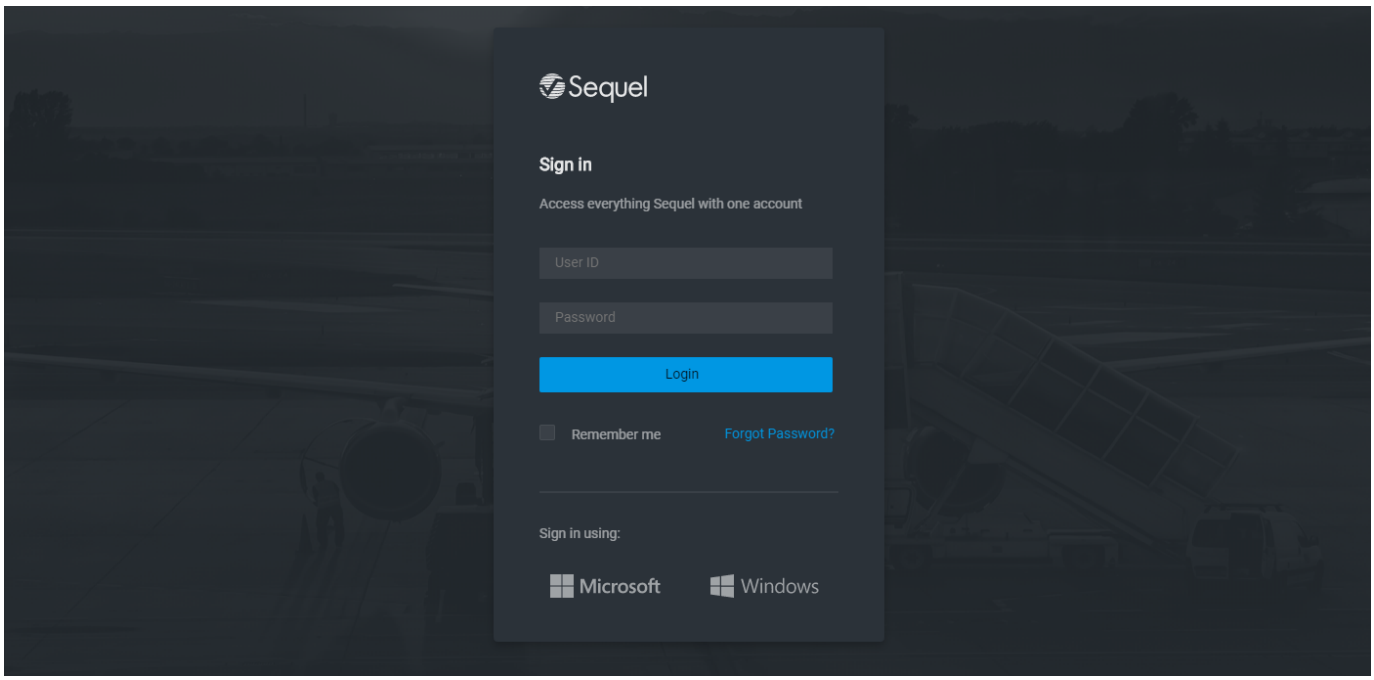
Authentication providers are configured at `appsettings.json` file at `LoginSettings.IdentityProvidersSettings`. Each provider has specific settings but all of them has an `Enabled` property to activate or deactivate it.

How authentication provider is selected

When just one authentication mechanism is configured this is used directly without asking to the user. If the selected method is:

- Built-in, then the login screen is displayed.
- Others, then the challenge/redirection to this authentication provider is started.

If there are multiple providers configured then the user is asked to select and start the authentication:



5.3.7 Password-based AuthN

Password-based authentication offers an easy way of authenticating users. In password authentication, the user must supply a password that is stored by Sequel Security Service. Password are stored hashed and have to follow a password policy.

Basic configuration

Password-based authentication can be enabled at `appsettings.json` in the authentication service: `LoginSettings.IdentityProvidersSettings.Sequel.Enabled`

Password policy

A password policy is a set of rules that govern how passwords are administered. Sequel Security service supports multiple password policies. The password policy can be partially configured to suit the security requirements at *Security Rest API's* `appsettings.json` file in the `PasswordPolicySettings` property.

Rule / Property	Description	Default
RequiredLength	The minimum length of the password	6
RequiredUniqueChars	Requires the number of distinct characters in the password.	5
RequireNonAlphanumeric	Requires a non-alphanumeric character in the password.	true
RequireLowercase	Requires a lowercase character in the password.	true
RequireUppercase	Requires an uppercase character in the password.	true
RequireDigit	Requires a number between 0-9 in the password	true
RequireUserNameCheck	Requires password does not contain the first 3 characters of username, first name or last name	true
ExpireAfterDays	Expires the password and forces to set a new one. Applies to users where `PasswordExpiryDateUtc` is populated.	90
RequireDifferent	Required new password must be different to previous one. This rule is not configurable and is always active.	true

Password lockout policy

Password lockout policies are used to lockout an account when someone tries to log on unsuccessfully several times in a row. We can usually assume that a legitimate user might type his or her password incorrectly once or twice, but not numerous times. Thus, numerous failed login attempts can indicate that someone is trying a brute-force password attack. Password lockout is not configurable and is based on:

- **LockoutMaxAttempts**: The account lockout threshold specifies the number of failed attempts at logon a user is allowed before the account is locked. This value is set to **3 attempts**. After the threshold has been reached, the account will be locked out.
- **LockoutDuration**: The account lockout duration specifies the time in minutes that the account can be locked out. This value is set to **5 minutes**. A user account is locked when property `LockoutEndDateUtc` is populated and has a value in the future.

The password lockout counter is reset after a valid login.

The lockout policy is enabled by default for all users, however this can be disabled for some users setting `LockoutEnabled` to false at user's record.

Password expiration

When creating a new user, a date when a password will be expired will be set too. This value will be 90 days by default and may be changed in the `appSettings.json` of Authorization. If the date when the user is trying to log in is later than the expire date a message will be sent in the login UI, requesting the user to change the password. The default expiration days can be customized at `appsettings.json` in SecurityApi with the property `PasswordPolicySettings.ExpireAfterDays`.

EDIT EXPIRED PASSWORD USER

An administrator user may change the expire date of an existing user by editing it in the edit drawer picking up a new date in the Date Picker. By default is not allowed to set a customized date when we create a user in Admin UI the default value will the one is set in the appsettings.

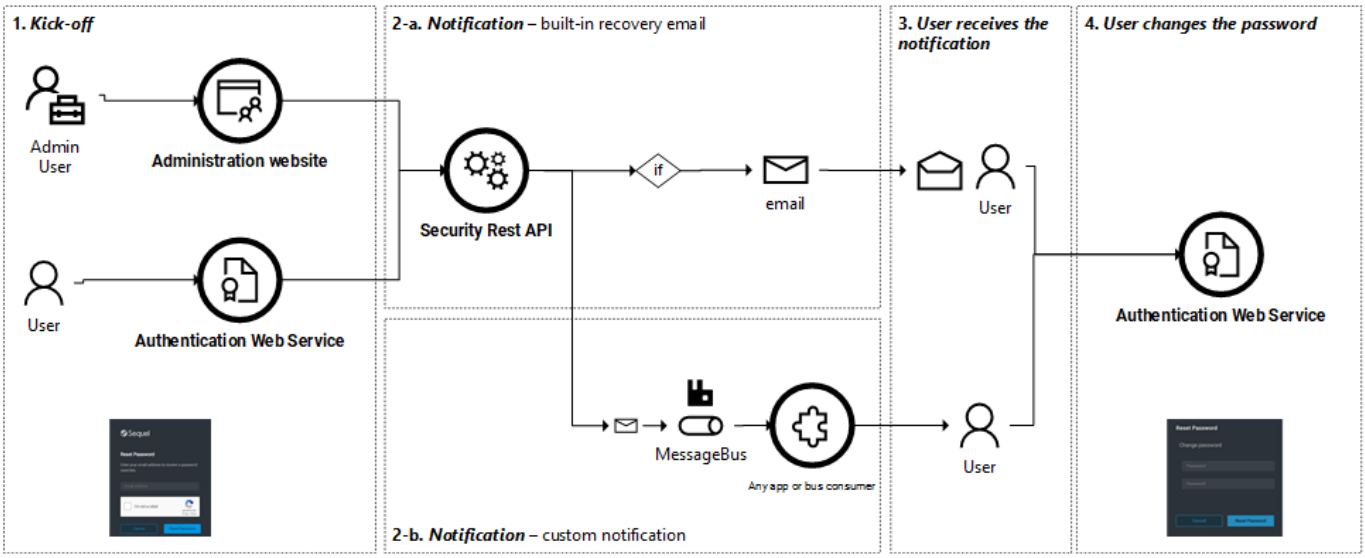
Forgot and reset password

The password reset option is defined outside of the scope of a logged-in user so that users who forgot their passwords could also reset their passwords. The *reset password flow* is based on a recovery email with a link that will kick off the password reset process. In this email the user receives a link that is valid for a short period of time; following this link the user will be able to reset his password.

The reset password is also the mechanism for providing a password to new users. There are two options:

- Administrator user forces to send the reset email, or
- User goes to login page an request to reset the password.

THE RESET PASSWORD FLOW



Kick-off

There are two methods for resetting a password:

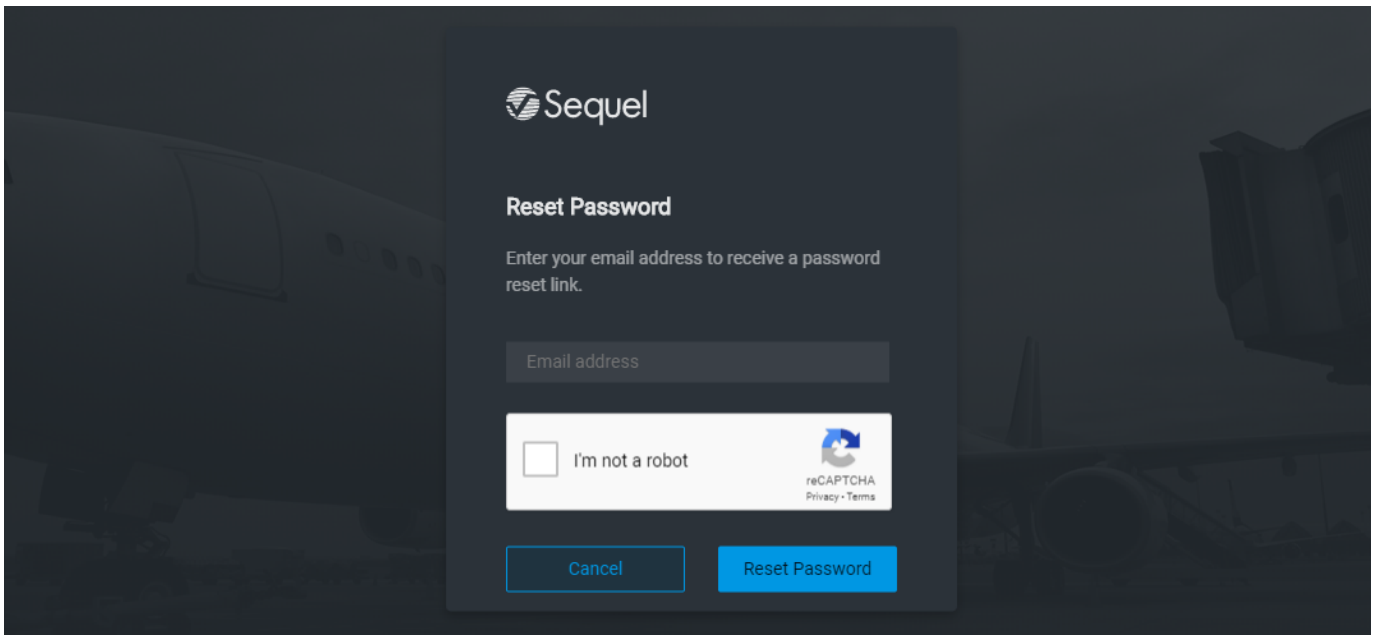
Not logged users

Anyone can request a password reset without being logged. To reset the password:

Go to **login** page.

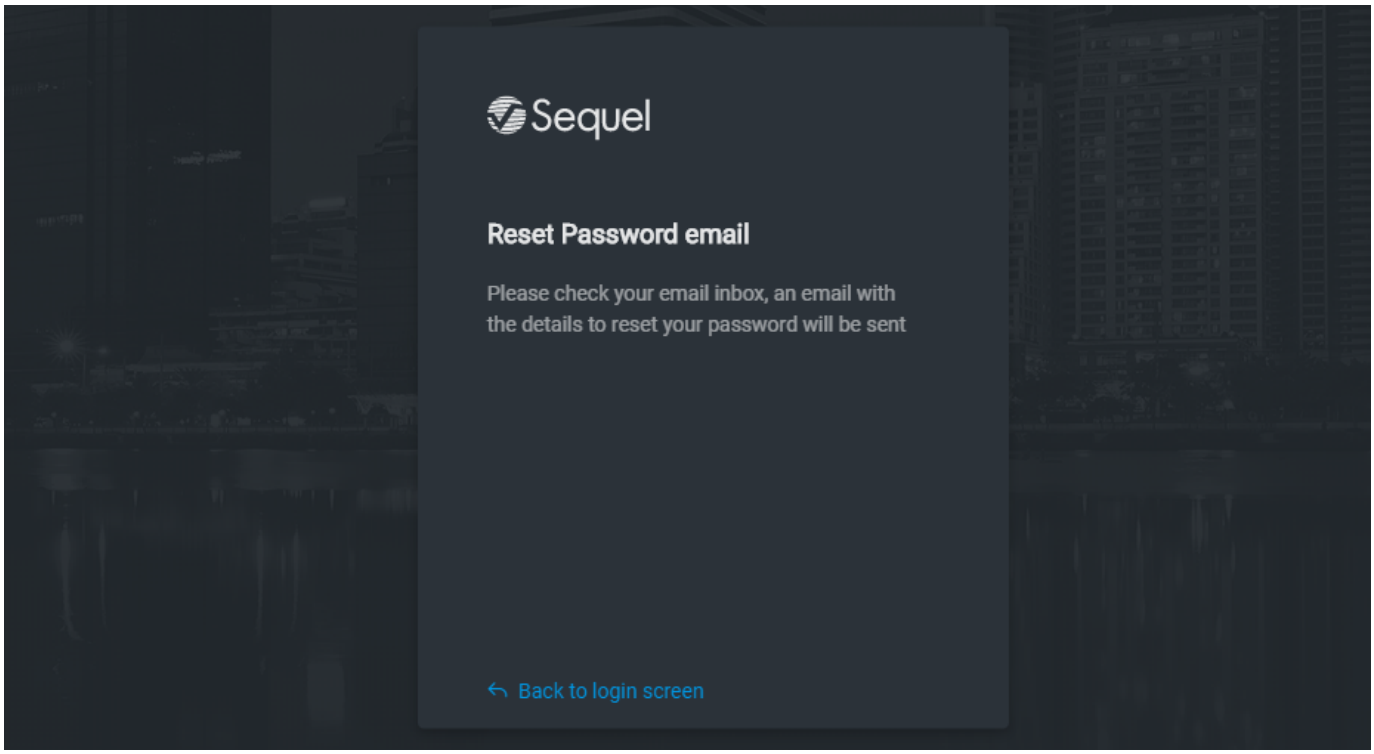
Click on *Forgot Password?*. The **Reset Password** ((`/ForgotPassword`)) page will be loaded.

At *Reset Password* page, introduce the user's email. Confirm the captcha if configured and click on *Reset password*.



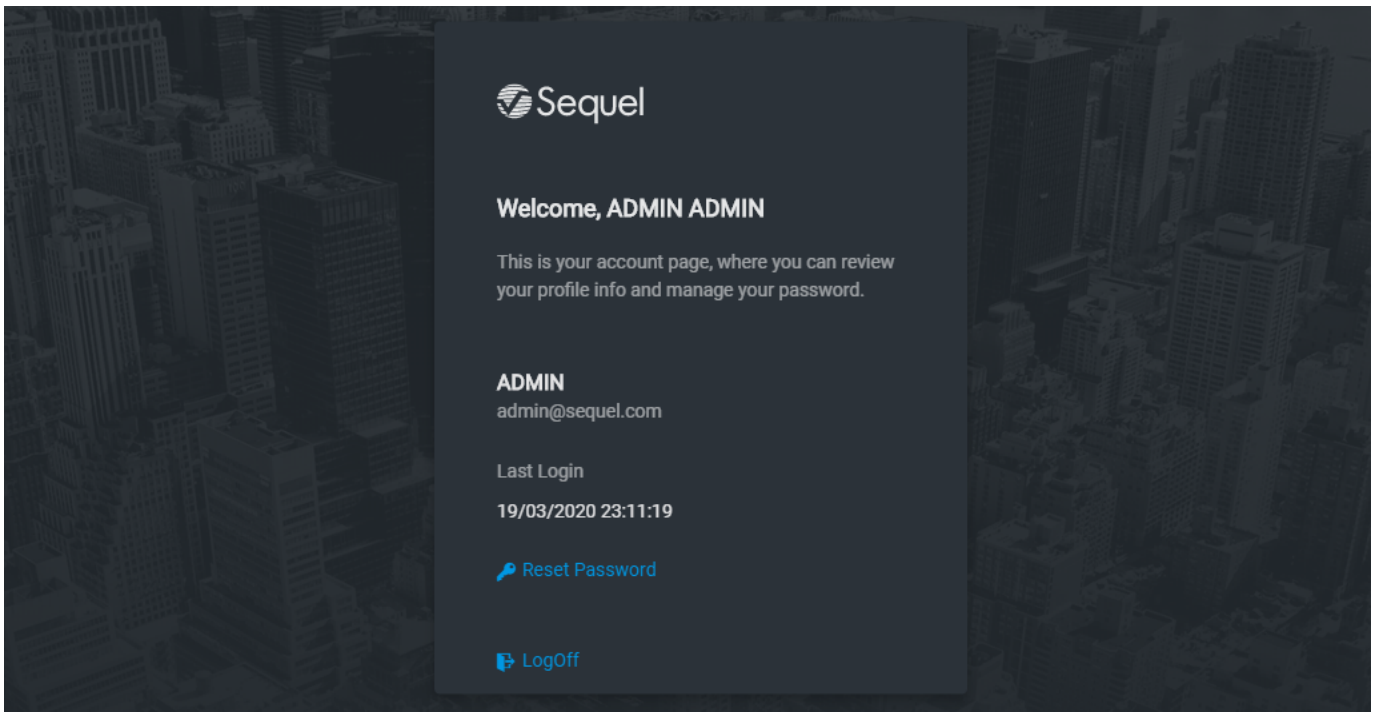
The **Reset Password email** confirmation page is always displayed:

- if the emails does not exists; this is done to do not confirm to malicious users that this emails exists in the system.
- if the email exists a notification is sent to the user associated to this email.

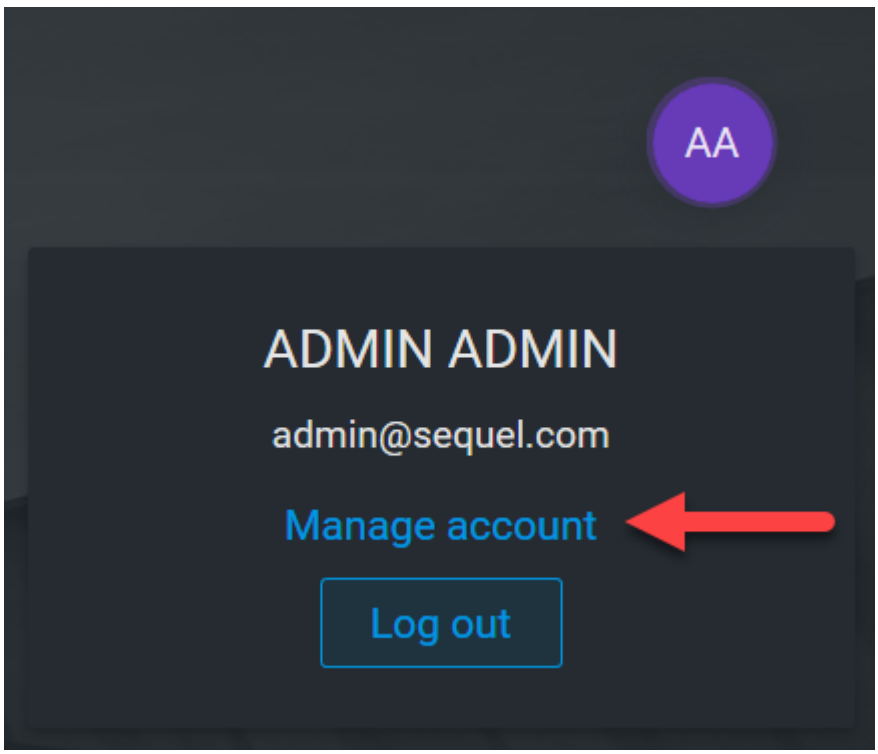


Logged users

A logged user can reset his password from the **MyAccount** page (`/MyAccount`), clicking on *Reset password*. Once at *Reset Password* page, the process is the same for *Not logged users*.



If user arrived to **MyAccount** page using the user's component from any application clicking on Manage account link the user will be redirected to de application after complete the process.



Administrator on behalf of any user

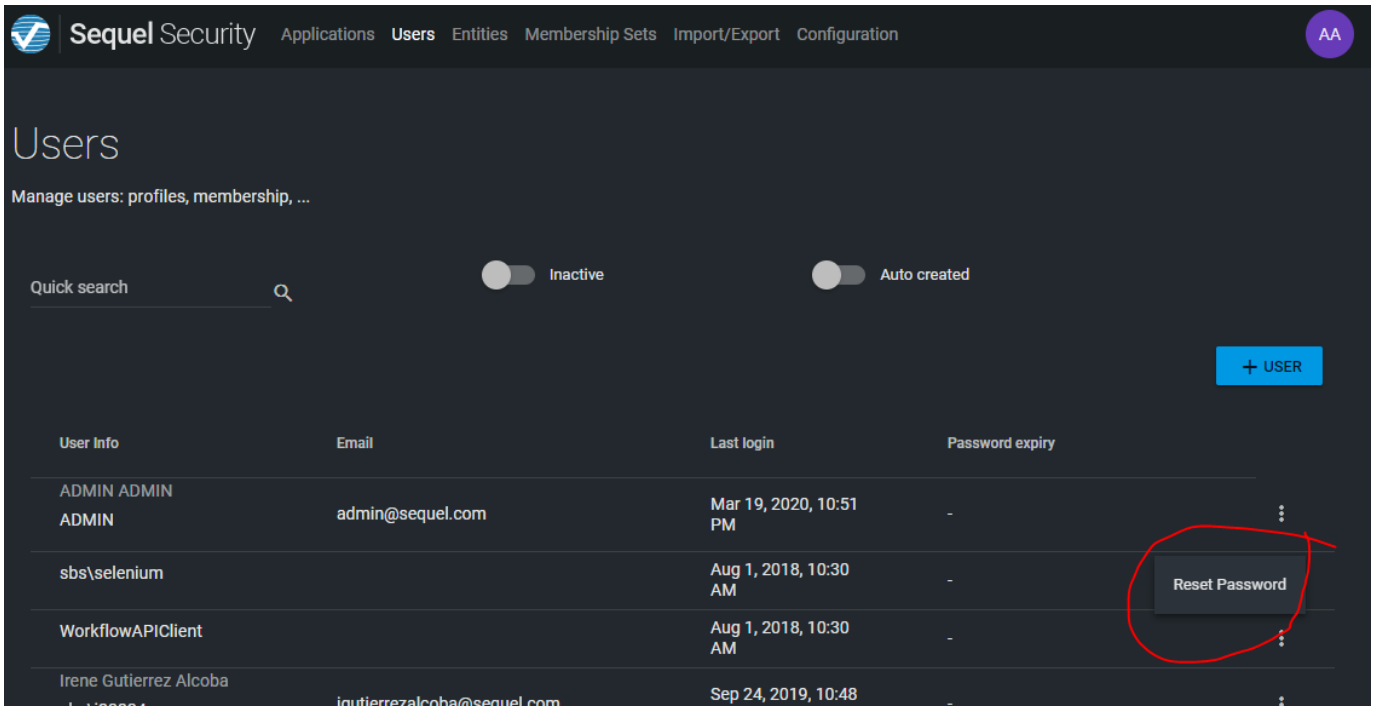
An administrator user can request a reset email for any user. For resetting the password of any user:

Go to Administration web site as an administrator

Go to Users page `/users`

Find the user that needs to reset his password.

Click on the contextual menu of this use and click on *Reset Password* option.



Notification

There are two mechanism for handling the reset password:

Built-in recovery email

The user will be notified with an email that contains the link for actually resetting the password. There are some important configurations that will affect to this delivery system:

- Send email support
  - Support for sending emails must be successfully done (at `appsettings.json` in Security API: `SendEmailSettings` property).
  - `SendEmailSettings.SendForgotPasswordEmail` is enabled.
  - The sender email address is defined at `SendEmailSettings.ForgotPasswordFromEmail`.
- Email template. Email sent can be customized with below settings stored in the database at `[configuration].[settings]` table.
  - `ForgotPasswordEmailSubjectTemplate`: Defines the subject of the email.
  - `ForgotPasswordEmailBodyTemplate`: Defines the boy of the email.
  - Both properties offers three tags for defining:
    - First name: `{user-firstname}`.
    - Last name: `{user-lastname}`.
    - Link for resetting password: `{url-resetpassword}`.

Message bus notification

Also a message of type `ForgotPassword` is published into the bus notifying the event. This is meant to be used by other products or scenarios where the reset password flow must be customized; on those scenarios is probable that `SendEmailSettings.SendForgotPasswordEmail` will be switched off.

User receives the reset link

At some point, the user will receive the link for resetting the password. This link will be valid for a limited period of time.

User resets the password

When the user clicks on the link, if the link is still valid then the user wil be allowed to introduce his password that must follow the password policies.

User is redirected to caller application (optional)

When the user started the reset password process from an application that was passing the return URL; the user will be redirected to this URL is reset is completed with success.

RECAPTCHA

Reset password can be protected from automated abuse and attacks enabling reCAPTCHA. We are using Google's reCAPTCHA v2 system (https://www.google.com/recaptcha) and it is necessary for every installation to have their own captcha created and configured from Google's reCAPTCHA admin console; please check the Security Installation Guide for more details.

5.3.8 Windows AuthN

On supported platforms, you can use our security services to authenticate users using **Windows Authentication** (e.g. against Active Directory).

The Windows authentication is triggered at the login page when the user clicks on the Windows icon or automatically if this is the single provider enabled.

Important note: for Windows Authentication to work, the application must be installed on a server within the same domain as the user, so that the Security application can consult the Windows AD and get the information of the user signing in. For example, if we are using an AWS instance to host the Security application but we are trying to use the Windows SSO functionality with our SBS user then it won't work as the AWS instance will most probably not be incldued in the SBS domain, therefore Security will try to retrieve the user information from the Windows AD of it's AWS instance, and of course it won't be there, because the SBS user information will be in the Windows AD of the SBS domain.

Basic configuration

Windows Authentication can be configured at `appsettings.json` in the authentication service: `LoginSettings.IdentityProvidersSettings.Windows`:

- `Enabled` for enabling this provider. Please, check installation guide as there are some requirements in the server and IIS configuration for supporting this provider.
- `IncludeGroups` for including groups of the user as claims in the generated token (by default *false*).

User matching

For being able to login into our system is required that the authenticated Windows user exists in our system. The user matching is done using below user's properties and in this order:

1. Match by `SsoUsername`.
2. Match by `Username`.

We recommend to store the `sAMAccountName` in the `SsoUsername` and the user name without domain or the `UserPrincipalName` as the `username`. Users can be synchronized with the Windows Active Directory associated using the active directory sync feature.

Browser configuration

It's also worth mentioning that in Windows we can set when the browser asks for our credentials when using this feature. In `_Internet Options \ Security tab \ Security level for this zone \ Custom level... \ User Authentication \ Logon_`, here we have 4 options:

- Anonymous login
- Automatic logon only in Intranet zone
- Automatic logon with current user name and password
- Prompt for user name and password

We recommend using the third or fourth option when using Windows Authentication in production, as the third will automatically log you in with your current Windows credentials if it can, and the fourth will simply prompt you to always manually introduce them yourself when signing in.

5.3.9 Azure AD AuthN

As part of the *federation gateway* feature the integration with **Azure AD Authentication** is possible. In this scenario, the users are authenticated against Azure AD and it is not required to store passwords in our system.

The Azure AD authentication is triggered at the login page when the user clicks on the Microsoft icon or automatically if this is the single provider enabled.

Azure AD was previously called *Microsoft Accounts*.

Authentication

BASIC CONFIGURATION

Azure AD Authentication can be configured at `appsettings.json` in the authentication service: `LoginSettings.IdentityProvidersSettings.Microsoft`:

- `Enabled` for enabling this provider. Please, check installation guide as there are some IT configurations required before using this provider.
- `UserPolicies` for configuring how user matching and sync will be performed.

USER MATCHING

For being able to login into our system is required that the authenticated Azure AD user exists in our system. The user matching is done following th matching rules defined at `LoginSettings.IdentityProvidersSettings.Microsoft.UserPolicies.MatchingFields`. The valid options are:

1. `Oid`: The immutable identifier for an object in the Microsoft identity system, in this case, a user account. In Sequel's user record this value is stored at `AzureUserIdentifier`.
2. `Email`.
3. `SsoUsername`. Any unique identifier for the user, different to the username. It could store an email, this email could be different of the user's email used for notifications.

USER SYNC DURING AUTHENTICATION

During the matching process, it is possible to update some user's properties using the collections `UserPolicies.FieldsToUpdatesWhenNull` and `UserPolicies.FieldsToUpdatesWhenDifferent`. The valid options for updates in both scenarios are:

Option	Azure AD claim	Sequel's field
`Oid`	`Oid`	`AzureUserIdentifier`
`Email`	`Email`	`Email`
`FirstName`	`GivenName`	`FirstName`
`LastName`	`Surname`	`LastName`
`SsoUsername`	`Email`	`SsoUsername`

The default configuration looks like:

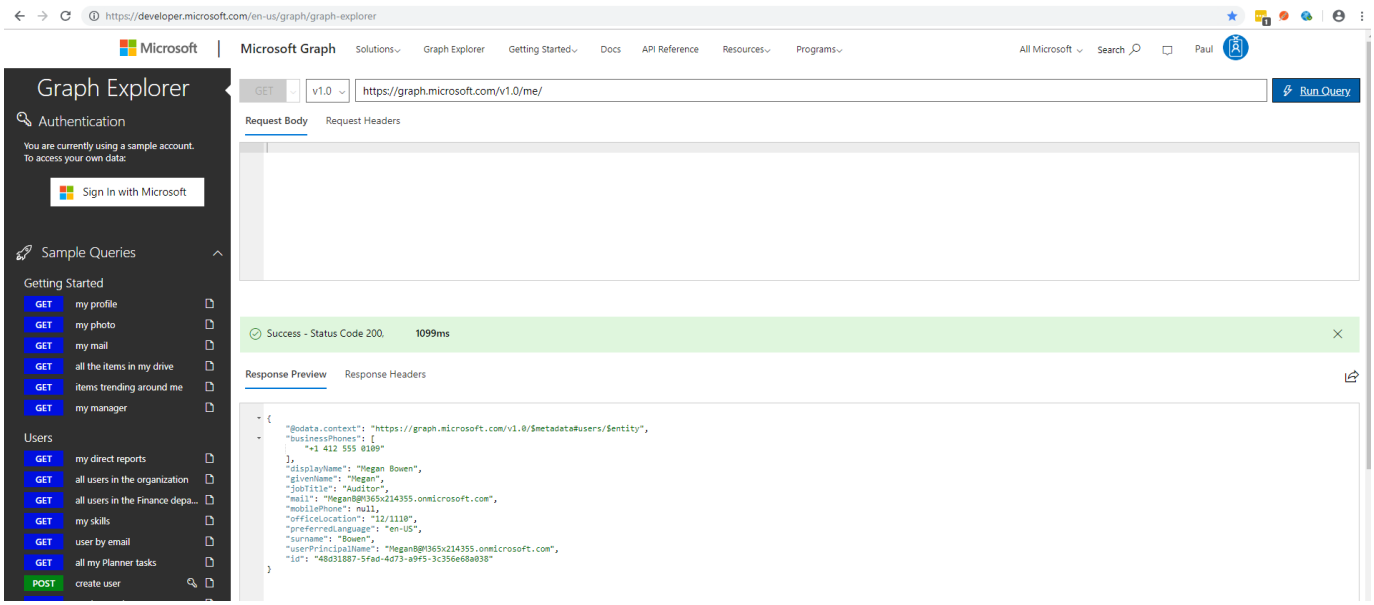
```
"Microsoft": {
  "Enabled": false,
  "TenantId": null,
  "ClientId": "_clientId_",
  "ClientSecret": "_secret_",
  "UserPolicies": {
    "MatchingFields": [ "Oid", "Email" ],
    "FieldsToUpdatesWhenNull": [ "Oid" ],
    "FieldsToUpdatesWhenDifferent": [ "FirstName", "LastName" ]
  }
```

If UserPolicies is missing or when some of its properties are null or empty default values will apply:

MatchingFields	FieldsToUpdatesWhenNull	FieldsToUpdatesWhenDifferent
Oid, Email	Oid	FirstName, LastName

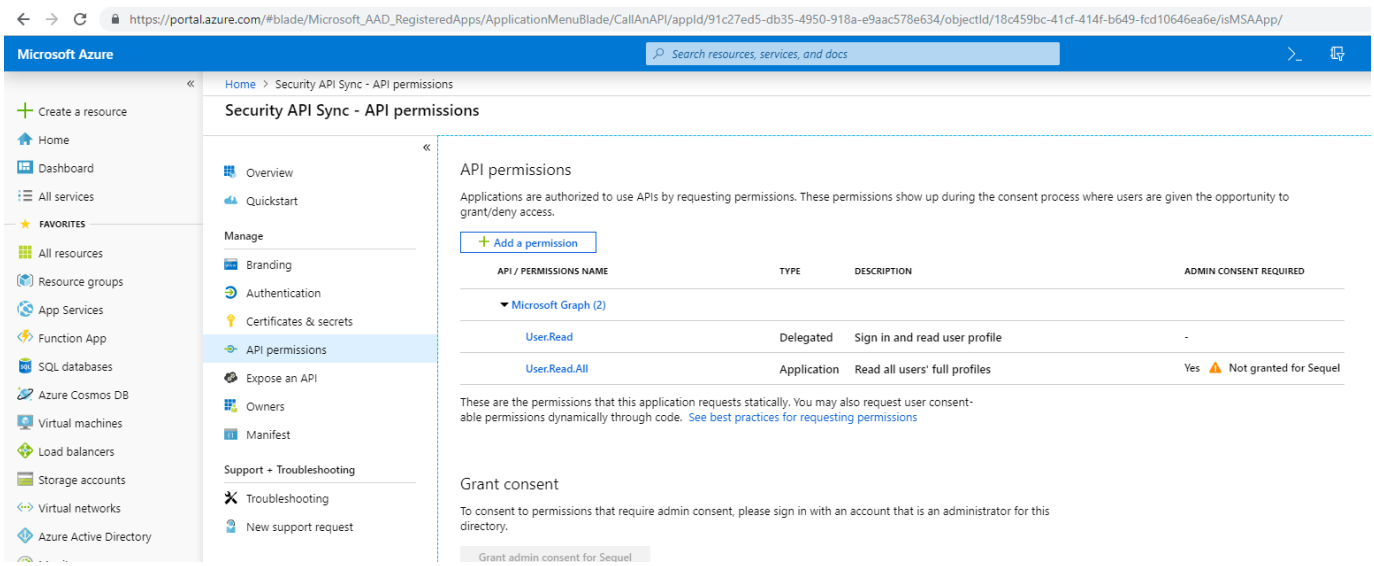
Azure AD Sync

This is a **experimental feature** that allows to synchronize users from Azure AD using Microsoft Graph. You can browse your AD contents using Azure AD Graph Explorer as here. https://developer.microsoft.com/en-us/graph/graph-explorer



You need to log in with your Azure AD account on the left hand side with the button called "Sign in with Microsoft". You can then run queries on you Azure AD.

For the Security Api to run queries against Azure AD we need to create an Azure Service Application and give it `Users.All.Read` as here: https://portal.azure.com/#blade/Microsoft_AAD_IAM/ActiveDirectoryMenuBlade/RegisteredAppsPreview:



When we create the application it is given an ApplicationID and we can create a password. We can then put these in the `appsettings.json` file *Security Rest API*:

```
  "SyncWithAzureUserSettings": {
    "ApplicationID": "--GUID of Application--",
```

```
    "Password": "--Password--"
  }
```

With these credentials SecurityAPI can get a authentication bearer token and we can use that to query the Azure AD API.

At Security Rest API a new method is exposed in `/Authorization/Users/{username}/syncWithAzure` for triggering the synchronization. This takes the username and updates the security system with the details from Azure AD.

5.3.10 ISO ClaimSearch AuthN

As part of the *federation gateway* feature is possible to authenticate users with **ISO ClaimSearch session management** (ClaimSearch). ClaimSearch AuthN is based on sharing authentication cookies (`ISOSESSIONID` cookie). Cookies can be only shared across the same domain or subdomain.

The authentication is triggered at the login page when the user clicks on the ISO ClaimSearch icon or automatically if this is the unique provider enabled.

Authentication

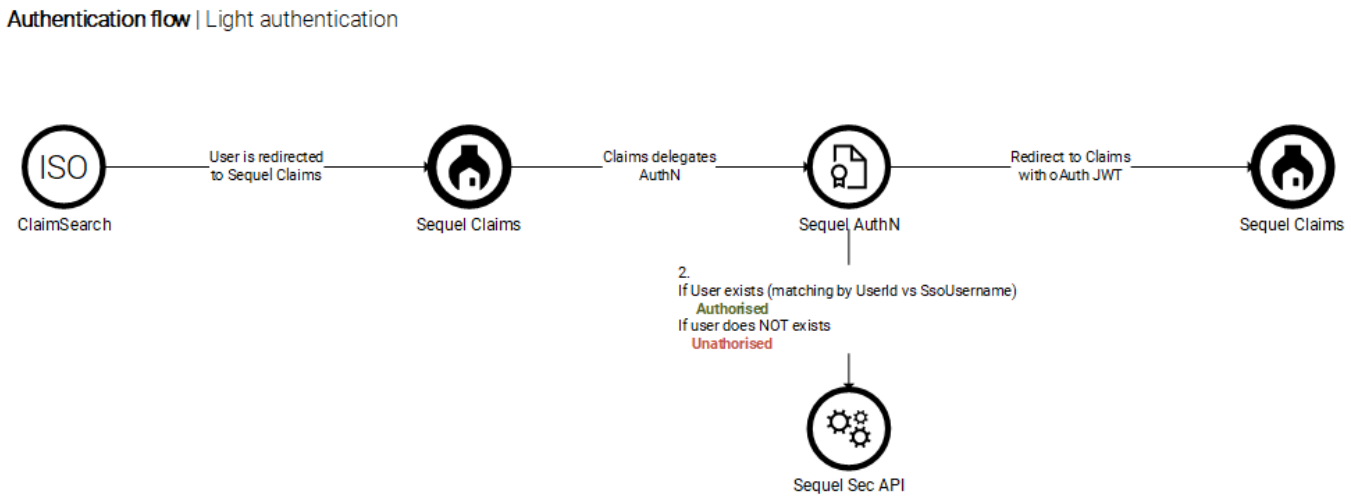
BASIC CONFIGURATION

Configure *ISO ClaimSearch AuthN* at `appsettings.json` file in the Authentication service, setting `LoginSettings.IdentityProvidersSettings.ClaimSearch` property. Some basic configurations are required:

- `Enabled` for enabling this provider.
- `SessionValidationEndpoint` is the endpoint to check if user have a valid active session in *ISO ClaimSearch* (i.e. https://servername/products/1.0/userinfo/)
- `LoginUrl` is the home page of *ISO ClaimSearch*. Users will be redirected to this URL if they are not already authenticated in ISO ClaimSearch.

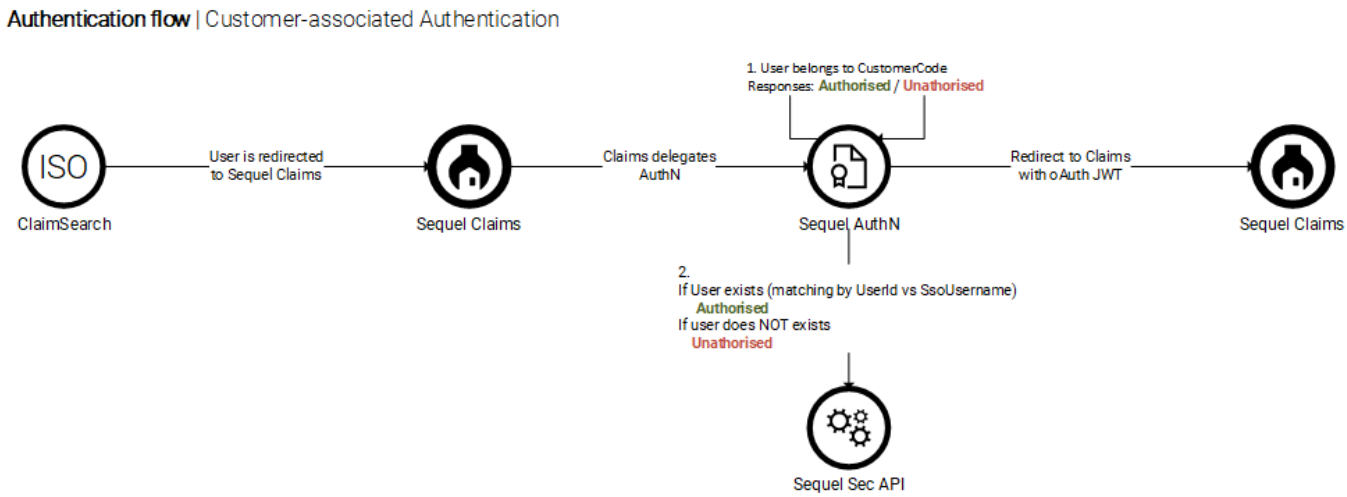
LIGHT AUTHENTICATION

The simplest and lightest authentication mode offered. This mode accepts all authenticated users by ClaimSearch if they already exists at Sequel's security services. The user matching is done comparing the ClaimSearch `userId` with the `SsoUsername` field at Sequel's.

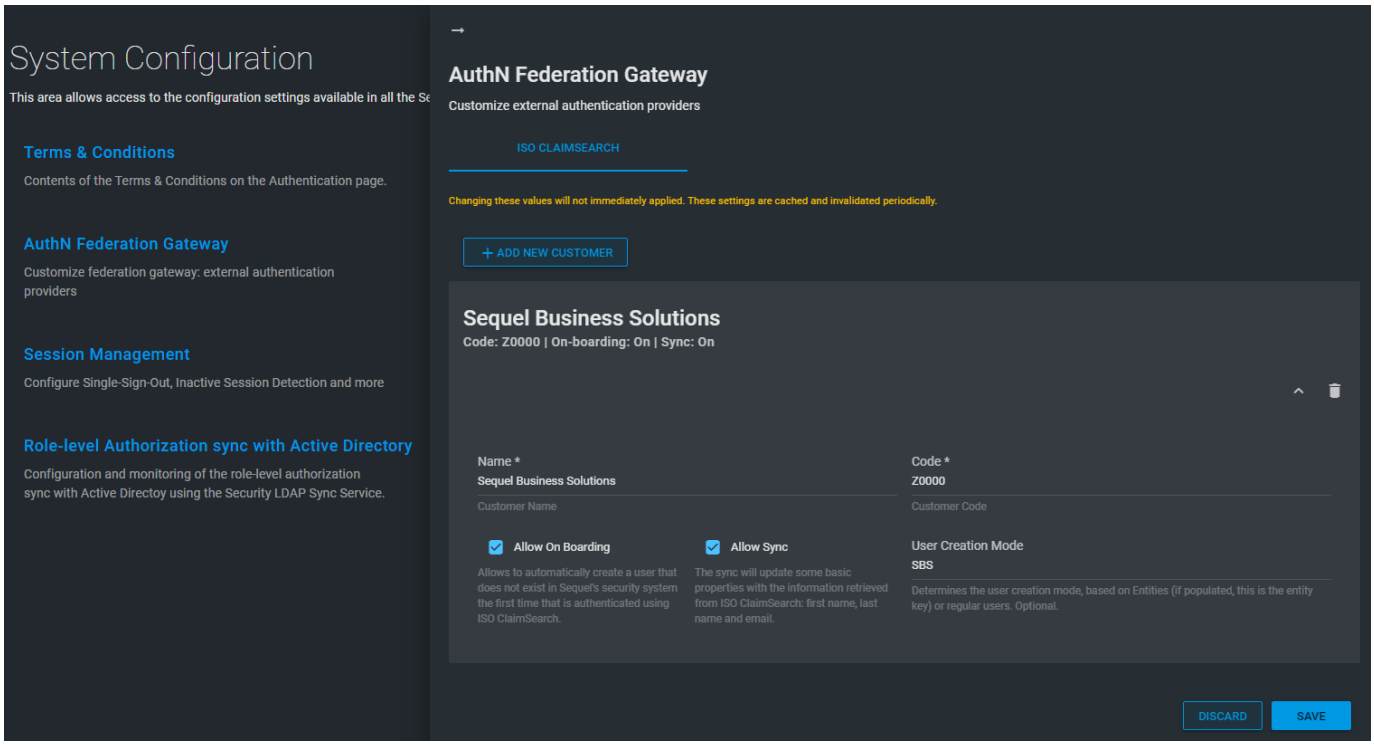


CUSTOMER-ASSOCIATED AUTHENTICATION

ClaimSearch, as a multi-tenant application, host multiple tenants or *customers*. Associate a Security instance with the accepted customer's codes in ClaimSearch is the more secure and recommended configuration to ensure that only users of those customers can access to this instance. When there are no customers configured, the *Light Authentication* mode is used.



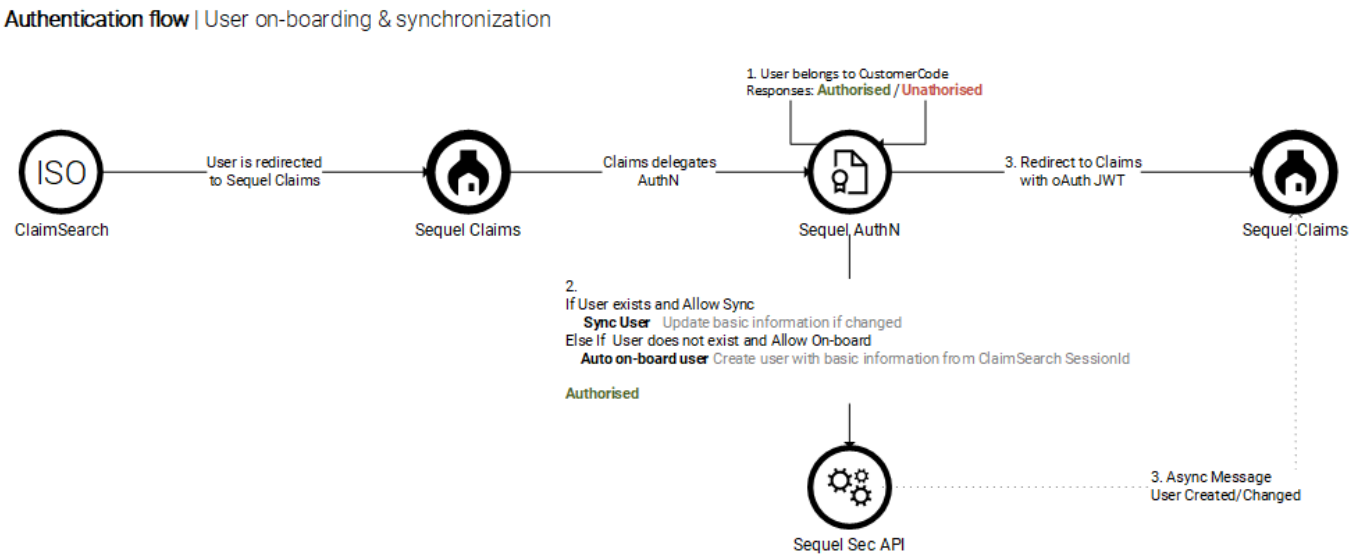
The list of associated *customers* is configured from the Security's Administration site in **AuthN Federation Gateway**. This section is protected by `Sec.ConfigFedGwy` securable.



User sync at authentication

Authentication requires that users exist in Sequel's Security; *auto on-boarding* and *synchronization* features allow to automatically create and keep in sync ClaimSearch users. Both process are triggered as part of each user authentication; so changes in users at ClaimSearch are not reflected in Sequel's Security until the user will successfully login again in Sequel.

These features reduce the maintenance effort of creating users and keeping them up to date. When this option is enabled, the authentication flow changes slightly:



On-boarding

Auto on-boarding process allows to automatically create a user that does not exist in Sequel's security system the first time that is authenticated using *ISO ClaimSearch*. The on-boarding is enabled by customer with the `AllowOnBoarding` property. The user can be created using two modes:

- As a member of an entity. Associate an entity key (`EntityKey`) to the customer entry and the user will be created as a member of this entity using the default permissions for this user.
- As a regular user. Those user can be authenticated in the system, but without permissions. So, they will require manual configuration for granting permissions before the user will be authorized to access any information or perform actions. Keep `EntityKey` empty for using this mode.

