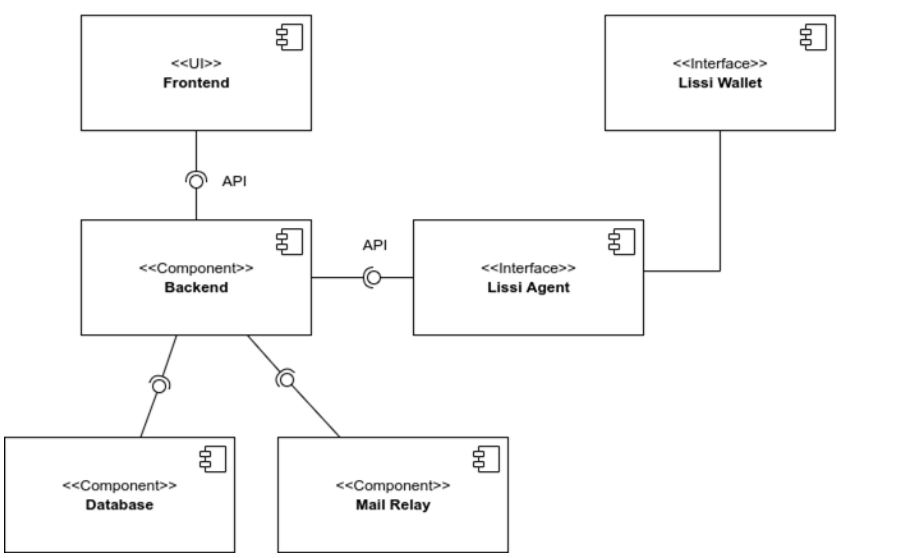
Software Architecture Description

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 link to the created connection, also 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 who are responsible for creating and maintain digital identities of the company.

Code Components

Backend: Spring

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

Repository

Repositories are used to connect and communicate 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. It also provides the API mapping that spring web implements.

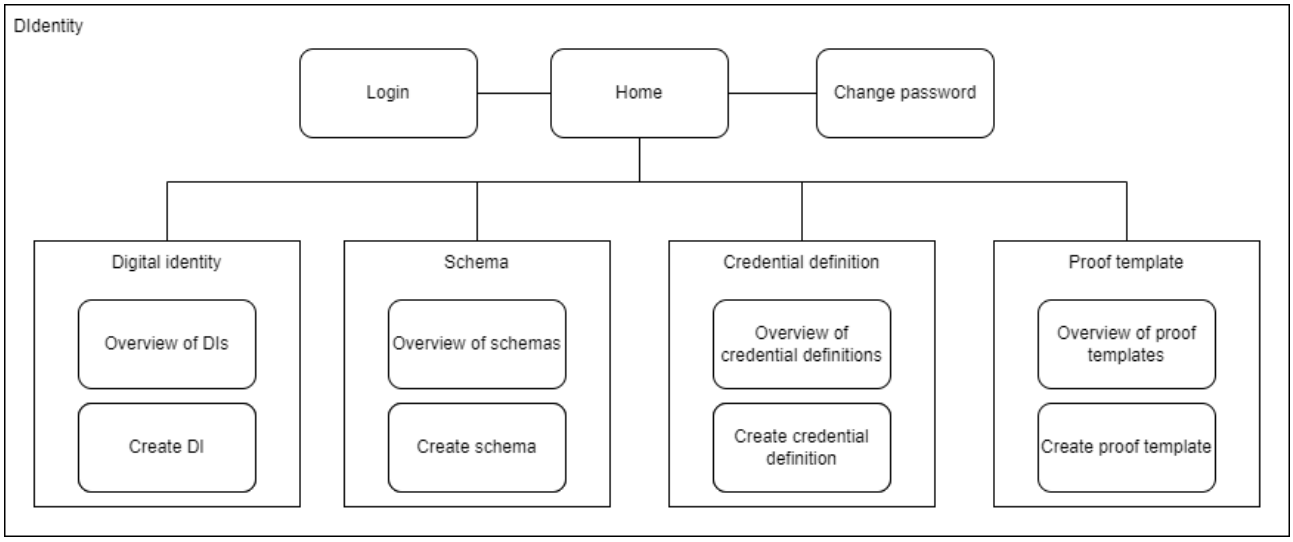
- **AuthenticationController** handles the login and registration process and also provides an update function for the user data. The AuthenticationController has access to the UserRepository.
- **ConnectionController** manages and retrieves connections/DIs, which represents an interface to the Lissi endpoint.
- **SchemaController** can create a new schema and return all created schemas saved in the Lissi backend to the frontend.
- **CredentialDefinitionController** can create a new credential definition and return all credential definitions saved in the Lissi backend to the frontend.
- **CredentialController** can issue a credential definition to a DI, which creates a credential, and return all credentials saved in the Lissi backend to the frontend.
- **ProofTemplateController** can create a new proof template and return all proof templates saved in the Lissi backend to the frontend.
- **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, in which the functionality is implemented. Additionally, there are the following services:

- **LissiApiService** is used by other services to communicate with Lissi via REST calls.
- **HttpService** is used by LissiAPIService. It provides an easy way to communicate with the Lissi backend.
- **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 add a created connection to the Lissi wallet app.
- **ResourceService** is used for getting files stored in our source.
- **StrongPasswordService** is used for creating a strong initial password when creating a new HR-employee.

Frontend: Angular



There are individual components that describe individual pages:

- **Login** describes the login page. A HR-employee can login here by typing in his/her mail and password.
- **Home** describes the landing/start page, which provides the navigation overview to all the other pages.
- **Change password** takes mail, password and new password as an input and sets the new password for the account if the data is valid.
- **Overview of DIs** describes the page with an overview of all registered digital identities. A DI consists of a connection from Lissi matched to a user information of the database including name, mail and password.
- **Create DI** allows to create new DIs. When creating a new DI the provided data is add to the database and a new connection is created in Lissi. The link to the connection is then send to the provided mail address.
- **Overview of schemas** describes the page with an overview of all created schemas. A schema is used for creating a new credential definition.
- **Create schema** allows to create a new schema. A schema consists of a name, a version and a list of attributes (at least one).
- **Overview of credential definitions** describes the page with an overview of all credential definitions. A credential definition is a collection of rights. Sending a credential definition to a DI creates a credential which can be displayed in the Lissi wallet.
- **Create credential definition** describes the page that allows to create a credential definition. A credential definition consists of a name, optionally a comment and the linked schema. The attributes of the schema will be added to credentials created with this credential definition.

- **Overview of proof templates** describes the page with 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. A proof template consists of a name and a version. It is linked one or more credential definitions and can also include self-attested attributes which can be filled out

Technology Overview

Technology Stack

Component	Technology/Tools	Version
Version Control	Github git	-
Frontend	Angular HTML CSS Typescript NodeJS Caddy Jasmine	13.3.3 18.6.0 2.5.2 4.3.0
Backend	Spring Boot Spring Web Spring Security Spring Data JPA Spring Mysql Connector Maven JUnit	2.6.7 2.6.7 2.6.7 2.6.7 3.6.3 3.8.6 5.7.2
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.