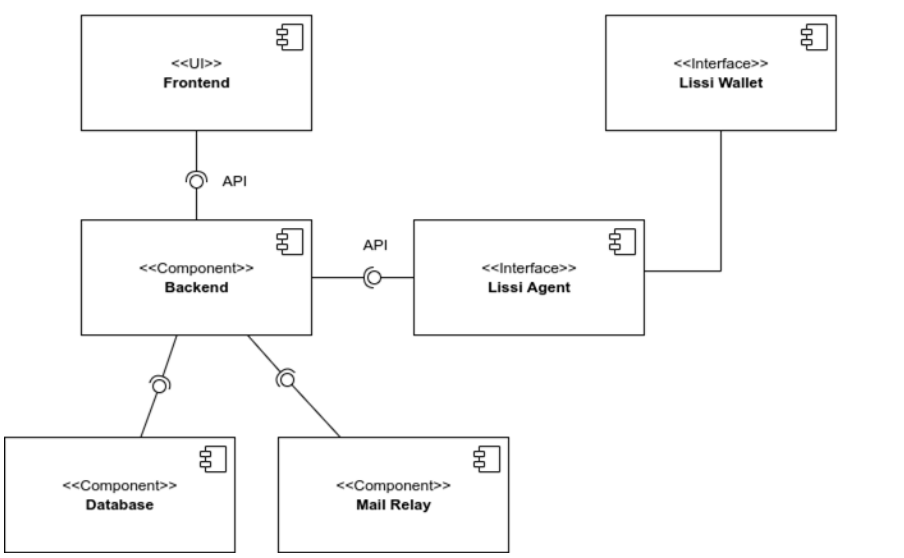
Software Architecture Description

AMOS SS2022

Project 4: Digital Identity

Runtime Components



The backend is the heart of our application. It is used to communicate with the Lissi Agent (external agent), which writes and retrieves data of the lissi wallet (also external). The lissi wallet is structured as a blockchain. Furthermore, the backend communicates with the database, where all digital identities are stored and with the mail relay to send a mail with a QR-code and initial password after creating a new digital identity. The functions of the backend can be used in the frontend. The frontend displays in a user-friendly environment all important functions required by our industry partner. The frontend will be used by HR-employees.

Code Components

Backend: Spring

The backend consists of a lot of classes which can be divided into repository, controllers and services. Additionally, there are some classes for enums, messages, models and security. Furthermore, there are a lot of tests in test classes which are written with the JUnit framework-

Repository

A repository is used for having a connection and communicating with the database.

- **UserRepository** is stored in a mysql database in conjunction with Spring Data JPA. The UserRepository is where the user entities are stored. The User entity class stores all private data such as name, email, password, but also the user role. The user role is used for authentication to use various methods.

Controller

Controllers in our Backend are called via API calls by our frontend.

- **AuthenticationController** handles the login and registration process and also provides an update function for the user data. It also provides the API mapping that spring web implements for us. The AuthenticationController has access to the UserRepository.
- **ConnectionController** manages and retrieves connections/Dis, which represents an interface to the Lissi Endpoint.
- **SchemaController** can create schemas and return all schemas to the frontend.
- **CredentialDefinitionController** can create a new credential definition and return all credential definitions.
- **CredentialController** can issue a credential definition to a DI, which creates a credential, and return all credentials.
- **ProofTemplateController** can create a new proof template and return all proof templates
- **PresentationProofController** can send a proof template to a DI, which creates a proof, and return all proof templates