Sync

User synchronization can be performed on each authentication challenge done between Sequel's AuthN service and ISO ClaimSearch.

Enable synchronization per customer with the setting `AllowSync` . The sync will update some basic properties with the information retrieved from ISO ClaimSearch: first name, last name and email.

The properties synchronized comes from the response of the UserInfo endpoint in ClaimSearch:

ClaimSearch property	Sequel property	Description
`userId`	`Username` , `SsoUsername`	Unique identifier of the user (name, id). In ClaimSearch is a 5 char string.
`firstName`	`FirstName`	First name
`lastName`	`LastName`	Last name
`customerCode`		Used to authorize users of the associated Customer. Multi-tenant management.
`emailId`	`EmailAddress`	Email Address. It is mandatory for a ClaimSearch user profile and also in Sequel.

Settings summary

RECOMMENDED CONFIGURATION

The recommended configuration of Security for being integrated with ISO Claims search requires some actions:

Infrastructure settings

Get and configure during security installation information related to *ClaimSearch URLs*: Validation Endpoint and Login page URL.

Associate to a customer

Get the *Customer code* associated to this instance and configure it in Configuration section in Administration site.

Configure user sync policy

First of all *design how users will be configured*: if users will be created automatically/manually, and if automatically design and create the entity and configure it. Then at Configuration section in Administration site configure those values for each customer associated.

AUTHENTICATION APPLICATION SETTINGS

Setting	Description
Enabled	Enables ISO ClaimSearch AuthN. Boolean.
SessionValidationEndpoint	Endpoint to validate sessions: like https://servername/products/1.0/userinfo/. Required if enabled.
LoginUrl	ClaimSearch home page or login page.

```
"ClaimSearch":
{
    "Enabled": true,
    "SessionValidationEndpoint": "https://claimsearch-test.iso.com/products/1.0/userinfo/",
    "LoginUrl": "https://claimsearch-test.iso.com/"
}
```

CONFIGURATION SETTINGS

Settings related to customer and how sync works are stored at database in `[configuration].[Setting]` table, with the key `ClaimSearchAuthN`.

Setting	Description
Customers	Collection of customers associated to this security instance. At least, one is required if enabled.
Customers[].Code	Customer code. String. Unique.
Customers[].DisplayName	Display name. String. Unique.
Customers[].AllowOnBoarding	Enable auto on-boarding. Boolean.
Customers[].AllowSync	Enable sync of user's basic details. Boolean.
Customers[].EntityKey	Determines the user creation mode, based on Entities (if populated, this is the entity key) or regular users. Optional.

```
{
    "Customers":
    [
        {
            "Code": "CODE_A",
            "DisplayName": "Customer A",
            "AllowOnBoarding": true,
            "AllowSync": true,
            "EntityKey" : "EntityKeyA" // Associated to entity EntityKeyA
        },
        {
            "Code": "CODE_B",
            "DisplayName": "Customer B",
            "AllowOnBoarding": true,
            "AllowSync": true,
            "EntityKey" : "" // Not associated to an entity
        }
    ]
}
```

5.3.11 Okta AuthN

As part of the *federation gateway* feature the integration with **Okta Authentication** is possible. In this scenario, the users are authenticated against company's Okta domain and it is not required to store passwords in our system.

The Okta authentication is triggered at the login page when the user clicks on the Okta icon or automatically if this is the single provider enabled.

Authentication

BASIC CONFIGURATION

Okta Authentication can be configured at `appsettings.json` in the authentication service: `LoginSettings.IdentityProvidersSettings.Okta`:

- `Enabled` for enabling this provider. Please, check installation guide as there are some IT configurations required before using this provider.
- `UserPolicies` for configuring how user matching and sync will be performed.

USER MATCHING

For being able to login into our system is required that the authenticated Okta user exists in our system. The user matching is done following matching rules defined at `LoginSettings.IdentityProvidersSettings.Okta.UserPolicies.MatchingFields`. The valid options are:

1. `Email`.
2. `SsoUsername`. Any unique identifier for the user, different to the username. It could store an email, this email could be different of the user's email used for notifications.

USER SYNC DURING AUTHENTICATION

During the matching process, it is possible to update some user's properties using the collections `UserPolicies.FieldsToUpdatesWhenNull` and `UserPolicies.FieldsToUpdatesWhenDifferent`. The valid options for updates in both scenarios are:

Option	Okta claim	Sequel's field
`Email`	`Email`	`Email`
`FirstName`	`GivenName`	`FirstName`
`LastName`	`Surname`	`LastName`
`SsoUsername`	`Email`	`SsoUsername`

The default configuration looks like:

```
    "Okta": {
      "Enabled": true,
      "Domain": "_domain_",
      "AuthorizationServerId": null,
      "ClientId": "_clientID_",
      "ClientSecret": "_clientSecret_",
      "UserPolicies": {
        "MatchingFields": [ "Email" ],
        "FieldsToUpdatesWhenDifferent": [ "FirstName", "LastName" ]
      }
    },
```

If UserPolicies is missing or when some of its properties are null or empty default values will apply:

MatchingFields	FieldsToUpdatesWhenDifferent
Email	FirstName, LastName

5.3.12 JumpCloud AuthN

As part of the *federation gateway* feature the integration with **JumpCloud Authentication** is possible. In this scenario, the users are authenticated against JumpCloud and it is not required to store passwords in our system.

The JumpCloud authentication is triggered at the login page when the user clicks on the JumpCloud icon or automatically if this is the single provider enabled.

Authentication

BASIC CONFIGURATION

JumpCloud Authentication can be configured at `appsettings.json` in the authentication service: `LoginSettings.IdentityProvidersSettings.JumpCloud`:

- `Enabled` for enabling this provider. Please, check installation guide as there are some IT configurations required before using this provider.
- `UserPolicies` for configuring how user matching and sync will be performed.

USER MATCHING

For being able to login into our system is required that the authenticated JumpCloud user exists in our system. The user matching is done following matching rules defined at `LoginSettings.IdentityProvidersSettings.JumpCloud .UserPolicies.MatchingFields`. The valid options are:

1. `Email`.
2. `SsoUsername`. Any unique identifier for the user, different to the username. It could store an email, this email could be different of the user's email used for notifications.

USER SYNC DURING AUTHENTICATION

During the matching process, it is possible to update some user's properties using the collections `UserPolicies.FieldsToUpdatesWhenNull` and `UserPolicies.FieldsToUpdatesWhenDifferent`. The valid options for updates in both scenarios are:

Option	JumpCloud claim	Sequel's field
`Username`	`username`	`Username`
`Email`	`Email`	`Email`
`FirstName`	`GivenName`	`FirstName`
`LastName`	`Surname`	`LastName`
`SsoUsername`	`Email`	`SsoUsername`

The default configuration looks like:

```
    "JumpCloud": {
      "Enabled": true,
      "SPEntityId": "_SPEntityID_",
      "X509SigningCertificate": "_signingCertificate_",
      "LoginUrl": "_loginUrl_",
      "UserPolicies": {
        "MatchingFields": [ "Username", "Email" ],
        "FieldsToUpdatesWhenDifferent": [ "FirstName", "LastName" ]
      }
    },
```

If UserPolicies is missing or when some of its properties are null or empty default values will apply:

MatchingFields	FieldsToUpdatesWhenDifferent
Username, Email	FirstName, LastName

5.3.13 User session

In this section we will cover how user's session is managed across all Sequel's products. It is not in the scope of this section to cover single sign-on with external providers, this is covered in the federation gateway sections.

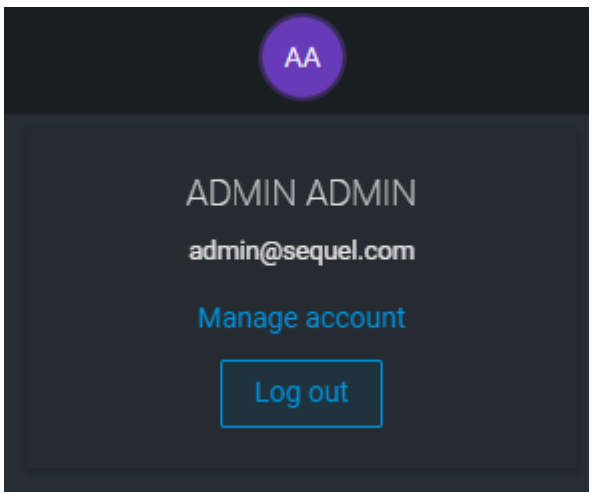
All features described here relays on the *user web component*.

User's session web component

The user's session web component provide three functionalities:

- feedback to the user about the current logged account.
- useful links.
- advanced user experience features like: *single sign-on*, *single sign-out* and *inactive session detection*.

You can know when this component is available if the application has below component:



See User Session Avatar for more information about implementation.

Single sign-on

Single sign-on (SSO) is a property of access control of multiple related, yet independent, software systems. With this property, a user logs in with a single ID and password to gain access to any of several related systems.

For all Sequel's products integrated with the Security Service, once logged from one we can access others without authenticating again. Even more, if we have multiple tabs of the same browser session open, where different Sequel's products are asking the user for credentials, if we log into one of them we will log automatically in all others tabs. This feature is the automatic SSO that is available when all products are installed under the same subdomain.

Single sign-out

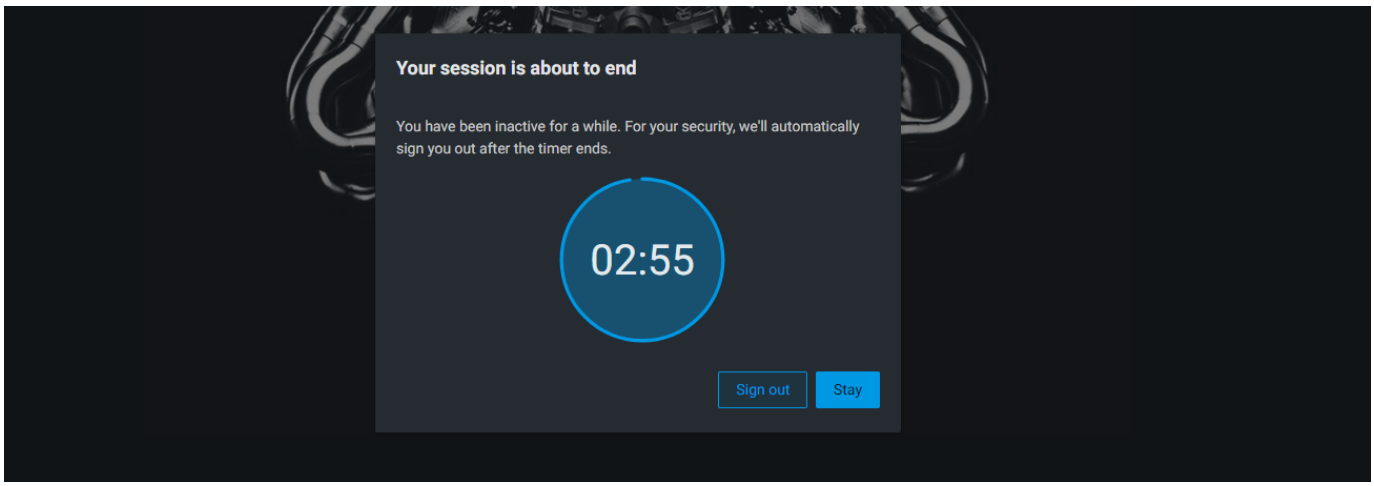
Conversely, single sign-out is the property whereby a single action of signing out terminates access to multiple software systems. This feature is available just for Sequel's products using the SSO and deployed in the same subdomain.

Inactive session

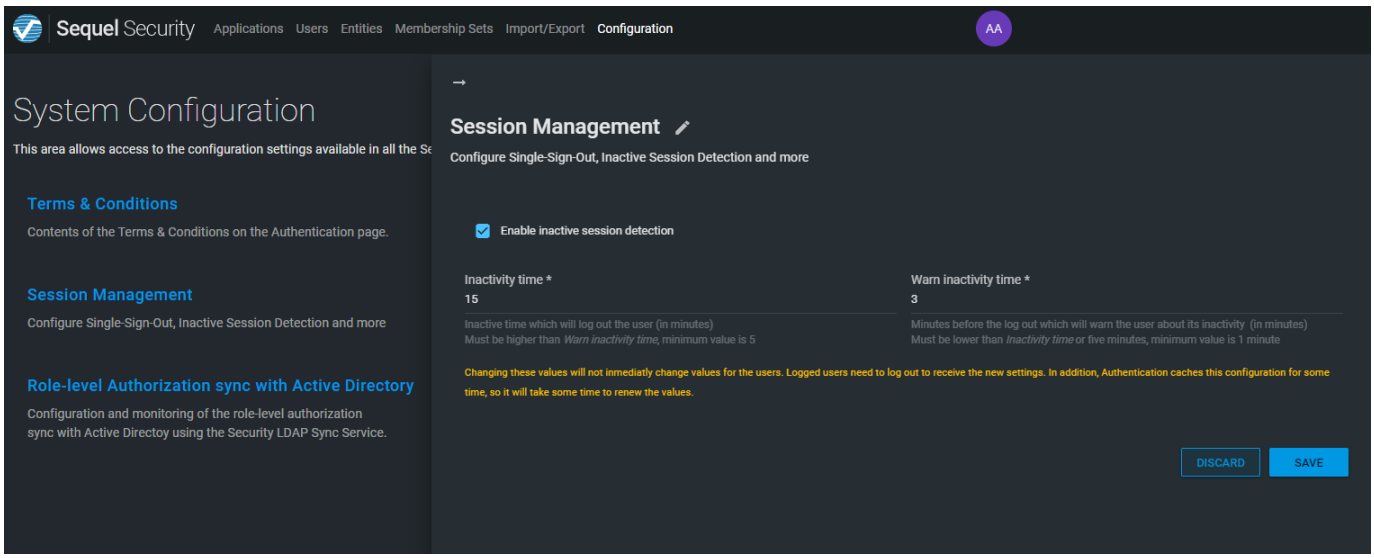
The **inactive session detection** feature allows to automatically force a single sign-out to those inactive sessions.

The inactive session works with two thresholds:

- *inactivity time*: Inactive time which will log out the user (in minutes). Must be higher than Warn inactivity time, minimum value is 5.
- *warn inactivity time*: Minutes before the log out which will warn the user about its inactivity (in minutes). Must be lower than Inactivity time or five minutes, minimum value is 1 minute



Configure it from *Administration website*, at *Configuration* section.



Warning

Changing these values will not immediately change values for the users. Logged users need to log out to receive the new settings. In addition, Authentication caches this configuration for some time, so it will take some time to renew the values.

5.4 Authorization

5.4.1 Authorization Model

Overview

Authorization or access management for resources and data is a critical function for any organization that is using the Sequel's products. **Role-based access control** (RBAC) helps you manage who has access to resources and data, what they can do with those resources, and what data they have access to. Our RBAC system provides fine-grained access management.

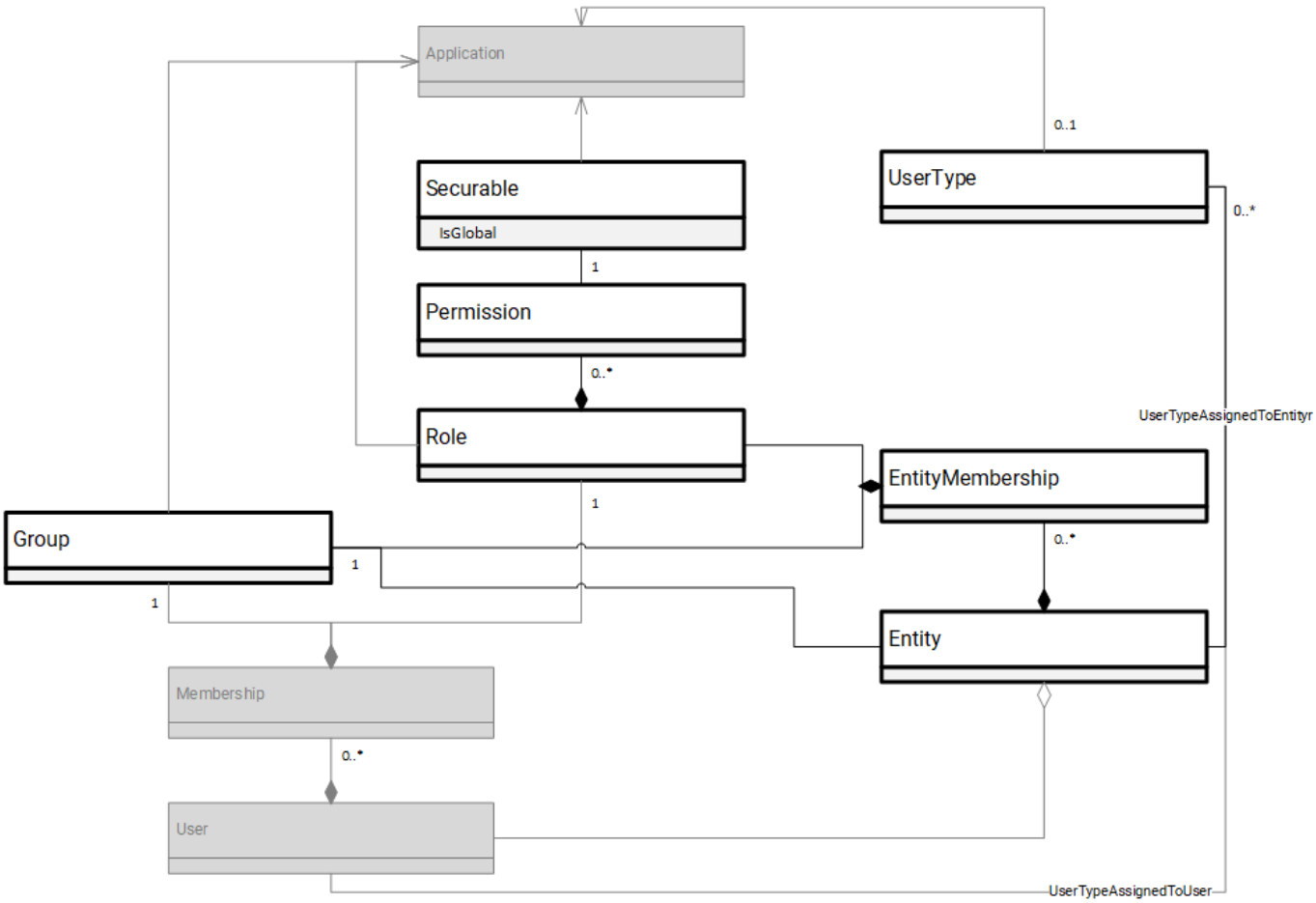
Our RBAC system is mainly based on two concepts:

- **Roles**, is what someone can do with some resources.
- **Groups**, is what someone can do with some data. Groups defined the access control level (ACLs).

A **permission** is a pair of one role and one group; defining what a user can do, based on its role, with some data protected by access control defined by the groups. Although, all permissions are applied always for an specific group and role; there are no concept of highest permissions as permissions are always clearly specified.

Domain model

Under the authentication model we include all domain models required to manage the authorization process: securables, permission, roles, groups and user types.



Also, in this section we will cover aspects related to authorization like entities, membership sets, LDAP synchronization,...

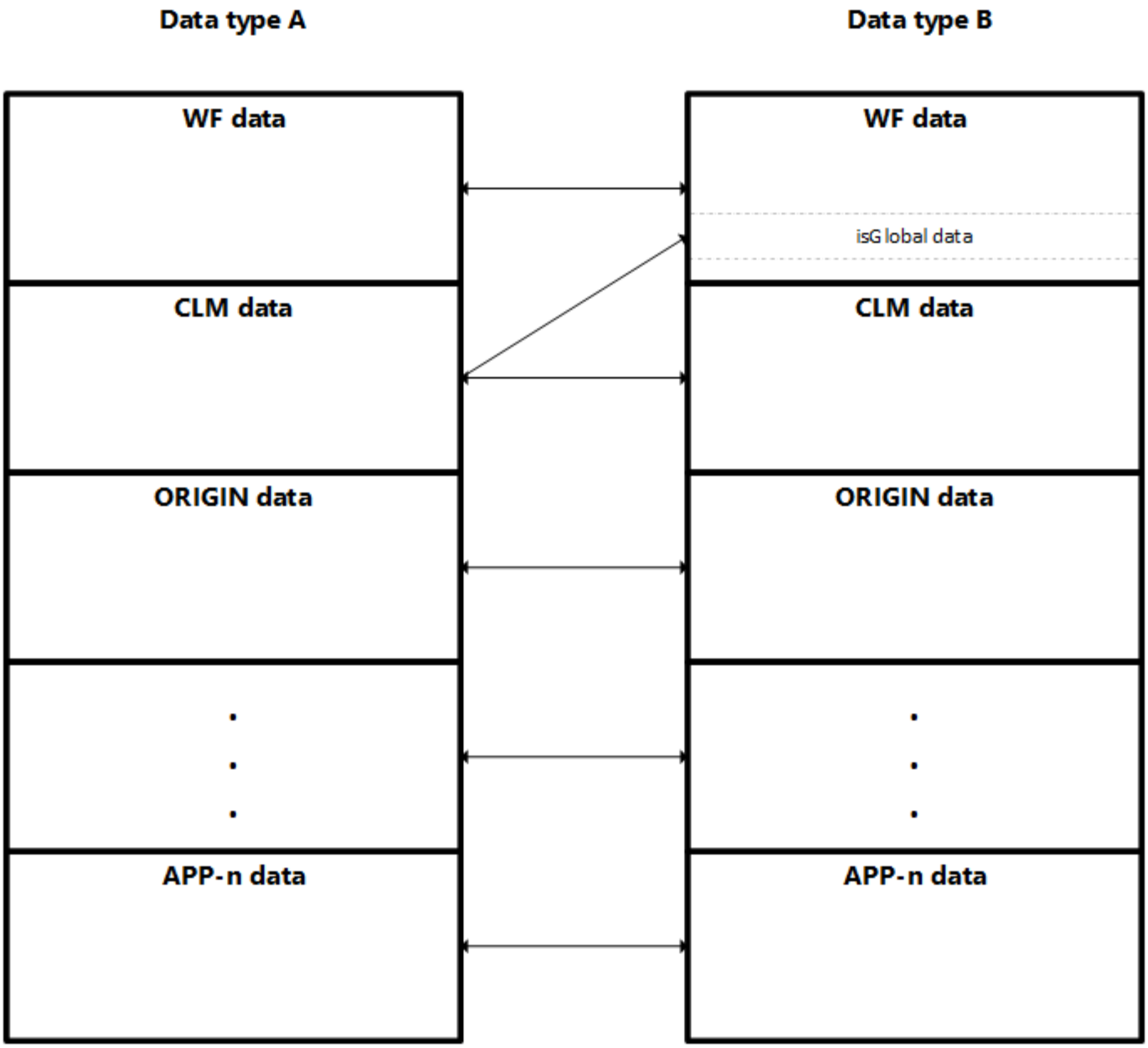
API first

All models and actions described in *authorization* are API first, that means all possible actions and data is exposed through a Rest API. This API is used by the *Administration website*. The API is fully documented using OpenAPI 2.0 specs (aka Swagger).

Applications

Authorization configuration is organized by applications. Configurations of different applications cannot be mixed or related; this isolation of the application's configuration allows us to manage and organise better the configuration. In most of the cases, it does not make sense to relate data of different applications. There is a single exception, it is the data entries flagged as `IsGlobal` that is allowed to be referenced from data owned by other applications: this is useful for applications like Workflow, where the data managed in their process belongs to other application.

We can think on this segregation as a "multi-application" repository, where all data is stored together in the same "table", however the data is never shared between different applications.

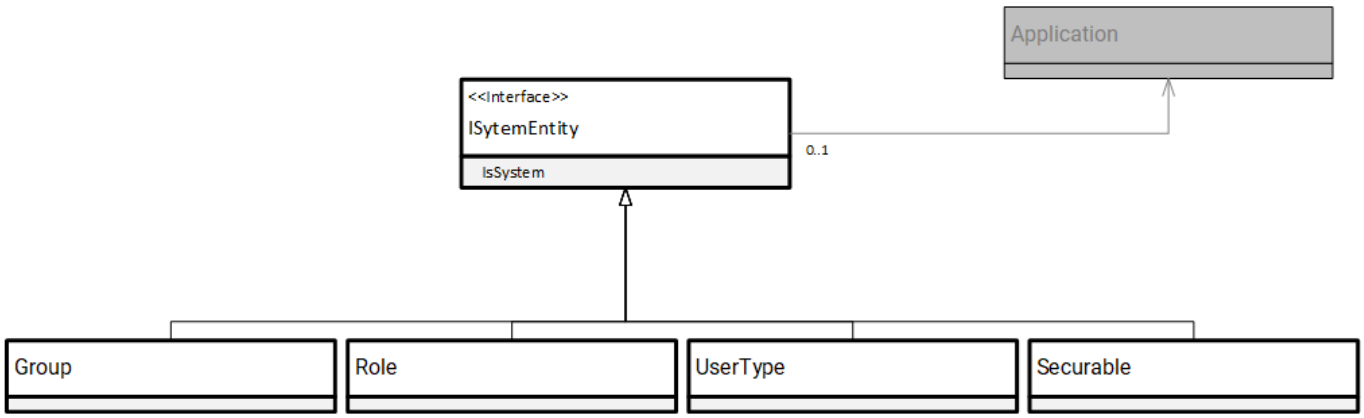


Process like data exchange (import and export) exposes a lot of flexibility on moving authorization data for all applications or just for an specific application.

System configuration

There are some entities that models configurations provided by each application and we consider them as **system configuration** that cannot be modified by the users. Those models are: Group, Role, User Types and Securables. There are some business rules that we can infer from this relationship:

1. Instances flag as `IsSystem=true` are considered System configuration and cannot be edited. Creation of System instances is not allowed as well using the API.
2. All "System configurations" are linked to an application.



5.4.2 Groups

A **group** models the **access control list (ACL)** for different resources. A group represents a set of users with permissions for accessing to an object. E.g. A user can be granted to modify Workflow Tasks; however, this user will just allowed to do it if this Task belongs to one of his groups; so, we are protecting instances of a resource to be edited by other users.

Groups can be part of a system configuration for an application.

All groups created by a user must be assigned to an specific application; for custom groups (not IsSystem) the Key must starts by the `ApplicationKey+"."`

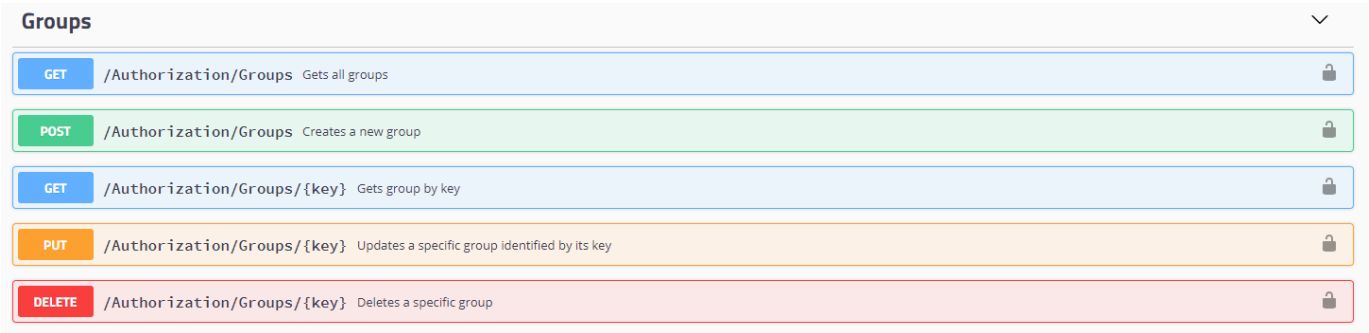
All applications have to define a group with the name *Public*. This is used as a default group; this is commonly used to protect actions in controllers where ACLs concept is not implemented.

Property	Description
Key	String. Unique Key. Required Original implementation was defining the Id as integer. In the new implementation we need to convert it to string to
ApplicationKey	String. Application.Key where this group is defined. Required
Name	String. Friendly name to refer the Application. Optional
AutomaticallyCreated	Boolean. Flags if this group manages automatically created users.
IsSystem	Boolean. Flags if this group is defined by the system or customized by the user. Required. System groups cannot be edited using the Security UI.

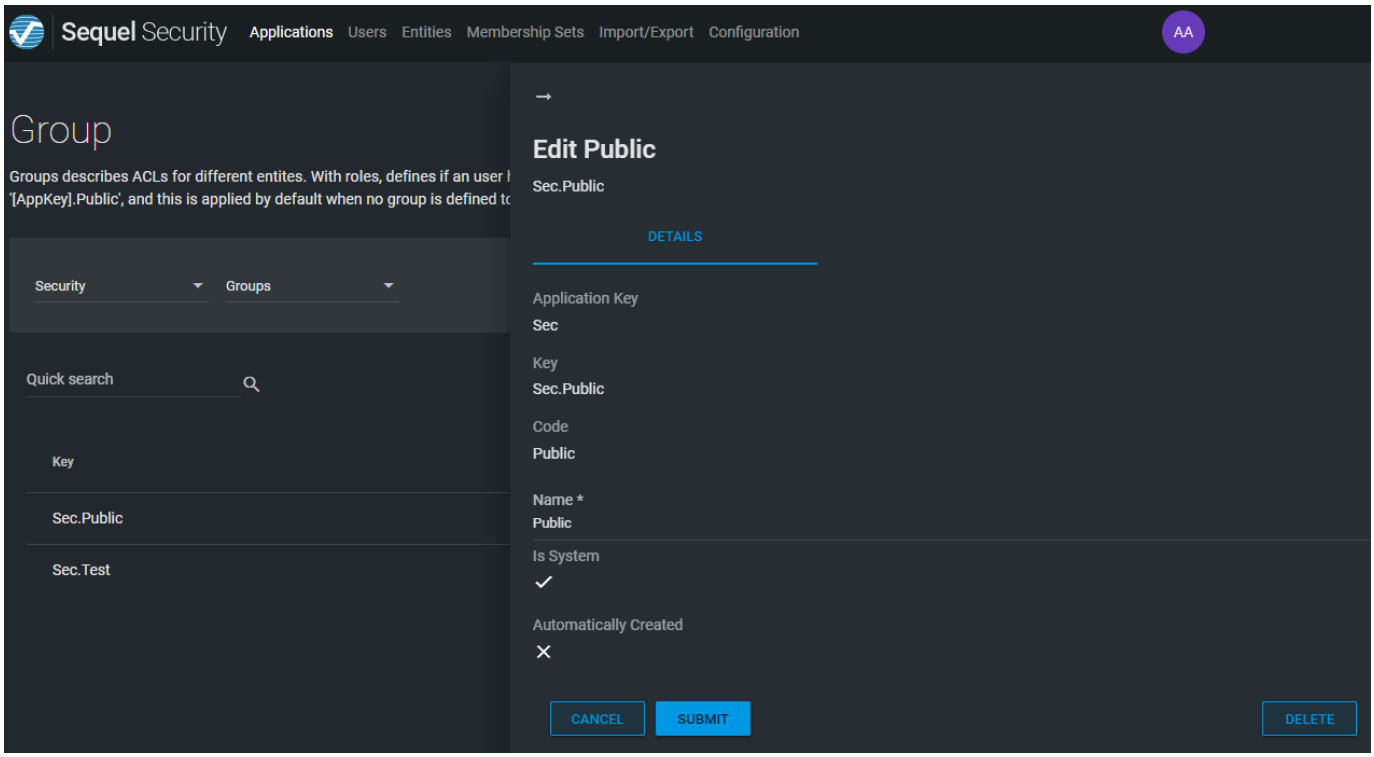
HOW TO MANAGE GROUPS

Roles can be managed from the API resource `\Authentication\Groups` and also from Administration website (`/applications/{ApplicationKey}/groups`). Both offers the same functionality and are protected by securable `Sec.Group` .

API



Administration site



5.4.3 Roles and permissions

Modelling *what a user can do* is done with the concepts or **securables**, **roles** and **permissions**.

Securables

A **securable**, in terms of the Sequel's security model, defines a resource or action that requires special authorization to be accessed or executed. There are mainly two types of securables: the built-in securables defined in the system, and securables created by the user. Built-in securables cannot be modified or deleted. Also, securables must be defined within an application.

Property	Description
Key	String. Unique Key. Required.
ApplicationKey	String. Application.Key where this scope is defined. Required
Name	String. Friendly name to refer the Securable. Required
Description	String. Provide details about the purpose of this securable. Optional.
IsSystem	Boolean. Flags if this securable is defined by the system or customized by the user. Required. System securables cannot be edited using the Security UI.
IsGlobal	Boolean. Flags if this securable is defined as global. Default value is false. Global securables can be assigned to roles of a different application.
IsCreateAllowed	Boolean. Indicates if action is allowed
CreateDescription	String. Description for action. Optional
IsReadAllowed	Boolean. Indicates if action is allowed
ReadDescription	String. Description for action. Optional
IsUpdateAllowed	Boolean. Indicates if action is allowed
UpdateDescription	String. Description for action. Optional
IsDeleteAllowed	Boolean. Indicates if action is allowed
DeleteDescription	String. Description for action. Optional

Securables are always associated to a specific application.

System's securables cannot be created, edited or deleted from the Security API and must be specially managed from the CLI import application (as we don't want the clients to change those configurations using the import tool).

Properties `IsCreateAllowed`, `IsReadAllowed`, `IsUpdateAllowed` and `IsDeleteAllowed` do not apply any extra logic and are used just for improving user experience during configuration.

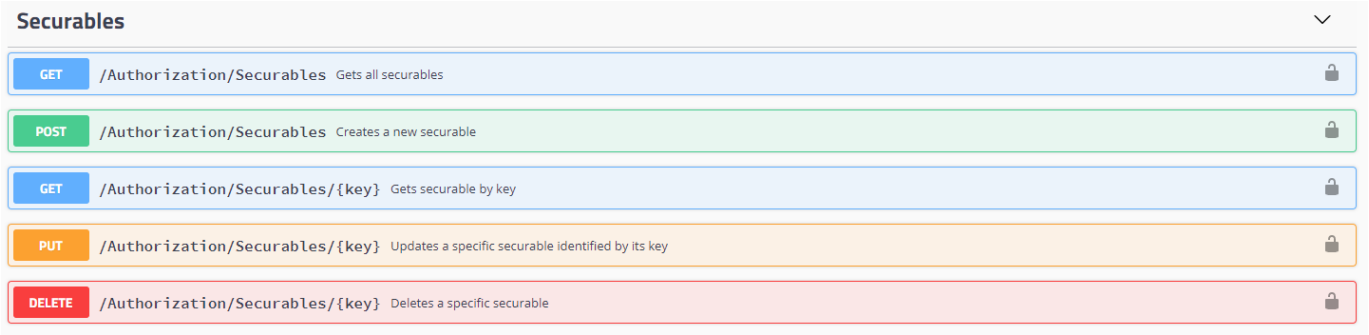
CUSTOM SECURABLE

Users can create its own securables (*custom securable*) and use it to customize functionalities; this will depend on each application customization level. As a sample, Workflow is highly customizable in terms of applying securables to workflows (steps, transitions,...). In other applications like this one, *Security*, it does not make sense to create custom securables as there are no options to protect resources with custom securables.

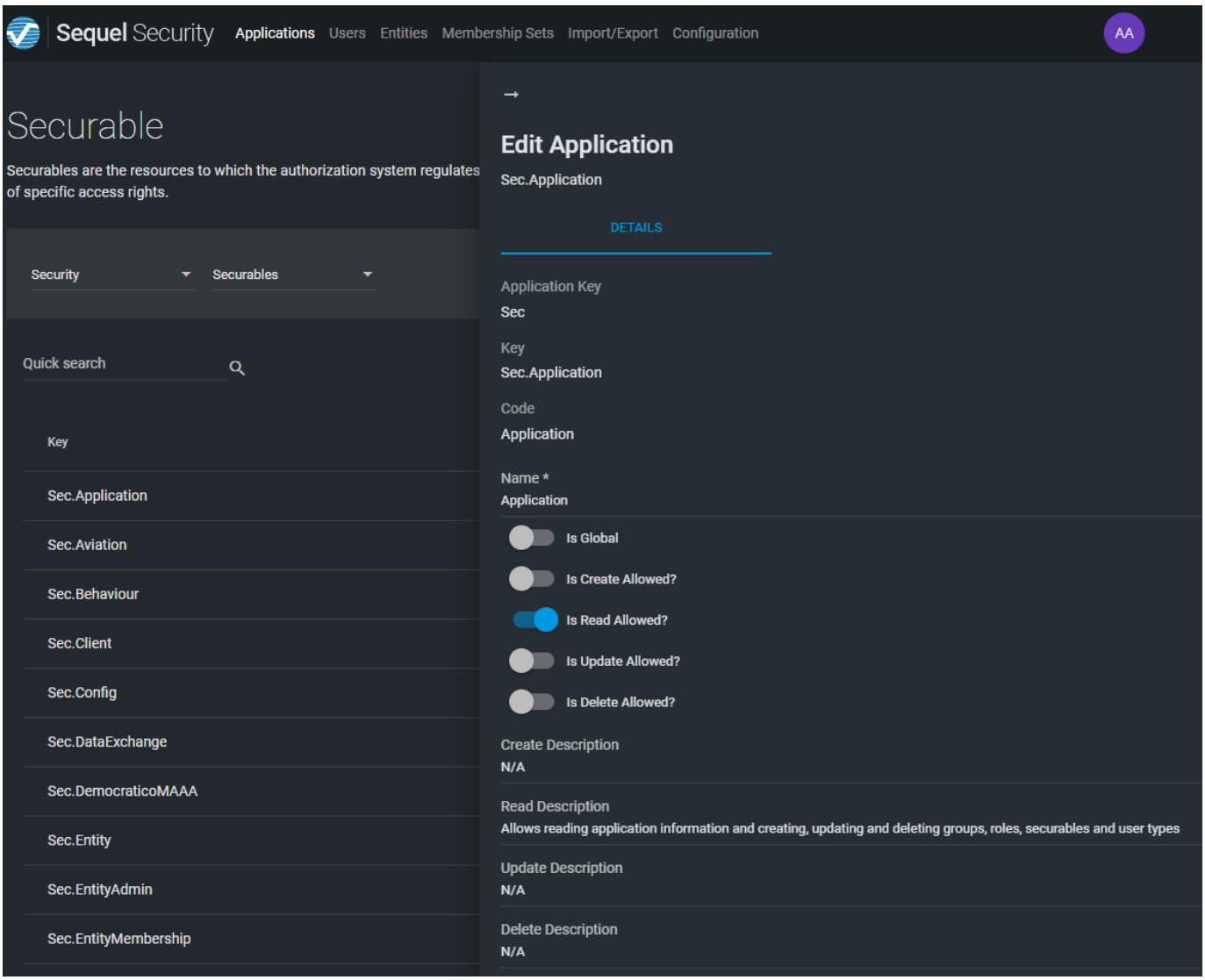
HOW TO MANAGE SECURABLES

Securables can be managed from the API resource `\Authentication\Securables` and also from Administration website (`/applications/{ApplicationKey}/securables`). Both offers the same functionality and are protected by securable `Sec.Securable` .

API



Administration site



Roles

A **role** describes a set of securables and actions over those securables. Users will be assigned to a role in order to define the effective permissions.

Roles can be part of a system configuration for an application.

All groups created by a user must be assigned to a specific application; for custom groups (not IsSystem) the Key must starts by the `ApplicationKey+"."`

Property	Description
Key	String. Unique Key (within the application) . Required.
ApplicationKey	String. Application.Key where this role is defined. Optional
Name	String. Friendly name to refer the role. Required
IsSystem	Boolean. Flags if this role is defined by the system or customized by the user. Required. System groups cannot be edited using the Security UI.

Also, a role is composed by a set of permissions.

PERMISSIONS

A **permission** is a pair of one role and one group; defining what a user can do, based on its role, with some data protected by access control defined by the groups. Although, all permissions are applied always for an specific group and role; there are no concept of highest permissions as permissions are always clearly specified. A permission allows to specify which CRUD actions (Create, Read, Update and Delete) are allowed over a specific securable for users assigned to a specific role.

Property	Description
Id	Integer. Unique id to refer the permission . Internal. Autogenerated
RoleKey	String. Role.Key where this permission is defined. Required
SecurableKey	String. Securable.Key which this permission is configuring. Required.
Create	Boolean. Create action is allowed. Required
Read	Boolean. Action is allowed. Required
Update	Boolean. Action is allowed. Required
Delete	Boolean. Action is allowed. Required

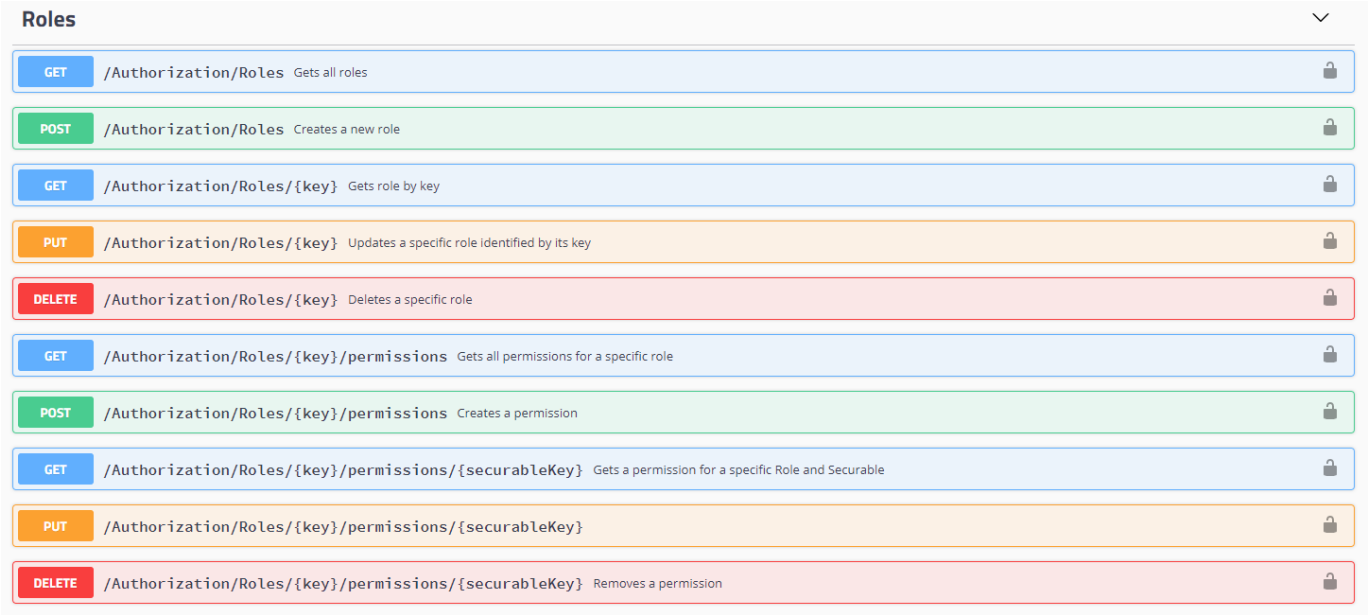
Same application rule

All roles belongs to an Application; we just can include permissions from the same application; with the exception of *global securables* that can be added to roles of a different application.

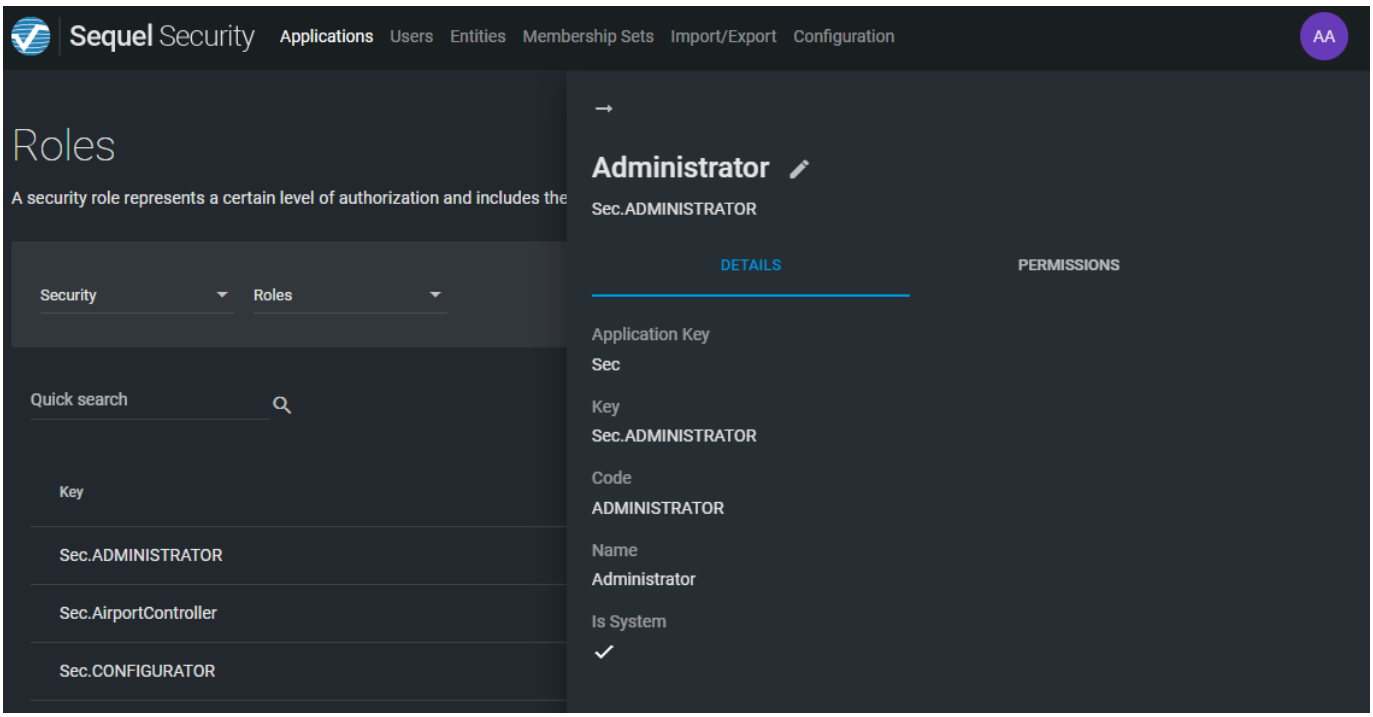
HOW TO MANAGE ROLES

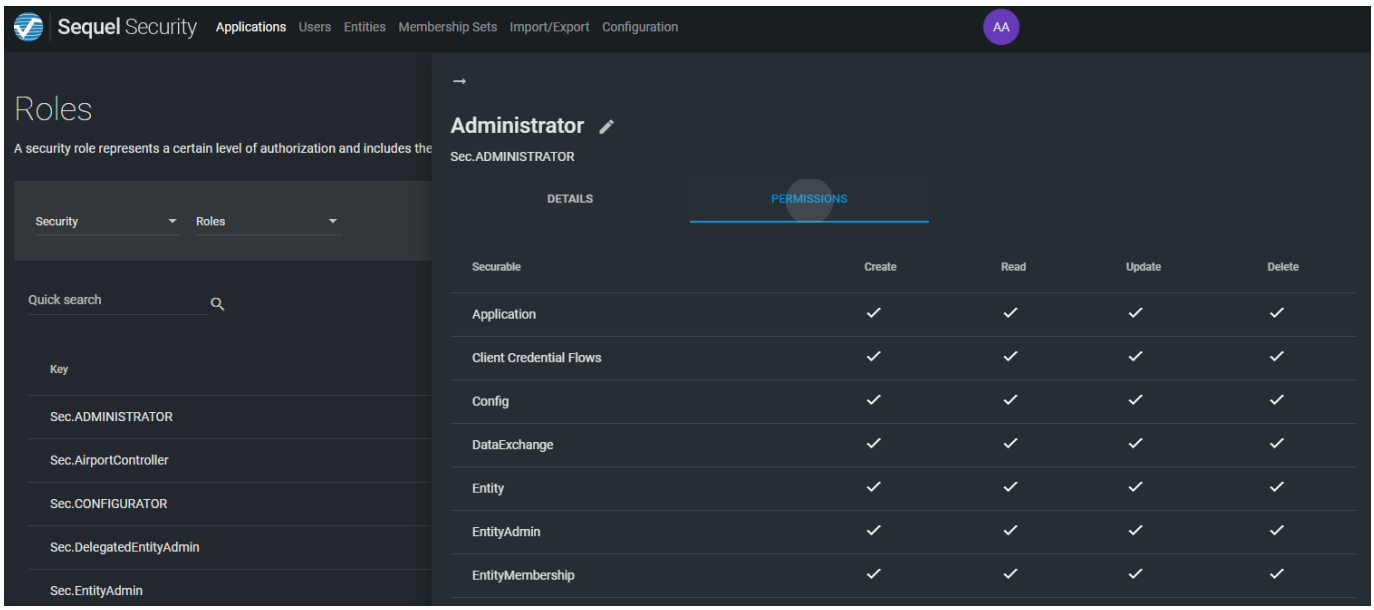
Roles can be managed from the API resource `\Authentication\Roles` and also from Administration website (`/applications/{ApplicationKey}/roles`). Both offers the same functionality and are protected by securable `Sec.Role` .

API



Administration site





5.4.4 User Types

A **user type** describes a mechanism to categorize users. It is not related to authentication or authorization and is used by applications to customize the user experience based on the type of the user.

