Generic Identity Service with(Mulittancy & Microservices)

Introducing SELISE `< blocks />`

What is SELISE < blocks />?

Previously known as SELISE ECAP/Core/Secure Link Services, we are rebranding our suite of components into `SELISE <blocks />` and transforming it into an open-ended full-stack platform for developing web and mobile applications.

Introducing new Identity in Selise V2

# Analysis of the problem of identity V1 in Selise (POC to Production)

1. Not fully support Oauth2
2. Dynamic Client Registration does not support
3. Not updated with Modern Architecture
4. Complex integration process
5. Introduce SSO and integration easily.

Solved problems with identity version 2

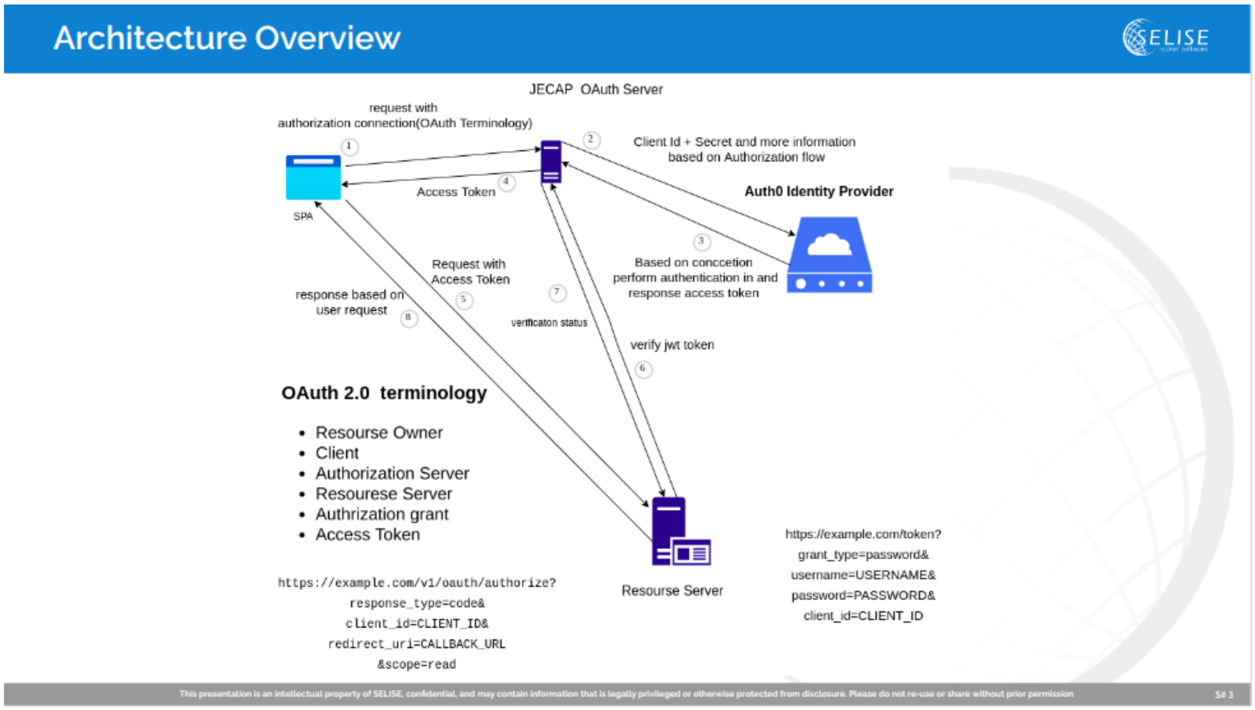
1. Introduced Scope base Permission.
2. Adopt with OAuth2 Architecture.
3. Backward Compatibility With Selise Blocks.
4. Generic Identity Support for all services.