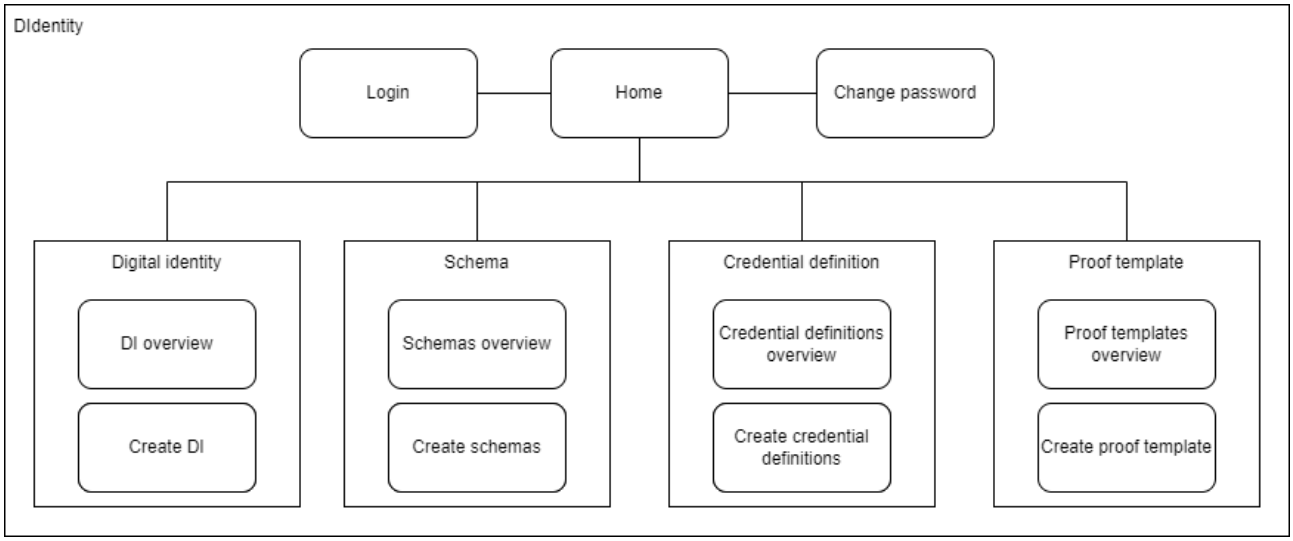
Services

There is a service for each controller, where the logic is stored. Additionally there are the following services:

- **LissiApiService** is used by services to communicate with lissi via REST calls.
- **HttpService** is used by LissiAPIService to have a more easy way to communicate with lissi.

- **MailService** is used to send an initial password and a QR-Code for a DI via Mail. It is the interface to communicate with the mail relay.
- **QrGeneratorService** is used for creating QR-Codes that can be scanned by a user to login with the lissi app.
- **ResourceService** is used for getting files stored in our source.
- **StrongPasswordService** is used for creating a strong initial password.

Frontend: Angular



There are individual components that describe individual pages:

- **Login** describes the login page, which gets an email address and a password and should let the user log in.
- **Home** describes the start page, where the user can navigate to all pages.
- **Change password** describes the page that requires an email, password and new password to change the password of an user.
- **Digital identity (DI) overview** describes the page that gives an overview of all registered digital identities. A DI consists of a connection from lissi matched to a user of the database including name, email and password.
- **Create Digital identity (DI)** describes the page that allows to create new DIs. DiI are stored in the database and in Lissi.
- **Schemas overview** describes the page that gives an overview of all created schemas. A schema is used for creating a new credential definition.
- **Create schema** describes the page that allows to create new schemas.
- **Credential definitions overview** describes the page that gives an overview of all credential definitions. A credential definition is a collection of rights. Sending a credential definition to a DI creates a credential.
- **Create credential definition** describes the page that allows to create a credential definition.
- **Proof templates Overview** describes the page that gives an overview of all proof templates. Sending a proof template to a DI creates a proof. A proof is used for validation of a credential.
- **Create proof template** describes the page that allows to create new proof templates.

Technology Overview

Technology Stack

Component	Technology/Tools	Version
Version Control	Github git	-
Frontend	Angular HTML CSS Typescript	13.3.3
Backend	Spring Boot Spring Web Spring Security Spring Data JPA Spring Mysql Connector Maven	2.6.7 2.6.7 2.6.7 2.6.7 3.6.3
Deployment	Docker	20.10.14
Database	MySQL Community Server	8.0.29
API Development	OpenAPI	3.1
Lissi Endpoint	Lissi Institutional Agent	

We chose Angular as our frontend because many team members already have experience with it. Angular can be used to create progressive web apps with the capabilities of a modern web platform. Angular apps are also available through the web browser but can also be ported to mobile devices very easily. With HTML, CSS and Typescript as mainstays, proven and well-known technologies are used.

Spring is the world's most popular Java framework and brings many features we need for a backend. It is very flexible and has a comprehensive set of extensions that allow developers to create almost any application imaginable. Spring Security helps us integrate our application with industry standard security schemes and deliver a trusted solution. With Spring Data JPA and Spring MySQL Connector, our database can be integrated very easily and many stumbling blocks can be overcome from the start.

With MySQL Community Server, a strong, widely used and proven relational database management system was chosen.

By using the OpenAPI standard we were able to develop our API very fast and understandable for everyone.