User Types can be part of a system configuration for an application.

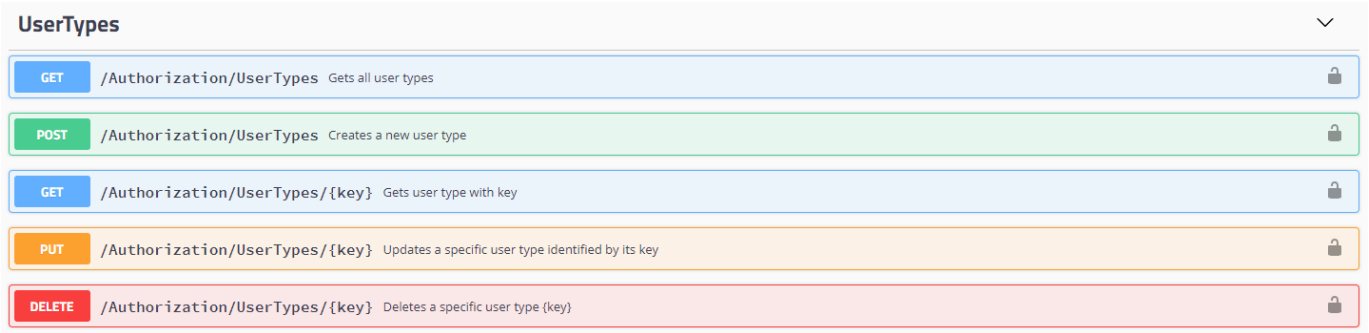
All User Types created by a user must be assigned to a specific application.

Property	Description
Key	String. Unique Key (within the application) . Required.
ApplicationKey	String. Application.Key where this role is defined. Optional
Name	String. Friendly name to refer the role. Required
IsSystem	Boolean. Flags if this role is defined by the system or customized by the user. Required. System user types cannot be edited using the Security UI. Previously this field was named IsInternal

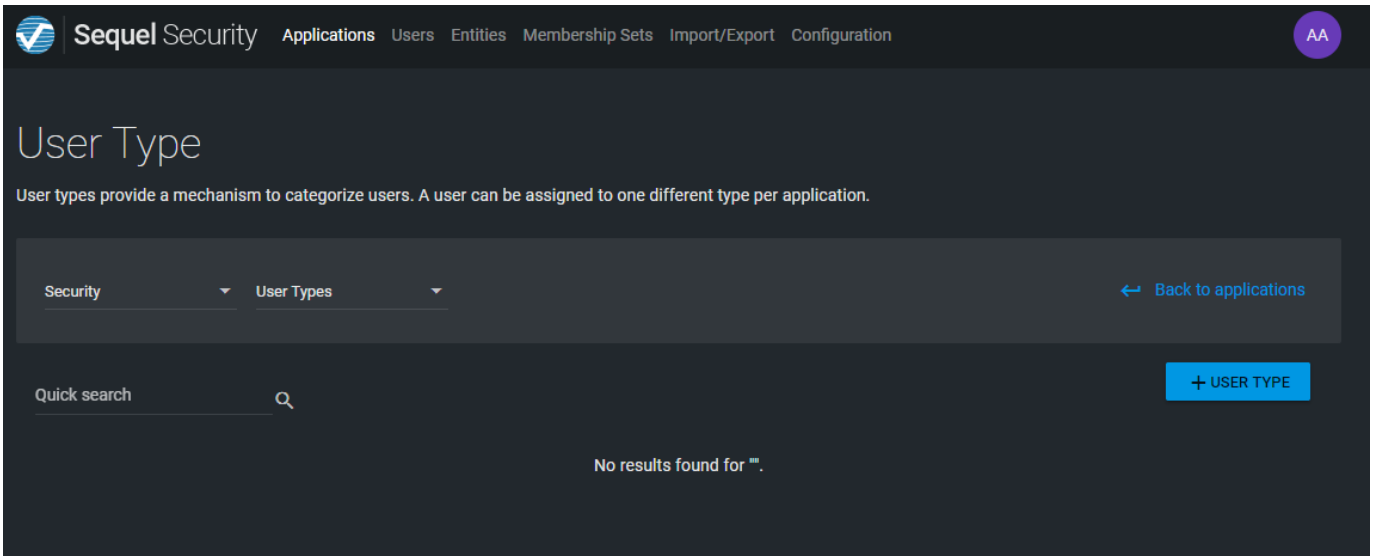
HOW TO MANAGE USER TYPES

Roles can be managed from the API resource `\Authentication\UserTypes` and also from Administration website (`/applications/{ApplicationKey}/user-types`). Both offers the same functionality and are protected by securable `Sec.UserType` .

API



Administration site



5.4.5 Entities

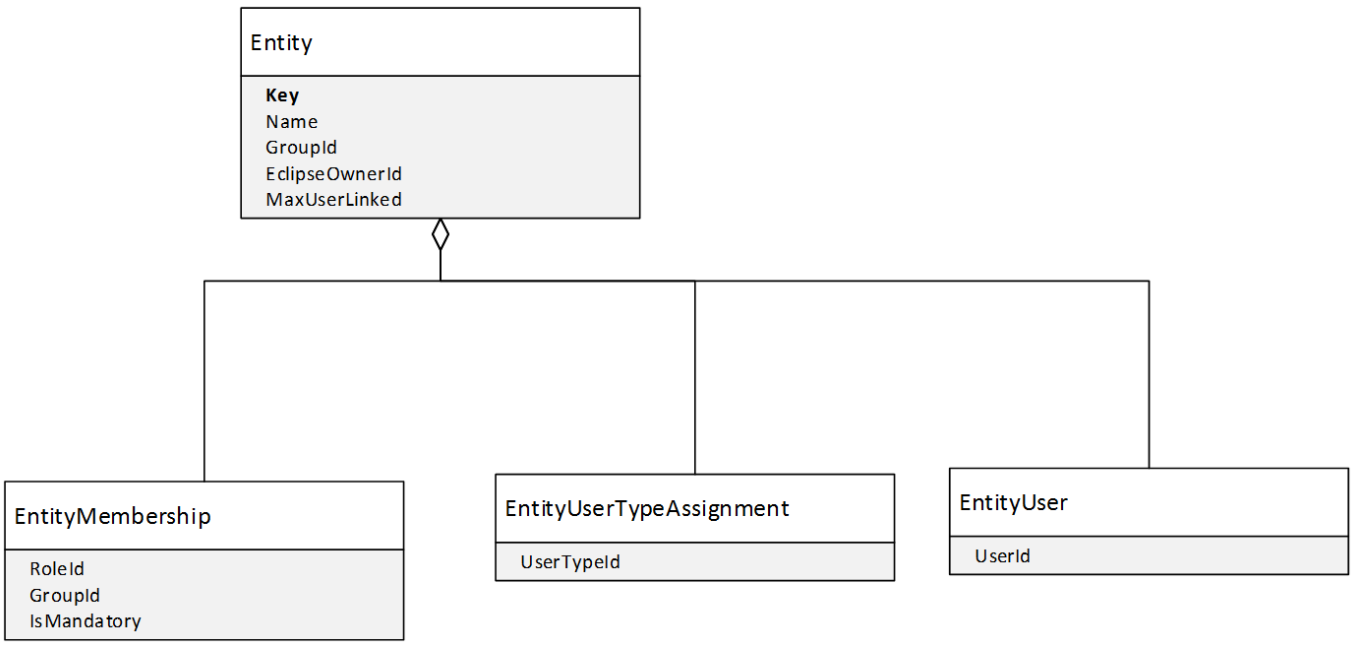
An *entity* offers a framework for managing a group of users that we grant autonomy for handling its own accounts based on a predefined set of rules. An entity can help to model a subdivision of an organization; or it could be used for allowing a company to expose services to partners (modelled as entities).

With an *entity* we can:

- Define a set of users that belongs to a same team. Linking this entity to a main security group.
- Define security templates for creating and managing those users: memberships and user types.
- Allow delegated management of user within an entity to "delegated entity admin" users.
- Driven creation of users based on templates.
- Limited management of users: change first, last names and some membership changes.
- Relate the entity to an "OwnerId" in the Eclipse system.
- Entity data support ACLs.

Data model

An Entity has been modelled as described below:



ENTITY

Defines an entity.

Field	Description	Validations
Key	Primary key, it's a string defined by the user at creation time	Required. Unique. MaxLength(270). Valid Code (see security documentation)
Name	Name used in UI to refer the entity	Required. Unique. MaxLength(270).
GroupId	Refers to a security group. Used for ACLs check.	
EclipseOwnerId	Optional value for linking the entity with an Eclipse Owner Id	Number
MaxUserLinked	Limits the number of active users allowed in the entity. 0 = Unlimited. >0 = Maximum number of active users.	Number

ENTITY MEMBERSHIP

Defines the available memberships for this entity. This information will be used later for:

- Creating a new user in an entity will automatically assign those memberships.
- Editing users in an entity will allow to manage just those memberships defined in this collection. Mandatory memberships cannot be changed; neither, other memberships assigned to a user that are not defined in this collection.

Field	Description	Validations
RoleId GroupId	Primary key. Defines a membership.	A valid membership (role and group belongs to the same application)
IsMandatory	Flags a membership as mandatory; later, when this is assigned to a user (in the context of an entity) it will not be possible to remove.	Boolean

By design, we cannot assume that users in an entity will have the same memberships that defined here; also, memberships are not synchronized with users.

ENTITY USER TYPE ASSIGNMENT

Defines the available user types for this entity. This information will be used later for:

- Creating a new user in an entity will automatically assign those user types.
- Editing users in an entity will allow to manage just those user types defined in this collection.

Field	Description	Validations
UserTypeId	Primary key.	A valid user type.

By design, we cannot assume that users in an entity will have the same user types that defined here; also, user types are not synchronized with users.

ENTITY USER

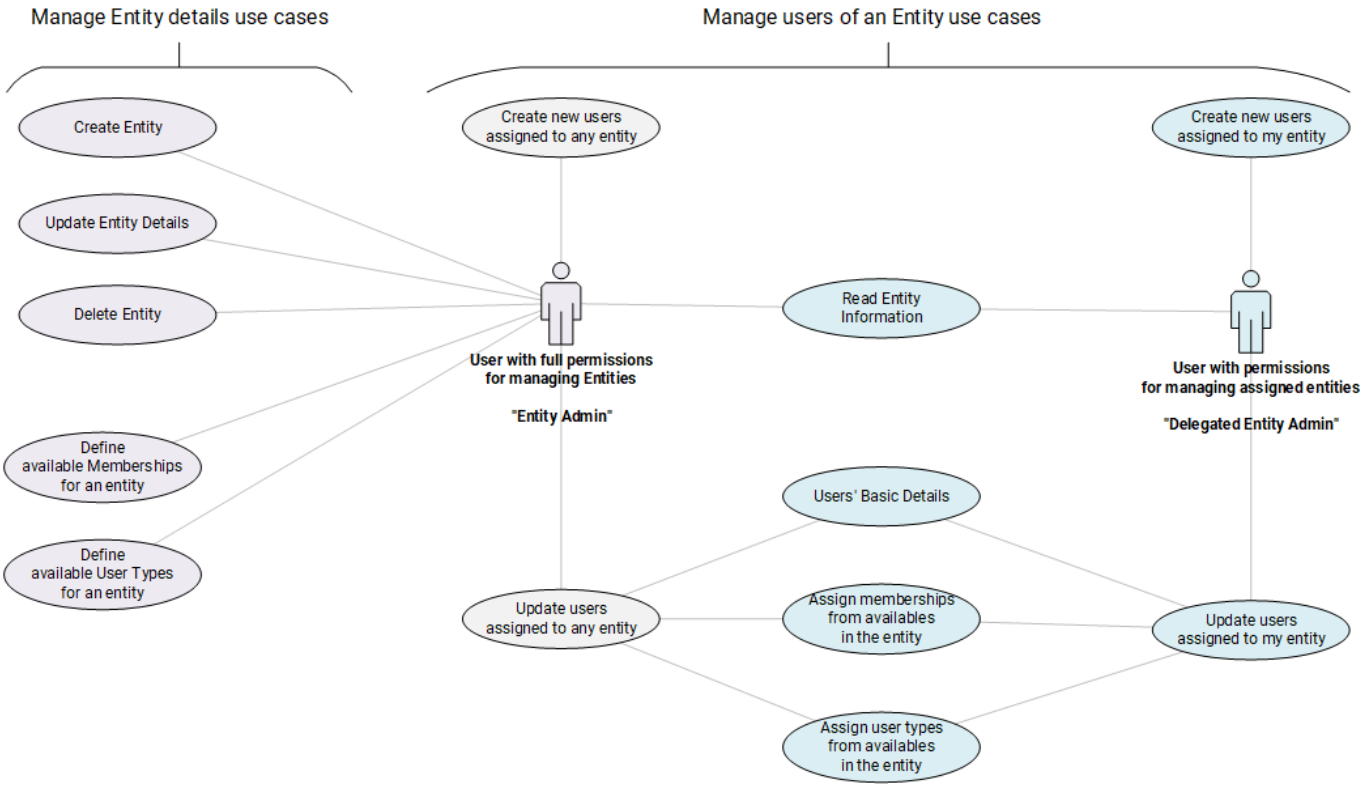
Stores the list of user included in an entity. From a persistence point of view, we support that a single user will be assigned to multiple entities (open design). However, this is not supported yet by the system.

The information offered about a user in the context of an entity doesn't contain all information in the user. It's a reduced view of the user by design; containing some editable information like: first and last name, email, active end date; and other information about the state of the user in read only mode.

Use cases

Obviously, entities can be managed by a "super security admin" user. However, makes more sense if we manage it and define the use cases based on two roles: an Entity Admin and a Delegated Entity Admin.

Below diagram shows all uses cases that we are covering with Entities:



Entity Admin represents a user with full permissions for managing entities; without any control of ACLs. This role, in term of security, is defined in Sec.EntityAdmin role; key setting here is full permission for Sec.Entity and Sec.EntityAdmin. This roles contains below permissions:

Securable	Create	Read	Update	Delete
Application	✗	✗	✗	✗
Entity	✓	✓	✓	✓
EntityAdmin	✓	✓	✓	✓
EntityMembership	✓	✓	✓	✓
EntityUser	✓	✓	✓	✓
EntityUsersMembership	✓	✓	✓	✓
EntityUsersUserType	✓	✓	✓	✓
EntityUserType	✓	✓	✓	✓
Group	✗	✓	✗	✗
Membership	✗	✗	✗	✗
Permission	✗	✓	✗	✗
Role	✗	✓	✗	✗
Securable	✗	✗	✗	✗
User	✗	✗	✗	✗
UserType	✗	✓	✗	✗

Delegated Entity Admin Represents a user that belongs to an entity and has been granted with permissions for managing users in this entity. This role, in term of security, is defined in `Sec.DelegatedEntityAdmin` role. Key setting here is don't add `Sec.EntityAdmin` permissions, allow just read for `Sec.Entity` and allow full permissions for `Sec.EntityUser*` securables. This roles contains below permissions:

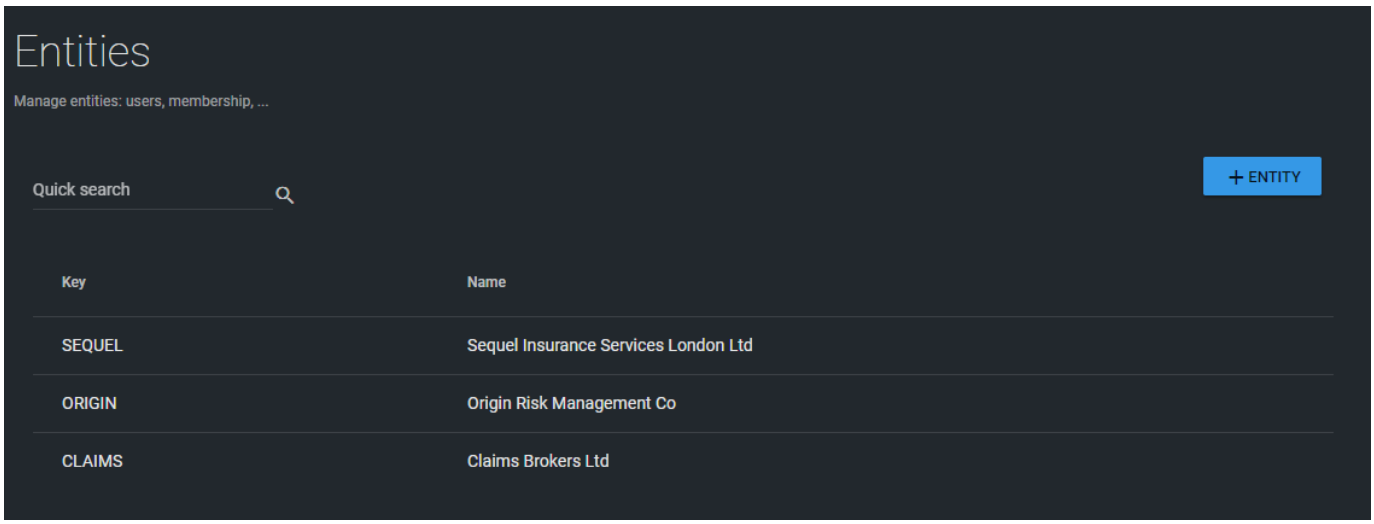
Securable	Create	Read	Update	Delete
Application	✗	✗	✗	✗
Entity	✗	✓	✗	✗
EntityAdmin	✗	✗	✗	✗
EntityMembership	✗	✓	✗	✗
EntityUser	✓	✓	✓	✓
EntityUsersMembership	✓	✓	✓	✓
EntityUsersUserType	✓	✓	✓	✓
EntityUserType	✗	✓	✗	✗
Group	✗	✓	✗	✗
Membership	✗	✗	✗	✗
Permission	✗	✓	✗	✗
Role	✗	✓	✗	✗
Securable	✗	✗	✗	✗
User	✗	✗	✗	✗
UserType	✗	✓	✗	✗

We recommend to use those roles to configure the users for managing both scenarios; as those roles contains the required permissions already configured.

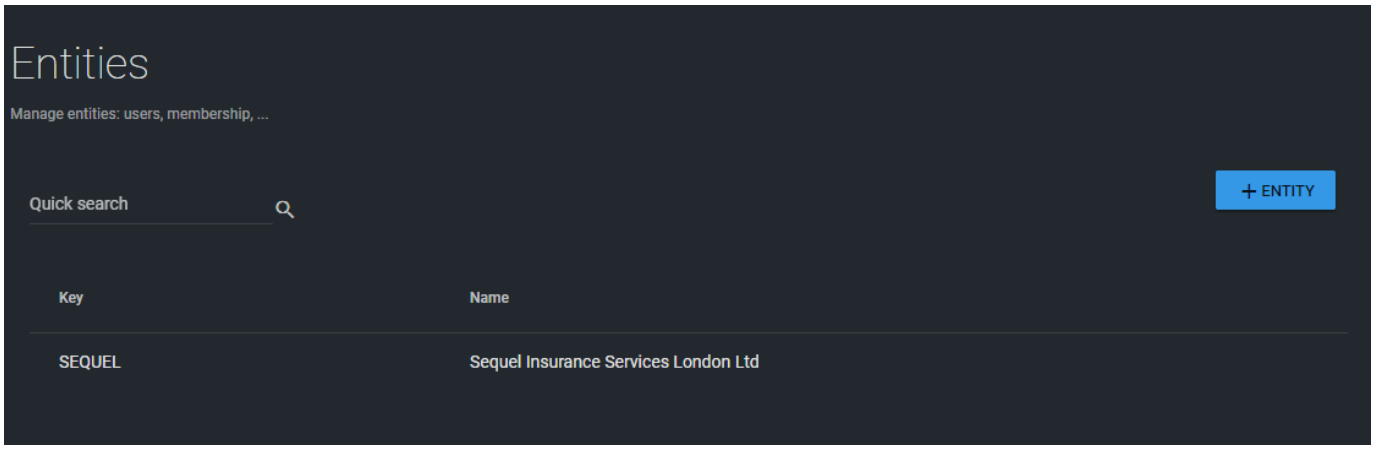
ENTITY MANAGEMENT USE CASES

Read Entity Information

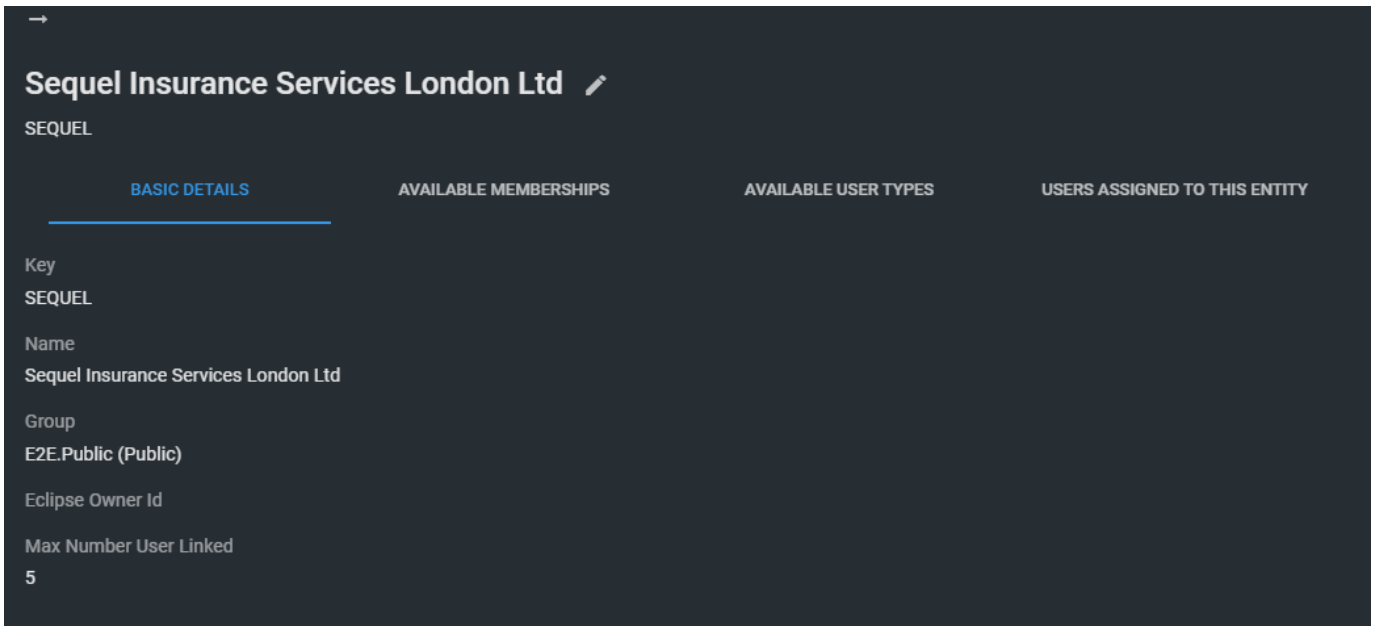
```
As a user with permission for [Sec.Entity|Read]
   And with permission for [Sec.EntityAdmin]
   When I navigate to Entity page
   Then I can see all entities
```




```
As a user with permission for [Sec.Entity|Read]
   And without permission for [Sec.EntityAdmin]
   And with ACLs permissions for some X entities
   When I navigate to Entity page
   Then I can see just X entities
```

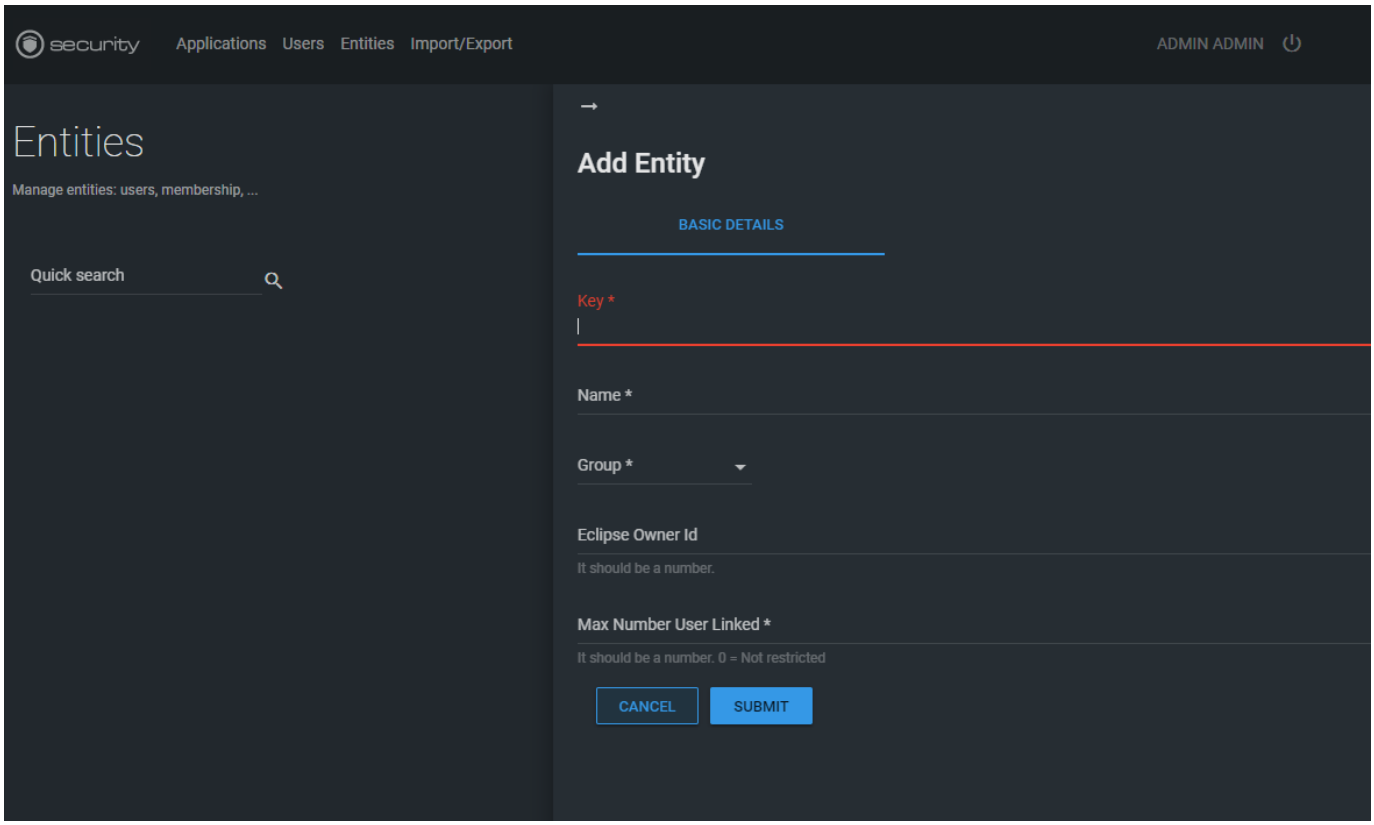


Selecting any entity; we will be able to open the entity screen. In read-only mode; the pencil for editing the entity is managed by `[Sec.Entity|Update]`



Create Entity

```
As a user with permission for [Sec.Entity|Create]
  And with permission for [Sec.EntityAdmin]
  When I navigate to Entity page
  Then I can create a new entity
```

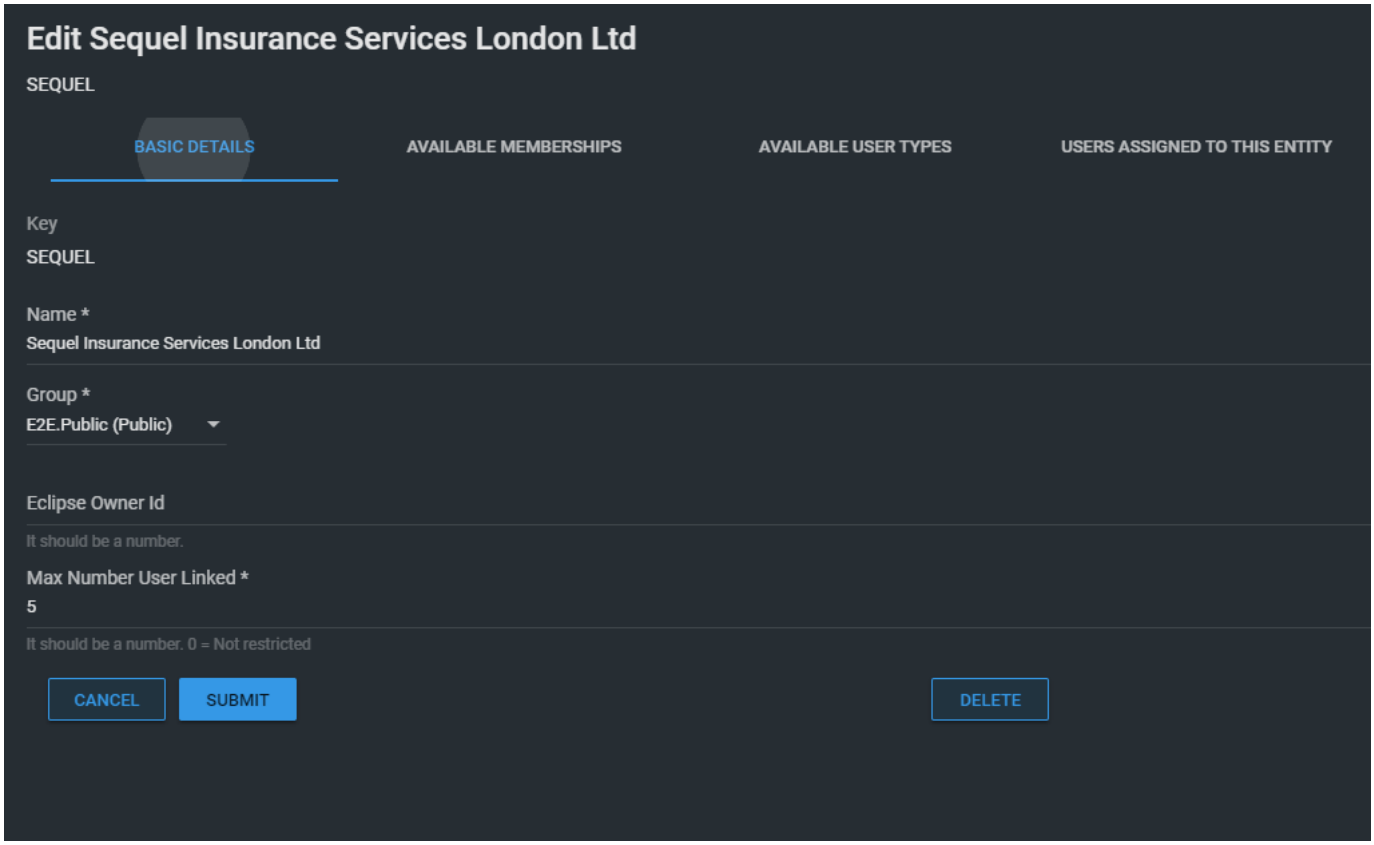


There are no restrictions on assigning a group. So, we can use public ones and reuse the same group with different entities. A good design of entities and groups are important. However, keep in mind that permissions for the user are not defined by this value. Permissions for the users are defined in the membership collection.

Concerns and future work: there are some conversations about the benefits of allowing multiple groups assigned to an entity or maybe define default groups by application for driving logic in the downstream of other applications. Nothing of this was included in the first version, considered the MVP.

Update Entity details

```
As a user with permission for [Sec.Entity|Update]
  And with permission for [Sec.EntityAdmin] or ACLs access to entity E
  When I navigate to entity E page
  Then I can edit  entity E
```



Delete Entities

```
As a user with permission for [Sec.Entity|Delete]
  And with permission for [Sec.EntityAdmin] or ACLs access to entity E
  When I navigate to entity E page
  Then I can edit  entity E
```

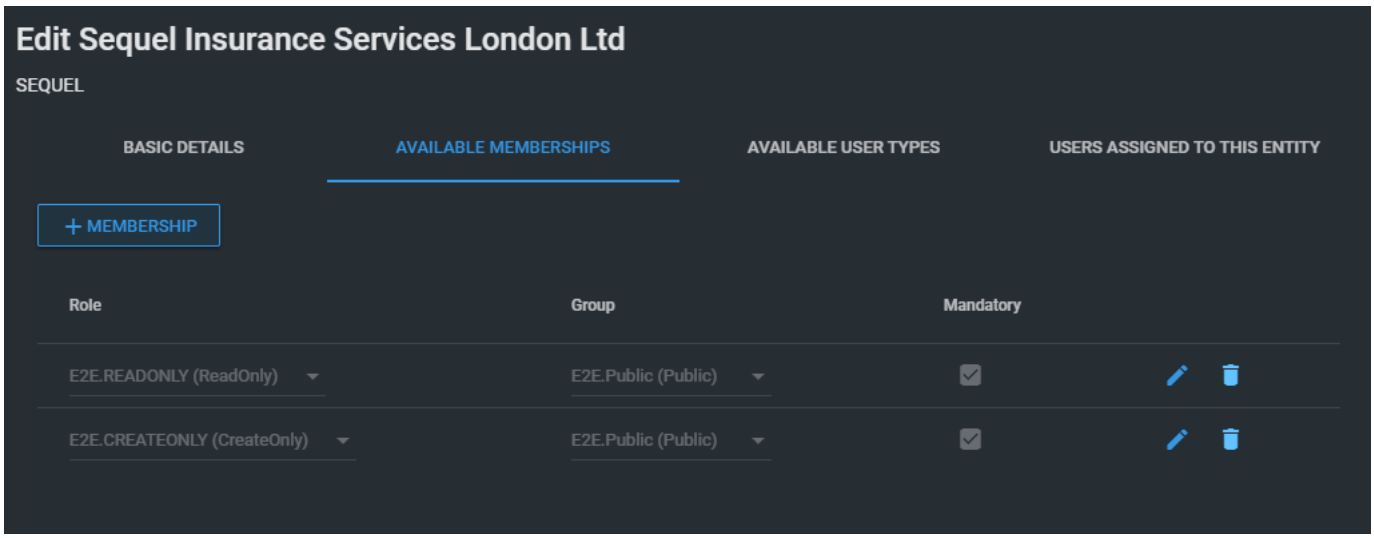
Deletion of entities is not allowed if there are users assigned to this entity.

Define Available Memberships for an entity

```
As a user with permission for [Sec.Entity|Update]
  And  with permission for
          [Sec.EntityMembership|Create] or/and
          [Sec.EntityMembership|Update] or/and
          [Sec.EntityMembership|Delete]
  And with permission for [Sec.EntityAdmin]
  When I navigate to entity drawer
  Then I can add/edit/delete memberships
```

Memberships included in the entity will drive the creation of users later and also the available (and editable) memberships for users within the context of the entity. Those limitations don't affect to user management screen and API endpoints.

Membership can be flagged as mandatory; this will prevent to `Sec.DelegatedEntityAdmin` users to remove those memberships; so we can protect the configuration against invalid options.



Define Available User Types for an entity

```
As a user with permission for [Sec.Entity|Update]
  And  with permission for
           [Sec.EntityUserType|Create] or/and
         [Sec.EntityUserType|Update] or/and
         [Sec.EntityUserType|Delete]
  And with permission for [Sec.EntityAdmin]
  When I navigate to entity drawer
  Then I can add/edit/delete user types
```

User types included in the entity will drive the creation of users later and also the available (and editable) user types for users within the context of the entity. Those limitations don't affect to user management screen and API endpoints. Just one user type per application can be configured

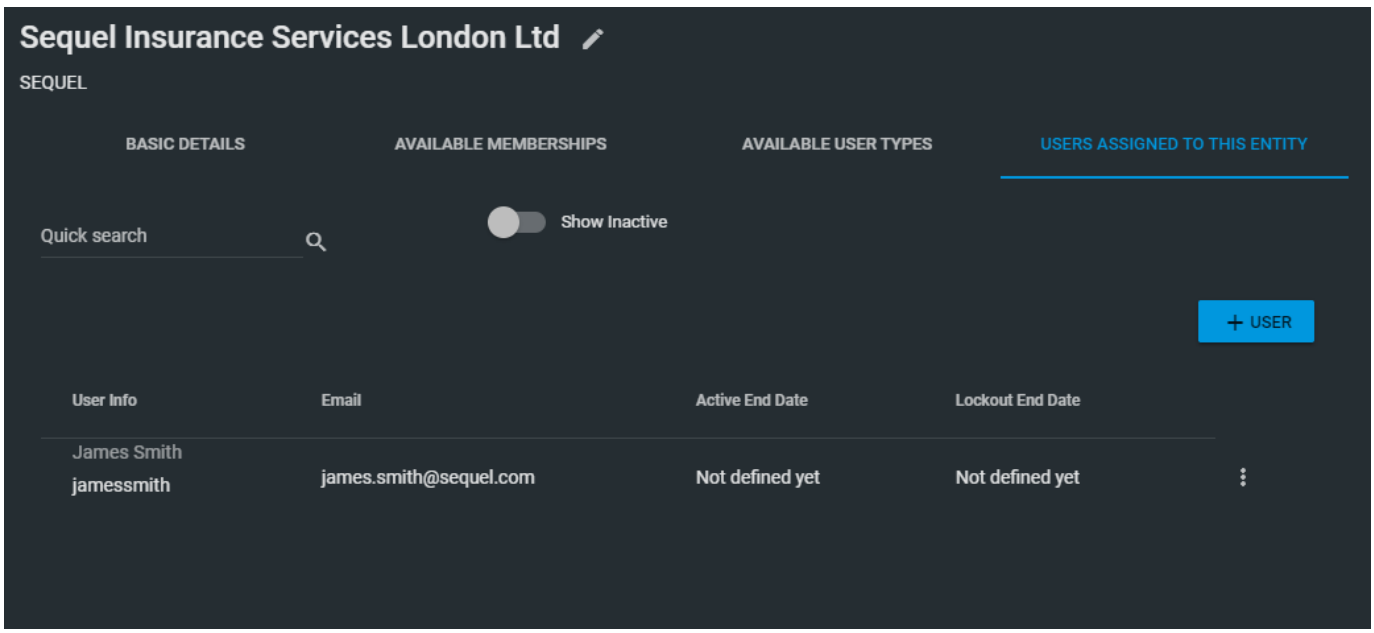
Concerns and future work: Allow to define multiple user types per application at entity level will allow to configure users in the entity with different user types. This is not possible currently.

Users of an entity

Users assigned to an entity are displayed in the "user assigned to this entity" tab. There are some basic filtering options.

Create user is protected by Sec.EntityUser-Create permission.

Clicking on a user, we will be able to access to the read only screen of this user. From this screen, we can edit it; in case this is allowed.



Concerns and future work:

- Deletion of users is not supported from the UI. Currently, this will mimic using active/inactive users.
- Unlink user from an entity is not supported. Requires analysis to decide the desired behaviour and implications of this action.

Create new users

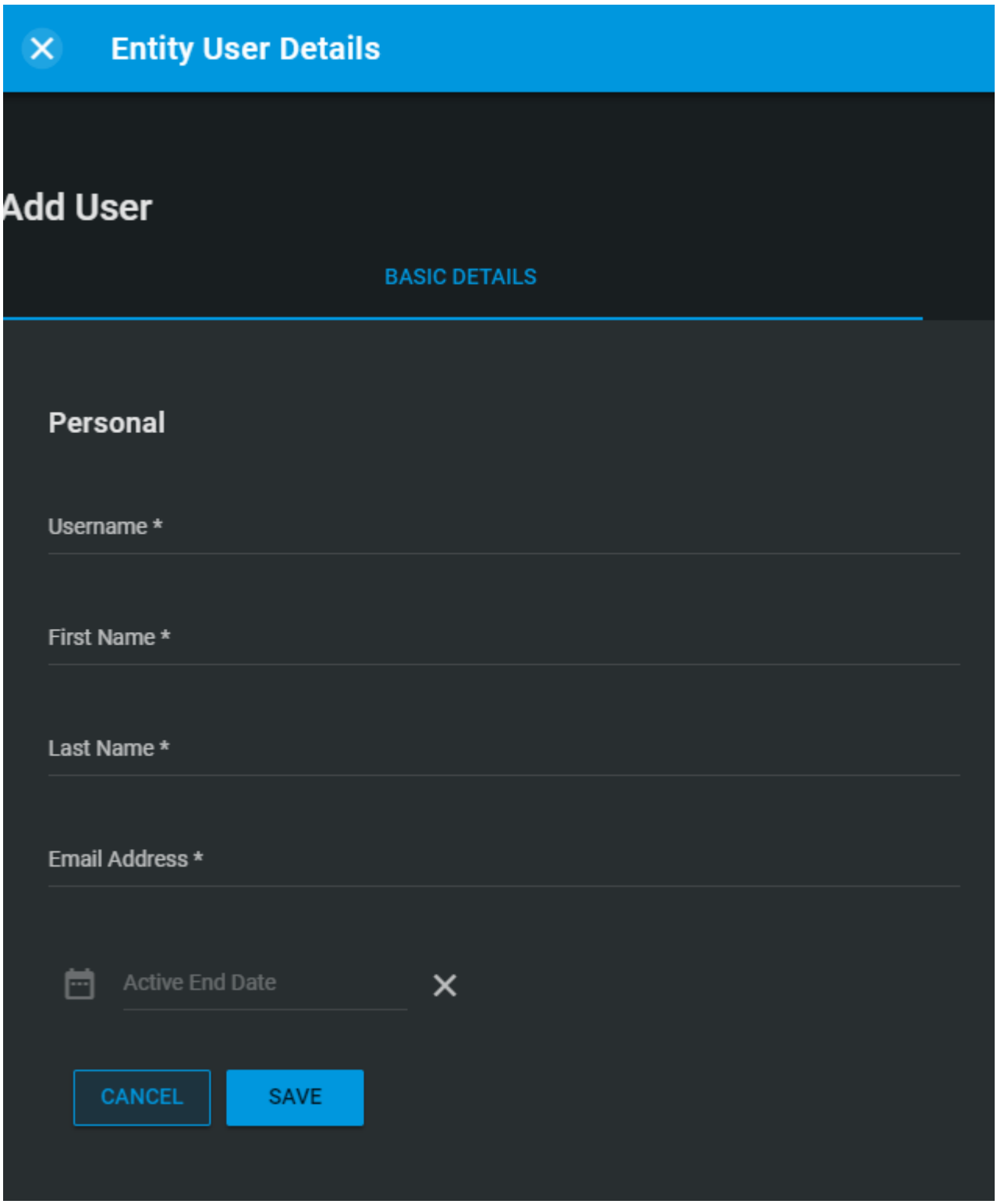
Common use case

```
As a user with permission for [Sec.Entity|Read]
  And with permission for [Sec.EntityUser|Create]
  And with permission for [Sec.EntityAdmin] or ACLs access to entity E
  When I navigate to entity E
  Then I can create a new user U associated to entity E
```

Creation of new users inside the entity will create a new user (similar to a user created from user screen) and also will:

- Include the user in the entity (EntityUser model)
- Populate the memberships of the user with all memberships defined in the entity membership list
- Populate the user types assignments of the user with the user types defined in the entity

User creation applies the same validations that regular user creation; plus maximum number of active users per entity.



Concerns and future work:

- There are no option to add existing users to the entity (further analysis about implications is required).
- There are no support for assigning the same user to multiple entities.

Assigned to any entity

For an EntityAdmin user, it will be possible to create users for any entity; doing this from the entity screen.

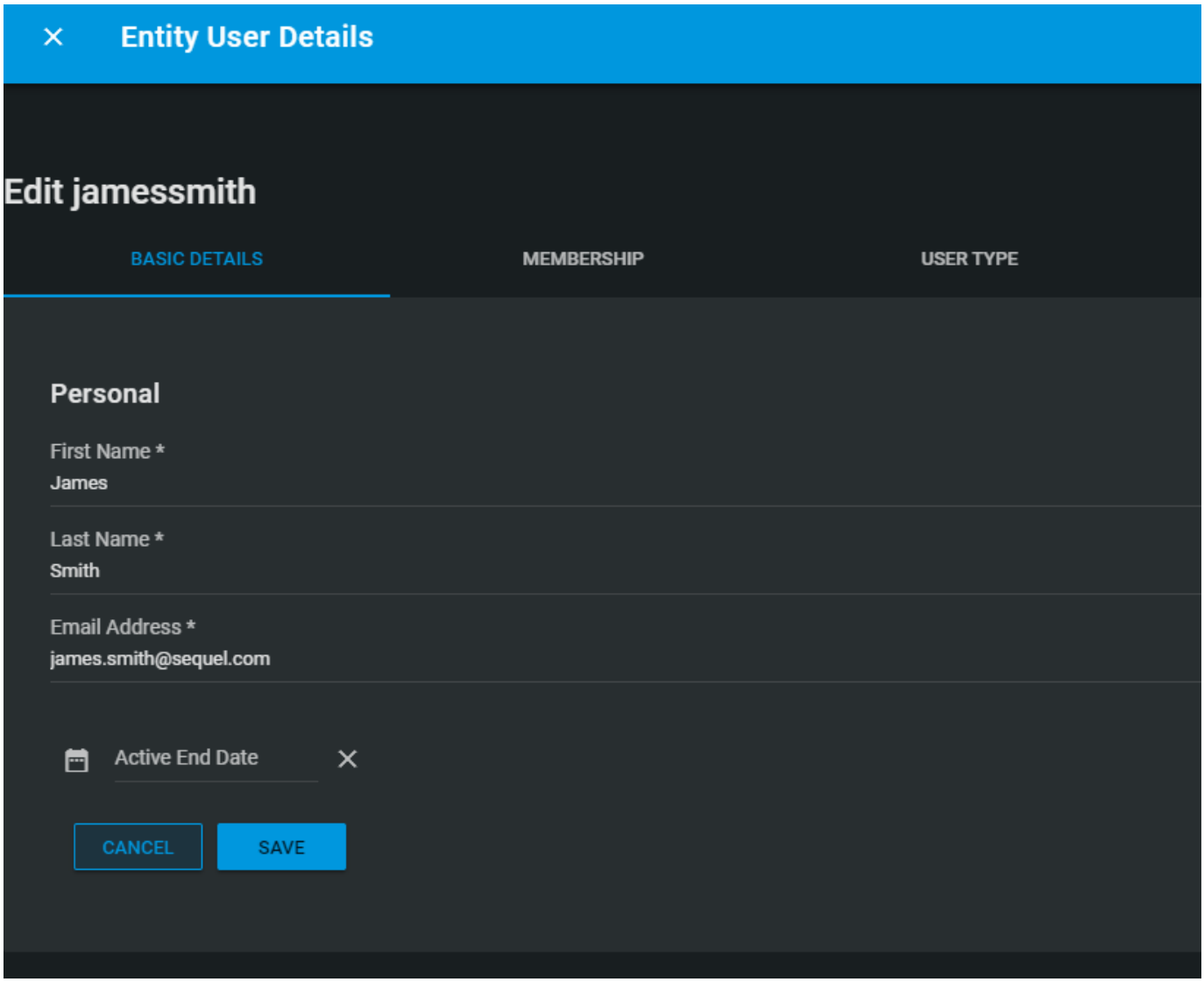
Assigned to my entity

For a DelegatedEntityAdmin, it will be possible to create user just for those entities with ACLs permissions.

Update users

Common use case

```
As a user with permission for [Sec.Entity|Read]
  And with permission for [Sec.EntityUser|Update]
  And with permission for [Sec.EntityAdmin] or ACLs access to entity E
  When I navigate to entity E
  Then I can update a user U associated to entity E
```



Assigned to any entity

For an EntityAdmin user, it will be possible to update users for any entity; doing this from the entity screen.

Assigned to my entity

For a DelegatedEntityAdmin, it will be possible to update user just for those entities with ACLs permissions.