Working Technology and Architecture :

1. Microservices
2. Militancy
3. OAuth2
4. JWT
5. Auth0
6. Messaging Service (RabbitMq)
7. Caching (Redis)

Back-End

1. Java
2. Spring Boot
3. MongoDB



Architecture Overview

JECAP OAuth Server
request with authorization connection(OAuth Terminology)
1
SPA
2
Client Id + Secret and more information based on Authorization flow
Auth0 Identity Provider
Access Token
4
3
Based on conccetion perform authentication in and response access token
Request with Access Token
5
response based on user request
8
7
verificaton status
verify jwt token
6
Resourse Server

OAuth 2.0 terminology

- Resourse Owner
- Client
- Authorization Server
- Resourese Server
- Authrization grant
- Access Token

```
https://example.com/v1/oauth/authorize?
response_type=code&
client_id=CLIENT_ID&
redirect_uri=CALLBACK_URL
&scope=read
```

```
https://example.com/token?
grant_type=password&
username=USERNAME&
password=PASSWORD&
client_id=CLIENT_ID
```

This presentation is an intellectual property of SELISE, confidential, and may contain information that is legally privileged or otherwise protected from disclosure. Please do not re-use or share without prior permission

Objective

The purpose of this document is to provide readers with an overview of SELISE's multi-tenancy architecture, with explanations and descriptions of key components. Developers at SELISE can use this document as a reference and support guide to gain an understanding of the different types of multi-tenancy implemented for applications.

Scope

This document will cover the three types of multi-tenancy configurations currently offered by SELISE to its clients. Sections of this document include architecture diagrams and details on the configurations used for tenants. SELISE makes use of .NET and MongoDB data services as key components within the multi-tenant system. The three types of multi-tenancy covered in this document are the following:

- Database Multi-tenancy
- Schema Multi-tenancy

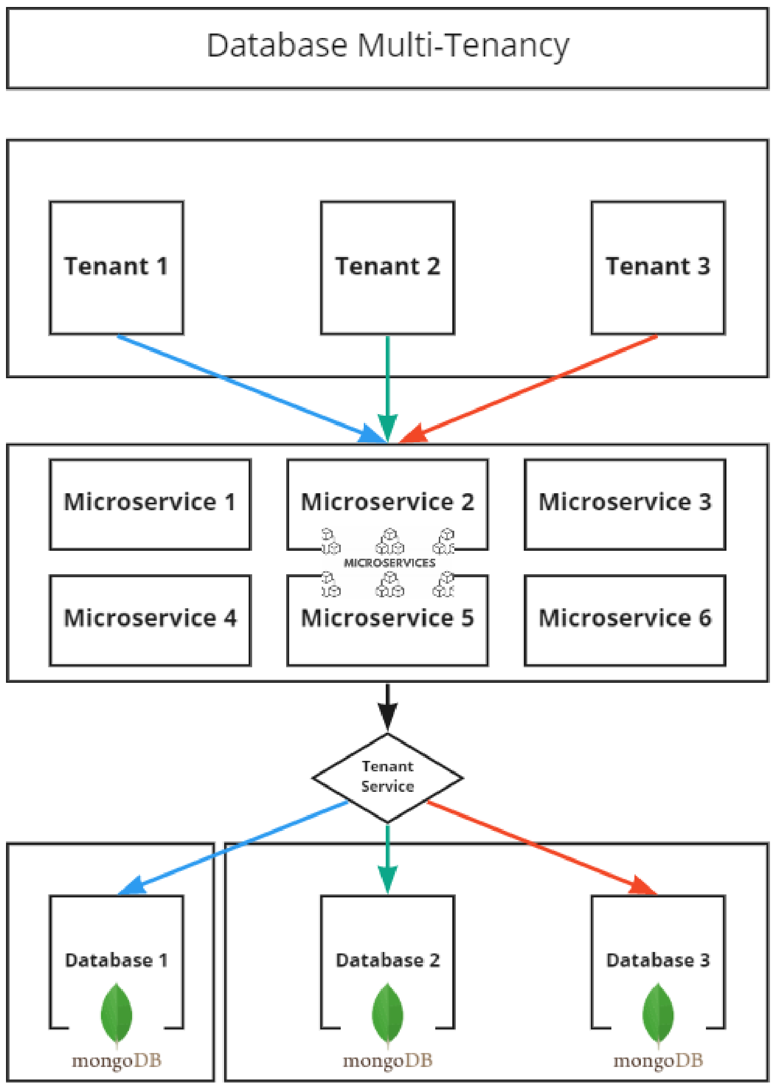
- Table Multi-tenancy

Database Multi-tenancy

In Database multi-tenancy, the application connects to a database and gets data while the tenancy logic is delegated to the ops layer. In an exemplary implementation, the application has no concept of tenants. As such, each tenant lives in a separate database. The database could be a separate server or merely a different database within the same MongoDB server. In hindsight, that flexibility is one of the most compelling reasons to do database multi-tenancy.

This is helpful when the customer of a SaaS application wants to have a completely separate instance of the application with their own Database. The most significant advantage of database multi-tenancy is peace of mind. It's simply impossible for one tenant to see another tenant's data due to a bug in the application layer.

The architecture of our Database Multi-tenancy is provided below.



Schema Multi-tenancy

In Schema multi-tenancy, your application connects to a database once and has some logic for choosing which schema to connect to when serving a particular tenant. Schema multi-tenancy is both compelling and popular. The tenancy logic still lives in the application, deciding which schema to access data from.

The Schema routing logic can be encapsulated in a small and well-guarded router. Isolating the connectivity logic solves most of the problems with table multi-tenancy while being operationally simpler than database multi-tenancy.

SELISE uses Schema Multi-tenancy when the customer does not require a completely separate database and is happy with a shared database structure. This ensures that customers' data is kept separate in the same cluster and other applications using the shared database cannot access the data of the customer.

The architecture of our Schema Multi-tenancy is provided below.

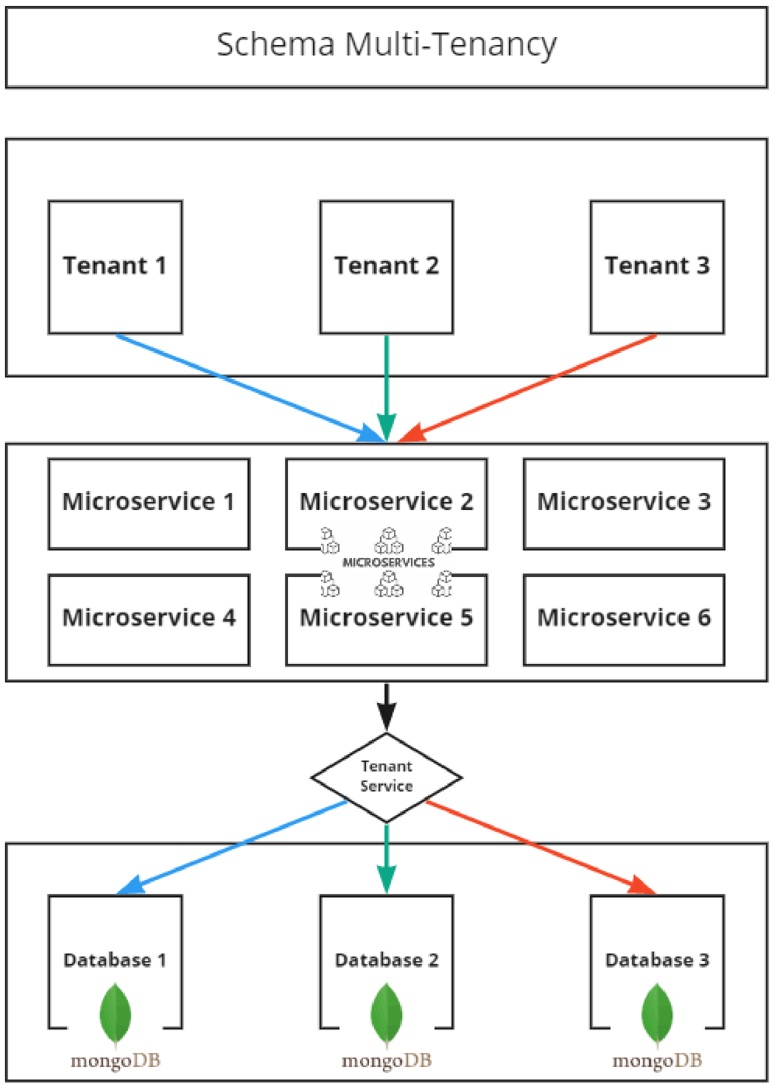


Table Multi-tenancy

In Table multi-tenancy, every row in every table is associated with a given tenant. The application doesn't need to worry about which schema or database it is connecting to. Instead, the business logic also maintains the tenancy logic. Our business deals with customer data. If we ever mistakenly shared one customer's data with another, it would be a disaster for them and us. Table multi-tenancy means the tenant logic lives in the application layer and thus offers many opportunities to make mistakes and leak data. By removing the tenancy logic from the application layer those mistakes become impossible. Making disasters impossible is a compelling reason to avoid table multi-tenancy.