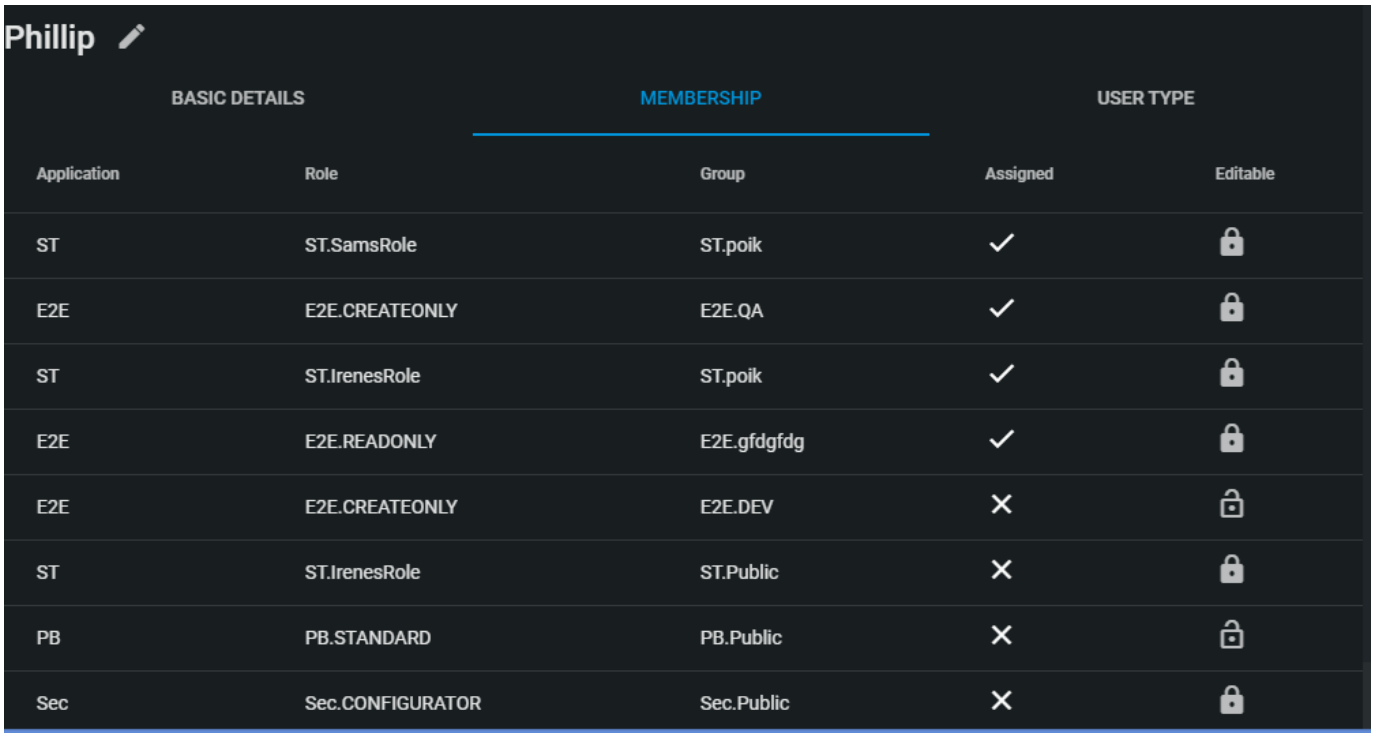
Assign memberships to a user

In this section, an EntityAdmin or DelegatedAdmin user, can configure the memberships of a user. The information presented in this screen contains:

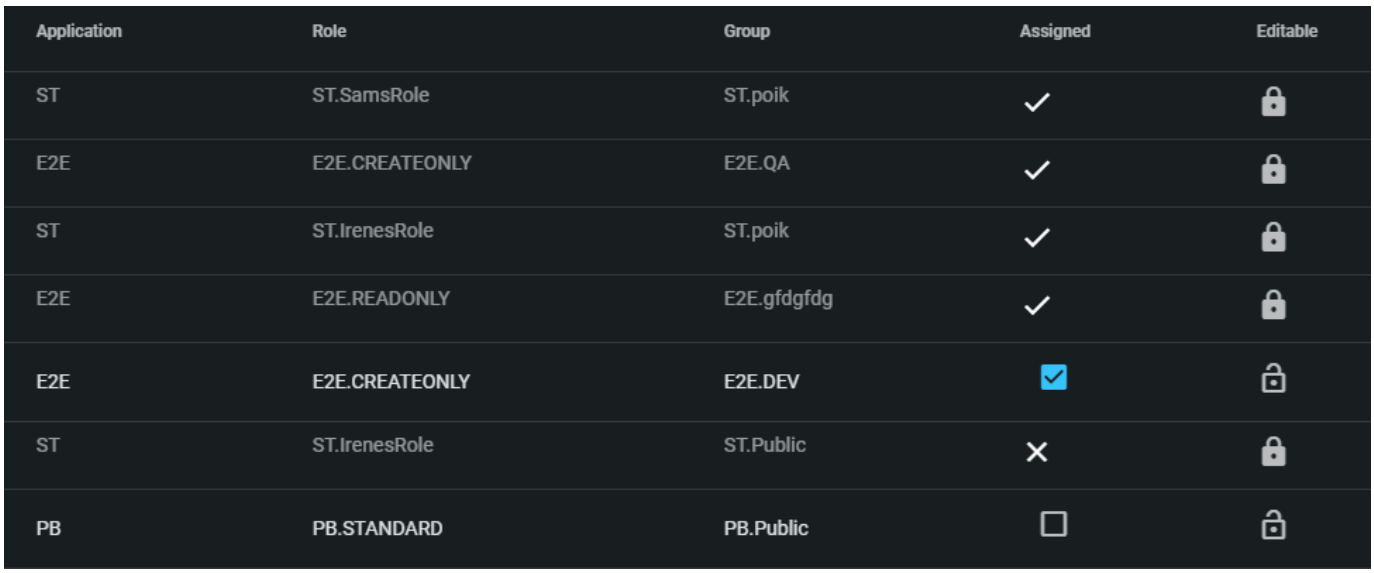
- All available memberships defined in the entity
- Plus any other membership assigned to this user; that it's not defined for the entity. This will allow to add special memberships to some users (ie DelegatedEntityAdmin role).

The memberships displayed show two columns:

- Assigned: if this user has this membership or not.
- Editable: Indicates when this membership can be assigned or removed. A membership is editable if the membership is defined in the entity and is not mandatory.



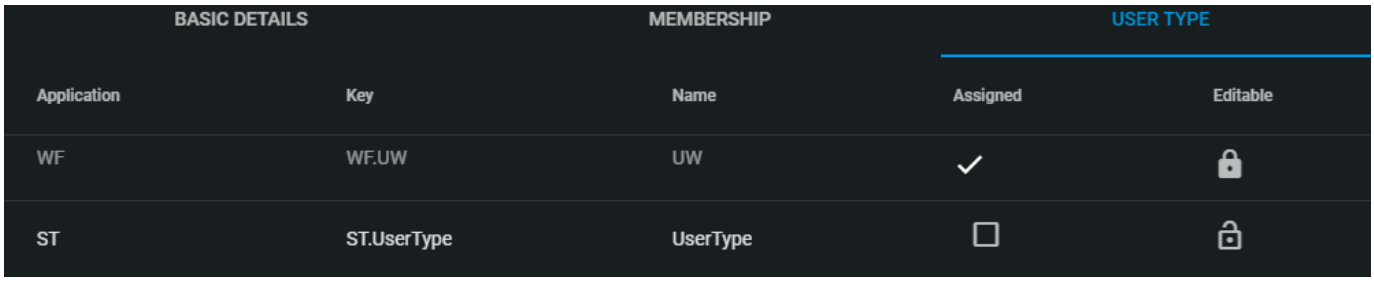
Entering in the edit mode will present below screen:



Just selecting/deselecting the assigned field the membership is assigned/removed from the user.

Assign user types to a user

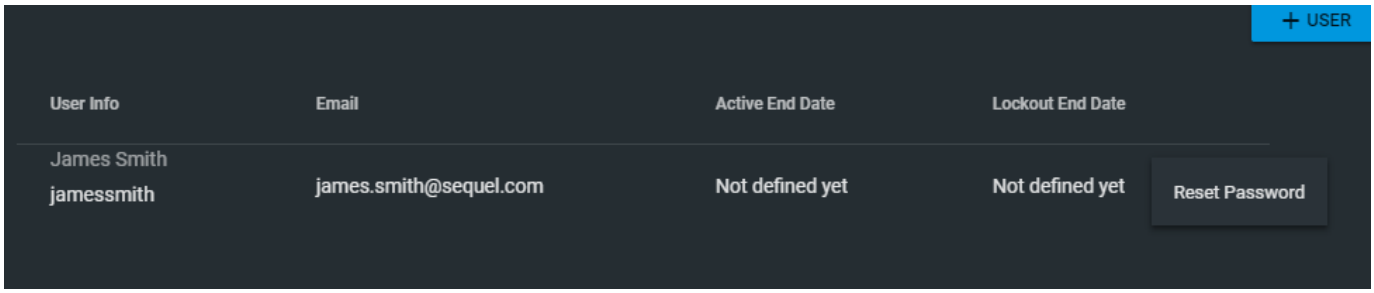
Works in a similar way than membership, with the different there is no concept of mandatory user type.



Request reset password for users

```
As a user with permission for [Sec.Entity|Read]
  And with permission for [Sec.EntityUser|Read]
  And with permission for [Sec.EntityAdmin] or ACLs access to entity E
  When I navigate to entity E
  Then I can request a reset password for a user U associated to entity E
```

We can do this from the context menu in the section of available users:



Bulk insert

Security Service offers the ability of bulk insert user assigned to an entity. This functionality will be exposed in the SecurityAPI, in the DataExchange specs (in swagger); in a new resource: **EntityUserBulkAction**. There will be two endpoints:

- For bulk import users in a json format
- For bulk import users from a XLSX file

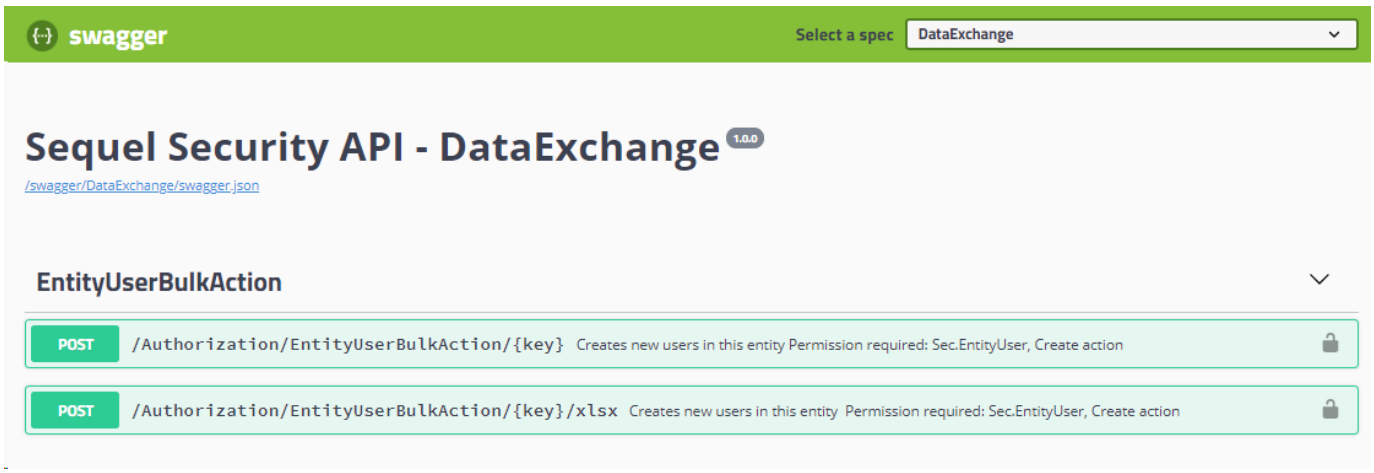
The bulk import will need to deal with potentially large sets of data; so, each item included will be committed independently. It's possible that a request with 10 items will save 6 and it will fail for 4. So, the response will always contain information about which entry failed (due to validations or any other error) and which one worked.

These action are protected by Sec.EntityUser securable and Create action; plus Sec.EntityAdmin and create action. Only EntityAdmin users can bulk insert users; this is not open for delegated admin users or any other roles with lower permissions.

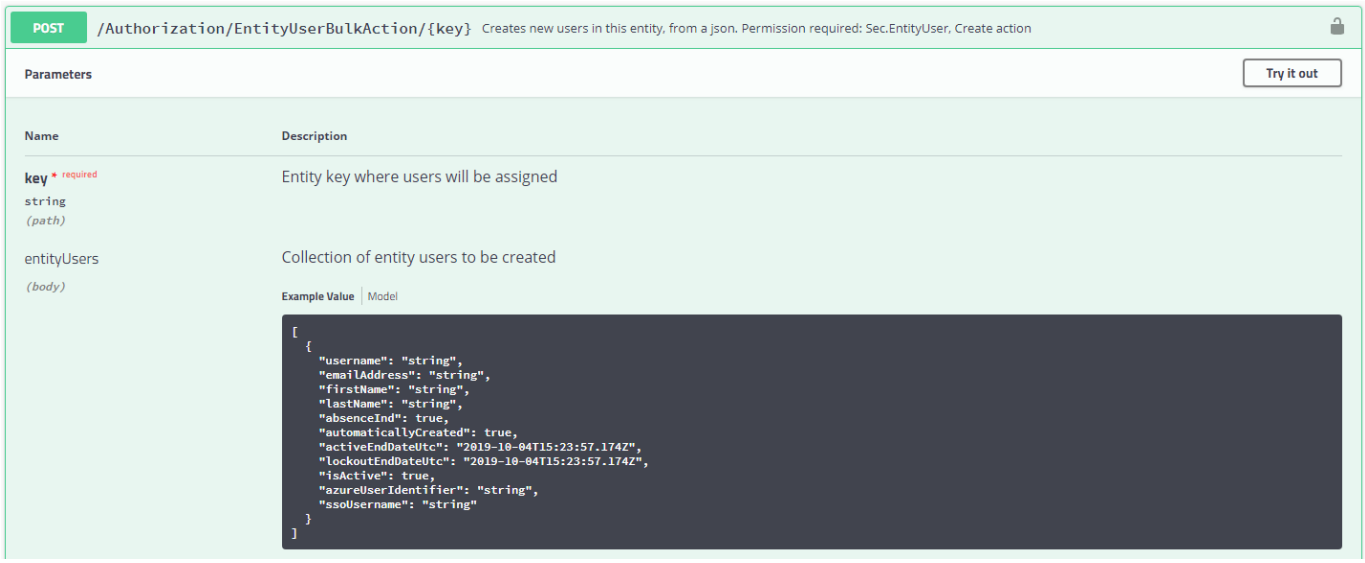
Adding UI for supporting this action is out of the scope because the first final users of this option will be Sequel Support team and they have expressed their desire of automate this process; so UI is not required in this first version.

API definition with OpenAPI - Swagger

All actions defined in this EntityUserBulkAction resource are defined in the context of an entity (parameter key). So, all request will create the users in the entity {key} passed as parameter in the path:



Bulk insert using json

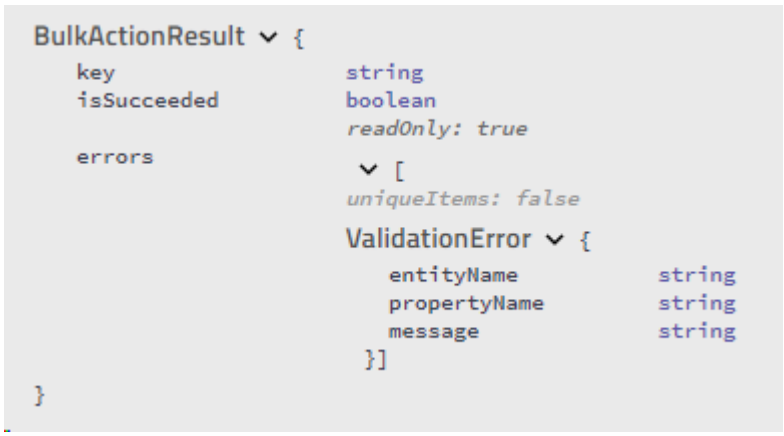


Request:

- key: Entity key. The entity must exists.
- entityUsers: collection of **EntityUser**.

Response:

- Collection of **BulkActionResult**.

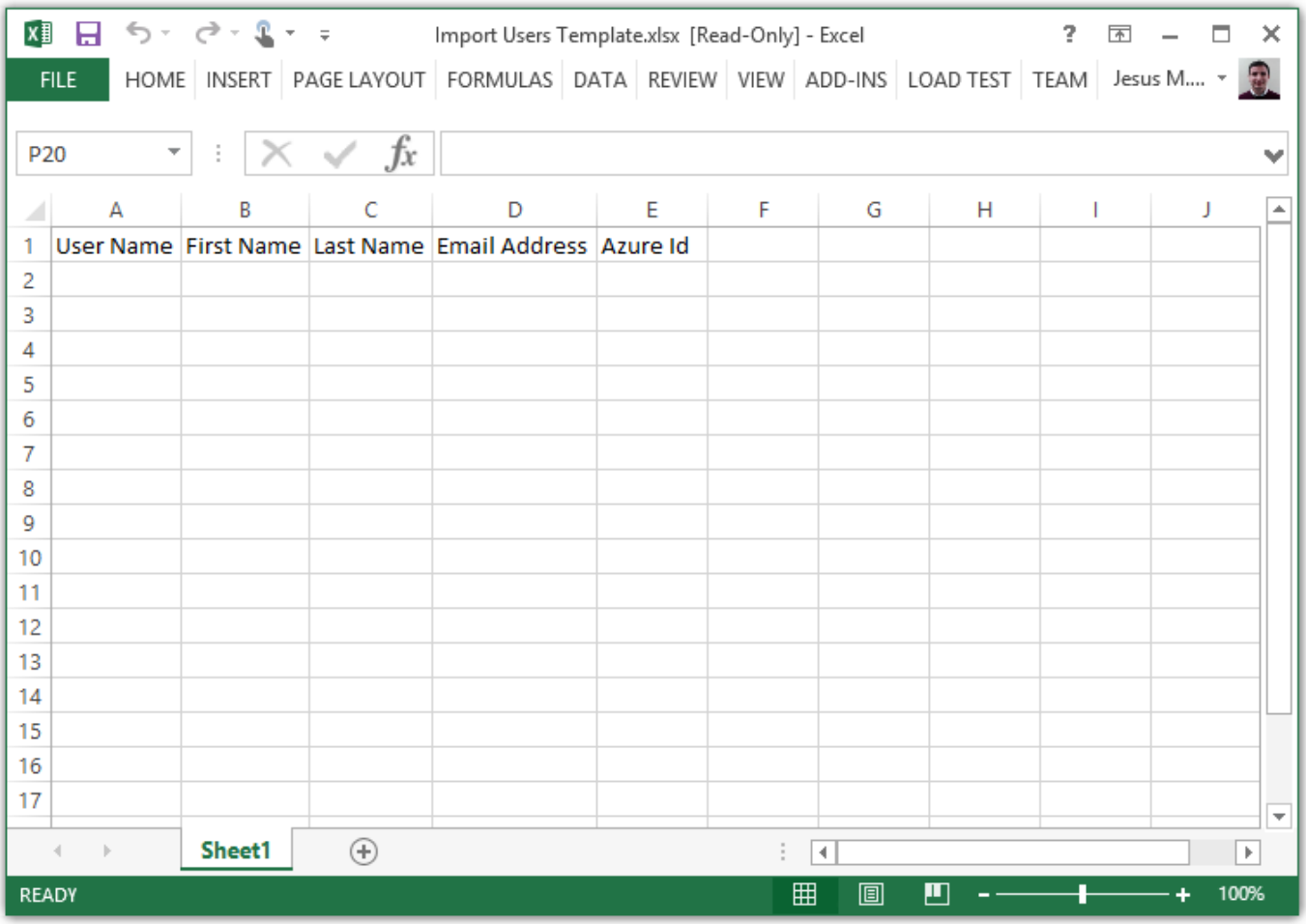


Response contains the same number of items than request; order is the same as request. So, first EntityUser has the result in the first element of response and so on.

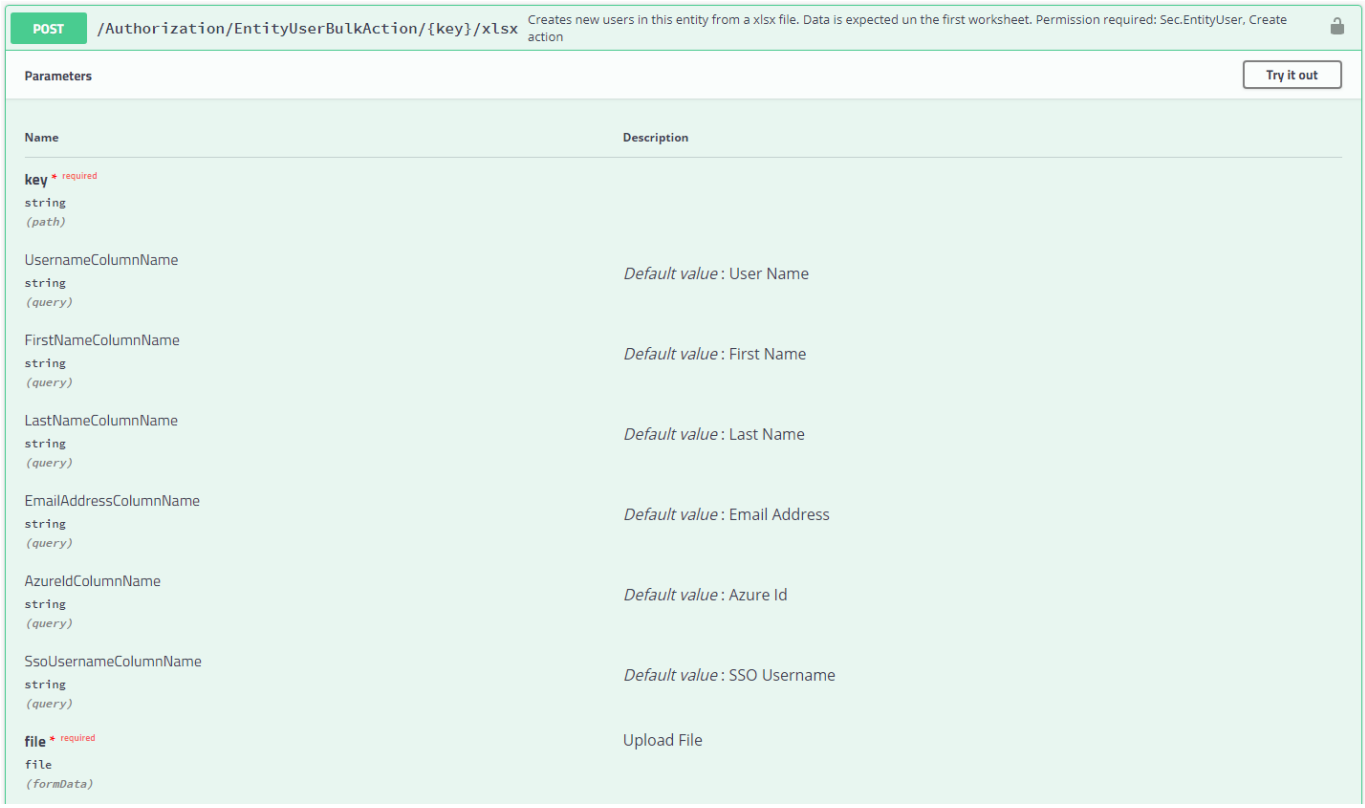
BulkActionResult: key contains the username, isSucceeded informs if the element was saved or not, and errors contains information when element couldn't be saved (due to validations or due to unexpected errors). When an internal error occurs, the validation error should contain a single element with message: 'Unexpected internal error. Please, review logs for further information' -details are logged and cannot be exposed in the API-

Bulk insert using XLSX file

The file expected must be similar to this one:



The request allows to define the mappings (column names where information is stored):



Request:

- key: Entity key. The entity must exists.
- file: xlsx file.
- mapping parameters (*ColumnName): allows to define the name of the column that contains this property. This is not required, and Swagger shows default values that matches with attached document.
- UsernameColumnName: contains the name of the column where username is expected.
- FirstNameColumnName: contains the name of the column where First Name is expected.
- LastNameColumnName: contains the name of the column where Last Name is expected.
- EmailAddressColumnName: contains the name of the column where Email Address is expected.
- AzureIdColumnName: contains the name of the column where Azure Id is expected.
- SsoUsernameColumnName: contains the name of the column where the SsoUsername is expected.

Response:

- Similar to **Bulk insert using json**

How-To

In this section, we will cover some basic configurations that we will expect our clients will do.

CREATE A NEW ENTITY

This action must be done by an Entity Admin user.

For creating a new entity, first of all we need to design the configuration. We will need:

- Determine a unique key for the entity.
- Collect basic details: name, eclipse owner id (if required)
- Determine the main group we will assign to the entity.

Once we have the information; we will follow below steps:

1. Go to Groups and create the group if doesn't exist yet.
2. Go to Entities page
3. Click on create new entity.
    a. Populate basic details: key, name
    b. Save
4. Edit the recently created entity for populating the memberships:
    a. Include mandatory memberships required for all users (traditionally related to public groups). Mark them as mandatory; because if those memberships are removed the user won't be able to access to the application.
    b. Add specific memberships for the user, related to the main group.
    c. Add other memberships related to groups different to the main group.

    This is the most important part of the configuration and must be designed by each product to fulfil the requirements.
5. Edit user types if required.

CREATE A DELEGATED ENTITY ADMIN USER FOR AN EXISTING ENTITY

This action must be done by a user with permissions for editing users.

For creating a Delegated Entity Admin; we need to create a user assigned to this entity and later add it to the Sec.DelegatedEntityAdmin role for Sec.Public group.

Steps are:

1. Go to an Entity page
2. Go to users assigned to this entity
3. Click on add new user
4. Populate all information
5. Save
6. User will be created and configured with all memberships defined
7. Go to user page
8. Edit recently created user.
9. Add membership: Sec.DelegatedEntityAdmin and Sec.Public.

Concerns and future work:

- Allow to an Entity Admin user to automatically assign/revoke roles of DelegatedEntityAdmin to a user in an entity.

5.4.6 Cross domain management

Cross-domain Identity Management

PURPOSE

For a large enterprise using Sequel's applications and an external identity provider (Idp), managing authentication (AuthN) and authorization (AuthZ) directly from the IdP instead of through the Sequel Security Services (Sequel's Security) application would reduce a lot of extra maintenance work. This functionality is based in two features:

- **User** synchronization from the IdP to Sequel's Security.
- User's **membership** synchronization, translating from IdP groups to memberships in Sequel's Security.

This integration will reduce the maintenance cost of user management at Sequel's Security to some initial configurations:

- *Roles*, specifying the granular permission for this role.
- *Groups*.
- *Membership set*, linked to an IdP group. In other words, define the mapping functions for translating from an IdP group to a set of memberships (pair of roles -permissions- and groups). Considering different level of complexity as: an IdP group could represent a Sequel's membership, role or group concepts.

Once the above manual configuration tasks are done, the daily maintenance of users will not require further manual actions at Sequel's Security Services for:

- Creation of new users
- Update changes on basic user information: first and last name and email
- Changes on user groups (member of).

Manual configuration will be required if a new AD group needs to be used in Sequel's application(s) or if memberships associated to an AD group need to be changed.

INTEGRATIONS

Current version of Sequel SEcurity Services can integrate with other IdP for syncing users and groups using:

- Azure AD with Microsoft's Graph API
- Windows AD with LDAP

USER INFORMATION STORE AND CACHE

Critically, when used as a proxy/middleware Sequel Security Service does not store user credentials (eg username, password), nor does it have knowledge of these sensitive credentials – the responsibility of identifying a user lies with the identity provider (eg Microsoft Azure). At no point are these sensitive credentials communicated, processed or stored by the Sequel Security Service. The Sequel Security Service is responsible for storing non-sensitive user information, within its User Information Store and Cache, and providing this user information to other products within the Sequel suite.

The Sequel Security Service is responsible for managing user's and their permissions within Sequel products, and providing (limited) information on users to Sequel products, principally a user's name, email address and their permissions/roles for each Sequel Product, which when connected to and IdP are defined as groups usually on the IdP (eg. At Azure Active Directory is defined by membership of Azure Active Directory groups). This information is stored within the Sequel Security Service (and Sequel products) and updated from data sourced from client's IdP. This store acts as a cache and avoids repeated, relatively slow requests to the identity provider (AAD).

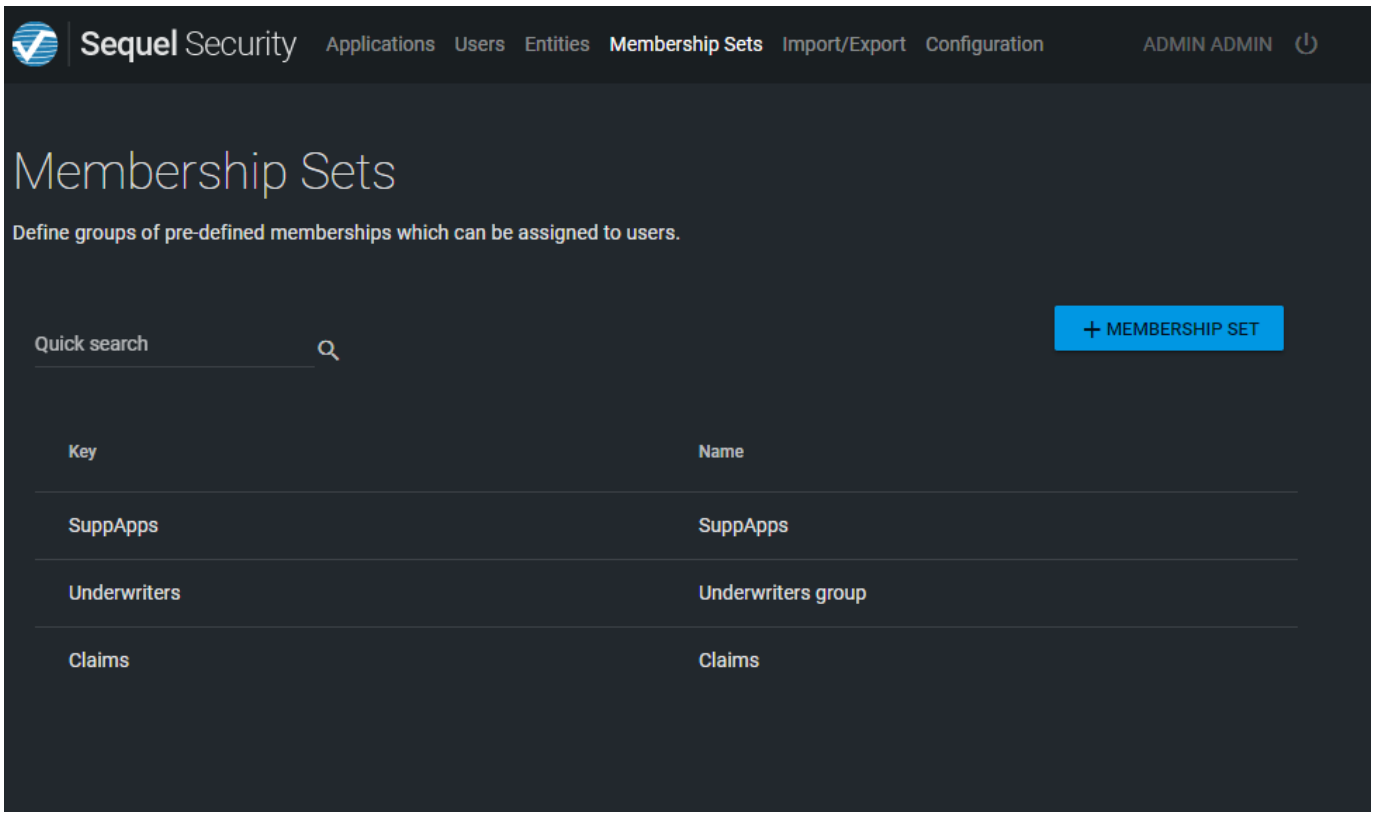
The Sequel Security Service both stores this user information within its own data store/cache and distributes the information to downstream Sequel product instances to update their own stores, to improve product instance performance and isolate product instances' workloads.

Membership Sets

MembershipSet is a concept introduced to Sequel Security to model templates of memberships that can be applied automatically to users as part of integration on cross-domain identity scenarios. A *membershipset* is composed of pairs of a role and a group. This set contains an unique identifier and a friendly name; and can be linked to a group on any identity provider (aka data source) supported in order to translate the role-level access from the IdP to Sequel Security service.

MEMBERSHIP MANAGEMENT

MembershipSet can be configured from the Security API and from Administration UI at the Membership Set section.

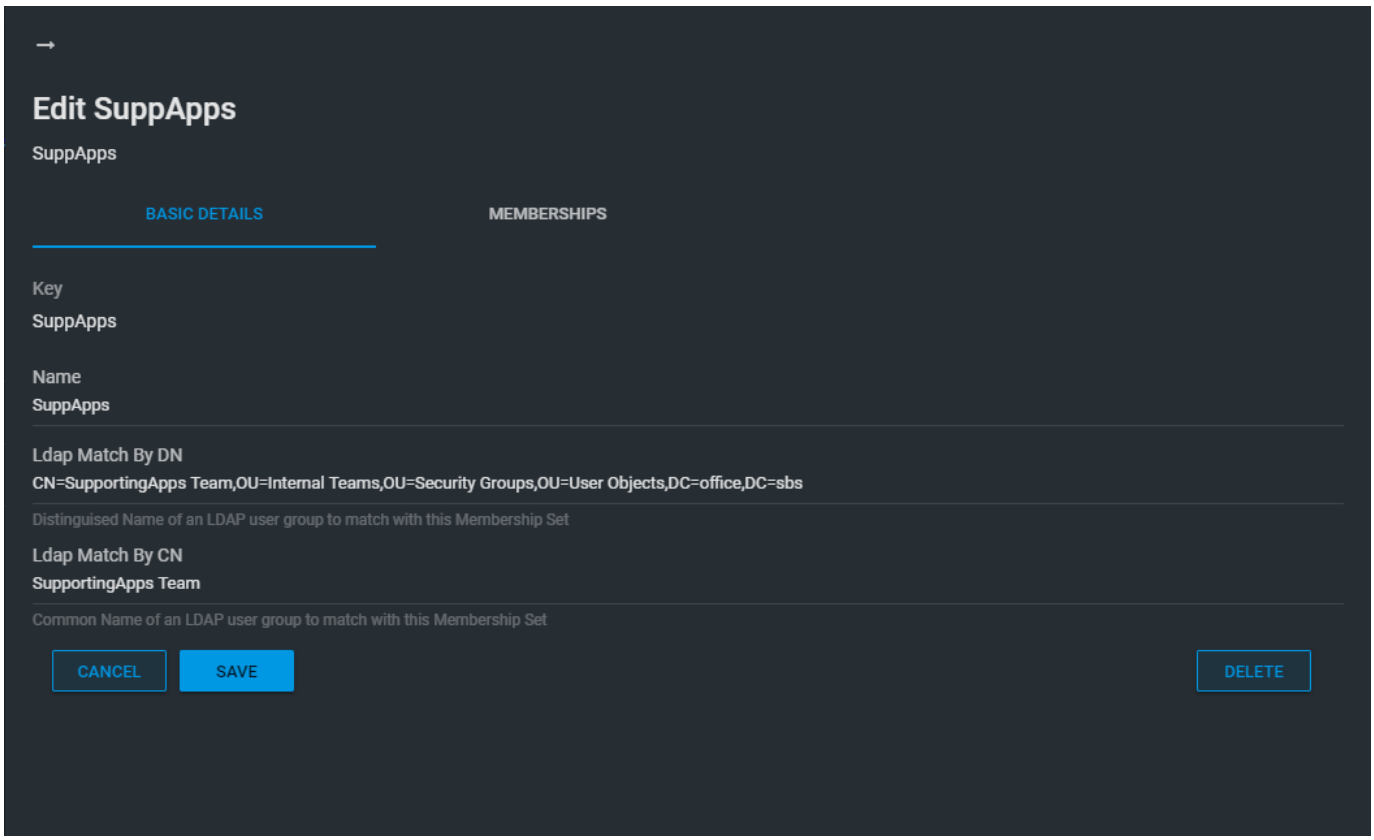


Membershipset configuration

The *membershipset* is defined by an identifier or key that must be unique in the system, a name and optionally matching conditions for linking this membership with an IdP group, it can be done defining:

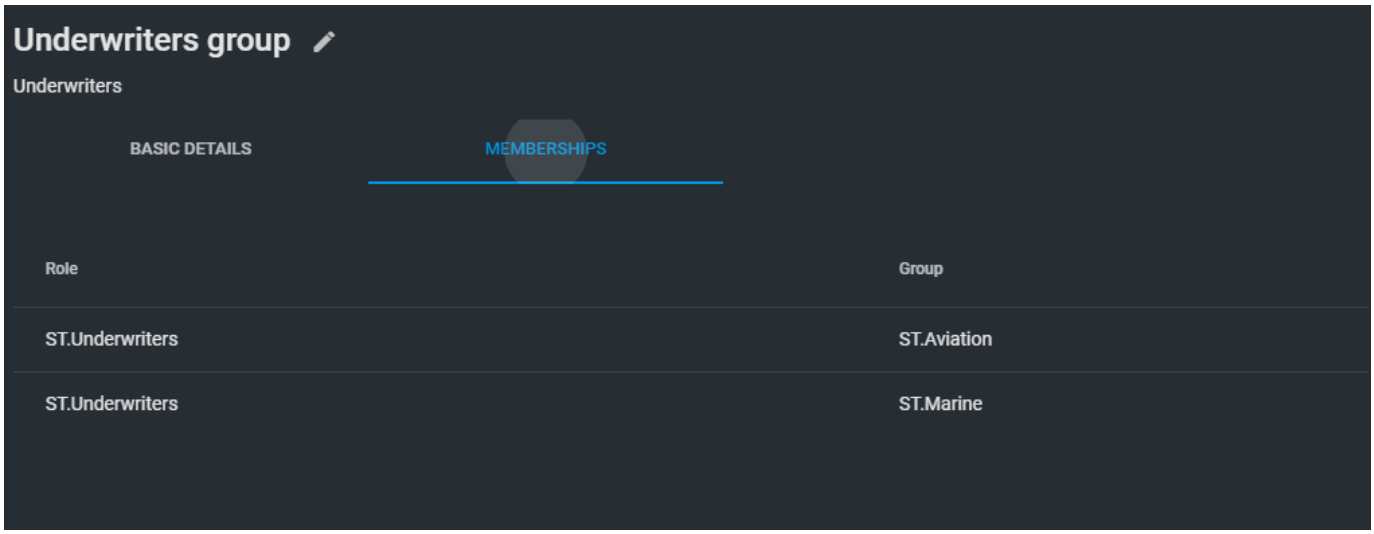
- Azure AD: `id` and `displayname`.
- LDAP: `DN` (Distinguished Name) or `CN` (common name).

The logic about matching is described below in the next sections.

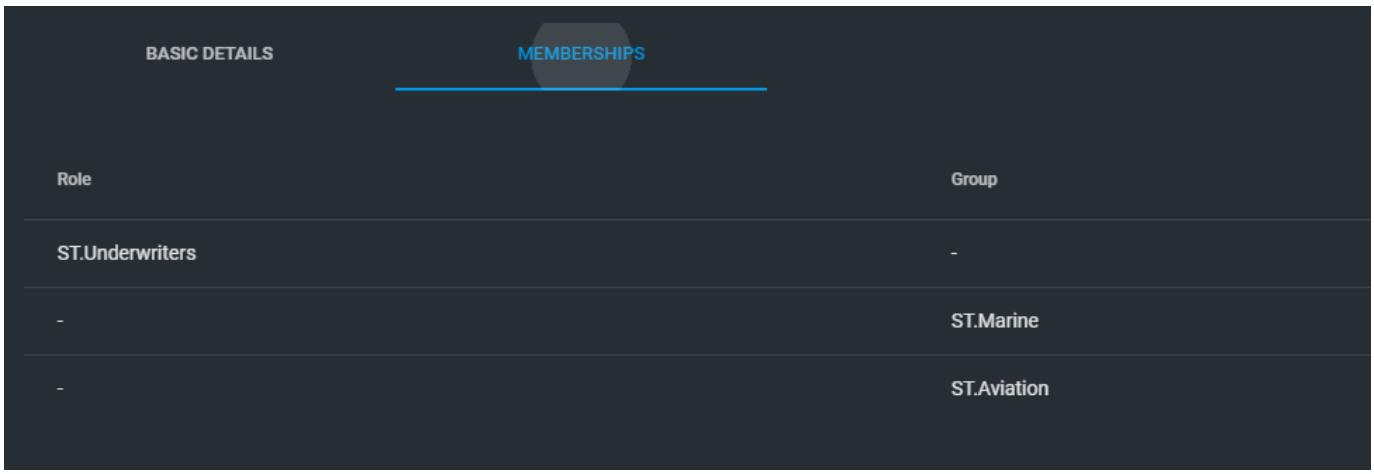


For each membership set, we can define the associated memberships. This information will drive the transformation logic applied to assign memberships to user; this is covered in *"Transform: Membership > Transformation Logic"*.

When a row defines both values, this will generate the same entry in the user.



When a row just define group or roles, means that all combinations of those values, in the same application, (in all entries of affected membership sets) will be generated (cross join).



Both samples in screenshots will produce the same results (if no other membershipset are involved).

> **Note**
>
> When a membership set configuration changes, the sync process is not automatically triggered. The new configuration will be applied in the next synchronization; scheduled or manually triggered.

TRANSLATION LOGIC

The mapping for memberships is more complex, as we have to transform a single value (IdP group) in a pair of *role* and *group*. For being able to map a IdP group to a Membership set, the matching fields should be configured depending on the type of integration used:

Matching fields

Azure AD

Integration type	Sequel's Field	Associated property on the IdP
Azure AD	`[Membershipset].[SyncMatchById]`	Id
Azure AD	`[Membershipset].[SyncMatchByDisplayName]`	DisplayName

- if both are empty this membership set will never be used for syncing,
- if `Id` is populated, the matching will be done using this field.
- if `DisplayName` is populated, the matching will be done using this field.
- if both are populated, both are used. So if matching with `Id` fails, the `DisplayName` is attempted.

Multiple membership set can be configured for matching with the same `Id` or `DisplayName`.

LDAP

Integration type	Sequel's Field	Associated property on the IdP
LDAP	`[Membershipset].[LdapMatchByDN]`	DN
LDAP	`[Membershipset].[LdapMatchByCN]`	CN

- if both are empty this membership set will never be used for syncing,
- if DN is populated, the matching will be done using this field.
- if CN is populated, the matching will be done using this field.
- if both are populated, both are used. So if matching with DN fails, the CN is attempted.

Those properties refers to a AD Group expressing DN or CN (keep in mind DN is more accurate -referring to a single object- and CN not).

- LdapMatchByDN: We will expect to follow a valid DN syntax. Sample: `CN=Claims,DC=sequel,DC=com`
- LdapMatchByCN: We will expect just the CN value. Sample: `Claims`

Multiple membership set can be configured for matching with the same DN or CN

Define translations

On most IdP a group can be used for different purposes; related to our problem, a group can model:

Access control and permissions defined in the same groups

A single group at the IdP defines ACLs (groups) and permissions (groups). Those IdP's groups could look like: *Underwriters_Marine, Underwriters_Aviation, Claims_Marine, Claims_Aviation*; and they could be translated to Sequel's roles/groups model as:

IdP group	Sequel Role	Sequel Group
Underwriters_Marine	VSBS.Underwriters	VSBS.Marine
Underwriters_Aviation	VSBS.Underwriters	VSBS.Aviation
Claims_Marine	VSBS.Claims	VSBS.Marine

Defining this scenario, is straight forward using the Membeship sets; as we just need to define the role+group pairs. When this membership set matches with an IdP group; the memberships fully defined are directly applied to the user. A

> **Note**
>
> In all samples, groups and roles defined at Sequel Security Service are prefixed with VSBS for clarity; this is not expected to be defined in this way on a real configuration.

Access control and permissions defined in different groups

A group at the IdP defines the ACLs (groups) or the permissions (groups); but not both at the same time.

- *Access control (group)*: the IdP group defines the ACLs, we can see as the team or business unit. The role/permissions about what you can do with this information is not defined here.

IdP group	Sequel Role	Sequel Group
Marine	??	VSBS.Marine
Aviation	??	VSBS.Aviation

- *Permissions*: the IdP group defines the role of the users; the actions they are allowed to perform. However, the access to the information is not defined at this group.

IdP group	Sequel Role	Sequel Group
Underwriters	VSBS.Underwriters	??
Claims	VSBS.Claims	??

With these IdP groups where role or group cannot be inferred directly; the translation mechanism provided by *membership set* allows to use the operator `*`. This operator defines a cross join with all other roles/groups defined using this operator. Let's review below definition:

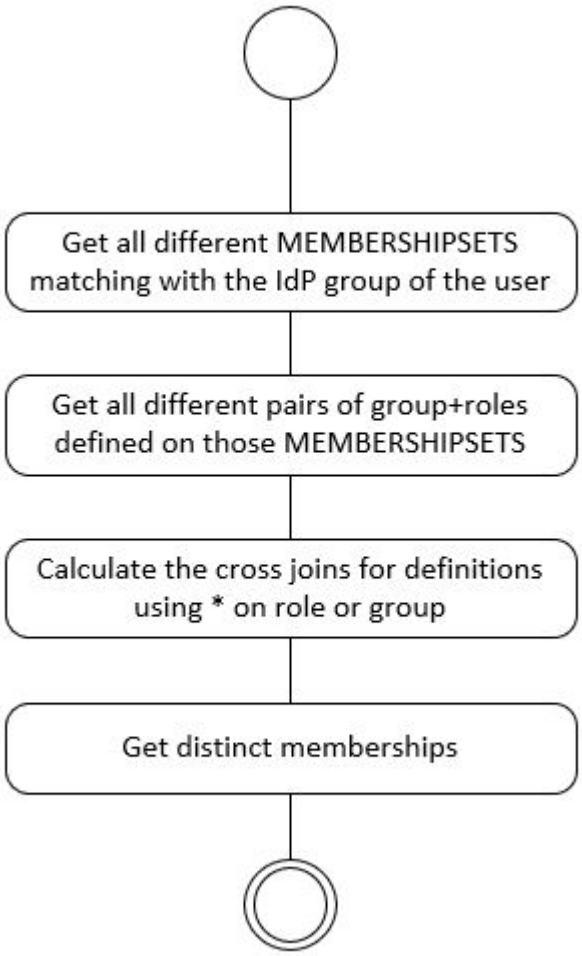
IdP group	Sequel Role	Sequel Group
Claims	VSBS.Claims	*
Underwriters	VSBS.Underwriters	*
Aviation	*	VSBS.Aviation
Marine	*	VSBS.Marine

This could generate for an user below memberships (pairs of role+group):

IdP group combinatios	Sequel Role	Sequel Group
Claims + Aviation	VSBS.Claims	VSBS.Aviation
Claims + Marine	VSBS.Claims	VSBS.Marine
Underwriters + Aviation	VSBS.Underwriters	VSBS.Aviation
Underwriters + Marine	VSBS.Underwriters	VSBS.Marine

Summary

The logic for translating IdP groups to memberships sets can be described then as:



SAMPLE

Let's assume below mapping configuration for translating groups to Sequel's memberships:

MembershipSet	IdP group for matching	Sequel MembershipSet[Role, Group]
#1	Underwriters_Marine	[VSBS.Underwriters, VSBS.Marine]
#2	Underwriters_Aviation	[VSBS.Underwriters, VSBS.Aviation]
#3	Claims_Marine	[VSBS.Claims, VSBS.Aviation]
#4	Underwriters	[VSBS.Underwriters, *]
#5	Claims	[VSBS.Claims, *]
#6	Marine	[*, VSBS.Marine]
#7	Aviation	[*, VSBS.Aviation]

For simplicity, we are defining just a membership on each membership set but it is possible to define many.

In this scenario, we will have below mappings for some users:

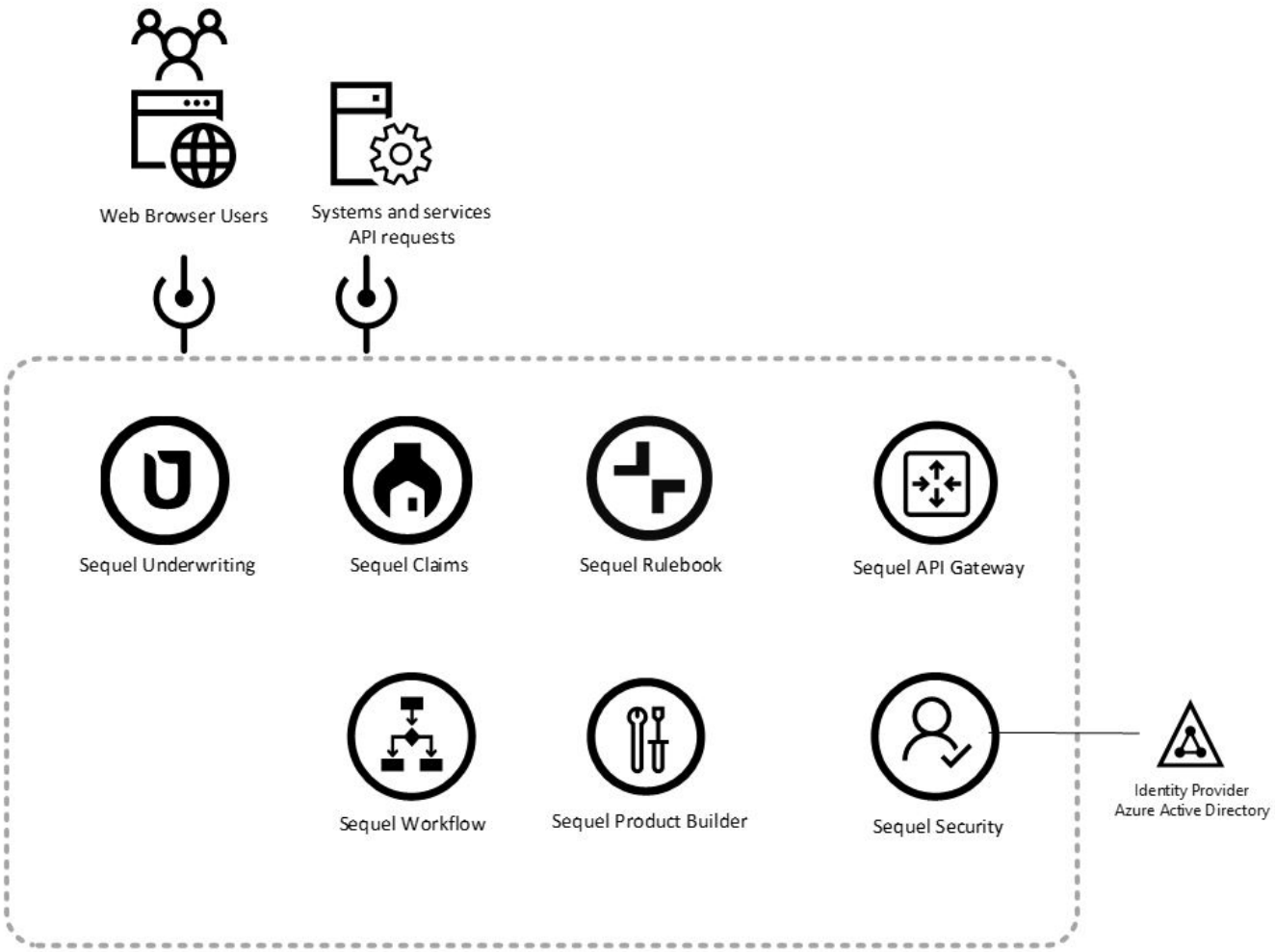
User	groups	Sequel Memberships
usr1	Underwriters_Marine	[VSBS.Underwriters, VSBS.Marine]
usr2	Underwriters_Marine, Underwriters_Aviation	[VSBS.Underwriters, VSBS.Marine], [VSBS.Underwriters, VSBS.Aviation]
usr3	Underwriters, Marine	[VSBS.Underwriters, VSBS.Marine]
usr4	Underwriters, Marine, Aviation	[VSBS.Underwriters, VSBS.Marine], [VSBS.Underwriters, VSBS.Aviation]
usr5	Underwriters	(nothing)

Azure AD with MsGraph

> **Note**
>
> This article covers the cross-domain identity management with Azure AD using Microsoft's Graph API (aka Azure AD Sync of users and groups)

INTRODUCTION

Whilst the Sequel Security Service can perform Authentication & Authorisation capabilities itself, it is typically used as a proxy to identity providers such as Microsoft who have responsibility for identifying a user and their permissions/roles/rights. This approach enables Sequel products to leverage the benefits and vendor implementations of multi-factor authentication (MFA), single sign-on (SSO), identity protection & monitoring etc., providing customers with a secure, trusted, tried and tested mechanism to authenticate and authorise users.

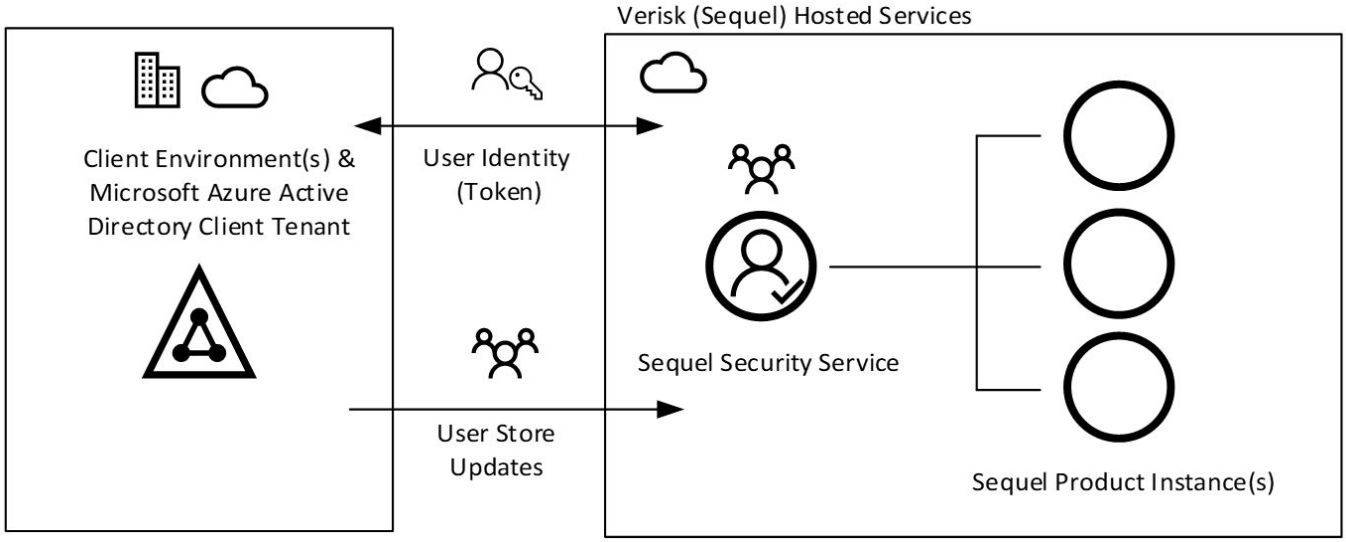


Critically, when used as a proxy/middleware Sequel Security Service does not store user credentials (eg username, password), nor does it have knowledge of these sensitive credentials – the responsibility of identifying a user lies with the identity provider (eg Microsoft). At no point are these sensitive credentials communicated, processed or stored by the Sequel Security Service. The Sequel Security Service is responsible for storing non-sensitive user information, within its User Information Store and Cache, and providing this user information to other products within the Sequel suite.

USER INFORMATION STORE AND CACHE

The Sequel Security Service is responsible for managing user's and their permissions within Sequel products, and providing (limited) information on users to Sequel products, principally a user's name, email address and their permissions/roles for each Sequel Product, which when connected to Azure Active Directory, is defined by membership of Azure Active Directory (AAD) groups. This information is stored within the Sequel Security

Service (and Sequel products) and updated from data sourced from client's Azure Active Directory. This store acts as a cache and avoids repeated, relatively slow requests to the identity provider (AAD). This information (sourced from the identity provider) is written to the Sequel Security Service data store when a user is authenticated by the identity provider and on a regular scheduled basis (e.g. every 30 minutes).



The Sequel Security Service both stores this user information within its own data store/cache and distributes the information to downstream Sequel product instances to update their own stores, to improve product instance performance and isolate product instances' workloads.

User Information Logical Schema

The Sequel Security Service (and other Sequel product) stores limited non-sensitive user information, sources from Azure Active Directory. User Credentials are not processed by or stored within the Sequel Security Service.

The scope of the dataset held within User Information Store and updated by the Sequel Security Service is controlled by membership of a configurable AAD group. The Sequel Security Service determines the users to retrieve information from AAD by identifying the members (users) within this AAD group .

The user details retrieved from AAD and stored within the User Information Store include:

- From users: `id`, `userPrincipalName`, `givenName`, `surname`, `mail` and `accountEnabled`
- From groups: `id` and `displayName`

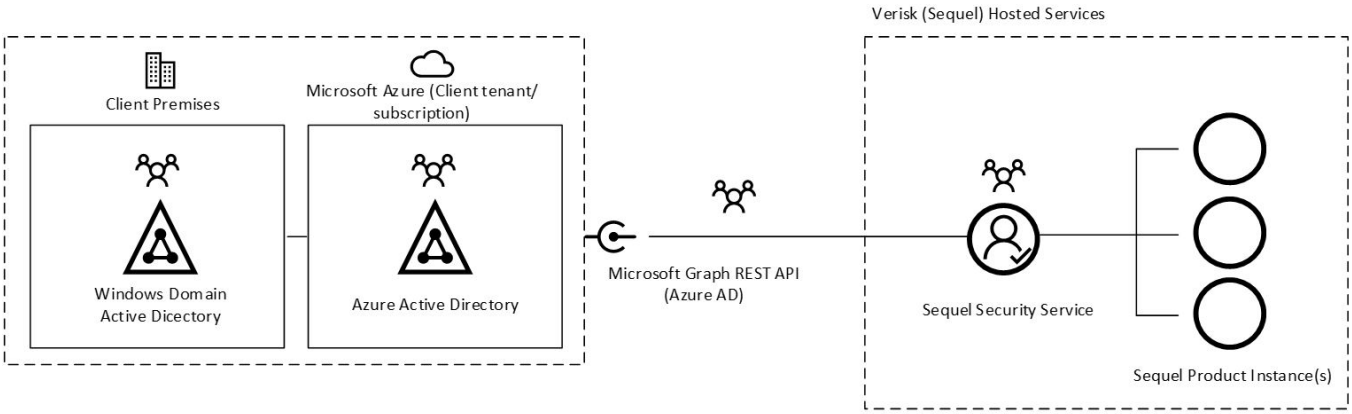
CLIENT DEPENDENCIES AND REQUIREMENTS

Clients are responsible for providing a functioning implementation of a Microsoft Azure Active Directory Tenant, populated with user identities and groups as per Microsoft's instructions. Since integration between Azure Active Directory and the Sequel Security Service requires configuration and information to be shared between the two services, the technical teams of the client and Sequel work collaboratively to enable integration.

There are two integration mechanisms between the Sequel Security Service and Microsoft Azure Active Directory:

- *User Authentication (out of scope)* – permits users of Sequel products (via the Sequel Security Service) to authenticate via Microsoft Azure Active Directory, using industry standard OpenIDConnect protocols. Integration permits the Sequel Security Service instance to redirect users to Microsoft to authenticate, and to be redirected back to the Sequel Security Service instance once successfully authenticated. This integration is not used by the User Information Cache and is not within scope of this document.
- *User Information Store and Cache Update* (in scope) – provides the Sequel Security Service instance permissions to connect to the Microsoft Azure Active Directory API and read limited user information (including group membership). This integration is only used by the User Information Store and Cache. The Sequel Security Service integrates with Microsoft Azure Active Directory (AAD) using Microsoft's Graph API, reading minimal user information and user group membership information. Legacy integration methods based on LDAP(S) requiring Windows Domain and VPN connectivity between client and Sequel hosting networks will be deprecated over time.

Clients are responsible for providing a configured and populated Azure Active Directory instance.



There is a registration form that covers how to set-up our Sequel Security applition on Azure, allowing to perform read queries to Azure through Microsoft's Graph API: Azure AD Sync Registration

SYNCHRONISATION PROCESS

The sync service periodically executes a *"sync process"*, this process requests information on groups and users, using Microsoft's Graph API. This information is translated to a Sequel User, representing a user with some key attributes retrieved from AAD, like name or groups that user is member of. The sync service will request the creation/update of this user to the Security API. This request will determine if this is a new user or an existing one, it will calculate the memberships for this user based on the information and it will apply the changes to the user (this calculation is coverere at MembershipSet section). All actions during the process are audited in the Security API, and error logged to core logging repository (central logging repository used by Sequel Apps).

The synchronisation process executes:

- after being started (*startup* of the daemon hosting the *sync process*).
- after a scheduled time (*polling*): scheduled by default to be executed each hour. This value can be customized. The configuration is refreshed from the SecurityAPI during the startup and in each polling/execution.
- a manual request (*manually-forced*): the process can be manually triggered from the Admin UI (also from Security API). For using this functionality is required to configure the message bus settings to the sync service. A manual request refreshes the configuration and restarts the polling.

SECURITY

The Sequel Security Service reads limited information from Azure Active Directory (AAD) using Microsoft Azure Active Directory (Graph) API. It does not write to AAD, nor should it be granted write access to AAD. The required permissions to be assigned on Azure for this app are: `Groups.ReadAll` and `User.Read.All`.

LDAP

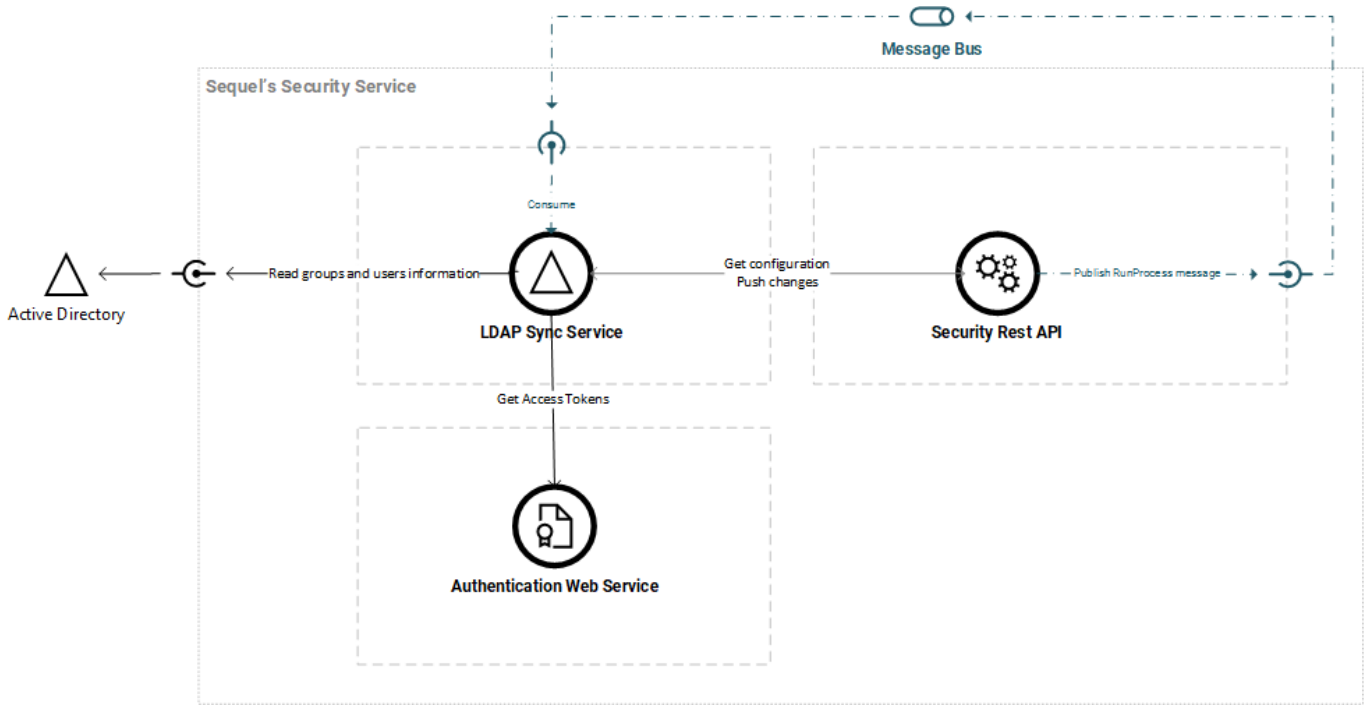
> **Note**
>
> This article covers the cross-domain identity management with Windows LDAP (aka Role-level authorization synchronization with Windows Active Directory using LDAP)

The **sync process** will select the users to be sync'd based on a list of LDAP queries that returns groups from the AD tree. Our suggestion is to create a Windows Group called "Sequel Application Users" and include all users to be sync'd in this group. However, it is possible to define custom LDAP queries and also multiple queries.

While being an expert on (https://ldap.com/) is not required; we highly recommend to be familiar with LDAP and understand properly how the AD tree is defined in the catalogue we want to sync from. Some basic concepts are important as: DNs, CN and basic query filters on groups.

ARCHITECTURE

Below diagram describes how the solution integrates with the Sequel's Security Architecture and the AD server.



The three services (not servers) involved in this solution are:

- A **Windows Active Directory** (AD). Supporting LDAP.
- A **"Sequel Security AD Sync" Windows Service** (sync service).
- A **Sequel's Security API** Service (Security API).

The *sync service* periodically executes a **"sync process"**, this process in requests information of groups and users, based on customized LDAP queries against the *AD*. This information is translated to an intermediate contract `LdapUser` that represents a user with some key attributes retrieved from AD, like name or groups that user is member of. The sync service will request the creation/update of this user to the *Security API*. This request will determine if this is a new user or an existing one, it will calculate the memberships for this user based on the information and it will apply the changes to the user. All actions during the process are audited in the Security API, and error logged to core logging repository (central logging repository used by Sequel Apps).

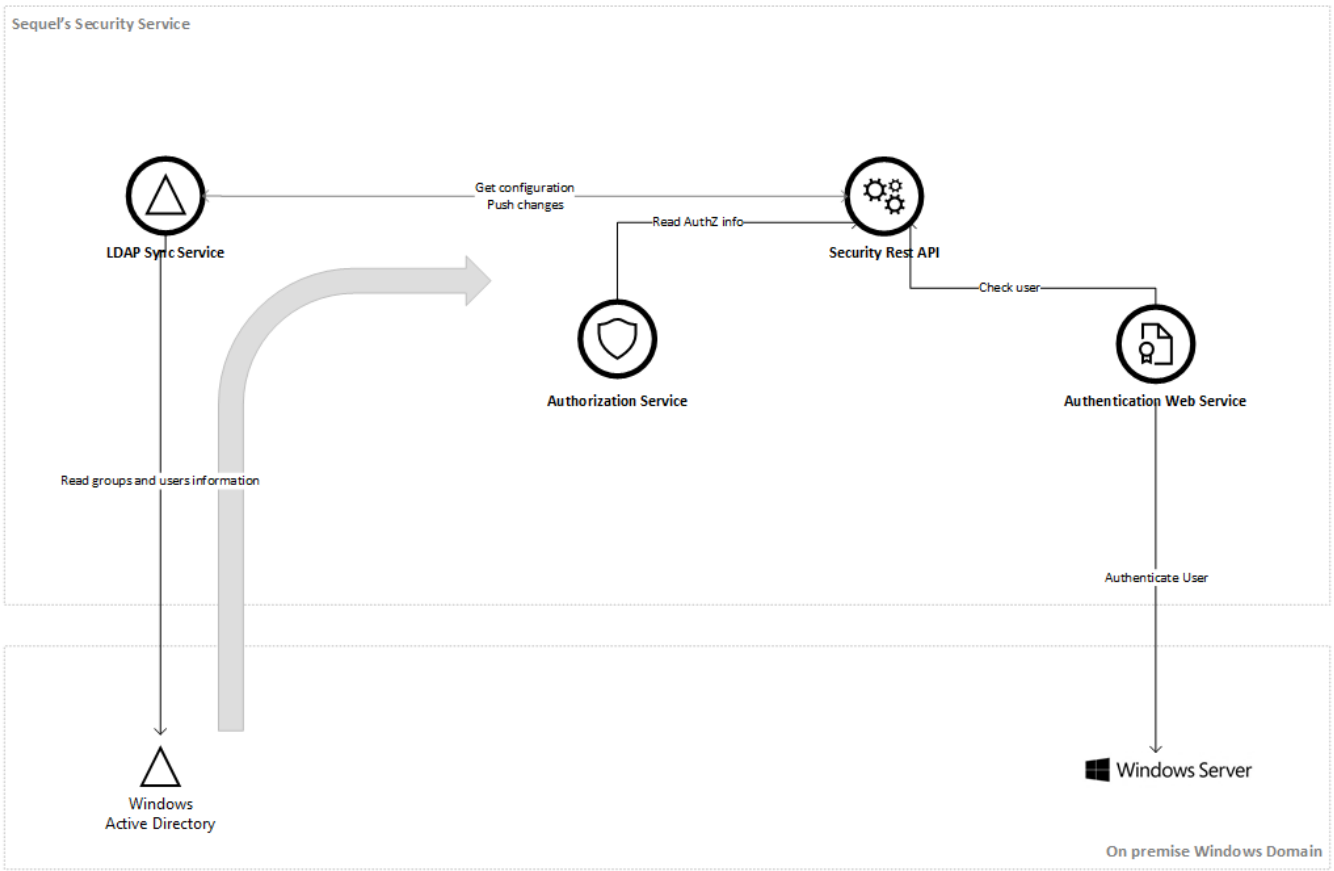
This solution is an ETL (extract, transform, load) where sync service is responsible of extracting data from AD using LDAP; and Security API is responsible of transforming and loading the data.

On premise deployment

The typical deployment of this feature is on a full on-premise installation where Windows AD and Sequel's products are installed on the customer's premises. Below diagram displays this topology, also in this sample authentication is configured using Windows.

Authentication with Windows and Sync with Windows Active Directory

All services deployed on premise

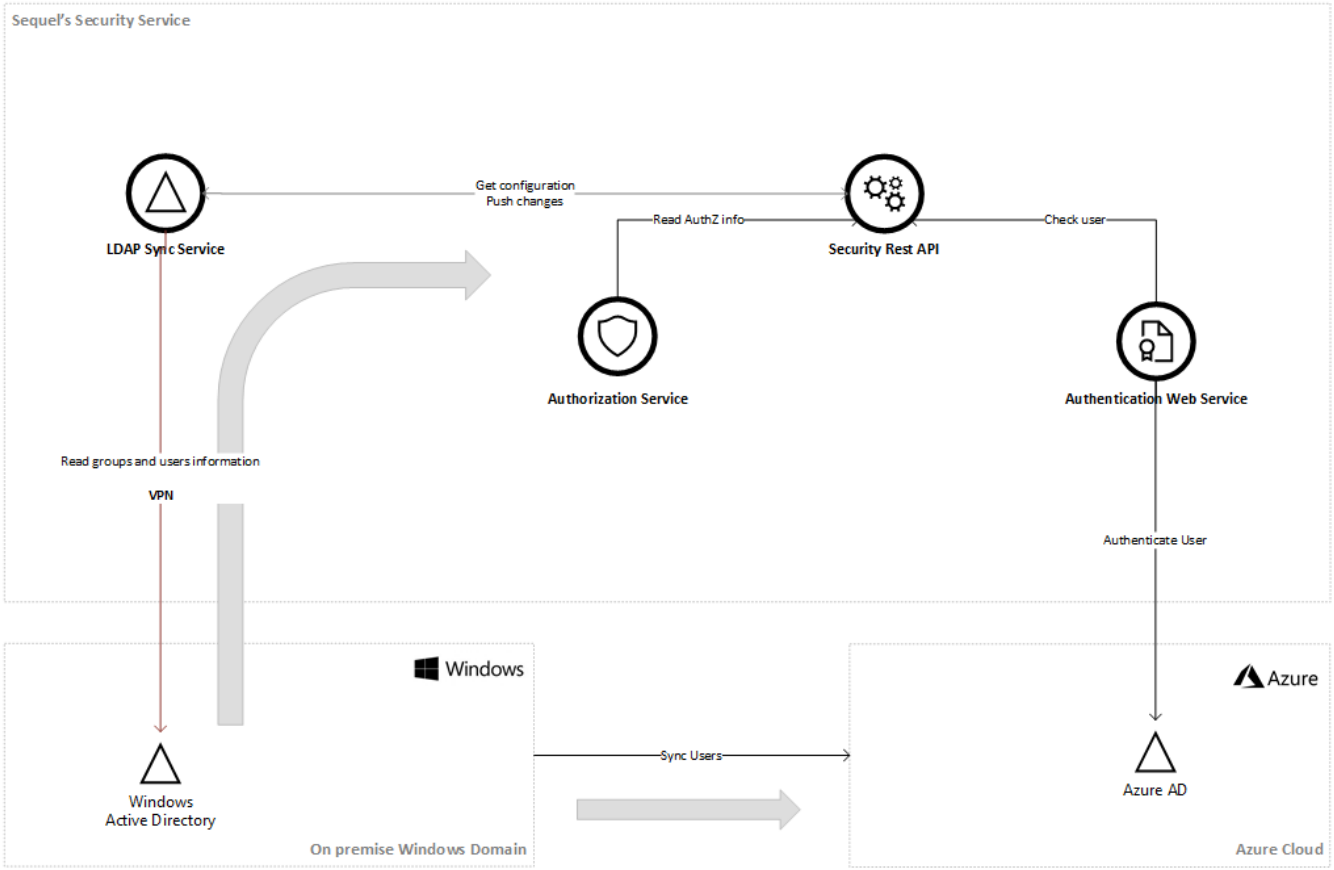


Hybrid deployment

For cloud deployments, when Windows AD is at customer's premises is still possible to get this synchronization working, the conceptual architecture is described below:

Authentication with Azure AD and Sync with Windows Active Directory

Sequel components on AWS + Active Directory on client premises + Azure AD



1. A VPN is required to connect the LDAP Sync Service with Windows AD
2. LDAP Sync Service is running hosted in the cloud and performs synchronization, allowing creating and maintenance of users and groups. This is required for Authorization purposes.
3. Authentication in this sample is done with Azure AD. The customer is responsible of synchronizing their Azure AD tenant with their Windows AD. We can assume this as a normal practice for all customers moving their authentication to Azure AD.

TERMINOLOGY

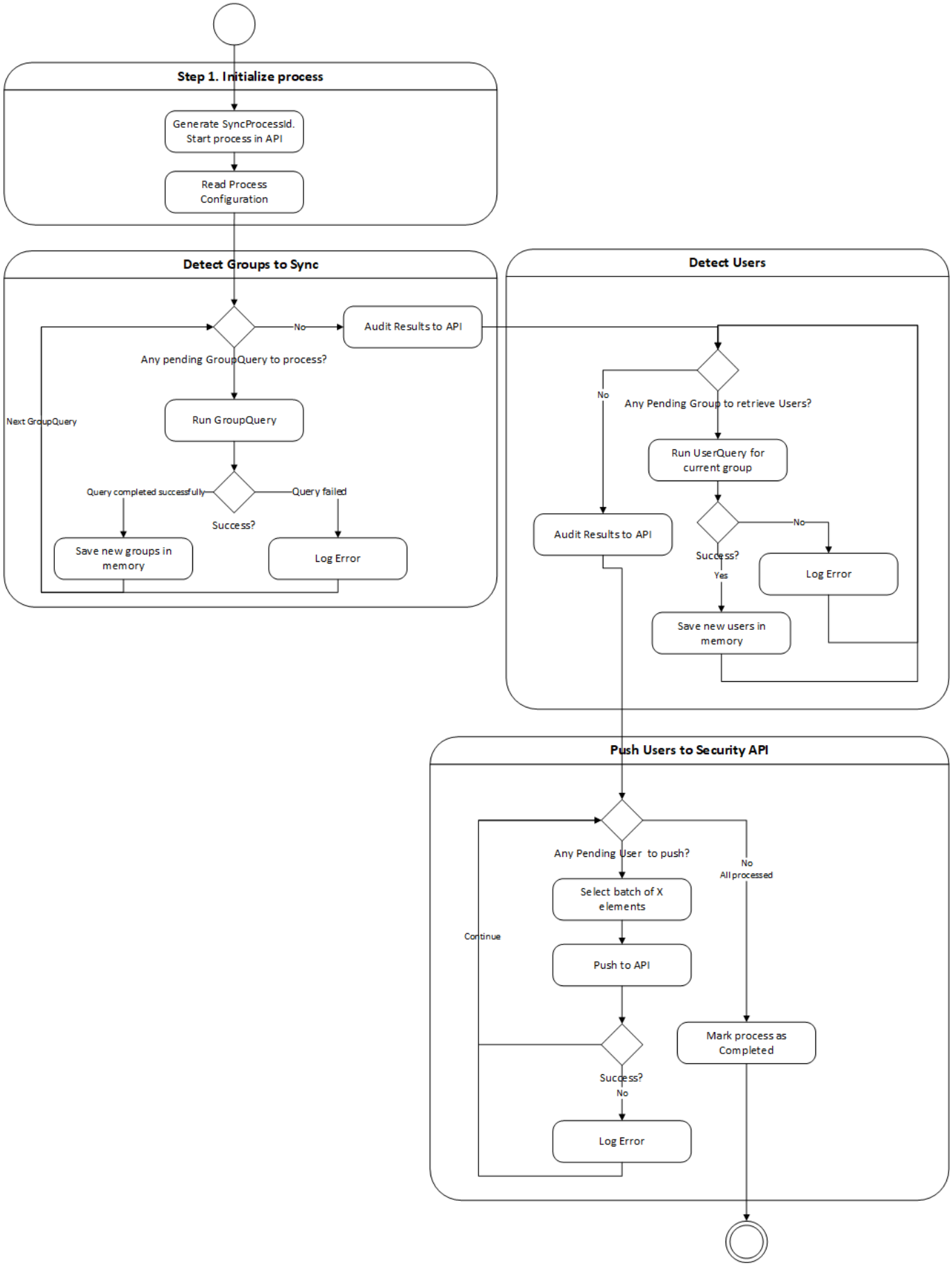
Sync Process

Each execution performed to synchronize the Sequel's Security information from an Active Directory.

Each sync process contains below information:

- Basic details: Unique Id, and when the process was started, completed or cancelled.
- Audit information: traces about how the process is behaving during the different phases of the extract, transform and load.
- Users to sync: list of users detected in the AD and the raw information collected and used to match and sync in the Sequel security service.

The sync process contains several models for configuring, executing and tracing the process; that we will be described in this document. As a first summary, all domain model for the sync process is included in the below class diagram:



Above models are domain model, and they are not representing the database models.

MembershipSet

Understanding what is and how works MembershipSet is recommended.

CONFIGURATION

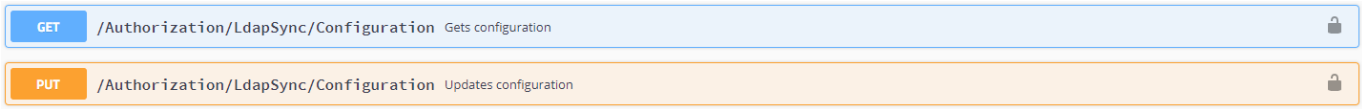
Membership management

MembershipSet can be configured from the Security API and from Administration UI at the Membership Set section. More info at MembershipSet management section.

Sync Configuration

LdapSyncConfiguration resource in API

The sync configuration is managed by Security API service, and is available at `LdapSync\Configuration` resource (protected by `LdapSyncConfiguration` securable). This configuration is stored in `[authorization].[LdapSyncConfiguration]` table in the security database.

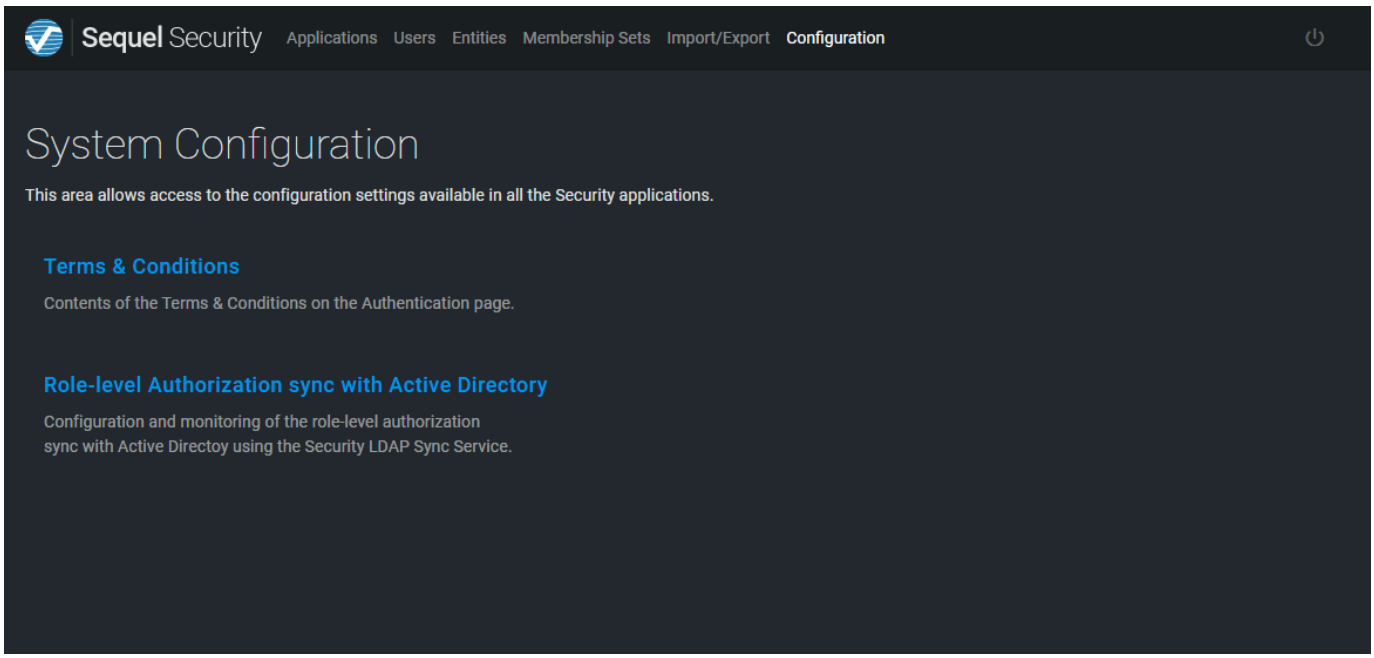


If there is not a configuration stored in the system; the GET endpoint returns a default configuration. If PUT is requested and there is not a configuration stored; then a new configuration is stored in the system. If there is already a configuration stored; this configuration is updated.

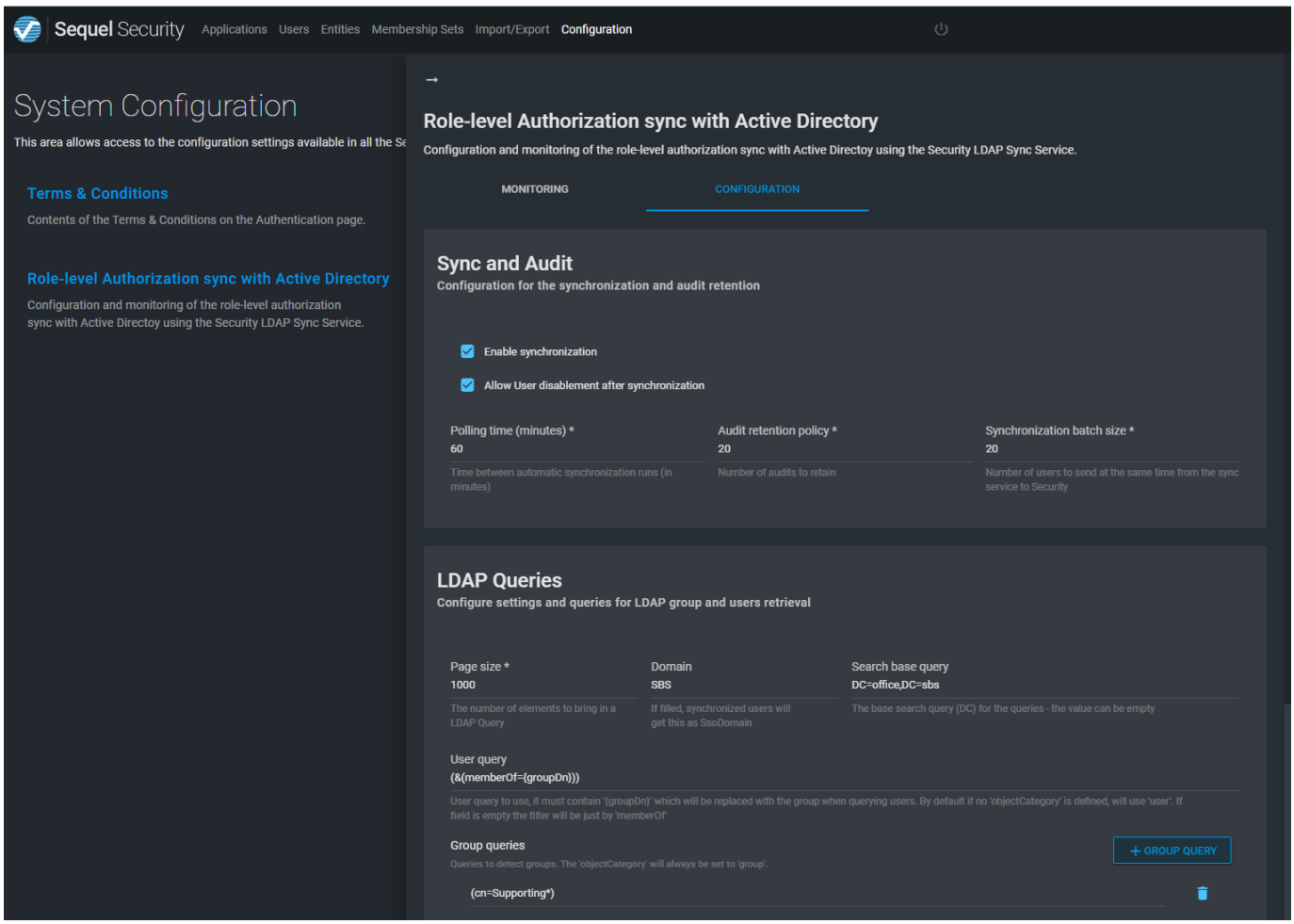
This resource is used by the Administration UI for configuring the sync process, al settings are described in the next section.

Configuring from UI

The configuration can be managed from the Admin UI as well from the *System Configuration* in the section *Role-level Authorization sync with Active Directory*.



Clicking on this option we will be able to configure the settings displayed above in below screen:



A `LdapSyncConfiguration` object contains below properties:

Property	Type	Description
Enabled	boolean	Determines when the sync is enabled or not. By default is true. However, if the sync service is not enabled and active the sync will not work.
GroupQueries	List of GroupQuery.	List of LDAP queries to retrieve the DN groups involved in the sync process. The Vanilla configuration contains a single entry with below LDAP query `(&(cn=Sequel Application Users))`. So, by default all users assigned to the group `Sequel Application Users` will be sync'd. If there are no groups queries, the process will be cancelled.
PageSize	int	Queries executed are paged using this setting as the page size. Default value is *1000*. This value should be lower or equal than the maximum number of objects returned by a single query in the AD server.
PollingMinutes	int	Polling frequency in minutes. By default *60* minutes. If changed and the service is already running this value will not be reflected until the next process execution.
ProcessAuditRetention	int	Maximum number of process audited retained in the system. By default value is *20*. When a process is completed, the housekeeping process is executed deleting the oldest entries and keeping just `ProcessAuditRetention` process in the audit tables.
Domain	string	Defines the domain assigned to the sync'd users. This domain will be stored in the user record. This setting is quite important as the user will be authenticated using {domain}{ssoUserName}.
SearchBaseQuery	string	The base distinguished name to search from. *Empty* by default.
SyncBatchSize	int	Number of users to include in each batch sent to sync. By default 20.
UserMapping	LdapUserMapping	Mapping definition for SsoUsername, Username, FirstName and LastName.
UserQuery	string	LDAP query to retrieve users. If empty/null, then the default query is used: `(&(memberOf={groupDn}))`, where `{groupDn}` is a token to be replaced with a DN group. If this query is customized, the token need to be present.

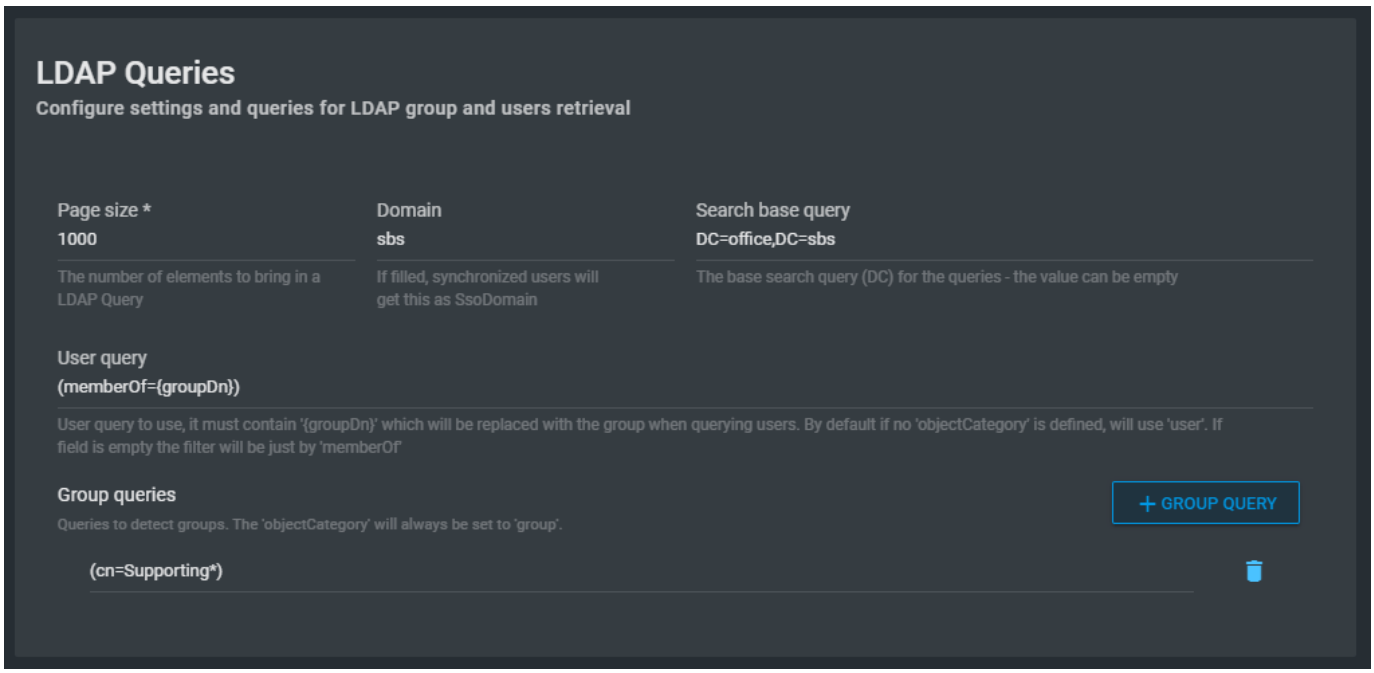
The `GroupQuery` object contains a single property called Query (string) that contains a LDAP query for retrieving group information. The Vanilla configuration contains a single entry with below LDAP query:

```
(&(cn=Sequel Application Users))
```

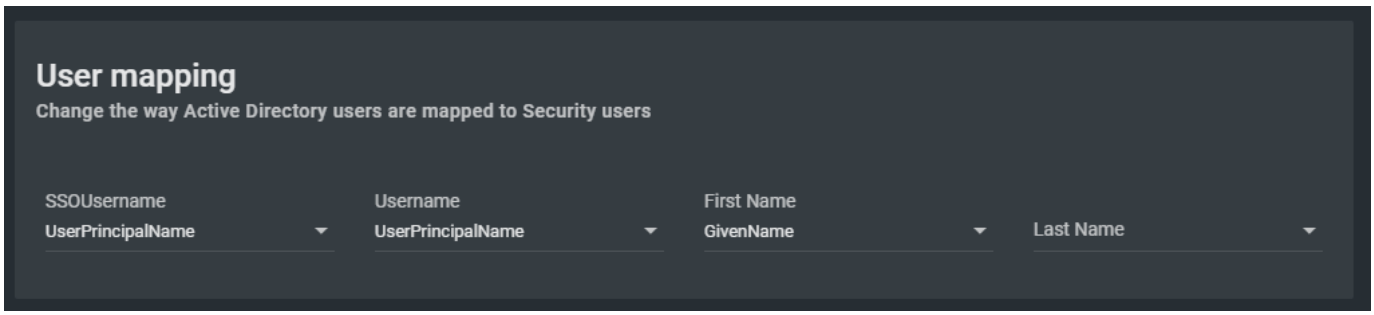
So, by default all users assigned to the group `Sequel Application Users` will be sync'd.

The `SearchBaseQuery` indicates a base domain to be used in the queries. This value must be set if request are done against the global catalog. The value should contain `DC` entries: if your base domain is `office.local`, the `SearchBaseQuery` should be `DC=office,DC=local`.

The approach followed to configure the LDAP queries reduces highly the risk of a LDAP injection, also this configuration can be done by security admin users so the risk is even lower. However, we recommend to use a service account for accessing Windows AD with read only permissions.



The `UserMapping` object contains a few properties to define the mappings; Maybe the most important are *SsoUsername* and *Username* that must be assigned to *SamAccountName* or *UserPrincipalName*; in order to identify the user and define how to map the user. The SsoUsername identifies the user in the AD.

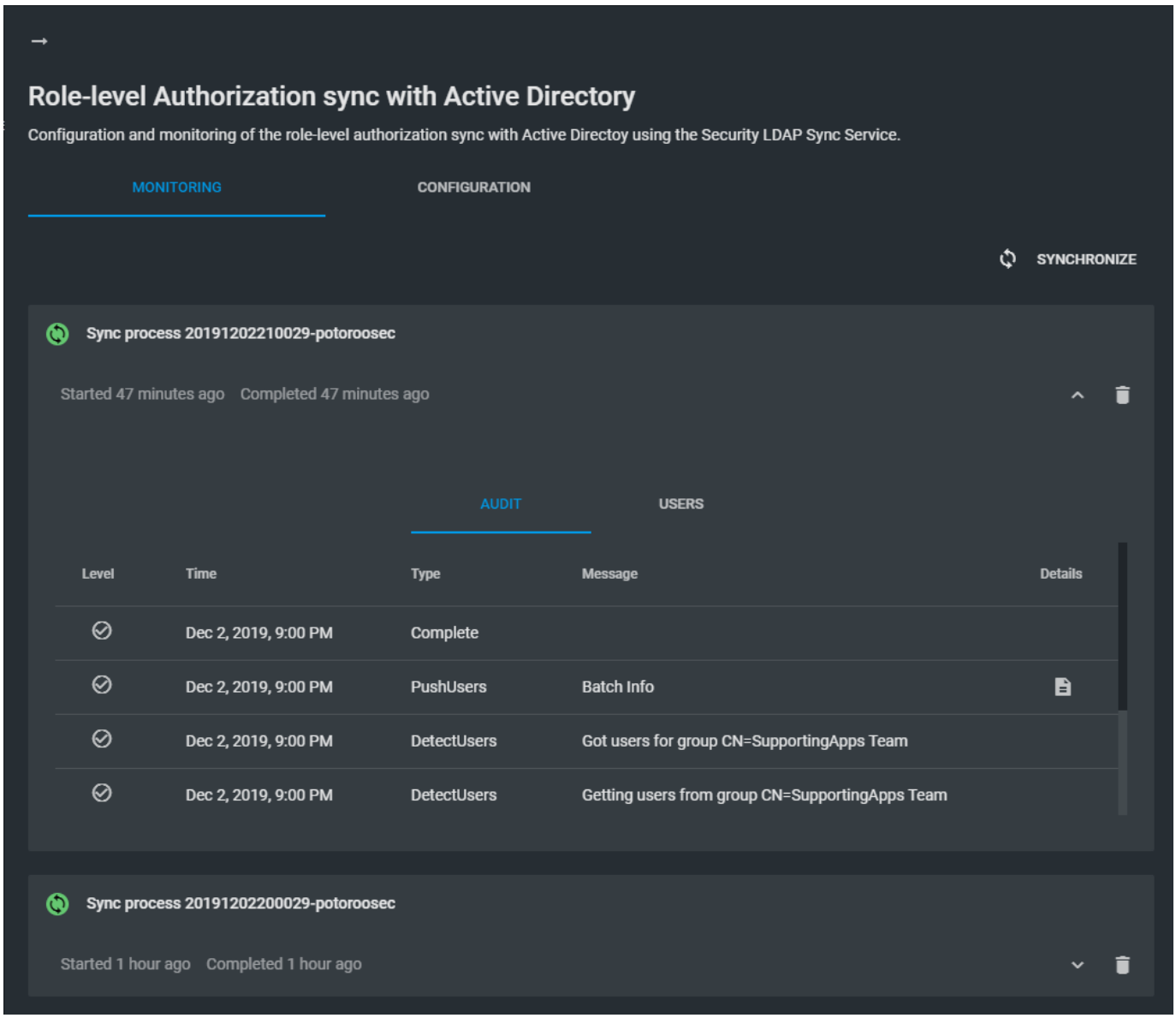


> **Info**
>
> This configuration cannot be exported currently.

Monitoring

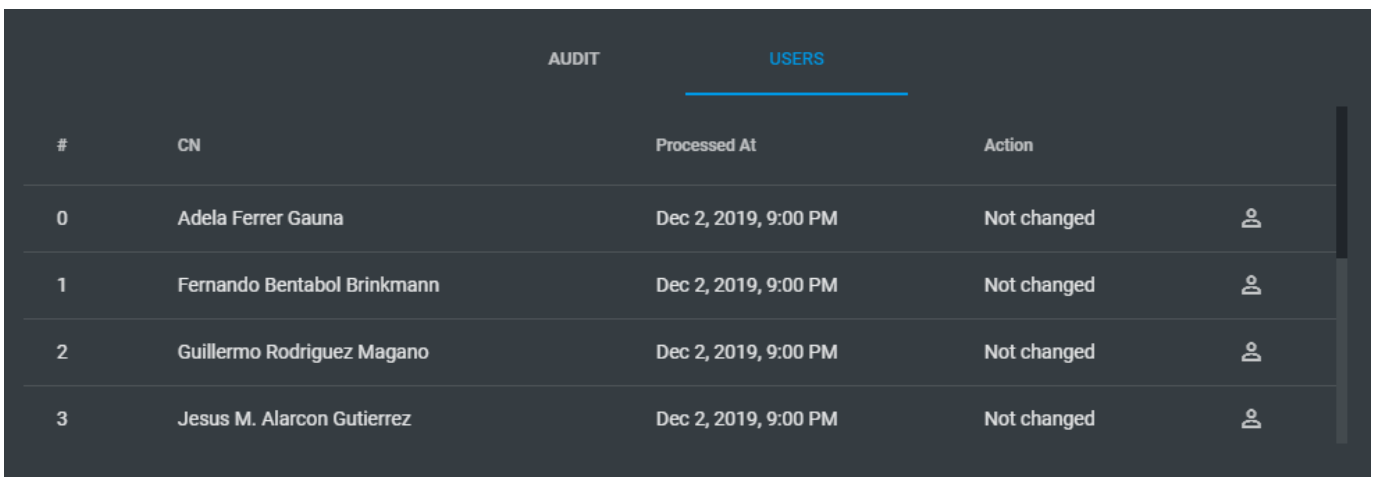
At monitoring page, we can review the list of recent sync process executed and its state. By expanding a process, you will be able to check the audit log and the synchronized users.

The monitoring will automatically refresh the information every 10 seconds when the user is in the *Monitoring* page.