Table multi-tenancy is used in SELISE’s SaaS Applications like SELISE Signature. The architecture of our Table Multi-tenancy is provided below.

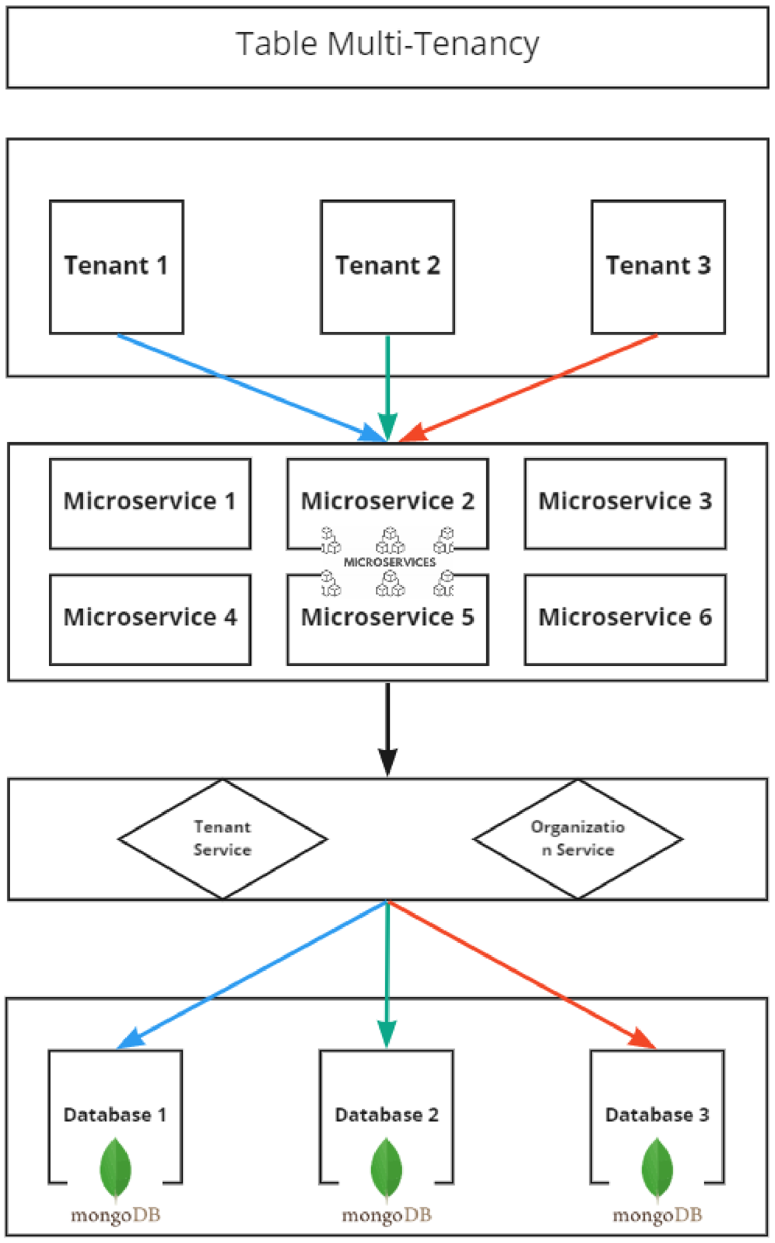


Table Multi-Tenancy
Tenant 1
Tenant 2
Tenant 3
Microservice 1
Microservice 2
Microservice 3
MICROSERVICES
Microservice 4
Microservice 5
Microservice 6
Tenant Service
Organizatio n Service
Database 1
Database 2
Database 3
mongoDB
mongoDB
mongoDB