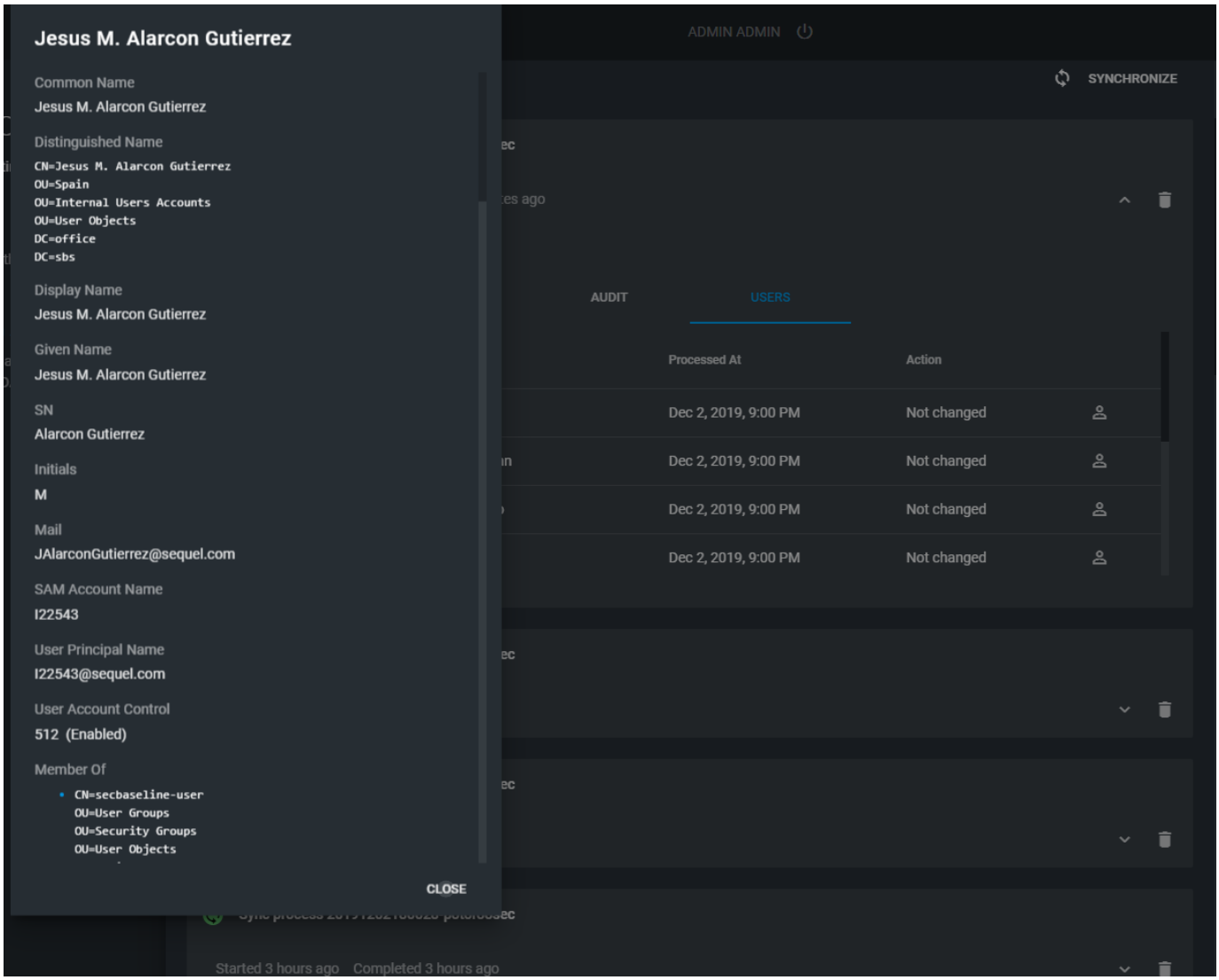


At the *audit* tab, all steps of the process are logged and some of them provide more details about the process. If any step contains errors, the step indicates this state and provide details about the problem.

The *users* tab offers information about the users detected in the sync process and the status of synchronization: created, updated, not-changed,...



Clicking on the user icon of each row, the information collected from AD is presented.

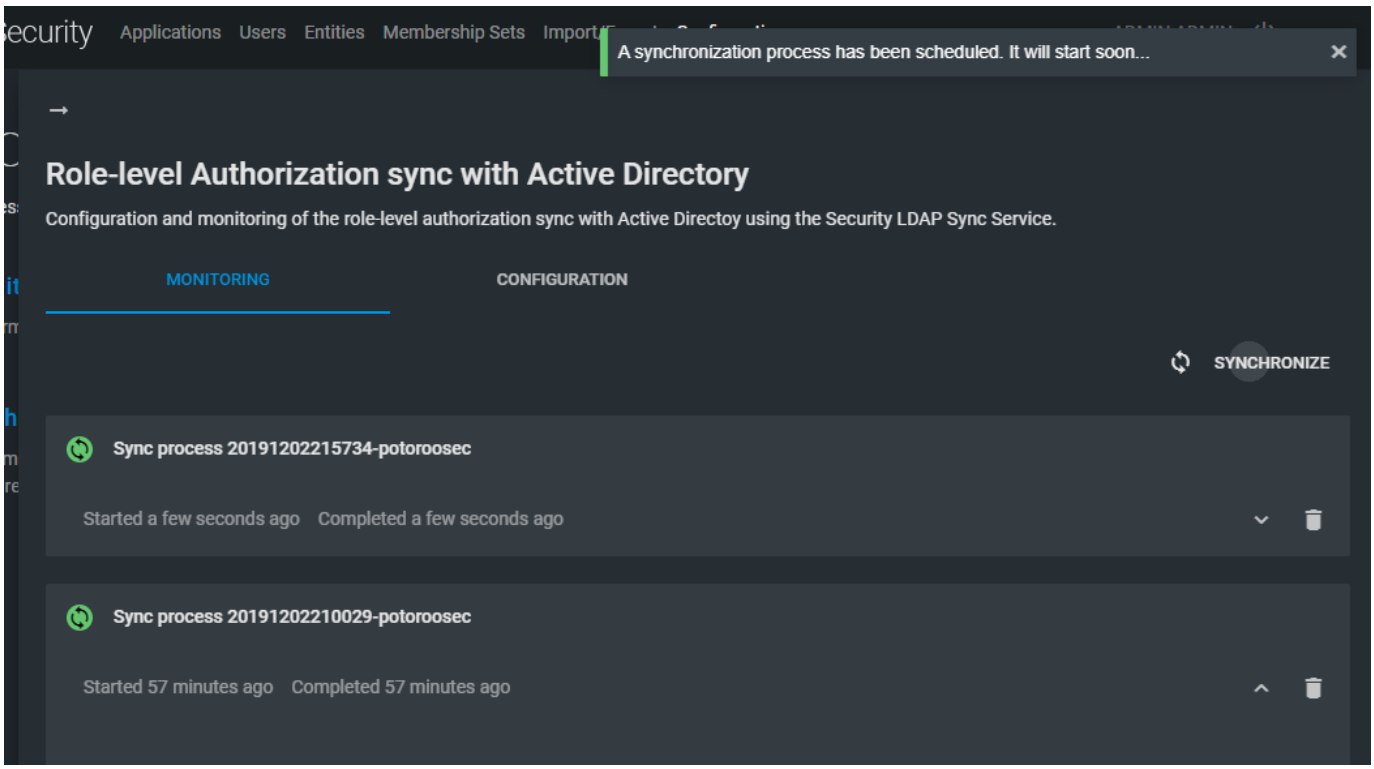


Related API endpoints for retrieving information for this screen are:

- `GET LdaPSync\Process` : returns all process registered by the system.
- `GET LdapSync\Process\{syncProcessId}\Audit` : returns all audit entries for sync process `syncProcessId` ordered by `AuditAt` field.
- `GET LdapSync\Process\{syncProcessId}\Users` : returns all `LdapSyncUser` entries for sync process `syncProcessId` ordered by `AuditAt` field. This model contains the `LdapUser` information and some control properties (when was requested the synchronization, the synchronization was completed, has created a new user, has updated an existing user,...).

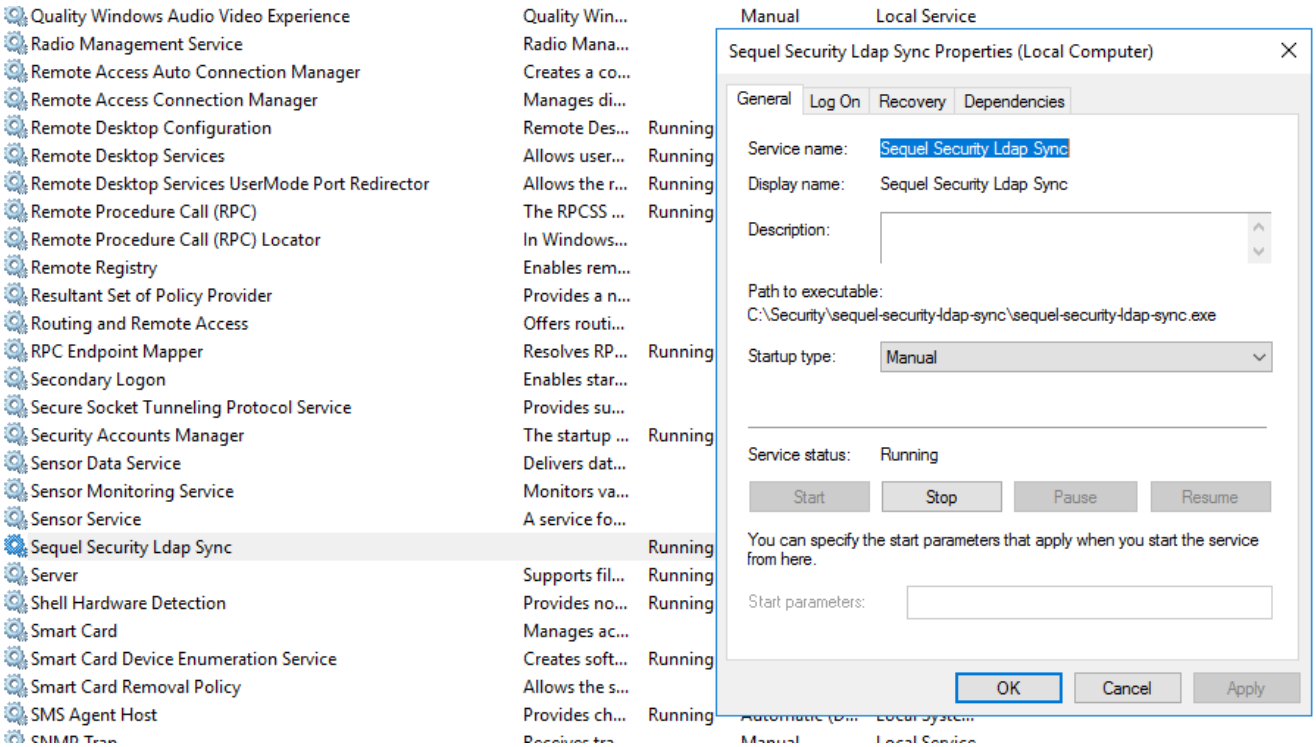
Force a manual sync

Also, from this screen a new manual sync can be forced. Clicking on the "*Synchronize*" button, a request to API to `POST LdapSync\Process\` is sent and the sync process is manually triggered. This functionality requires the service bus; as the process is triggered asynchronously.



EXTRACT: THE SYNC SERVICE

Sync service is a .net core application that can be executed as a console application and also hosted as a Windows Service; and it is responsible of querying the AD server for getting the affected users and push these information to the Security API for reflecting the changes in the Sequel's security schema.



The sync service (`sequel-security-sync-ldap.exe`) executes the extract process:

- after being started (startup).
- after a scheduled time (polling): scheduled by default to be executed each hour. This value can be customized. The configuration is refreshed from the SecurityAPI during the startup and in each polling/execution.
- a manual request (manually-forced): the process can be manually triggered from the Admin UI (also from Security API). For using this functionality is required to configure the message bus settings to the sync service. A manual request refreshes the configuration and restarts the polling.

Dependencies

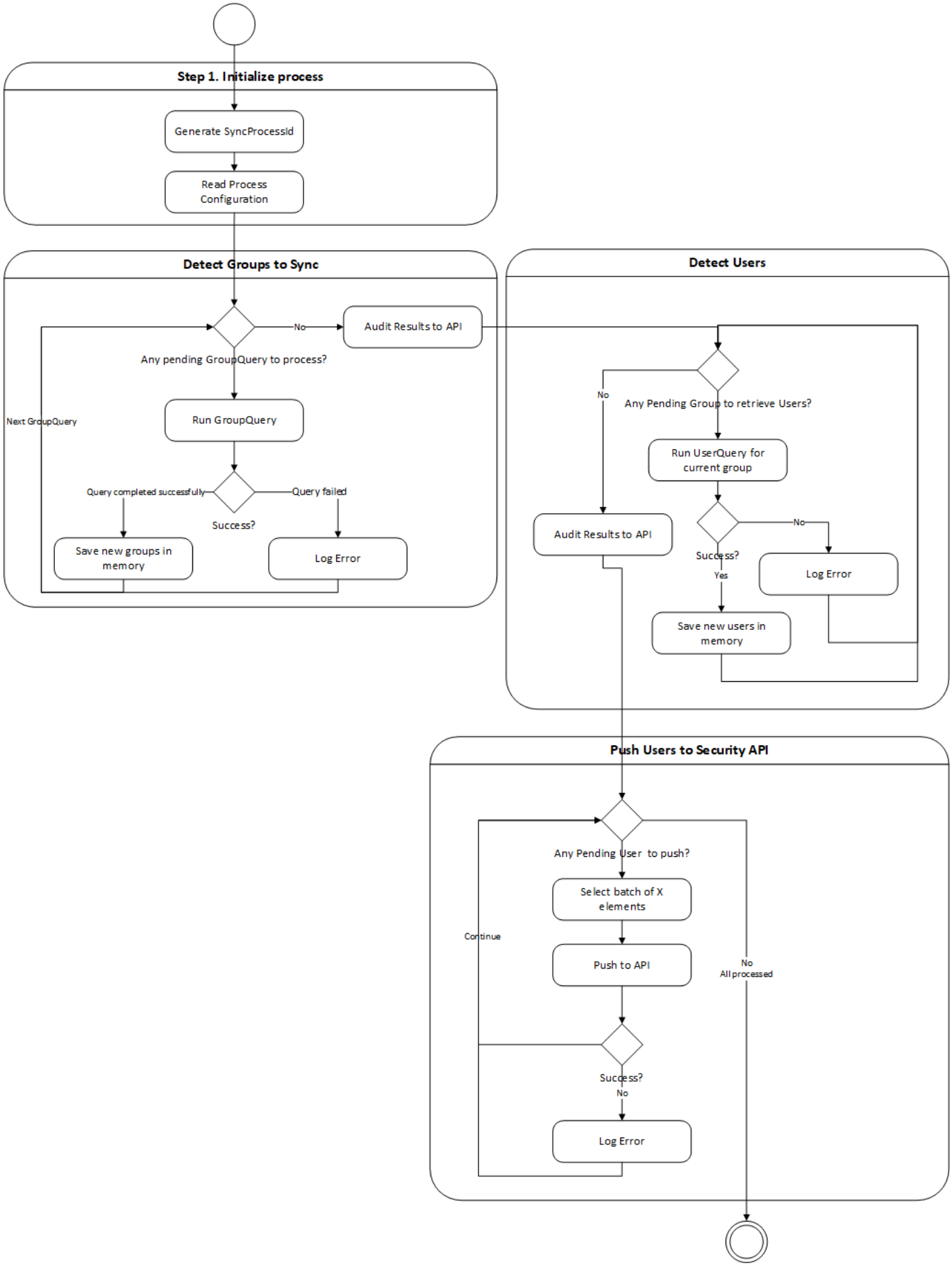
The sync service has dependencies with:

- Windows AD - LDAP server. (`LdapConnection`)
- Sequel's Authentication Service
- Sequel's Security API.
- Message Bus. (`MessageBusSettings`)
- Sequel's Logging repository. (`LoggingSettings`)

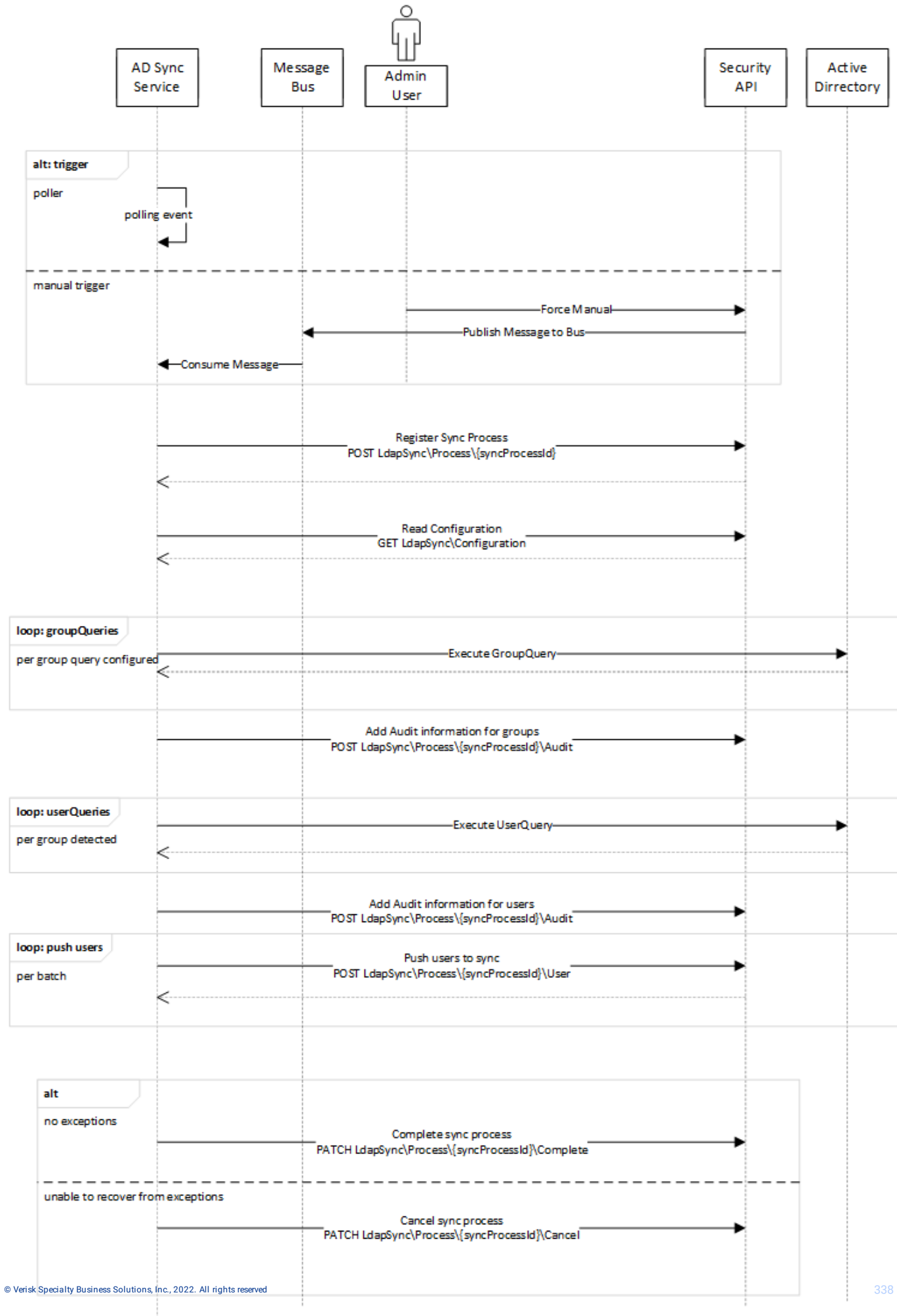
Process description

The extract part of the sync process is described with below activity and sequence diagrams.

Activity diagram



Sequence diagram



Starting the process

The process start because the service was started, it was scheduled or a message was received in the bus (`Sequel.Security.MessageBus.Contracts.LdapSync.v1.RunProcess`). If a request is received during the execution of the process, this request is ignored by the system, sending a Cancel message to the API.

Once started, the extract process is divided in four steps: initialize, detect groups, detect users and push users.

> **Note**
>
> *at startup, if there are more than one messages pending to be consumed, those will be cleared and only one synchronization will be run.*

Step 1. Initialize process

At this step the process is initialized:

- a unique identifier for the process is generated. The ID will be generated as `YYYYMMDDhhmmss-MachineName`.
- and, configuration is retrieved from the Security API service. As part of this request, the process is registered in the Security Service.

For starting the process, the sync service will call to `POST LdapSync\Process\{SyncProcessId}`. This endpoint will create entries on the `[authorization].[LdapAudit]` table. Once received the response, the sync service requests the current configuration and applies it to perform the process. The polling time can be affected with the refresh of the configuration.

If the sync process is disabled in configuration; the request to create a new one will return bad request; and the sync process will send a request for cancelling the process.

Always, the latest configuration will be requested and applied; even if the process is cancelled. So, the next execution is configured with the latest configuration (waiting time)

Step 2. Detect groups to sync

In this phase the groups to be sync'd are detected. The configuration process contains a list of LDAP queries to retrieve the DN of the groups that in the next phase will be used to retrieve the users to be sync'd. A sample query for retrieving groups will look like:

```
(&(cn=Sequel Application Users))

(&(cn=*Underwriters*))
```

The configuration contains a list of queries for retrieving DN of groups. All those queries have to be executed. Once all queries have been executed, this has to be notified to the security API calling to `POST LdapSync\Process\{SyncProcessId}\Audit` where the body contains below contract:

```
[
    {
        "Level" : "Error" | "Information" | "Warning",
        "Type"  : "DetectGroups" | "DetectUsers" | "PushUsers",
        "Message" : "Brief message describing the error/action",
        "Details" : "Details related to the error/action: query executed, parameters passed,...",
    },
    {
        "Level" : "Error",
        "Type"  : "DetectGroups",
        "Message" : "Error executing query X",
        "Details" : "Error message received when the query was executed"
    },
    {
        "Level" : "Information",
        "Type"  : "DetectGroups",
        "Message" : "Executed query X",
        "Details" : "Found n groups."
    }
]
```

The DNs retrieved from the queries will be stored by the process in memory managing below requirements:

- Groups members of other groups are not processed. Inheritance of groups is not supported.
- List of groups cannot contain duplicated groups to avoid duplicated request in the next steps.

Step 3. Detect users

With the list of DN groups from the previous phase; we will detect the users to be sync'd. The query for retrieving users is defined in the `UserQuery` property of the configuration; using the default one or the custom one. The query will be validated ensuring the query contains the token `{groupDn}` . As this token is replaced with the DN groups detected in the previous phase.

The list of attributes requested in this query to AD are:

Attribute	Description
CN	Common Name. Maps to 'Name' in the LDAP provider. CN is a mandatory property. See also SamAccountName.
DN	Distinguished Name. A DN is a sequence of relative distinguished names (RDN) connected by commas. DN is simply the most important LDAP attribute.
DisplayName	Display name
GivenName	First name
Initials	Initials
Mail	User's email
Name	Exactly the same as CN.
SamAccountName	The old NT 4.0 logon name
SN	Last name or surname.
UserAccountControl	Used to disable an account. A value of 514 disables the account, while 512 makes the account ready for logon. More info https://support.microsoft.com/en-us/help/305144/how-to-use-useraccountcontrol-to-manipulate-user-account-properties
UserPrincipalName	Often abbreviated to UPN, and looks like an email address. Note UPN must be unique in the AD.
MemberOf	List of groups where user is member. Groups are described in DN format

The user information stored in memory uses the `LdapUser` model contract used in `POST LdapSync\Process\{SyncProcessId}\Users` for pushing changes to API. This information is stored in the database for tracing purposes as well at `[authorization].[LdapSyncUser]` .

Similar to the previous step, this process needs to notify to security API about the current state calling to the same endpoint `POST LdapSync\Process\{SyncProcessId}\Audit` where the body contains below contract:

```
[
    {
        "Level" : "Error" | "Information" | "Warning",
        "Type"  : "DetectGroups" | "DetectUsers" | "PushUsers",
        "Message" : "Brief message describing the error/action",
        "Details" : "Details related to the error/action: query executed, parameters passed,...",
    },
    {
        "Level" : "Error",
        "Type"  : "DetectUsers",
        "Message" : "Error executing query for group DN",
        "Details" : "Error message received when the query was executed"
    },
    {
        "Level" : "Information",
        "Type"  : "DetectUsers",
        "Message" : "Detected users for group DN",
        "Details" : "Found n users."
    }
]
```

The aggregated list of users detected cannot contain duplicated entries, so we can ensure we will not request duplicated syncs for a given user.

Step 4. Push users to Security API

At this step, the process is managing a list of users to be sync'd. Following the indications of the configuration, the process creates batches of users and post them to `POST LdapSync\Process\{SyncProcessId}\Users`, using below contract in the body:

Attribute	Type	Data Type	Description
SyncProcessId	url section	string	Unique identifier of the sync process
BatchSeqNumber	body	int	Sequence number of the batch
TotalUsers	body	int	Total number of users to sync in this process
TotalUsersInBatch	body	int	Total number of users in this batch. This must be the same value than users.Count
FromUserNumber	body	int	Position of the first user included in this batch in the complete list of users to sync.
ToUserNumber	body	int	Position of the last user included in this batch in the complete list of users to sync.
Users	body	LdapUser[]	List of LdapUser to sync

Completing the process

When the process is completed, this is notified to the API calling to `PATCH LdapSync\Process\{SyncProcessId}\Complete`.

In general, only unhandled errors will force to stop the process. In this case, errors are logged to the Sequel.Core.Logging repository; and cancellation will be notified to the Security API (if possible) `PATCH LdapSync\Process\{SyncProcessId}\Cancel`, including the reason: the error message of the unhandled exception.

Infrastructure settings

All infrastructure dependencies are reflected in the `appsettings.json` file located at the LDAP service installation folder (e.g.: `C:\Security\sequel-security-ldap-sync`). For further information, see also AD Sync Registration.

```
{
  "LdapConnection": {
      "Host" : "host-name (string)",
      "Port" : 3268,
      "SecureConnection": true,
      "DN" : "dn username",
      "Password" : "password encrypted"
  },
  "SecurityConnection": {
    "ClientId" : "sec.ldapsync",
    "ClientSecret" : "secret",
  },
  "ServiceDiscoverySettings": {
    "Mode": "PointToPoint",
    "RequiredServices": {
      "SecurityApi": {
        "InternalUrl": "<INTERNAL-URL>",
        "ExternalUrl": "<EXTERNAL-URL>"
      },
      "Authentication": {
        "InternalUrl": "<INTERNAL-URL>",
        "ExternalUrl": "<EXTERNAL-URL>"
      }
    }
  },
  "MessageBusSettings": {
    "Application": "SecurityLdapSync",
    "Instance": "{machine-name}",
    "RabbitMqSettings": {
      "Password": "pwd-encrypted",
      "ServerUri": "rabbitmq://sbsmpormqsmt.office.sbs/SuppAppTests",
      "UserName": "security",
      "PurgeOnStartup": true,
      "Timeout": "0.00:45:00.0000",
      "BindMessageExchanges": true
    },
    "Consumers": [
      {
        "ConsumerType": "Sequel.Security.Tools.LdapSync.Application.Consumers.RunProcessConsumer, Sequel.Security.Tools.LdapSync.Application, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=null",
        "MessageType": "Sequel.Security.MessageBus.Contracts.LdapSync.v1.RunProcess, Sequel.Security.MessageBus.Contracts, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null",
        "QueueName": "LdapSyncRunProcess",
        "QueuePerInstance": false
      }
```

```
    ]
  },
  "LoggingSettings": {
    "MsSqlLoggingSettings": {
      "LoggingEnabled": true,
      "ConnectionString": "sql-connection-string",
      "IgnoredLogTypes": [],
      "MinimumLogLevel": "Debug"
    },
    "RollingFileLoggingSettings": {
      "LoggingEnabled": true,
      "PathFormat": "C:\\Temp\\log-{Date}.txt",
      "PropertiesTemplate": "{Message}{MessageTemplate}{TimeStamp}{Level}{Exception}{Properties}{LogEvent}{ErrorGuid}{TraceGuid}{SystemName}{ExecutionTime}
{MachineName}{LogType}{UserName}",
      "PropertyFormat": "{{PropertyName}:{PropertyValue}}{Tab}",
      "IgnoredLogTypes": [],
      "MinimumLogLevel": "Debug"
    }
  }
}
```

LdapConnection

Parameter	Type	Description
Host	string	A host name or a dotted string representing the IP address of a host running an LDAP server. It may also contain a list of host names, space-delimited. Each host name can include a trailing colon and port number
Port	integer	The TCP or UDP port number to connect to or contact. The default LDAP port is **389** and LDAPS is **636** (enabled if `SecureConnection` is true). The port parameter is ignored for any host name which includes a colon and port number. **Port 3268.** This port is used for queries specifically targeted for the global catalog. LDAP requests sent to port 3268 can be used to search for objects in the entire forest. However, only the attributes marked for replication to the global catalog can be returned. For example, a user's department could not be returned using port 3268 since this attribute is not replicated to the global catalog. **Port 636.** Port for LDAPS. **Port 389.** Cannot be used as this port doesn't support the type of queries we are using.
SecureConnection	boolean	If the value is `true`, then uses TLS for the connection to LDAP (LDAPS). By default, is set to `false`.
DN	string	If non-null and non-empty, specifies that the connection and all operations through it should be authenticated with DN as the distinguished name. We highly recommend to use a service account or user with only read permissions on Windows AD.
Password	string / encrypted	If non-null and non-empty, specifies that the connection and all operations through it should be authenticated with DN as the distinguished name and this argument as password. This value is encrypted in the appsettings file.

SecurityConnection

Contains information for authenticating and connecting to the Security API, following the standard defined in other components of security services

Parameters	Type	Description
ClientId	string	Client id for this application: `sec.ldapsync`. The client requires access to `sec.api` scope
ClientSecret	string / encrypted	Secret for the client.

MessageBusSettings

Defined following schema provided by `Sequel.Core.MessageBus`, and configured with a single consumer `Sequel.Security.Tools.LdapSync.Application.Consumers.RunProcessConsumer`, for the contract `Sequel.Security.MessageBus.Contracts.LdapSync.v1.RunProcess`. The API publishes this message with a time to live of 5 minutes by default.

LoggingSettings

Defined following schema provided by `Sequel.Core.Logging`.

TRANSFORM: MEMBERSHIPSET

When a request is received to `POST LdapSync\Process\{SyncProcessId}\Users`, two main actions occurs:

- Transform a `LdapUser` to a Sequel's users.
- Load User (upsert).

In this section, we will cover the transformation logic.

Transformation logic

The transformation logic is function that transforms a `LdapUser` into a Sequel's User. We have to differentiate two types of information.

Basic information

Including information as username, first name, last name or email address. This mapping is defined with default settings, and some of them can be customized. Below table describes the available fields and the default mappings:

Properties in LdapUser object	Mapped	Properties in Sequel's User object
CN		-
DN		-
DisplayName		-
GivenName	Yes	FirstName
Initials		-
Mail	Yes	EmailAddress
Name		-
SamAccountName	Yes	SsoUsername
SN	Yes	LastName
UserAccountControl	Yes	IsActive (If UserAccountControl is disabled)
UserPrincipalName	Yes	Username
MemberOf	Yes	Membership using MembershipSets

The mapping for `SsoUsername` is quite important; as this will allow the user to SSO using Windows Authentication. But to complete the login, `SsoDomain` must be defined in the configuration (using UI) so both `${SsoDomain}\${SsoUsername}` will form the valid username for the Authentication.

> **Note**
>
> *`SsoDomain` is populated automatically by the LDAP Synchronization*

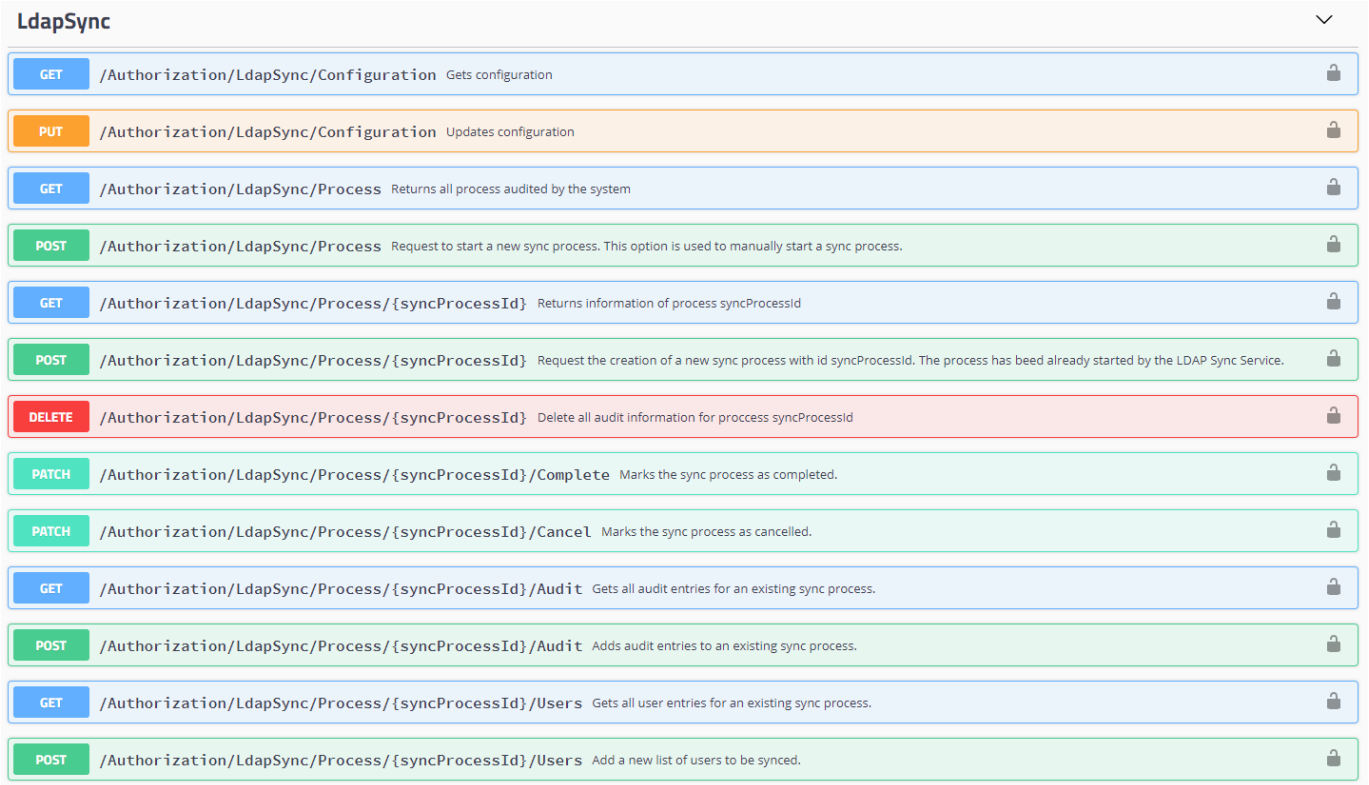
The configuration for mappings (`SamAccountName` and `UserPrincipalName`) can be done from the Ldap Configuration API endpoint and associated UI (card "User mappings").

Membership information

The memberships translation logic is convered with details at membership set management section.

LOAD: THE SECURITY API ENDPOINT

The transformation and load parts of this ETL sync process are managed in the Security API; including configuration and monitoring functionalities; all defined under the `LdapSync` resource.



The creation of a new sync process will always audit the request; when the sync is disabled the response will be a BAD REQUEST with a validation error in the body indicating the sync process is disabled.

Process endpoints

GET /Process

Returns all process audited by the system.

POST /Process

Publishes a message in the bus that will be consumed by Sync Service and then the service will start the sync process.

GET /Process/{syncProcessId}

Returns information for `syncProcessId`

POST /Process/{syncProcessId}

Registers the start of a new sync process. This will record information in the sync audit tables a new entry of type Started.

DELETE /Process/{syncProcessId}

Delete all audit information for process syncProcess (auditing and user information).

PATCH /Process/{syncProcessId}/Complete

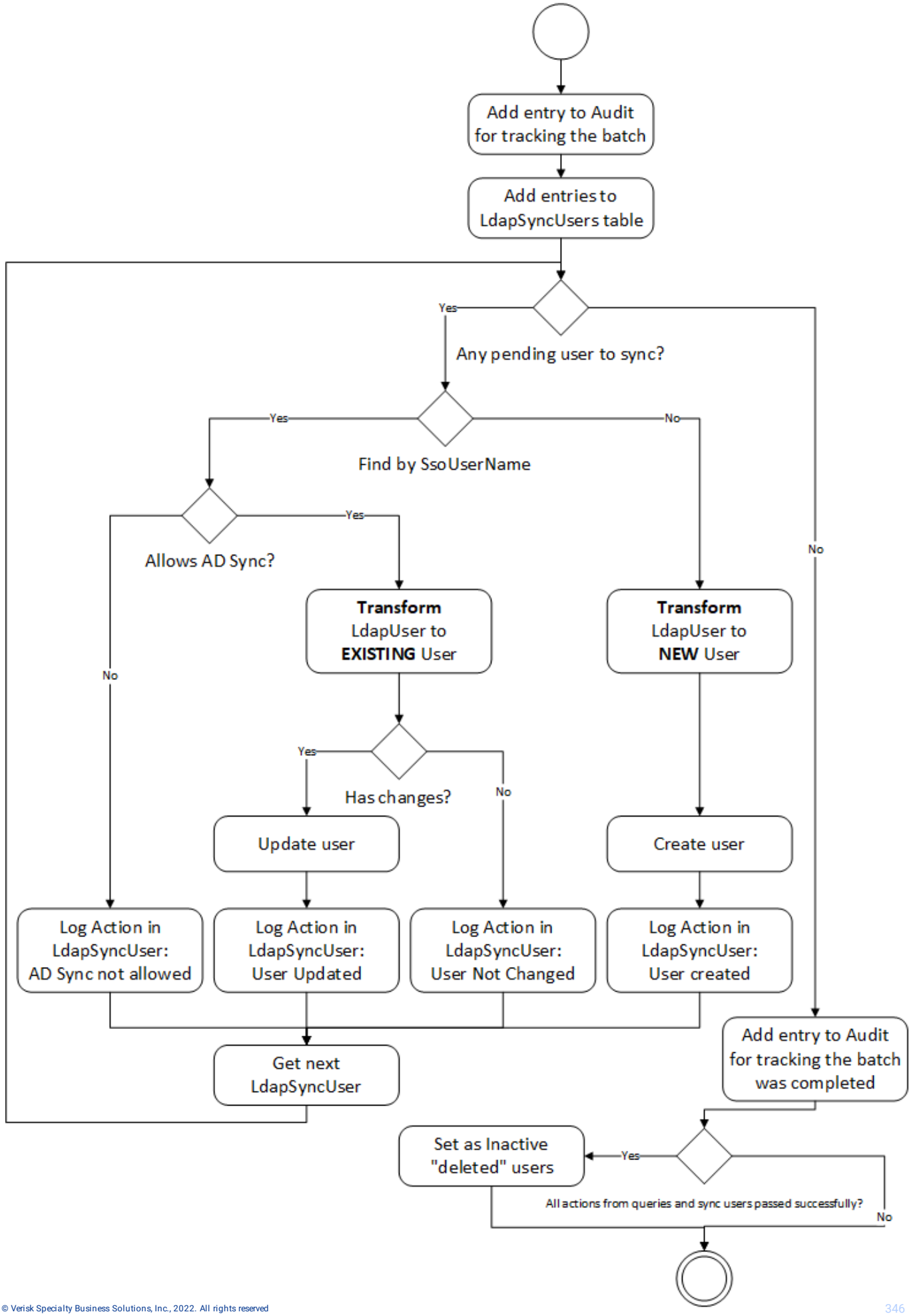
Completes the process. The underlying action is adding a new entry in Audit with the type Completed. If all process is successfully completed without errors, the detection of deleted users is executed. The audit retention policy is applied here.

PATCH /Process/{syncProcessId}/Cancel

Completes the process. The underlying action is adding a new entry in Audit with the type Cancelled. The audit retention policy is applied here.

Sync users

Users are stored in the Sequel's AuthZ schema when receiving a request in `POST LdapSync\Process\{SyncProcessId}\Users`. The logic for this save is described below:



A few rules applies to this process:

- All request are included in the `LdapAudit` when starting and when completing.
- All `LdapUsers` included in the request are stored in the `LdapSyncUser` table; linked to the `LdapSyncProcess`.
- All `LdapUsers` have to be processed. If one of them fails, the information is logged and audited; and the process carries on with the next one.
- Users contains a flag to indicate if LDAP Sync is enabled (`LdapSyncEnabled` bit). If false, users are not synchronized. Also, a new field `LdapSyncUpdatedAt` : nullable datetime updated each time the user is updated due to the sync process.
- Membership entries contains a flag to indicate when the entry has to be or not included in the synchronization (`LdapSyncEnabled`). All entries created by the sync process will be flagged as true. Only `LdapSyncEnabled` entries can be removed during the sync process.
- Users will be updated only when there are changes: basic details or memberships. When the user is created or changed, the related message bus are expected to be published.
- Saving/Creating users/memberships should apply the same rules applied to any action done to users/memberships from API.

Deleted users

Once the sync process is completed without errors; the "deleted user detection" will be triggered but only if enabled in configuration. This process will mark as inactive all users with `LdapSyncEnabled` enabled and were not included in the sync process.

STARTING GUIDE

For starting working with the role-level based authentication sync with Active Directory we need:

- The LDAP Sync service installed and running (see Installation Guide)
- Role-level based Authentication Sync minimal configuration (see Azure AD AuthN or Azure AD Authentication Registration if necessary). Assuming all default settings are used.
- At least, the domain for the new users created must be populated.
- Depending on Active Directory configurations; it could be required to review some LDAP query settings.
- Membership Sets created and configured in Security; if there are no created Membership Sets created the sync process could create users but they will not have permissions to access.

APPENDIX

Models

LdapProcess model

Models a Sync Process. Contains below information:

Property	Type	Description
Id	string	Unique identifier of the sync process. The preferred format is `YYYYMMDDhhmmss-MachineName`
StartedAt	DateTime	When the process was started. In UTC format.
CancelledAt	DateTime	When the process was cancelled. In UTC format.
CompletedAt	DateTime	When the process was completed. In UTC format.

Authentication.User model

Changes to user model due to Ldap Sync:

Property	Type	Description
LdapSyncEnabled	boolean	Flag indicates if the user can be synchronized by Ldap Sync Process. New users are created with `false` as default; users created by the Ldap Sync Process are created with `true`.
LdapSyncUpdatedAt	DateTime	When the sync process applied changes for last time. In UTC format.

Authentication.Membership model

Changes to membership model due to Ldap Sync:

Property	Type	Description
LdapSyncEnabled	boolean	Flag indicates if the user can be synchronized by Ldap Sync Process. New users are created with `false` as default; users created by the Ldap Sync Process are created with `true`.

Enabling LDAP Sync on existing installation

If we enable LDAP Sync for an existing installation is important to analyse how this sync will affect to existing users. By default, users are flagged to do not sync with LDAP, just those ones created by the automated sync are created with this flag. There are a few points to bear in mind when LDAP Sync is enabled for an existing Security installation:

- Existing users could be flagged with LDAP Sync disabled. Review if those existing users need to be synced. If those users are changed to enable sync, membership must be reviewed as described in the next point.
- Review memberships defined for users with LDAP Sync enabled. In general, users with LDAP sync activated are expected to do not define membership manually (however, there are exceptions where this could be done in purpose).

Performance

Some performance tests have been done to determine the expected execution times. The test has been done in a server, with below specs, where all services of security where installed.

Server Specs	
Operating System Version	Microsoft Windows Server 2016 Standard
Processors	DUAL Intel(R) Xeon(R) CPU E5-2683 v4 @ 2.10GHz
Installed Memory (RAM)	4 GB
Total disk space (Free)	59.51 GB (26.2 GB)

The AD server was in the same private network, and the tests was done against a single group with 445 users; where each membershipset was configured with three entries.

LDAP Sync Process was executed with the default settings.

Scenario 1: All new users

In this test all users were created..

Five executions.	Min	Max	Avg
Total Time	24.4s	32.4s	26.8s
Collect AD	0.4s	0.4s	0.4s
Sync users	24s	32s	26.2s

Scenario 2: Updated 80% users

In this scenario, 80% of users were updated due to changes in the memberships.

Five executions.	Min	Max	Avg
Total Time	22.4s	25.4s	24.4s
Collect AD	0.4s	0.4s	0.4s
Sync users	22s	25s	24s

Scenario 3: Updated 5% users

In this scenario 5% users were updated due to changes in the memberships.

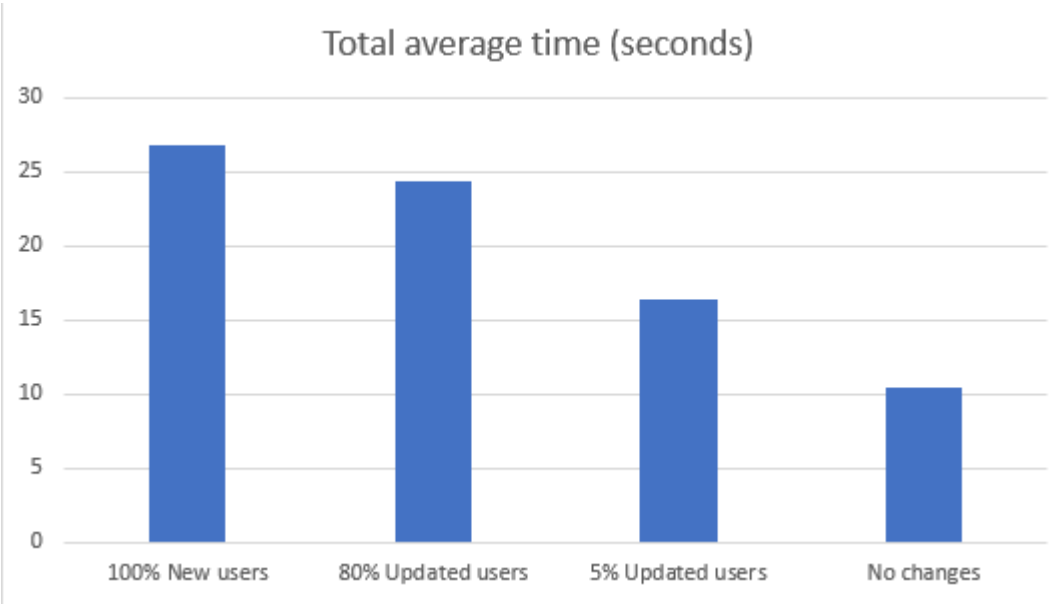
Five executions.	Min	Max	Avg
Total Time	11.4s	19.4s	16.4s
Collect AD	0.4s	0.4s	0.4s
Sync users	11s	19s	16s

Scenario 4: No Changes

In this scenario, the most probably, there were no changes detected and users were not changed.

Five executions.	Min	Max	Avg
Total Time	9.4s	12.4s	10.4s
Collect AD	0.4s	0.4s	0.4s
Sync users	9s	12s	10s

Comparative



5.5 Users

5.5.1 Users

A user is a human that is using a registered client to access resources. The information managed by the system is the minimum set of data required to identify the user and manage the security concerns. All other information associated to the user but specific to a given application must be considered *user profile information* and managed by each application. In this set of information we have to include the User Type model existing in the legacy security model. The user information is stored at `[authorization].[user]` table in the database.

Models

USER'S BASIC DETAILS

Personal information

Property	Description
Username	string. Unique Identifier for the user.
EmailAddress	string. Email address of the user. Must be unique. Required.
FirstName	string
LastName	string
PhoneNumber	string
AutomaticallyCreated	bool. Flags if the user was created automatically (e.g. Origin Portal)

Security

Property	Description
LastLoginDateUtc	DateTime?. Last login date (stored at UTC)
ActiveEndDateUtc	DateTime?. User will be inactive from this date (UTC)
PasswordHash	string. Password hashed
PasswordExpiryDateUtc	DateTime?. Date when password will expire (UTC)
SecurityStamp	GUID. The security timestamp is used for tracking changes made to the user profile. It is used for security purposes when important properties of a user change, such as changing the password.
AccessFailedCount	int. Counts the number of failed access since last successful access.
LockoutEndDateUtc	Date?. Locked till this date. Keeping Lookout* for implementing this functionality in the new service.
LockoutEnabled	bool. Enables/disables lockout mechanism

Single sign-on

Property	Description
AzureUserIdentifier	string. To store the oid provided by Azure AD for users using Azure AD authentication
SsoUserName	string. User name used for matching SSO with external providers.
SsoDomain	string. User's domain used for matching SSO with external providers.
LdapSyncEnabled	boolean. Determines where this user will be included in the AD/Azure AD sync
LdapSyncUpdatedAt	datetime. When the user was synced with AD/Azure AD for the last time.

MEMBERSHIP

A permission defines which CRUD actions (Create, Read, Update and Delete) are allowed over a specific securable for users assigned to a specific role. Role and group have to belong to the same application.

Property	Description
Key	Integer. Unique Key to refer the permission . Required. Autogenerated
GroupKey	String. Group.Key where this membership is defined. Required
RoleKey	String. Role.Key which this membership is configuring. Required.
LadpSyncEnabled	Boolean. Indicates if the membership is affected by LDAP/Azure AD sync.

USERTYPEASSIGNEDTOUSER

Each user can be assigned to one User Type per Application. Same UserType cannot be assigned more than once.

Property	Description
Key	Integer. Unique Id to refer the UserType assignation. Required. Autogenerated
UserTypeId	String. UserType.Id assigned to this user. Required

HOW TO MANAGE USERS

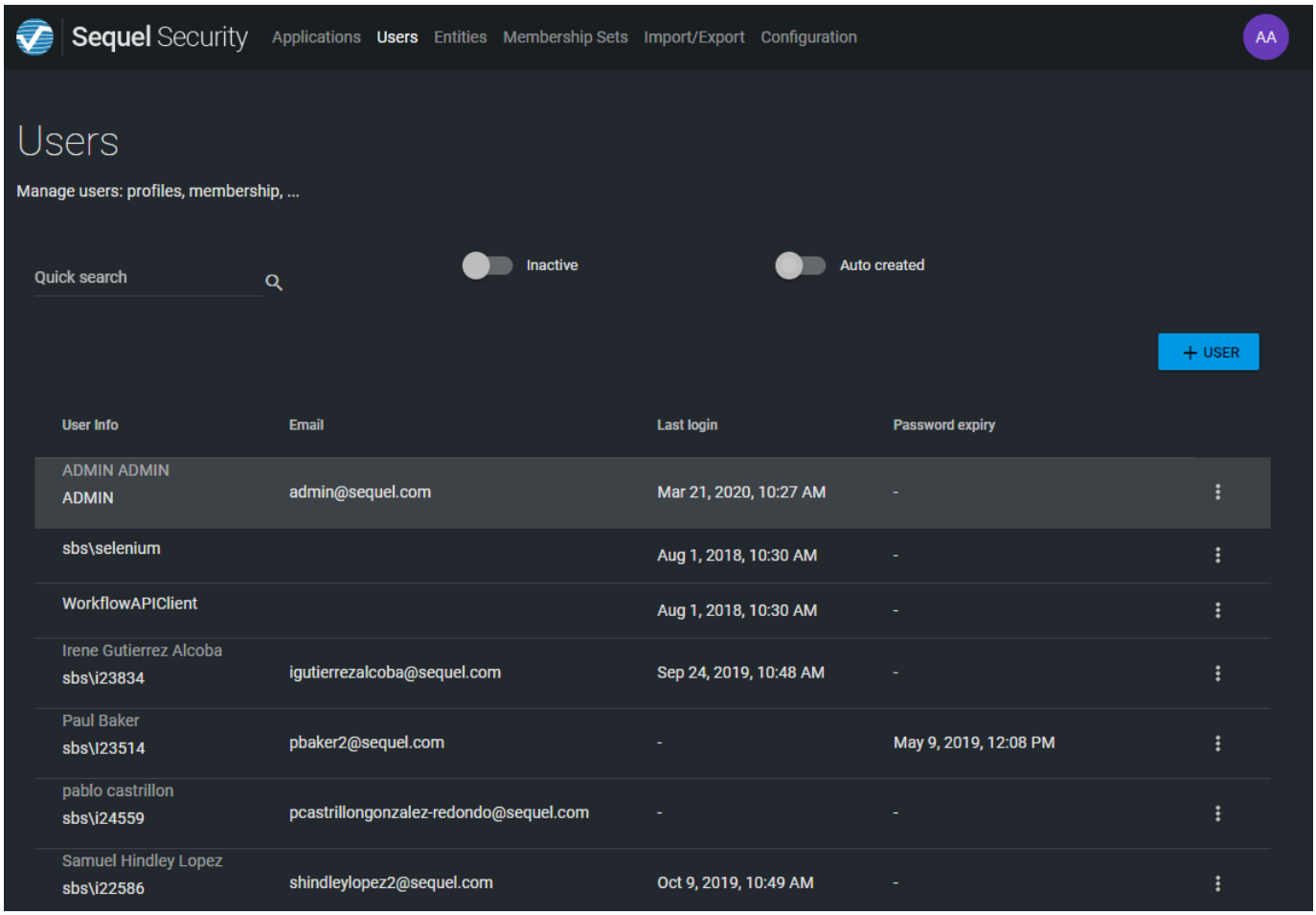
Users can be managed with the administration UI, using directly the API or importing a user package:

- *UI* at `/applications/users`. In the next sections, we will describe with more detail how to manage users using the UI.
- *API* at `/Authorization/Users`. Please, use OpenAPI specs offered by Swagger at `https://your-security-server/SecurityApi/swagger/`.
- *sequel-security* tool with command `-import`, `-export` and `-add-admin-user`. More information at sequel-security tool page.

Actions performed with UI and API are protected by securable `Sec.Users`.

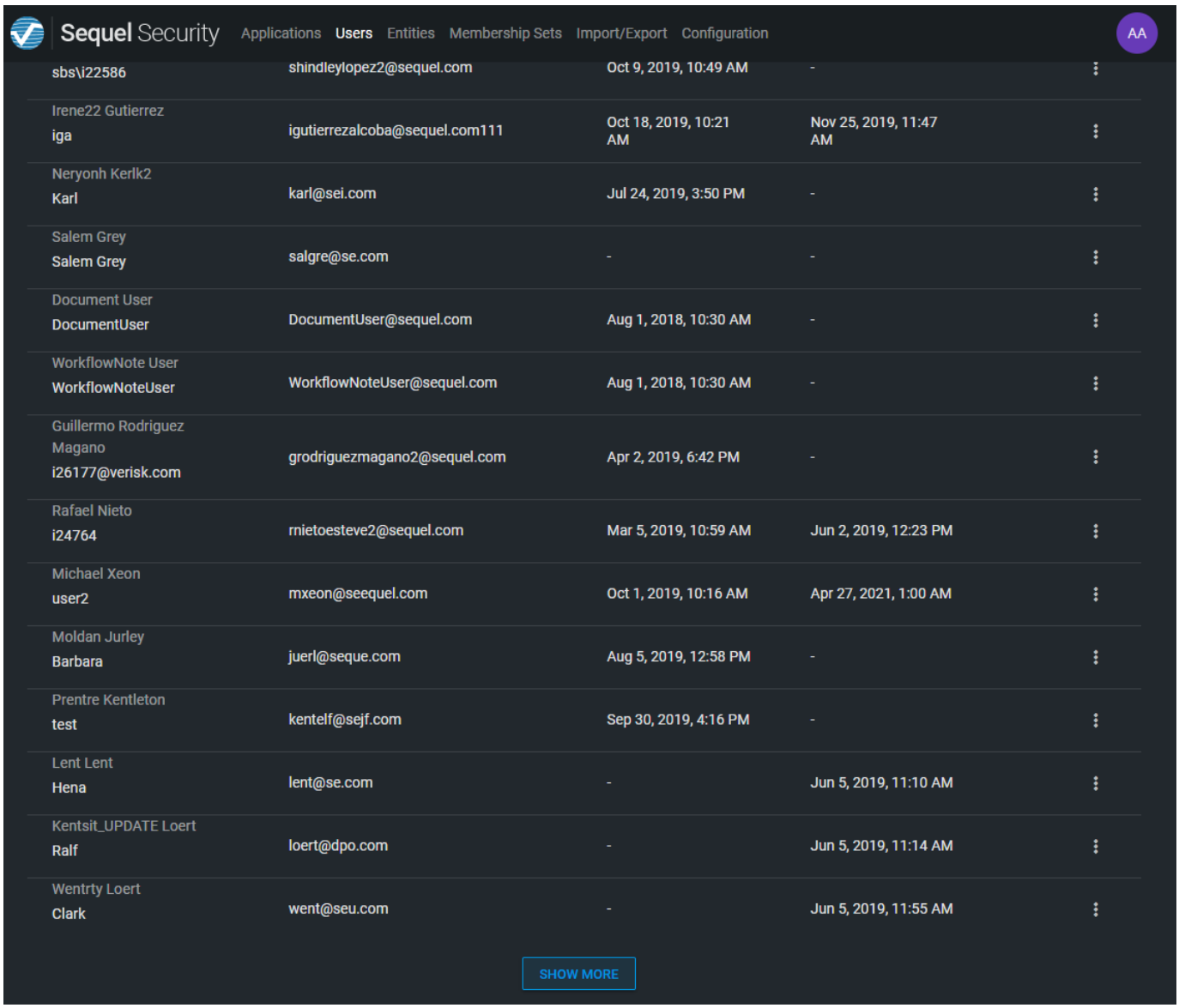
5.5.2 Search for users

User's home page at `/Authorization/Users` allows to search users.



- *Quick search*: search if user contains this criteria at username, first name, lastname or email.
- *Inactive*: . Turn on to include in results inactive users.
- *Auto created*: Turn on to include in results user `AutomaticallyCreated` .

Results are returned paged, if there are more results affected in the query, click *Show more* to return the next page.



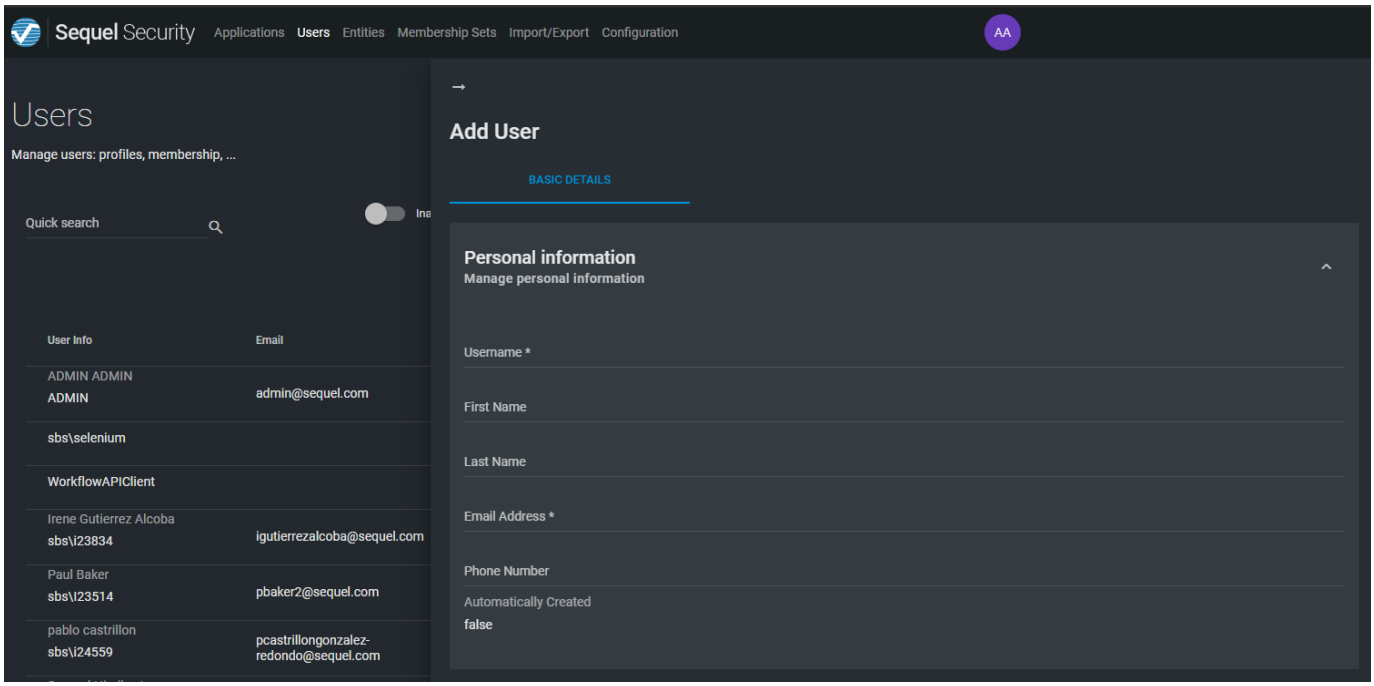
5.5.3 Create new users

From user's home page at `/Authorization/Users` click on *+ User* button. It is required to have create permissions in `Sec.User` securable.

Required information

PERSONAL INFORMATION

Fill in all **personal information** of the user:

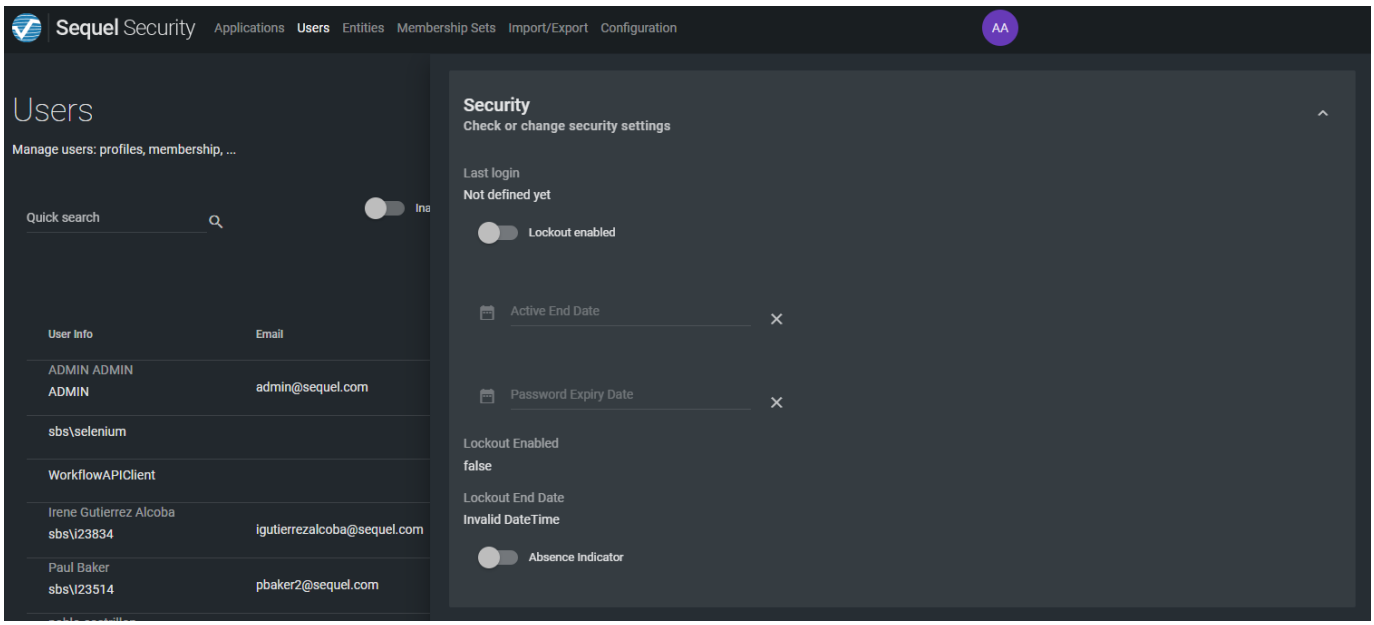


The *username* must be unique in the system and follow below rules:

- Valid characters:
  - Alphanumerics: uppercase A to Z and lowercase a to z latin letters, and digits 0 to 9.
  - Most non-alphanumerics: .@!'+-_#$%&*\=?^`{|}~;
  - White space.
  - Backslash is a valid character because it is allowed for supporting usernames in format `domain\username`; however, we do not recommend to use this format here. There is a separate field for storing these kind of user names (SsoUsername).
  - Email addresses: valid email addresses can also be used as usernames.
- Non-valid characters:
  - Some non-alphanumerics: /[]<>, these non-alphanumeric characters are not allowed as they are often used in URLs and can cause issues.
- Length: must be between 1 and 50 characters.

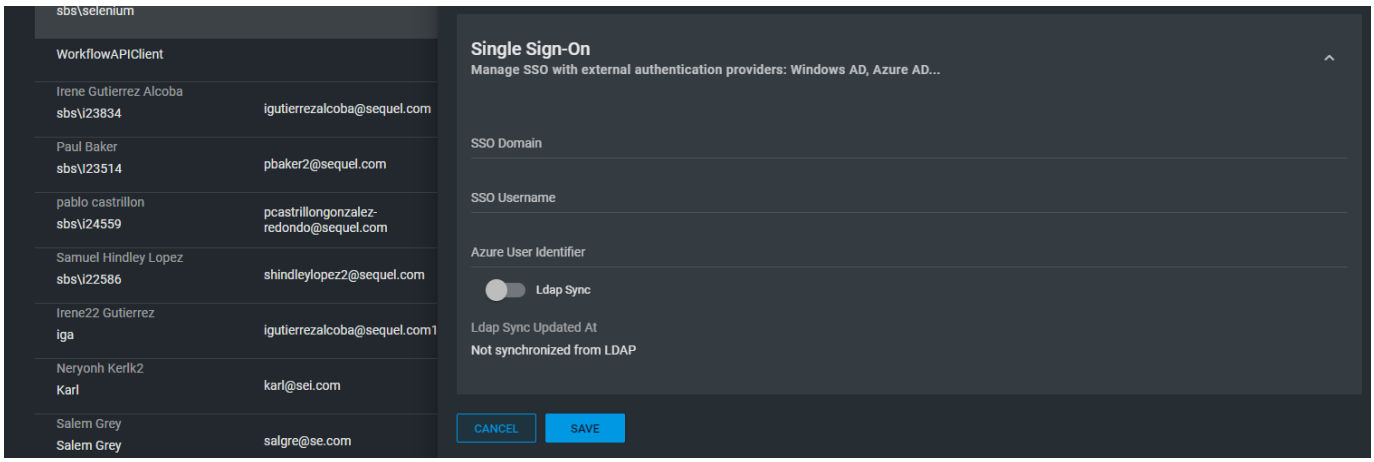
SECURITY

At **security section** is possible to configure important security aspect for the account like **password lockout** policy.



SINGLE SIGN-ON

Single sign-on section is required for configurations where SSO is enabled for Windows Authentication or other federation gateway, like Azure AD.



Membership and user types assignation

Once the user is created, it is not assigned to any group and role (membership) so the user will not able to access to any application. The next action is editing the user for assigning memberships and optionally user types.

Password setup

The last action for completing the creation of the user is setting up the password in case the user will be authenticated using the password-based authentication. In this case, there are two options:

- Administrator user forces to send the reset email from the contextual menu selecting this user, or
- User goes to login page an request to reset the password.

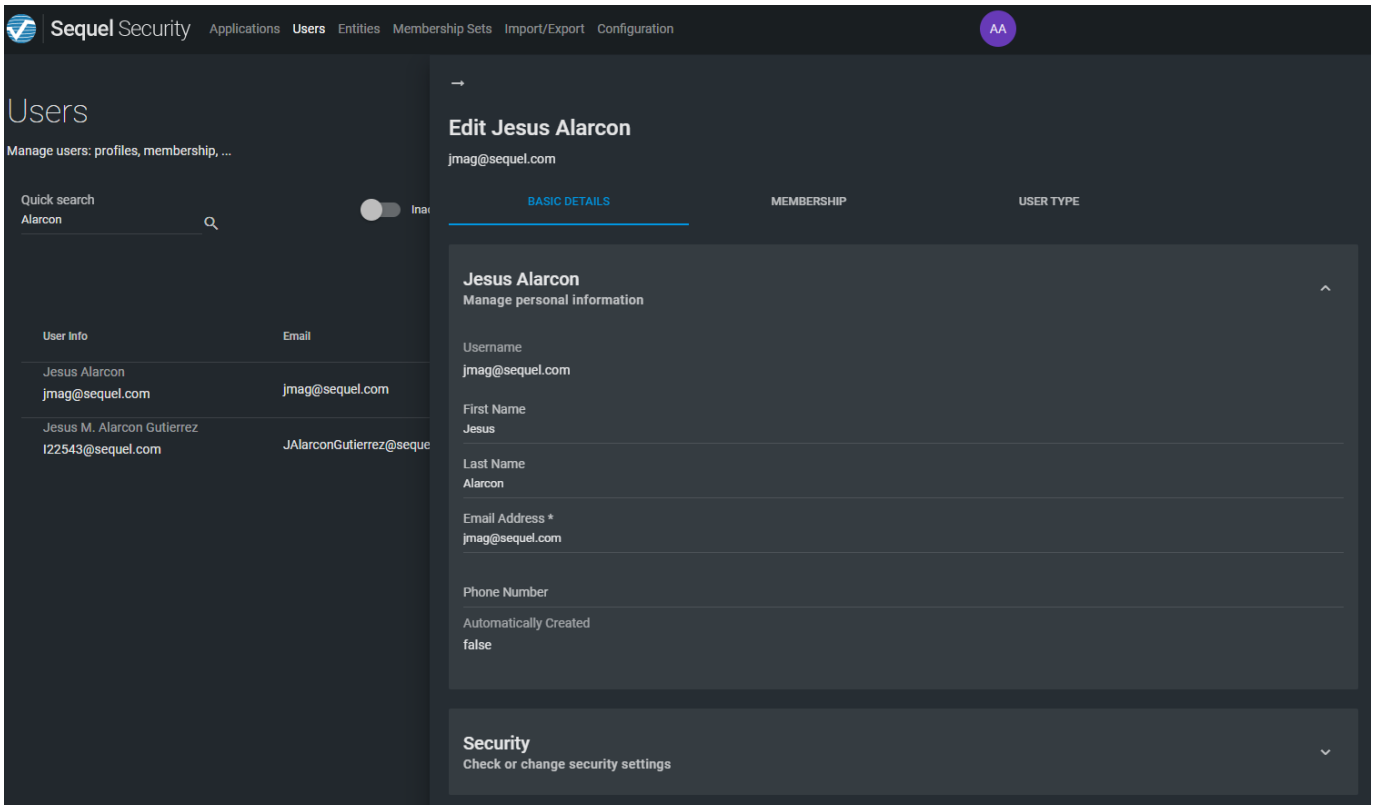
Password based section describes widely password management and creation, including policies and rules.

5.5.4 Edit users

Manage basic details

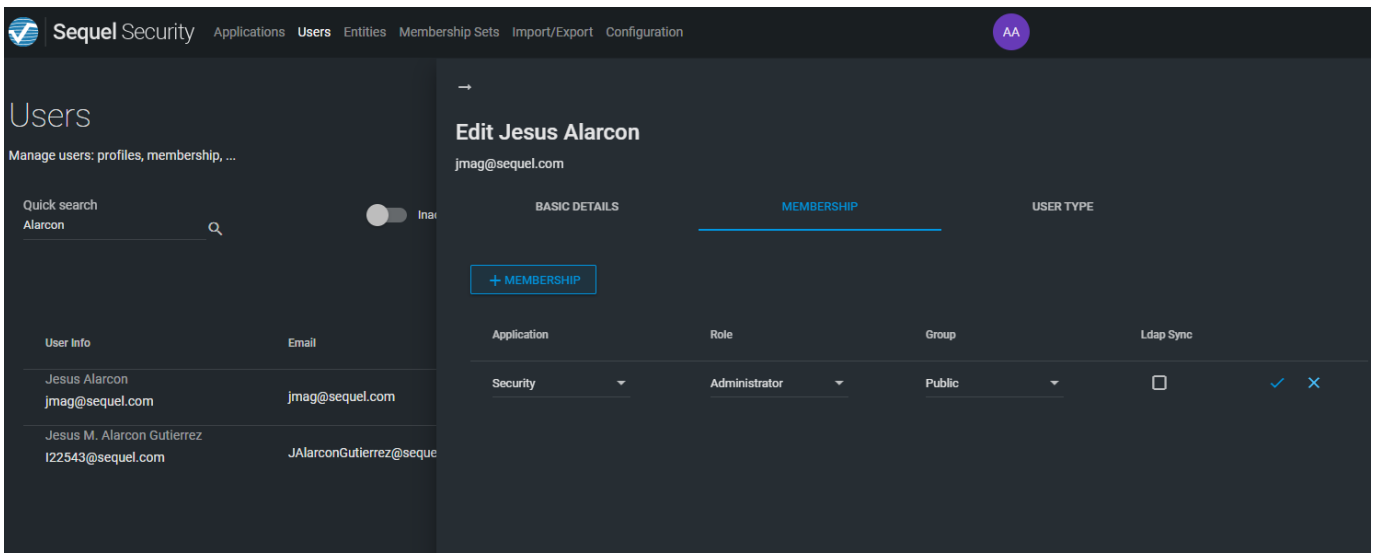
Click on *edit icon*.

Apply changes to basic *details tab* and press *save* for confirming changes.



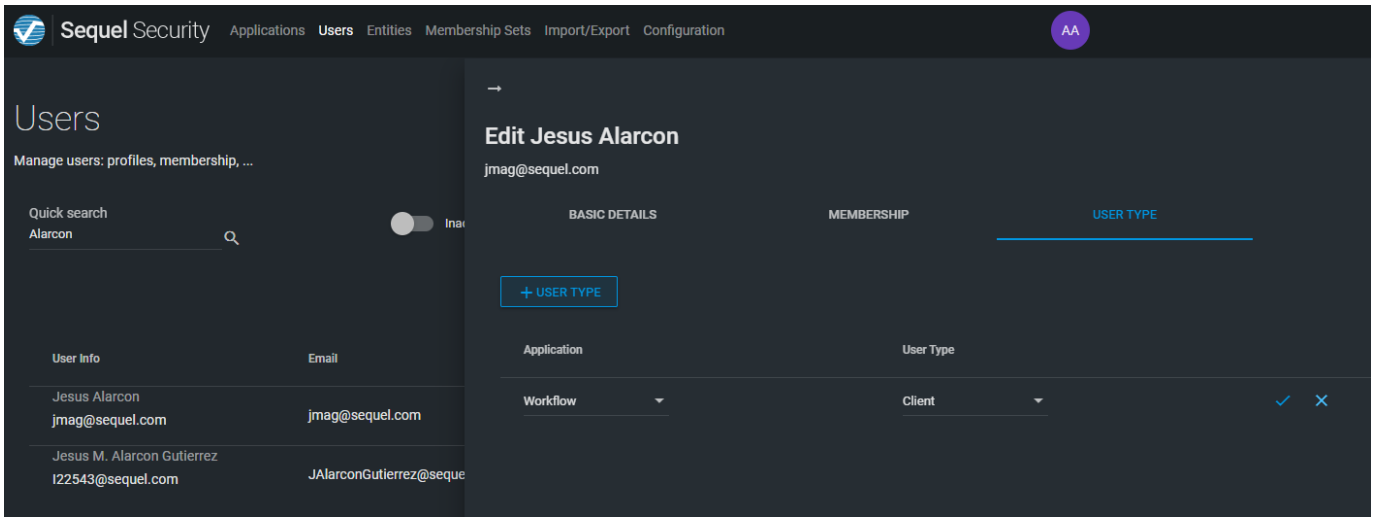
Manage memberships

At membership tab, you can add, edit or delete memberships. First select the application, and then a role and a group from this application. Data is saved each time a row is confirmed.



Manage user types

At user types tab, you can add, edit or delete user types. First select the application that the user types belongs and then the user type. Data is saved each time a row is confirmed.



5.5.5 Inactive and delete a user

When a user is no longer required there are two options:

- Set the user account inactive.
- Delete physically the user account.

There are some implications in both approaches

Inactive users

Declaring a user inactive starting from a date (`ActiveEndDateUtc`) will forbidden this account to access to the system; without removing all information related to this user.

A inactive user can be activated again resetting the `ActiveEndDateUtc` .

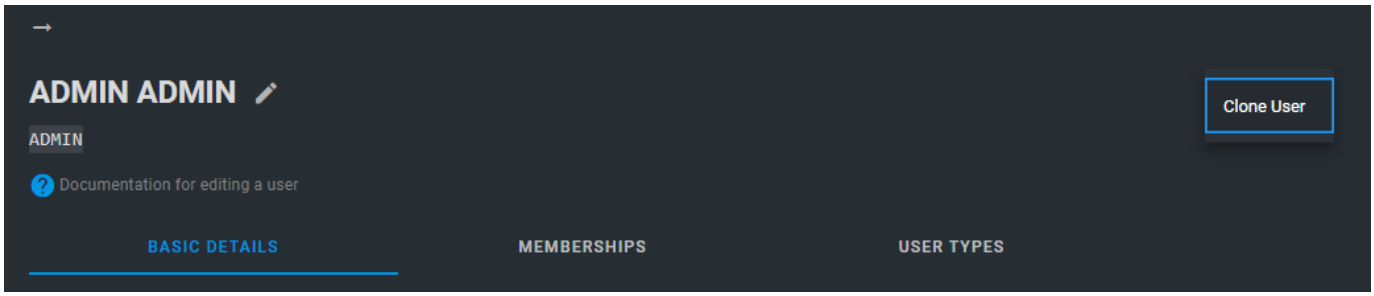
Deleted users

Deleting physically a user is possible, however we have to consider how this will affect to the downstream. As we could be referencing this user in many places.

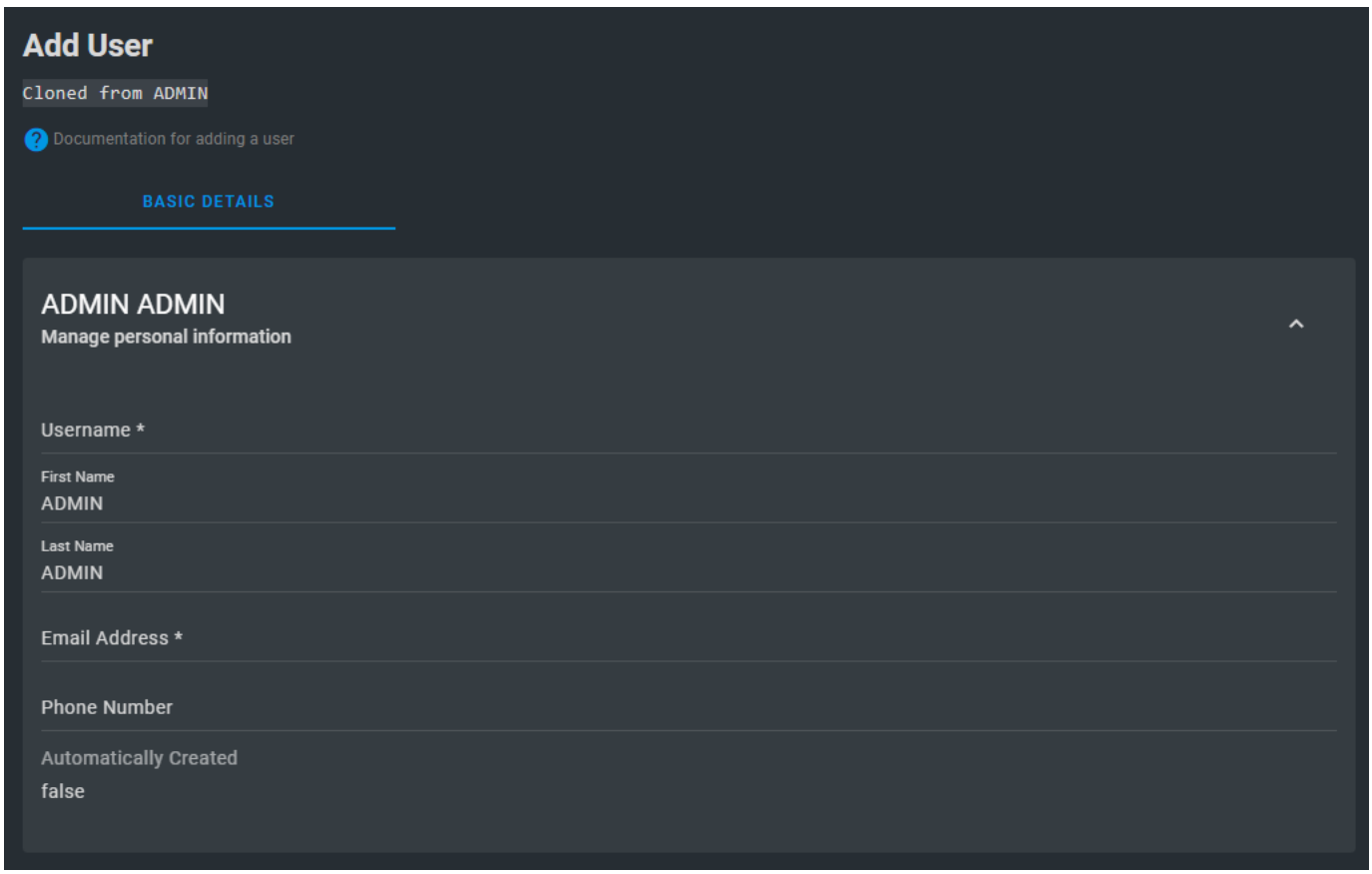
This option is not available from UI, it must be used directly calling to the API: `DELETE /Authorization/Users/{username}` .

5.5.6 Clone existing users

You can also clone existing users, you just have to click on the desired user to be cloned, go to the top right corner of the window and in the three dots menu select `Clone User`.

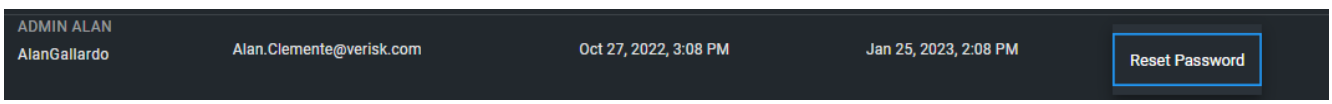


An exact copy of the user will appear, except for the user name, email address, password and other sensitive data like Single Sign On settings.



Warning

To set a new password you will have to reset it by clicking in `Reset Password` found in the three dots at the right side of the users list:



It will send you a new mail with a link in it to set the new password.

Info

Memberships and User Types will also be cloned but they will only appear after the new user is saved.

5.6 Data exchange

5.6.1 Data exchange

Security services offers different way and really flexible for exchanging security data. Please, keep in mind that restoring a security database must be done with care as there are settings in the database that contains URLs that depends on the environment. Our suggestion is to use the data exchange options described in this section.

Concept

Data exchange has been organised around two main concepts: *applications* and *domains*.

APPLICATIONS

We are already familiar with the **application** concept and applied to data exchange, all operations can be performed in the ambient of an application or globally. That means we can move all data or just data related to an application. This approach give as the required mechanism to keep application's configuration independent of other application at the same time that we manage all information in a central service.

DOMAINS

Domains organize the data in different areas: *Authorization*, *Authentication*, *Users*, *Entities*, *MembershipSets*. This separation of areas responds to different use case behind each domain:

- *Authorization*: the audience are DevOps teams and deployment teams. This is related to the pure installation of the application.
- *Authentication*: In the initial installation, this is installed with a Vanilla configuration. During configuration process, business analyst will be working with this domain (groups, roles, permissions,...)
- *Users*: It is not common to move users from one environment to other, but it is possible with this domain. This is really important for setting up automated testing environments.
- *Entities*: Allows to move entity configuration across different environments.
- *MembershipSet*: Allows to move membership sets configuration across different environments.

Clients

Data exchange functionality is offered in three different clients:

- **Administration UI**, that actually is calling to API.
- **API**, documented in the *Data Exchange* specification at Swagger in Security Rest API.
- **sequel-security tool**, a console tool that offers multiple operations for managing security configuration. This is the most powerful and complete client.

Operations

IMPORT AND EXPORT

Import and export security configuration. Not all operations, in purpose, are exposed with the same functionality depending the client. These commands are fully supported in the console tool; however in the API and UI is just possible to manage the *Authorization* domain.

Package format

The import/export package is a zip file that contains a predefined folder structure and json files. The structure is organised in applications and domains, as described below:

```
\Applications
            \{ApplicationKey}
                        \Application.json
                        \Authentication
```

```
                                        \ApiResource.json
                                        \Clients.json
                            \Authorization
                                        \Groups.json
                                        \Roles.json
                                        \Securable.json
                                        \userTypes.json
        \{ApplicationKey}
                            \Application.json
                            \Authentication
                                        \ApiResource.json
                                        \Clients.json
                            \Authorization
                                        \Groups.json
                                        \Roles.json
                                        \Securable.json
                                        \userTypes.json
```

This zip file can contain more than one application; however when installing only domains and applications passed in the request will be processed. When exporting, the folders structure will be followed even if the export parameters are just affecting one or two files.

SYNC

The **sync** operations helps to resynchronize a legacy security database with the latest security data. It is available in the console tool and also in the UI and API; however works in different ways.

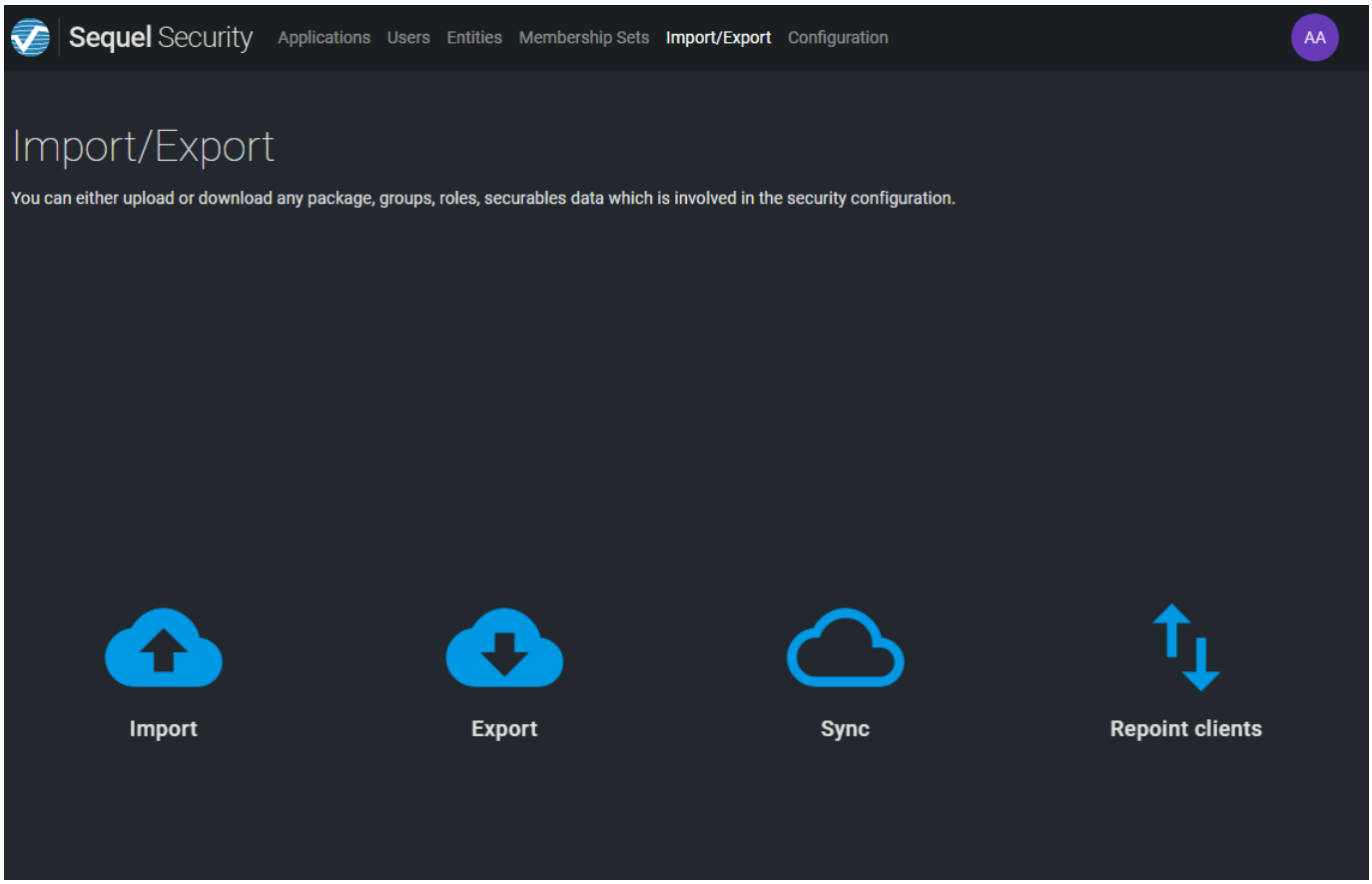
- **sequel-security**: the synchronization is done directly reading data from one database and storing data in the legacy database. This is useful for testing environments.
- **API** and **UI**: the synchronization here is just the act of starting the sync process by emitting a message to the bus for let legacy sync consumers know that a full sync is required.

REPOINT CLIENTS

Clients on security contains information that are specific of an environment: URLs and Origins. In some scenarios, like when restoring a database from production to a testing environment, it is required to add the URLs for the new environment. This option allows to do this task in a really simple way, based on providing the original pattern for selection the affected data and the new pattern (ie moving from `http://old-server-name/Auth` to `http://new-server-name/Auth`)

5.6.2 Data exchange at UI

The data exchange operations are available at `/administration/importexport`. Permissions for securable `Sec.DataExchange` are required.

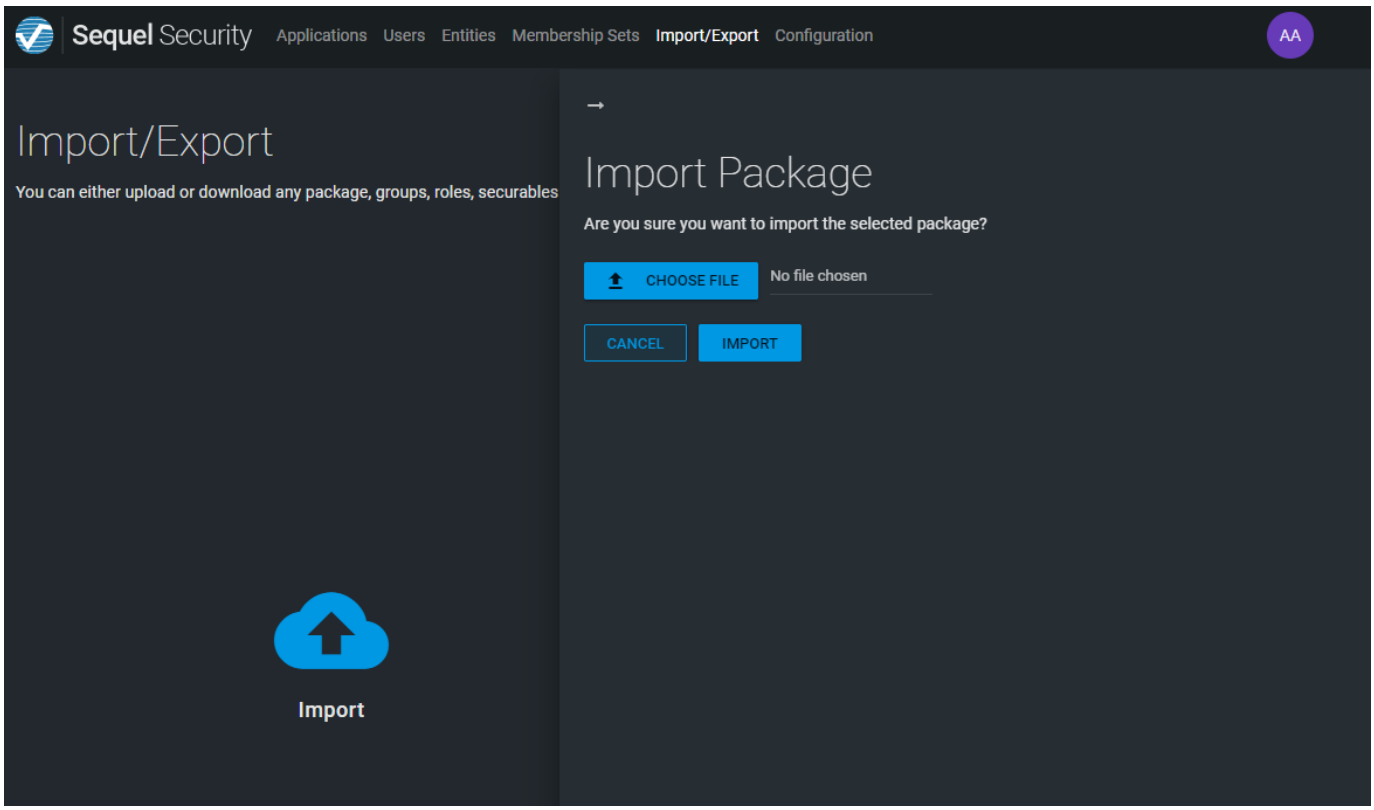


Import

Click on *Import* icon.

Select a file in zip format that contains a security package.

Click *Import* button.



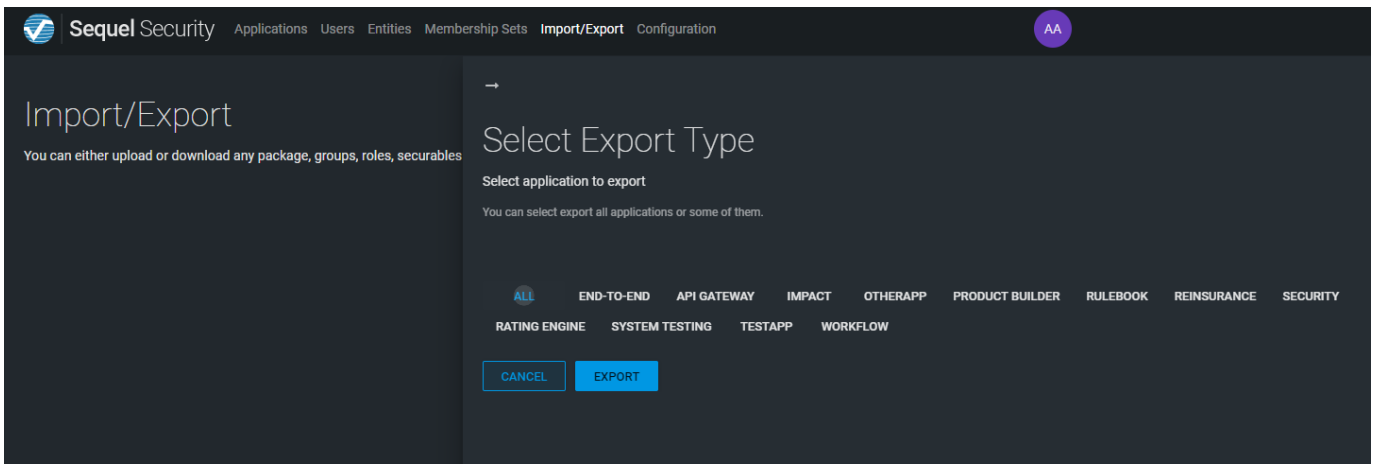
From UI the unique domain available is *authorization*.

Export

Click on *Export* icon.

Select *all* applications or one of the existing applications.

Click *Export* button.



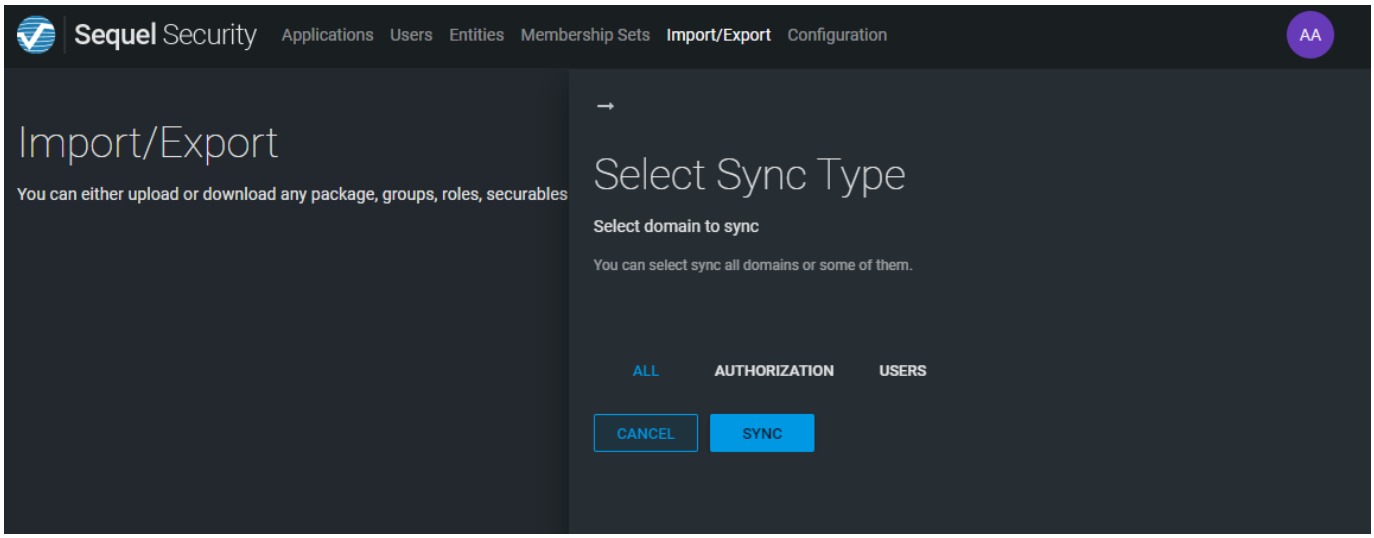
From UI the unique domain available is *authorization*.

Sync

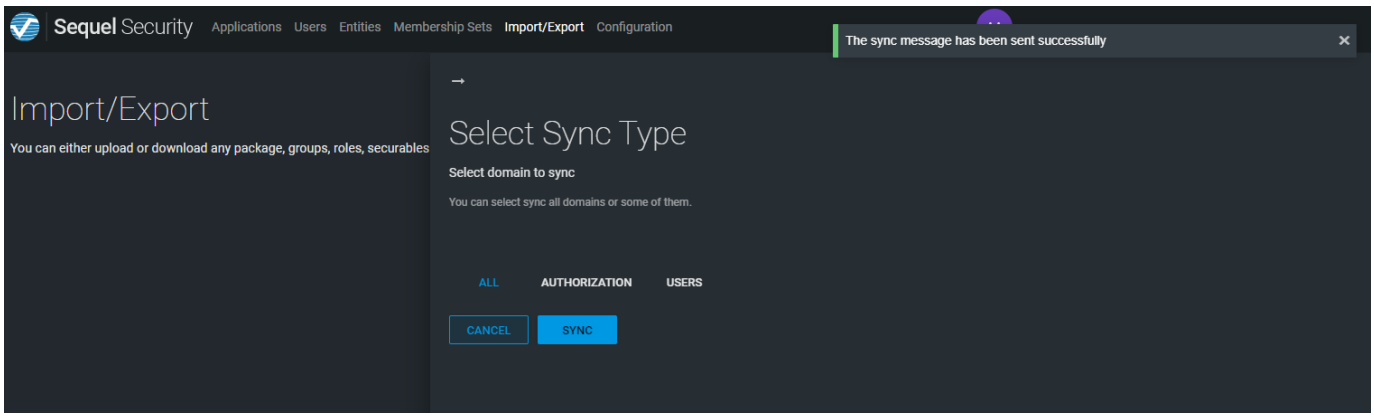
Click on *Sync* icon.

Select the domain to be synced: *authorization*, *users* or both (*all*).

Click on *Sync* button.



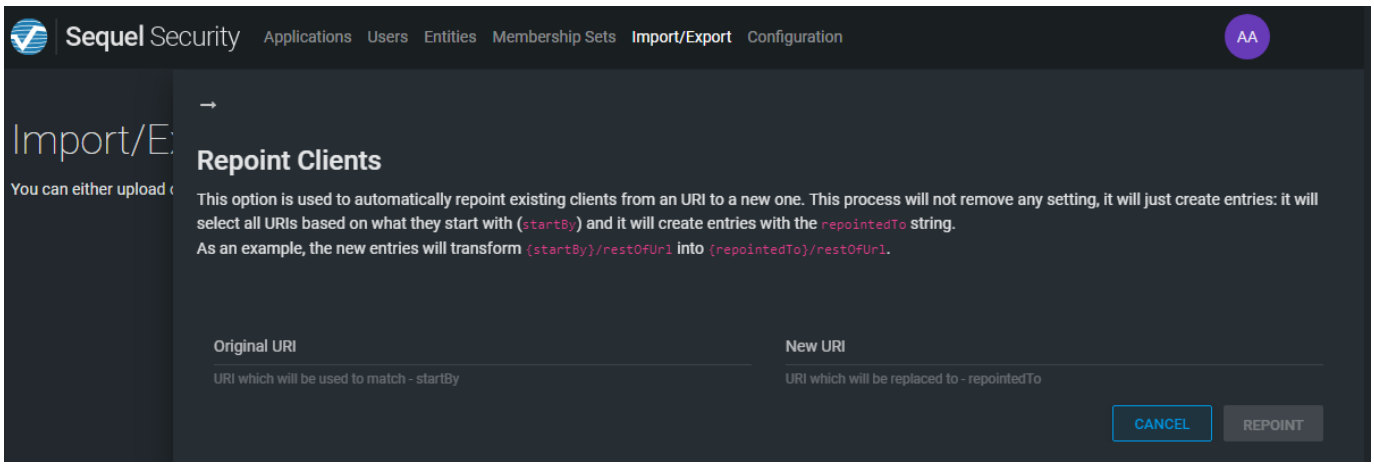
When the message is sent we receive confirmation. However, because sync depends on an external consumer we do not have confirmation that sync has been actually done.



Repoint client

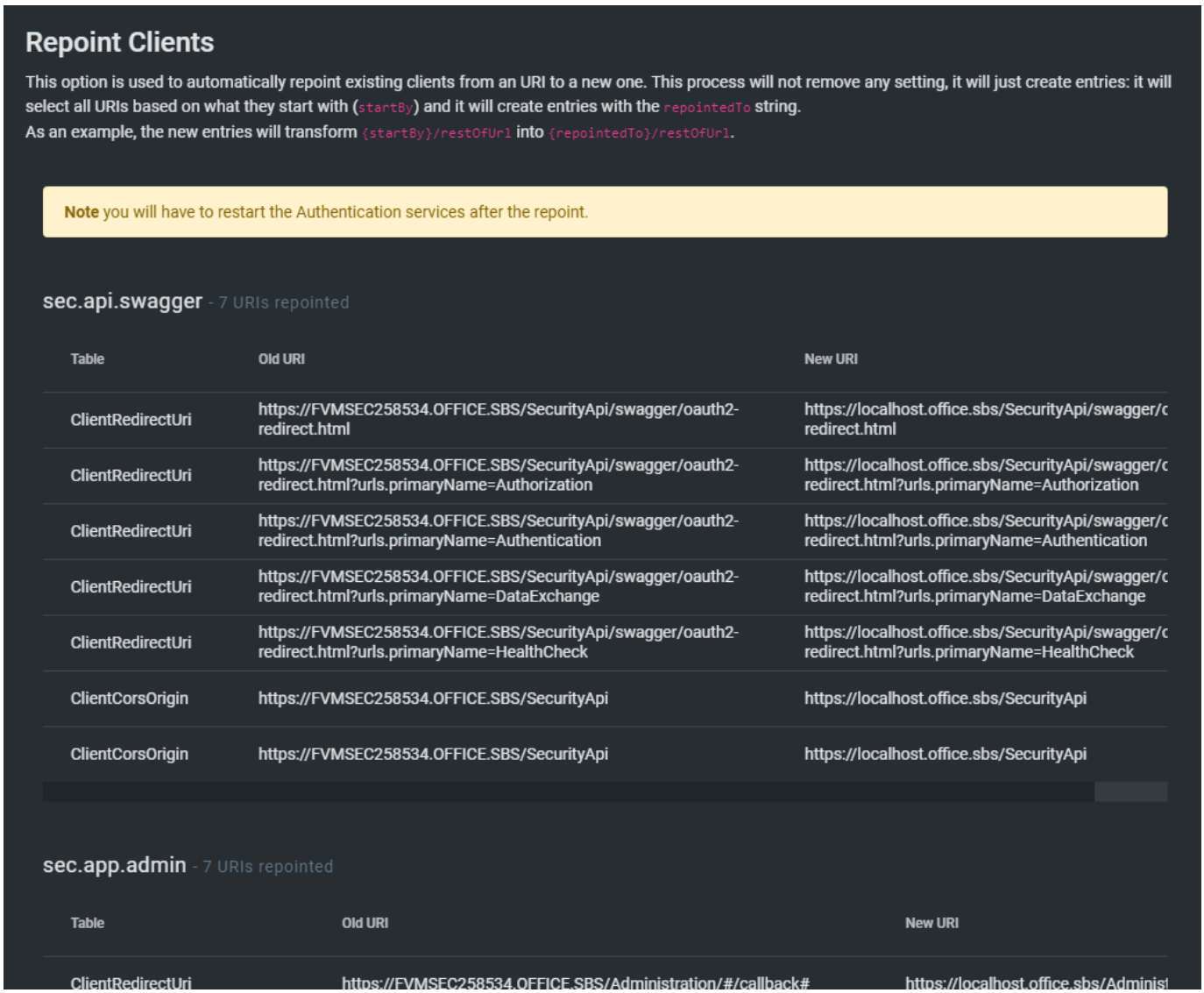
Click on *Repoint clients* icon.

Select the original and new URI. Click on `repoint`.



This option is used to automatically repoint existing clients from an URI to a new one. This process will not remove any setting, it will just create entries: it will select all URIs based on what they start with `startBy` and it will create entries with the `repointedTo` string. As an example, the new entries will transform `{startBy}/restOfUrl` into `{repointedTo}/restOfUrl`.

As result of the repoint, a summary with the changes applied will be displayed:

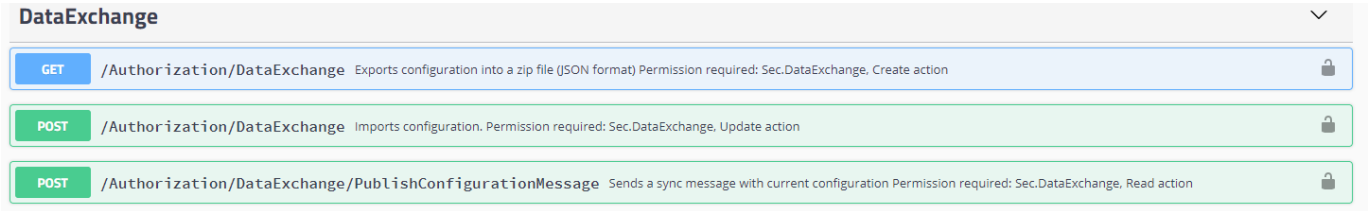


Authentication settings are cached

Keep in mind that changes to authentication settings are not reflected automatically to the Authentication services; so, **you will have to restart the Authentication services after the repoint**.

5.6.3 Data exchange at API

The data exchange operations are available at `/Authorization/DataExchange`. Permissions for securable `Sec.DataExchange` are required. The operation exposed are:



5.6.4 Sequel Security Tool

This document provides useful information about the `sequel-security` command line tool.

Installation

`sequel-security` is a global tool implemented using the Global Tools feature in .NET Core 6.0. The .NET Core 6.0 runtime must be installed for executing the tool.

For **production**, get the version of this tool (Sequel.Security\Packages\Tools\sequel-security\sequel-security.exe) from the same **Sequel.Deployment.Manager** you work with, for example:

```
\\buildoutput.office.sbs\drops\Security\stable\**version-number**\Sequel.Security\Packages\Tools\sequel-security\sequel-security.exe
```

The tool has to be executed with a user with enough permissions in the database for adding, deleting and updating data on security database.

Functionality

The `sequel-security` tool has been implemented to cover the following functionality:

- `import` data packages into the new **Sequel Security Service**, either migrated from a legacy system or from the new one.
- `export` data from the new **Sequel Security Service**.
- `add-admin-user` to the new **Sequel Security Service** and assign the membership: `SEC.CONFIGURATOR` (Role) / `SEC.PUBLIC` (Group).
- `sync` synchronizes a legacy security database with a Security database.
- `migrate` from legacy systems (Claims and Origin) to the new **Sequel Security Service**, so data migration packages can be generated with the tool and imported later into the new service.
- `migrate-validate` the compatibility from legacy systems to the new **Sequel Security Service**, and generates a report with the issues detected.
- `repoint-client` changes the *Redirect*, *Post Logout Redirect* and *CORS Origin* URIs from one value to another.
- `idsrv-license` commands for managing Identity Server licenses.
- `multitenancy` commands for manipulating tenants configuration.

Commands

After installing the global tool as described above, the easiest way to learn how to use it is by just running the following command from any location:

```
sequel-security -h
```

That command will show the help page with detailed information about each allowed command.

IMPORT

`Import` data packages into the new Sequel Security Service, either migrated from a legacy system or from the new one.

Required parameters:

```
    -d|--domain         /* Domain to be imported. Expected values:  authentication | authorization | users | entities | configuration */
    -a|--application    /* Valid when domain is equal to [authentication | authorization]. Application key used to filter data to be imported.*/
    -i|--input          /* Path to the folder or zip file that contains the package to be imported. */
    -c|--connection     /* Connection string to the target Security database. */
```

Optional parameters:

```
    -e|--entities               /* Entities to be imported (all by default) related to domain. Expected values (multiple allowed):
                                   [authentication] => clients | api-resources
                                   [authorization] => applications | securables | roles | groups | user-types | permissions | users | memberships | user-type-
assignments
                                   [users ] => users | memberships | user-type-assignments
                                   [entities] => entities | entityMemberships | entityUserTypeAssignments | entityUsers */
    -ieu|--includeEntityUsers   /* Valid when domain is equal to 'entities'. Import entities including EntityUsers */
```

```
    -tr|--tokenReplacement      /* Replacement of tokens values in the package before being imported (multiple allowed). Expected syntax: {token}{value} */
    -trf|--tokenReplacementFile /* Path to the text file that contains the tokens to replace in the package before being imported. Text file contains one line per
token with the expected syntax: {token}{value} */
    -tp|--tracedPackage         /* Path (folder or zip file) where the traced package will be exported. The trace package will contain updated entities (and tokens
resolved if some token replacement parameter defined too). */
    -v|--verbose                /* Activates verbose mode */
    -s|--silence                /* Execute tool in silence mode for executing in background tasks */
    -st|--strategy              /* Import strategy, options are merge (default) or replace.
                                   Merge is not deleting items in collections.
                                   Replace ensures collections at the system are equals to the definition on the import file.
                                   Only applies to User domain. */
```

Example 1: We want to import all Workflow authorization entities into a security database called SECURITYDB located at DBSERVER server from a ZIP file called Package.zip:

```
sequel-security import -a WF -d authorization -i Package.zip -c "Server=DBSERVER;Database=SECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true"
```

Example 2: We want to import Workflow membership entities into a security database called SECURITYDB located at DBSERVER server from a ZIP file called Package.zip:

```
sequel-security import -a WF -d users -e memberships -i Package.zip -c "Server=DBSERVER;Database=SECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true"
```

Example 3: We want to import all Workflow entities (entities, entityMemberships, entityUserTypeAssignments, entityUsers) into a security database called SECURITYDB located at DBSERVER server from a ZIP file called Package.zip:

```
sequel-security import -a WF -d entities -ieu -i Package.zip -c "Server=DBSERVER;Database=SECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true"
```

Example 4: We want to import Security authentication into a security database called SECURITYDB located at DBSERVER server from a ZIP file called Package.zip. The values of tokens such as **ClientSecretHash**, **SecurityApiUrlExternal**, **SecurityAdminUrlExternal** and **SecurityAuthorizationUrl** will be replaced in the package before being imported:

```
sequel-security import -a Sec -d authentication -i Package.zip -c "Server=DBSERVER;Database=SECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true" -tr
"{__ClientSecretHash__}{kx7uWQCFrSHtUXMs0jfZ83Ljf15QyxA6hLPKtFX1I08=}" -tr "{__SecurityApiUrlExternal__}{https://FVMSEC256333.OFFICE.SBS/SecurityApi" -tr
"{__SecurityAdminUrlExternal__}{https://FVMSEC256333.OFFICE.SBS/Administration" -tr "{__SecurityAuthorizationUrl__}{https://FVMSEC256333.OFFICE.SBS/Authorization}"
```

Example 5: We want to import Security authentication into a security database called SECURITYDB located at DBSERVER server from a ZIP file called Package.zip. Each line of the TEXT file called tokens.txt contains the values of tokens that will be replaced in the package before being imported:

```
sequel-security import -a Sec -d authentication -i Package.zip -c "Server=DBSERVER;Database=SECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true" -trf
tokens.txt
```

where a tokens.txt file sample is the following:

```
{__ClientSecretHash__}{kx7uWQCFrSHtUXMs0jfZ83Ljf15QyxA6hLPKtFX1I08=}
{__SecurityApiUrlExternal__}{https://littensec.office.sbs/SecurityApi}
{__SecurityAdminUrlExternal__}{https://littensec.office.sbs/Administration}
{__SecurityAuthorizationUrl__}{https://littenauth.office.sbs/Authorization}
```

Example 6: We want to import Security authentication into a security database called SECURITYDB located at DBSERVER server from a ZIP file called Package.zip. At the same time we want to generate a traced package in a ZIP file called TracedPackage.zip with updated entities and replaced tokens.

```
sequel-security import -a Sec -d authentication -i Package.zip -c "Server=DBSERVER;Database=SECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true" -trf
tokens.txt -tp TracedPackage.zip
```

EXPORT

`Export` data from the new Sequel Security Service.

Required parameters:

```
    -c|--connection     /* Connection string to the source Security database. */
    -o|--output         /* Output path (folder or zip file) where the package will be exported. */
```

Optional parameters:

```
    -d|--domain                 /* Domain to be exported (all by default). Expected values (multiple allowed): authorization | authentication | users | entities |
configuration */
```

```
    -ieu|--includeEntityUsers   /* Valid when domain is equal to 'entities'. Export entities including EntityUsers */
    -a|--application            /* Valid when domain is equal to [authentication | authorization]. Application key used to filter data to be imported. If not
specified then data for all the applications in the input will be exported. */
    -v|--verbose                /* Activates verbose mode */
    -s|--silence                /* Execute tool in silence mode for executing in background tasks */
```

Example 1: We want to export all application with all entities from a security database called SECURITYDB located at DBSERVER server and store it in a ZIP file called AllApplicationsExport.zip:

```
sequel-security export -o AllApplicationsExport.zip -c "Server=DBSERVER;Database=SECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true" -ieu
```

Example 2: We want to export only Workflow and Security entities related to authentication from a database called SECURITYDB located at DBSERVER server and store it in a folder called WF_Sec_AuthenticationExport:

```
sequel-security export -a WF -a Sec -d authentication -o WF_Sec_AuthenticationExport -c
"Server=DBSERVER;Database=SECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true"
```

Example 3: We want to export only Workflow and Security entities (entities, entityMemberships, entityUserTypeAssignments) without entityUsers from a database called SECURITYDB located at DBSERVER server and store it in a folder called WF_Sec_AuthenticationExport:

```
sequel-security export -a WF -a Sec -d entities -o WF_Sec_AuthenticationExport -c
"Server=DBSERVER;Database=SECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true"
```

ADD-ADMIN-USER

`add-admin-user` command adds an administrator user to the new Sequel Security Service and assign the membership: `SEC.CONFIGURATOR` (Role) / `SEC.PUBLIC` (Group). If the user already exists, then the password, email and membership are updated.

Required parameters:

```
    -n|--name           /* Admin user username.*/
    -m|--mail           /* Admin user email address. */
    -p|--password       /* Admin user password. */
    -c|--connection     /* Connection string to the target Security database. */
```

Optional parameters:

```
    -s|--silence        /* Activate silent mode */
```

Example: We want to add the admin user named ADMIN_NAME whose email address is ADMIN_MAIL and the password is ADMIN_PASSWORD into a security database called SECURITYDB located at DBSERVER server:

```
sequel-security add-admin-user -n ADMIN_NAME -m ADMIN_MAIL -p ADMIN_PASSWORD -c
"Server=DBSERVER;Database=SECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true"
```

SYNC

`sync` command synchronizes a legacy security database with a Security database.

Required parameters:

```
    -c|--connection            /* Connection string to the source Security database. */
    -lc|--legacyconnection     /* Connection string to the destination database with the legacy [security] schema. */
```

Optional parameters:

```
    -d|--domain                /* Domain to be synchronized (authorization and users by default). Expected value: authorization | users*/
    -dp|--deletepolicy         /* Delete policy applied during synchronization process (PhysicalThenLogical by default). Expected value: Logical | Physical |
PhysicalThenLogical
    -v|--verbose               /* Activates verbose mode */
    -s|--silence               /* Execute tool in silence mode for executing in background tasks */
```

Example 1: We want to synchronize all applications with their authorization entities and users from a security database called SECURITYDB located at DBSERVER server into a legacy security database LEGACYSECURITYDB located and DBSERVER:

```
sequel-security sync -c "Server=DBSERVER;Database=SECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true" -lc
"Server=DBSERVER;Database=LEGACYSECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true"
```

Example 2: We want to synchronize all users from a security database called SECURITYDB located at DBSERVER server into a legacy security database LEGACYSECURITYDB located and DBSERVER using Logical deletion:

```
sequel-security sync -d users -c "Server=DBSERVER;Database=SECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true" -lc
"Server=DBSERVER;Database=LEGACYSECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true" -dp Logical
```

MIGRATE

`migrate` from legacy systems (Claims and Origin) to the new Sequel Security Service, so data migration packages can be generated with the tool and imported later into the new service.

Required parameters:

```
    -c|--connection     /* Connection string to the database with the legacy [security] schema to be migrated. */
    -o|--output         /* Output path (folder or zip file) where the migration package will be generated. */
```

Optional parameters:

```
    -a|--application    /* Application key to choose from WF, Sec, Origin, PB, CLM (by default all of them) to be included in the migration package. (multiple
allowed)*/
    -d|--domain         /* Domain to be exported (all by default). Expected values (multiple allowed): authorization | users */
    -val|--validate     /* Validates package before creates output file. */
    -v|--verbose        /* Activates verbose mode */
```

Example 1: We want to migrate all Workflow entities from a legacy database called LEGACYDB located at DBSERVER server and store it in a ZIP file called WorkflowMigration.zip:

```
sequel-security migrate -a WF -o WorkflowMigration.zip -c "Server=DBSERVER;Database=LEGACYDB;Trusted_Connection=True;MultipleActiveResultSets=true"
```

Example 2: We want to migrate only Workflow entities related to users from a legacy database called LEGACYDB located at DBSERVER server and store it in a folder called WorkflowMigrationOnlyUsers:

```
sequel-security migrate -a WF -d users -o WorkflowMigrationOnlyUsers -c "Server=DBSERVER;Database=LEGACYDB;Trusted_Connection=True;MultipleActiveResultSets=true"
```

MIGRATE-VALIDATE

`migrate-validate` the compatibility from legacy systems to the new Sequel Security Service, and generates a report with the issues detected.

Required parameters:

```
    -c|--connection     /* Connection string to the database with the legacy [security] schema to be validated. */
```

Optional parameters:

```
    -r|--references     /* JSON filename with references to the legacy database columns of other applications (WF, PB, Origin, CLM) with security data which will be
validated too. */
    -v|--verbose        /* Activates verbose mode */
    -s|--silence        /* Activate silent mode */
```

JSON file format is described in below sample:

```
{
  "Groups": [
    {
      "DbSchema": "workflow",
      "DbTable": "DiaryAction",
      "DbColumn": "assignedGroupId"
    },
    {
      "DbSchema": "workflow",
      "DbTable": "DiaryActionAcls",
      "DbColumn": "GroupId"
    }
  ],
  "Roles": [],
  "Securables": [
    {
      "DbSchema": "workflow",
      "DbTable": "DiaryActionType",
      "DbColumn": "securableId"
    }
  ],
  "Users": [
```

```
    {
      "DbSchema": "workflow",
      "DbTable": "Alert",
      "DbColumn": "userId"
    }
  ],
  "UserTypes": []
}
```

Example 1: We want to validate the users and authorization entities of all applications from a legacy database called LEGACYDB located at DBSERVER server:

```
sequel-security migrate-validate -c "Server=DBSERVER;Database=LEGACYDB;Trusted_Connection=True;MultipleActiveResultSets=true"
```

Example 2: We want to validate the users and authorization entities of all applications **plus** a list of columns with security data defined in a JSON filename called SECURITY_COLUMNS from a legacy database called LEGACYDB located at DBSERVER server:

```
sequel-security migrate -c "Server=DBSERVER;Database=LEGACYDB;Trusted_Connection=True;MultipleActiveResultSets=true" -r "SECURITY_COLUMNS"
```

REPOINT-CLIENT

`repoint-client` repoints one or more Clients from old URIs to new URIs after a database restore or URL changes in a environment. The match for old URIs is done from the start of them (`https://example.com`). The CORS Origins will also be replaced if the Origins does not contain a path but the old URI contains one (Origins should not have a path in them).

Arguments:

```
  CURRENT_VALUE  Start value from URIs that are going to be replaced
  NEW_VALUE      Start value from URIs that will point to after repoint
```

Options:

```
  -c|--connection <CONNECTION_STRING>  [REQUIRED] Connection string to the source Security database
  -f|--fail                            [OPTIONAL] If set, the command will fail if no clients are repointed
```

Example: You have restored a database from a client and to make it work in a support environment, these URIs must be repointed. The client has all products under the URI `https://client.example.com` , and now they will point to `https://secclientsuppenv.office.sbs` :

```
sequel-security repoint-client -c 'Server=DBSERVER;Database=SECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true' 'https://client.example.com' 'https://
secclientsuppenv.office.sbs'
```

NORMALIZE-CORS

`normalize-cors` normalizes CORS origins stored in database to match the pattern: `protocol://host:port` . Invalid and duplicated values will be removed.

Options:

```
  -c|--connection <CONNECTION_STRING>  [REQUIRED] Connection string to the source Security database
```

Example:

```
sequel-security normalize-cors -c 'Server=DBSERVER;Database=SECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true'
```

ADD-CONFIG-SETTING

`add-config-setting` command adds (or update if exists a setting with same key) a new configuration setting.

Arguments:

```
  KEY       Configuration setting key
  VALUE     Configuration setting value
  CATEGORY  [OPTIONAL] Configuration setting category
```

Options:

```
  -c|--connection <CONNECTION_STRING>  [REQUIRED] Connection string to the source Security database
```

Example: Add/Update subject for forgot password email:

```
sequel-security add-config-setting 'ForgotPasswordEmailSubjectTemplate' 'You have requested a password reset!' 'ForgotPasswordEmailTemplate' -c
'Server=DBSERVER;Database=SECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true'
```

IDSRV-LICENSE

`idsrv-license` show a set of commands for managing Identity Server licenses.

```
idsrv-license <COMMAND>
```

add-license

`add-license` command adds (or update if exists a license with same key) an Identity Server license.

Arguments:

```
  KEY       License key
  VALUE     License value
  PRIORITY  [OPTIONAL] Priority for the license if several licenses are valid for the   same interval of time
```

Options:

```
  -c|--connection <CONNECTION_STRING>  [REQUIRED] Connection string to the source Security database
```

Example: Add/Update license:

```
sequel-security idsrv-license add-license 'licenseKey' 'LicenseValueJWT' -c
'Server=DBSERVER;Database=SECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true'
```

del-license

`del-license` command deletes an Identity Server license.

Arguments:

```
  KEY       License key
```

Options:

```
  -c|--connection <CONNECTION_STRING>  [REQUIRED] Connection string to the source Security database
```

Example: Delete *MyLicenseKey* license:

```
sequel-security idsrv-license del-license 'MyLicenseKey' -c 'Server=DBSERVER;Database=SECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true'
```

list-license

`list-license` command list available Identity Server licenses.

Options:

```
  -c|--connection <CONNECTION_STRING>  [REQUIRED] Connection string to the source Security database
```

Example: :

```
sequel-security idsrv-license list-license -c 'Server=DBSERVER;Database=SECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true'
```

get-license-ring

`get-license-ring` Gets the Identity Server licenses ring for current configuration.

Options:

```
  -c|--connection <CONNECTION_STRING>  [REQUIRED] Connection string to the source Security database
```

Example: :

```
sequel-security idsrv-license get-license-ring -c 'Server=DBSERVER;Database=SECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true'
```

get-license-info

`get-license-info` Gets the Identity Server info info for current configuration.

Options:

```
  -c|--connection <CONNECTION_STRING>  [REQUIRED] Connection string to the source Security database
```

Example: :

```
sequel-security idsrv-license get-license-info -c 'Server=DBSERVER;Database=SECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true'
```

get-license-settings

`get-license-setting` Gets the Identity Server license settings.

Options:

```
  -c|--connection <CONNECTION_STRING>  [REQUIRED] Connection string to the source Security database
```

Example: :

```
sequel-security idsrv-license get-license-settings -c 'Server=DBSERVER;Database=SECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true'
```

set-license-settings

`set-license-license` command sets Identity Server license settings.

Arguments:

```
  VALIDATION_MODE      License validation mode.                                  Allowed values are: Complete, PayloadOnly
  REMAINING_DAYS       Number of days before show license expiration warning
```

Validation mode must be set to *Complete. Payload Only* mode is only for testing.

Options:

```
  -c|--connection <CONNECTION_STRING>  [REQUIRED] Connection string to the source Security database
```

Example: Sets Identity Server license settings with Complete validation mode and remaining days to 30:

```
sequel-security idsrv-license set-license-settings Complete 30 -c 'Server=DBSERVER;Database=SECURITYDB;Trusted_Connection=True;MultipleActiveResultSets=true'
```

MULTITENANCY

`multitenancy` subcommands serves to operate the tenants configuration with ease from the tool.

configure

```
Usage: sequel-security multitenancy configure [options]

Options:
  -c|--connection <CONNECTION_STRING>                      [REQUIRED] Connection String to the Multitenancy database.

  -tk|--tenant-key <TENANT_KEY>                            [OPTIONAL] The key of the tenant (by default is
                                                           "Default").
                                                           Default value is: Default.
  -tn|--tenant-name <TENANT_NAME>                          [OPTIONAL] The name of the tenant (defaults to the key).
```

```
-tcs|--tenant-connection-string <TENANT_CONNECTION_STRING>  [REQUIRED] The connection string of the tenant.
-h|--help                                                  Show help information.
```

Configures a tenant with a name and connection string. It can also be used to configure the default tenant but just providing the tenant connection string.

6. Release notes

6.1 Current version

Release Notes	
Description of changes	See **Release Notes** section for major changes and **Changelog** for a detailed report of minor improvements and bug fixes.
Environment	See *Platform Specs* document
Installation / Running Instructions	See *Platform Specs* and *Installation Guide* documents
Issues	See *Troubleshooting guide* for common issues during deployment and at runtime.

6.1.1 Release Notes

.NET Core 6.0 App

v3.0

In this version Sequel Security's platform specs have been migrated from .NET Core 3.1 to .NET Core 6.0. Also, EntityFramework has been updated from 3.1.13 to 6.0 and IdentityServer4 has been upgraded to Duende IdentityServer v6.

.NET Core 3.1 App

v2.0

In this version Sequel Security's platform specs have been migrated from .NET Core 2.1 to .NET Core 3.1. Also, EntityFramework has been updated from 2.0 to 3.1.13.

Role-level authorization sync with Active Directory

v1.40

For a large enterprise using Sequel's applications and Windows Active Directory (AD), managing Authentication (AuthN) and Authorization (AuthZ) directly from AD instead of through the Sequel application would reduce a lot of extra work. While AuthN is already supported; AuthZ is not. The new feature helps to reduce the manual administration required to manage authorization in the system. The solution is based on two main functionalities: *User synchronization from Windows Authentication to Sequel Security Services* and *synchronization of Windows User Groups (AD groups) to Sequel's Security Memberships*, using the new *membershipset* model. More information can be found at the product's handbook.

Health Check Endpoint

v1.36, v1.38

Security services expose HTTP endpoints for reporting the health of app infrastructure components. Different levels of information are provided based on permissions. More information on the *troubleshooting guide*.

6.1.2 Breaking Changes notes

This section will include breaking changes introduced.

V3.125.23080.1

As part of PBI 483843 (Remove requirement of Hash Routing on URL) the URL Rewrite module is required for IIS. Please review the platform specs for further information.

V3.0

Since v3 the NET6 runtime is required. Please review the platform specs to make sure the environment is using the latest recommended version of the .NET Core 6. Please note that the testing performed for this upgrade was made from a v2.x version to a v3.x version.

V2.0

- All of Sequel Security applications have been migrated to use .NET Core 3.1 and EntityFramework 3.1.13. Due to this we highly recommended doing the following when updating to v2.0 of Security:
  - Backup the current Security database
  - Export the current configuration with the `sequel-security` tool
  - Review the platform specs to make sure the environment is using the latest recommended version of the .NET Core 3.1.
  - Begin with the update of the databases with the Deployment Manager first, as there are a lot of schema changes that need to be done first before updating the applications.

V1.46.20072.01

- The administration user created by Deployment Manager is part of the database installation. This will break compatibilities with previous environment.json files used to install the application.

6.1.3 Changelog

3.129.23132.1 - 12 MAY 2023

Bug fixes

- 493455 - EKS deployment generates a wrong URL
- 492854 - User import is not removing item collections not defined on the import file
- 494385 - Azure AD Sync - All users are made inactive after synchronization

3.127.23115.1 - 25 APRIL 2023

Improvements

- 479794 Invalid CORS detection
- 481567 Username should support standard UPN lengths when integrated with Azure
- 491173 Username should support standard UPN lengths when integrated with Okta
- 491174 Username should support standard UPN lengths when integrated with Jumpcloud
- 483353 Define Issuer on discovery endpoint as defined on RFC
- 485138 Data Archiving - Logs, Traces and metrics
- 487756 Okta AuthN - Allow to match by Okta User Name

Bug fixes

- 487185 Admin - Entities freezes after failing and is unable to recover
- 490055 Changes when an user is disabled as part of AzureAD Sync are not being notified
- 490112 Release for AWS ECS creates wrong secrets
- 491974 Generated ZIP from Security tool uses backslash separators (not Linux friendly)
- 491495 Admin - List of users is not refreshed when creating a new user
- 492590 sequel-security tool shouldn't apply application filtering to memberships and user types when importing ALL users
- 489214 jaeger storage memory issue: do not track healthcheck calls

3.125.23080.1 - 21 MARCH 2023

Improvements

- As part of the Database deployment with EF Migrations:
- 411073 Phase 1 - All schemas w/o audit triggers
- 483867 Phase 2 - Add audit triggers to Sequel's schemas
- 483868 Phase 3 - Add audit triggers to IdentityServer's schema
- 483869 Phase 4 - Deployment: new and upgrades
- 476459 Improve appsettings read mechanism in Kubernetes
- 477132 AD Sync with Azure AD MsGraph - Frontend
- 478985 Expose at Admin site the Import/Export Memberships(Set)
- 483843 Remove requirement of Hash Routing on URL
- 483870 Configuration deployment on EKS
- 483962 Single tenant deployment (self)
- 484038 Azure AD Sync - Ability to select users by group's display name too
- 485711 Refactor all forms in Administration

Bug fixes

- 483919 Email validation is only done by the front-end (ZD136062)
- 484208 Membership Sets - 'SyncMatchById' & 'SyncMatchByDisplayName' fields are not exported in config
- 484209 Synchronization - Improvement in the recovery of Azure AD when incorporating incorrect mappings

3.122.23026.1 - 26 JANUARY 2023

Improvements

- 471850 Vulnerability detection & SCA pipelines
- 479733 Vulnerability checks - Triage and win-win fixes
- 476499 Expose message bus public contracts on a NuGet package
- 476605 QA - Update to the latest stable version of Selenium
- 477667 Endpoint with groups that belong an user
- 472649 QA - API tests to cover AuthZ cache
- 468153 Tech Upgrade - Security Services - IdentityServer License management - Basic
- 468168 Tech Upgrade - Security Services - IdentityServer License management - UI
- 412341 Tech Upgrade - Security sync service in AWS Serverless

As part of EKS work made for UW:

- 471540 Conflict when performing a release
- 472728 UW Integration - Independent deployment of Workflow (based on EKS) - Security Deployment

Bug Fixes

- 479803 Admin UI - Redirect URIs and Post-logout URIs are not stored
- 471491 Admin UI - Console Error when access to Reset Email Template
- 473826 Admin UI - Entities - Unable to assign or unassign a membership to a user
- 473979 Admin UI - Import/Export - Blank page when accessing Repoint clients
- 476170 Authentication - Start-up fails if 'IdentityServerLicenseSettings' setting is not found

3.116.22322.1 - 18 NOVEMBER 2022

Improvements

- 412283 Ability to import/export security configuration table
- 460610 Create users based on existing users
- 432684 Jumpcloud integration
- 464472 Add Telemetry settings to Release

Bug fixes

- 467223 Azure - Find user with empty MatchingFields

3.115.22300.1 - 27 OCTOBER 2022

Improvements

- 354565 Okta integration

Bug fixes

- Bug 464527: SEC - Replayable Password Reset Code
- Updated AWS Dynatrace tags for use with Sequel HUB due to `Bug 466151: HUB - Cloudformation fails with current tags for dyna`

3.114.22279.1 - 6 OCTOBER 2022

Please, review breaking changes section.

Features

- 376755 Security upgrade: NET6 & IdentityServer v6

Improvements

- 455891 Telemetry and Logs enhancements
- 460611 Create clients based on existing clients

Bug fixes

- 463105 User creation with spaces on start or end " username "
- 462677 Email validation prevent users emails addresses like name.surname@company.XXXXX (5 digit extensions)

2.109.22223.1 - 11 AUGUST 2022

Improvements

- 265031 Clients - Improve validations in Admin UI

Bug fixes

- 455264 Unexpected error when publishing ConfigurationChangedMessage for synchronization

2.108.22208.1 - 27 JULY 2022

Improvements

- 453118 Improve validations on import command
- 365062 sequel-security tool detects duplicate entries when importing configuration

- 403802 Node v16
- Upgraded Sequel.Core.MessageBus and Sequel.Core.AppBuilder to latest versions due to `Bug 443541: BUS - Different scopes do not allow to debug consumers`
- Upgraded Sequel.Core.Logging to latest versions and added ability to write logs to Console (e.g. can be used in AWS CloudWatch) or MsSql `Product Backlog Item 411104: LOG - Stop using database for logging on AWS environments`

Bug fixes

- 445063 Security upgrade does not migrate ApiResourceScopes
- 440911 sequel-security tool import fails with closed connection

2.104.22143.1 - 23 MAY 2022

Improvements

- 431820 Add support to Authentication for Microsoft + DUO
- 430882 Extract sequel-security tool into independent artifact

Bug fixes

- 436990 Admin - Configuration - Auth Fed GWY - Customers are not deleted
- 437061 Reset Password - Email field does not show entire email address

2.101.22111.1 - 21 APRIL 2022

Improvements

- 393213 UserSessionAvatar - SQ.sso cookie validation
- 413309 Allow to install Security Sync and LDAP Sync on the same Windows Server
- 415627 Rebranding Verisk - App

Bug fixes

- 397927 AD Sync - User Sync - Ldap sync enabling for a user's membership is not working
- 417490 Redeployment locks the admin account

2.84.21224.01 - 12 AUGUST 2021

Bug fixes

- 382659 id_token contains multiple audiences

2.81.21189.01 - 8 JULY 2021

Features

- 345136 UI - Enhancements
  - Migrate to `react-md` v2
  - Refactor components to support new validations mechanism
  - Include Sequel Security documentation directly in Security Admin UI

Improvements

- Updated `Sequel.Core.MessageBus` to v2.3.21154.1 due to `Product Backlog Item 366373: Improved performance registering IMessageBusPublisher as singleton`

Bug fixes

- 352201 SSO Cookie performance issue on encryption
- 367301 Admin UI - First client secret is always the one that is deleted/updated

2.78.21145.01 - 25 MAY 2021

Features

- 333157 Upgrade to .NET Core 3.1

Improvements

- 354227 Use diagnostic health in Swagger
- 341762 Include ReturnUrl parameter on ResetPassword link when redirected from an application
- `Sequel.Core.HealthCheck` updated due to work done in `Bug 352048: HEALTH - MessageBus not registered properly in .NET Core app`

Known Issues

- There is a known issue in this version of Security that affects the ability of 3rd party apps (like Workflow, Product Builder, etc.) to keep their session alive by refreshing the access token automatically after it has expired. Instead, when the access token has expired, even if the refresh token hasn't, the current session is simply closed. This is resolved in `v2.81.21186.01`.

Bug fixes

- 343720 Admin - Not able to access Clients section when the user only has permissions for Clients and Applications
- 353641 DM - Error deploying new Sec DB when Data Loss flag is set to true

1.72.21057.01 - 26 FEBRUARY 2021

Improvements

- 345406 ClaimSearch Authentication is not refreshing session where flow is auth_code
- 342317 Improve HealthChecks to reduce a potential DoS attack on diagnostic mode

1.71.21043.01 - 12 FEBRUARY 2021

Improvements

- 276173 Route all errors from stdout to logs database

1.70.21028.01 - 28 JANUARY 2021

Improvements

- 330937 Manager access to Absence Indicator

Bug fixes

- 331146 Admin UI - Wrong redirection when discarding password reset email template changes

1.69.21012.01 - 12 JANUARY 2021

Bug fixes

- 331183 "Show More" button not working in Users tab
- 329953 Identity events are not logged using the underlying EventType and always logged as Information

1.67.20350.01 - 15 DECEMBER 2020

Improvements

- 316936 Force clients to require their secrets when asking for a token
- 315620 Assign default actioner user to a client
- 306413 Apostrophes in group names & Security sync process timeout
- 267574 Manage EmailTemplate from AdminUI

Bug fixes

- 319872 SecurityAPI (AuthN section) Swagger is not loading correctly
- 315461 "Invalid Client" error using a client created using Admin
- 313092 Dot added to Security Domain setting in DM
- 312443 Administration - Absence indicator is editable on user creation
- 307536 Admin UI - "\" char can't be used in user search

1.61.20275.01 - 01 OCTOBER 2020

Improvements

- 264697 Manage PersistedGrant housekeeping from UI
- 305097 Security Username / Email restrictions
- 290994 Reset Password navigation back to caller application
- 302681 Deploy security configuration independently from the schema deployment
- 296131 SaaS - Add endpoint to "restart" a service for an instance

Bug fixes

- 303513 Unable to logout due to `samesite` none cookie
- 305583 Timeout importing users
- 289960 Two sessions can be opened in the same browser
- Upgraded AppBuilder to v0.2.20241.1 and MessageBus to v2.1.20244.1 as part of `290735 BUS - StopRabbitMqConsumers doesn't stop consumers`
- Upgraded user session web component as part of `286576 Web Component - Prevent negative time`
- 285251 Inconsistency at groups validation in sequel-security tool
- 287887 Import failure of not having the application.json of an app is not handled

1.53.20168.01 - 16 JUNE 2020

Features

- 260912 Security Services Farm - SaaS

Improvements

- 282755 Integration to update the SSO cookie when using only Bearer Authentication
- 267555 Repoint clients commands (tool and admin)
- 281329 Resolve HIGH risk vulnerabilities caused by ForgotPassword page

Bugs fixed

- 280857 Admin - Bad request when signing in after lockout

- 279887 Ampersand not being accepted as non-alphanumeric char for passwords
- 279476 PenTest - Buffer overflow in ForgotPassword
- 282799 Show more users does not work first time if a filter is applied
- 281706 sequel-security command line tool fails when importing a big amount of users
- 278767 Deployment of security configuration is not rerunnable
- 278957 Handle duplicates entries at client collections: secrets, URIs and origins
- 282326 security-saas-create-database READ_COMMITTED_SNAPSHOT OFF

1.50.20126.01 - 5 MAY 2020

Improvements

- 269940 SIU - User Sync
- 274828 Support Reverse Proxies/Load Balancers on K8S and AWS

1.49.20115.01 - 24 APRIL 2020

Improvements

- 274718 Health - Enable Non-intrusive and diagnostic modes

1.49.20112.01 - 21 APRIL 2020

Improvements

- 270334 Scale-out DataProtection using database
- 267989 Scale-out DataProtection using AWS SSM Data Protection Provider
- 244144 Issues with backslashes in API parameters when hosted in IIS
- 265033 SaaS Support Security in containers Docker
- 265036 SaaS Reverse proxy
- 268335 Added documentation of maintenance and logging

Bugs fixed

- 273214 PersistedGrant is not deleting by token type

1.47.20093.01 - 2 APRIL 2020

Improvements

- 265030 Data protection scale-out in AWS based on AWS System Manager (beta version)

Bugs fixed

- 269991 sequel-security tool sync command validates wrongly the legacy database

1.47.20086.01 - 26 MARCH 2020

Improvements

- 253311 Security Admin Logos replacement with new SVG and tweaks

Bugs fixed

- 267173 Create ADMIN user creation in DatabaseMetadata at Deployment Manager
- 264004 Security API fails during start-up due to error in healths checking databases

1.46.20072.01 - 12 MARCH 2020

Improvements

- 246349 Administration UI - Improve error page message
- 243211 Client management
- 256557 Housekeeping on Authentication.PersistedGrant table
- 256553 Improve database access during login process for getting ApiScope configuration

Bugs fixed

- 256550 Many refresh token created during login process (Integration NuGet Package)
- 260273 Admin UI - Configuration panel resizes after opening card in Monitoring tab

1.44.20031.01 - 31 JANUARY 2020

Epic 173720 Security - User Info Web Component - Common

This new epic offers a shared web component for all of Sequel's applications for displaying information of the current user and for managing it's session.

This new web component gives us a more consistent user experience across our applications and also solves some technical debt with the current session management providing a web component package available in our npm repository that can be easily integrated by registering this component in our applications.

It contains UI (user info and a close session option) and also some logic (single-sign-out event detection and inactive session detection).

Improvements

- 245895 User screen Organise information in cards
- 238262 UserWebComponent - Configure Session management
- 237969 UserWebComponent - Integration in Security Administration
- 237972 UserWebComponent - Integration in Security Authentication

Bugs fixed

- 255402 Password Reset - Server error during password reset procedure
- 240089 Session with two different users can be opened in the same browser using SEC ADMIN app

1.43.20021.01 - 21 JANUARY 2020

Bugs fixed

- 250550 Administration site stuck into an infinite loop
- 246803 Unable to delete audit processes when synchronization is disabled

1.40.19346.01 - 12 DECEMBER 2019

Epic: Security - Role-level authorization sync with Active Directory

Feature 204397 Sync Users from AD

- 232137 AD Sync - Extract: The AD Sync Service
- 232138 AD Sync - Transform & Load: LdapSync Endpoints
- 232140 AD Sync - Configuration and Monitoring

Improvements

- 212278 Performance - Reduce calls number when token is renegotiated
- 214478 Use AuthorizationCode flow instead of Implicit flow. Security recommendation of OpenId Foundation and IdentityServer.
- 242539 HealthCheck - Change memory check output to not give info on OS.

Bugs fixed

- 238427 Security API & AuthZ Swagger configured to internal Auth URL instead of public
- 238436 Wrong error handling when SecurityApiSettings.AuthenticationApiKey is wrongly configured.
- 238865 Admin UI - Deleting/Editing memberships depends on order they are saved
- 245888 Unclear message in MyAccount page

1.38.19325.01 - 22 NOVEMBER 2019

Bugs fixed

- 237827 Update health check NuGet with a version following new version number format

1.38.19317.01 - 13 NOVEMBER 2019

Improvements

- 207140 Configure TTL for EffectivePermissions message
- 236116 Variable names in environmentConfig

Bugs fixed

- 232515 Health check error handling for message bus & databases
- 233802 Unable to edit T&C
- 235532 Admin UI - Wrong message when removing permissions
- 239949 Unable to authenticate from Admin UI in AWS

Epic 204375 Security - Role-level authorization sync with Active Directory

Feature 204398 MembershipSets:

- 232102 MembershipSets API
- 232103 MembershipSets UI
- 232105 MembershipSets Import/Export

1.36.19293.02 - 20 OCTOBER 2019

Improvements

- 215544 Improved HealthCheck Integration
- 182022 Improve error handling when reading appsettings from security services
- 214481 Protect "Sync" button with securable when user has no permissions
- 230329 Improve Show more User UX
- 230338 Disable autocomplete in search
- 230355 Review confirmation popup in Admin
- 220867 Security API authentication failure caused by invalid ClientRedirectUri entries
- 227179 Set default DB settings for isolation level
- 223731 Security Tool is returning "import completed" message when the App imported doesn't exist
- 227571 Ability to configure Windows Authentication in Release

Bugs fixed

- 208444 SSO failing when multiple tabs are open
- 213511 EffectivePermissionMessages are not being triggered when creating securables
- 213845 Authentication between Origin/Claims and Supporting Apps timeout in some random scenarios
- 225815 Session with two different users can be opened in the same browser using SEC ADMIN app
- 227581 UI issue in login with external auth provider
- 229860 Remove ENTITYUSERS help from sequel-security

Known Issues

- *Bug 239949 Unable to authenticate from Admin UI in AWS*, fixed in 1.38.19317.01

1.0.19241.01 - 29 August 2019

Improvements

- 207180 Use authentication data for managing AllowedCorsOriginsCurrent
- 210509 User filter of an entity shouldn't be case sensitive
- 211443 Reset password email subject configurable
- 216237 Azure AD Authentication when application is registered only for users in the same organisation (single tenant)

Bugs fixed

- 196188 Session is being saved when we are in a different URL
- 212082 Failed login attempts counter keeps growing after a user gets locked out
- 212947 PasswordPolicySettings not being used
- 213511 EffectivePermissionMessages are not being triggered when creating securables

1.0.19218.01 - 6 August 2019

Improvements

- 206442 Prevent Security API access to [master] database
- 206415 Forgot email - Ability to configure SSL mode-> AWS deployment only
- 206421 Add ApplicationKey header to all messages
- 200128 Add permissions descriptions to Vanilla config
- 208295 Use substrings for quick search filter in Security Admin
- 211443 Reset password email subject configurable
- 188755 Terms & Conditions - Improve WYSIWYG editor
- 187846 LoggingTermsAndConditions isn't found when a new environment is deployed
- 206092 Ensure all EffectivePermissions consumers receive the message
- 203290 RE - WF - PB Update Verisk Favicon in all Supporting Apps applications

Bugs fixed

- 205672 Incomplete error message when introducing incorrect credentials in login
- 209908:Security tokens are generating activity in WF/PB/RE
- 205581 Security tool create admin when already exist throw and error
- 201514 Security administration site hangs after idle for some time
- 197632 Authorization cache not invalidating for no application key when roles permissions are changed
- 205161 Blocked user is able to login
- 200025 Ensure all consumers are configured as "BindMessageExchanges": true
- 195820 Inactive user login with Microsoft Account
- 190470 Error when importing an App with permissions for a global securable

Note 1: new Sequel Bus version being used as per 198481:BUS - Automatically configuration of shared & independent consumers

Note 2: new Sequel Logging version being used as per 113299:LOG - Log Table optimizations

1.0.19128.01 - 8 May 2019

Improvements

- 175445 Ability to synchronize information from MS Accounts with Sequel Users using Microsoft Graph
- 186123 Investigate performance issue reported by Api GWY - Part 2
- 189309 New logo in header