Lufthansa Test Project

Work by: Ergys Rrjolli

1. Introduction

In this paper I will be explaining the process of building out the lufthansa test project I was assigned.

The project aims to be an information system for keeping track of annual leave requests within a company. The system should be able to handle things such as leave request approval or denial, email notifications, password reset functionality and more.

In the duration of this paper I start with a detailed requirement specification of the system and what it needs to do. From this I will also extract out the actors of the application and their use scenarios.

Afer that I will discuss about the overall system design and how it all fits together.

Following system design, there will be a discussion about system implementation which will discuss the implementation details of the components of the system.

2. Requirement Specification

In this section I will be describing in detail all the system requirements and specifications so we can understand exactly what we need to build and how to build it.

Let’s start with functional requirements.

2.1 Functional Requirements

In software enginnering, a functional requirement defines a function of a system or its components, where a function is defined as a specification behaviour between inputs and outputs.

I will divide these requirements in three main categories.

2.1.1 Authentication Requirements

The authentication requirements of the application are:

- The system will provide an interface for users to log in the system
- The system should use token based authentication.
- A role should be assigned to each user after authentication to determine the available services.
- User data used for authentication will be stored in a relational database.
- A user’s password can be reset using email based notifications.

2.1.2 Administrator Requirements

The administartor requirements of the system are:

- The admin main view will give him access to most of the services the system provides.
- The admin must have an easy way to add new users into the system.
- The admin must have an easy way to modify and delete users of the system.
- The admin must be able to change his personal information and password.
- The admin must be able to reset his password using the reset password web flow.

2.1.3 Supervisor Requirements

The supervisor requirements are as follows:

- Supervisors will be able to see the list of all the users they are supervising.
- A supervisor can see the list of requests of a given user.
- The supervisor can see the list of all requests of its users.
- The supervisor can approve/deny the request of one of its users.

- The supervisor has the ability to export the requests of a user to excel format.
- The supervisor must be notified via email every time a new request is made.
- The supervisor must be able to change its personal information and password.
- The supervisor must be able to reset its password using the reset password web flow.

2.1.4 User requirements

The user requirements are as follows:

- The user will be able to see a list of its previously made requests.
- The user will be able to make new requests.
- The user will be able to delete one of its requests.
- The user must be able to change its personal information and password.
- The user must be able to reset its password using the reset password web flow.

2.2 Non functional requirements

In software engineering, a non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. They are different from functional requirements that define specific behavior or functions.

2.2.1 User interface

There should be a login page provided from which a particular user/supervisor/admin can log in.

After login the main page is divided into 2 main sections. The navigation bar and the display page. The display page should be consisten throughout the application and contain the content of various pages.

The navigation bar should be to the left of the display page and should contain all the different options for navigating to different pages.

2.2.2 Usability

Each component of the page should as common as possible, this is to make sure that if the user is familiar with other sytems of the same kind the transition from that system to this one should be as easy as possible.

2.2.3 Performance

The system is intended to be used in the browser platform through communicating with the api.

So there are two main time and space requirements we should look for:

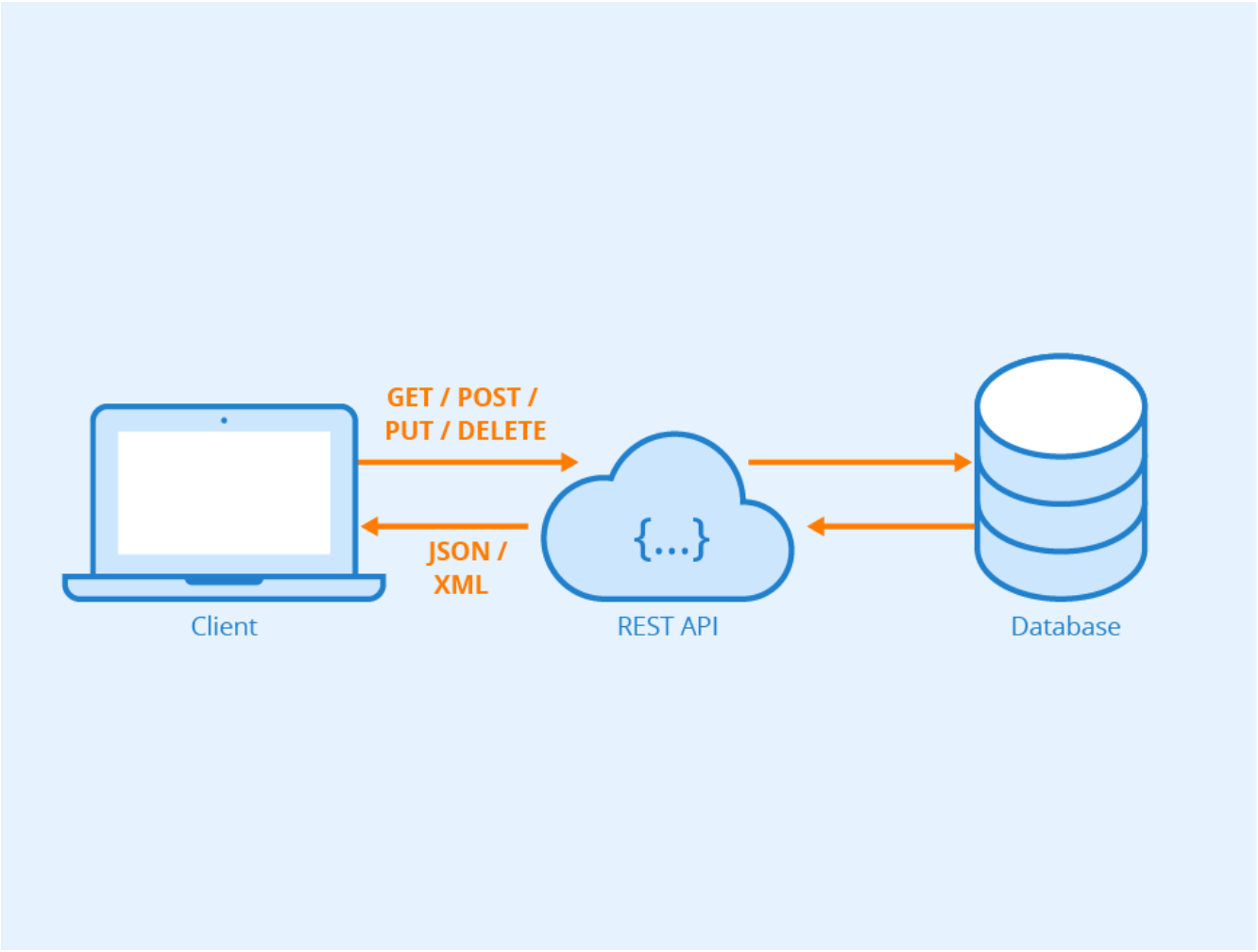
- The speed of use and start up of the application in the browser. The application itself is completely in javascript so everytime the user loads the page, the whole javascript bundle will be downloaded. The bundle after production optimization should be no more than 300 KB and should be downloadable in less than 100ms in an average internet connection of 5 MB/s including the propagation delay.
- The response time and storage requirements of the back end. The response time should not be a problem since we are using java and spring container, where the objects that are used the most are created as beans so that they can be initialized once and be used multiple times. We can also utilize the multithreading capabilities of java, which are very well defined and supported.

3. System design

The system should be built using the client-server architechture. Where the client, i.e browser client, is completely seperate from the back end api, and the back end is also seperate from the database. They will communicate using a RESTful api using the HTTP protocol.

And the back end component will communicate with the database either locally using the mysql protocol or via a network connection.

The below picture illustrates it very well:



This design follows the microservices architecture, which lets us deploy, develop and scale different components of the system seperately.

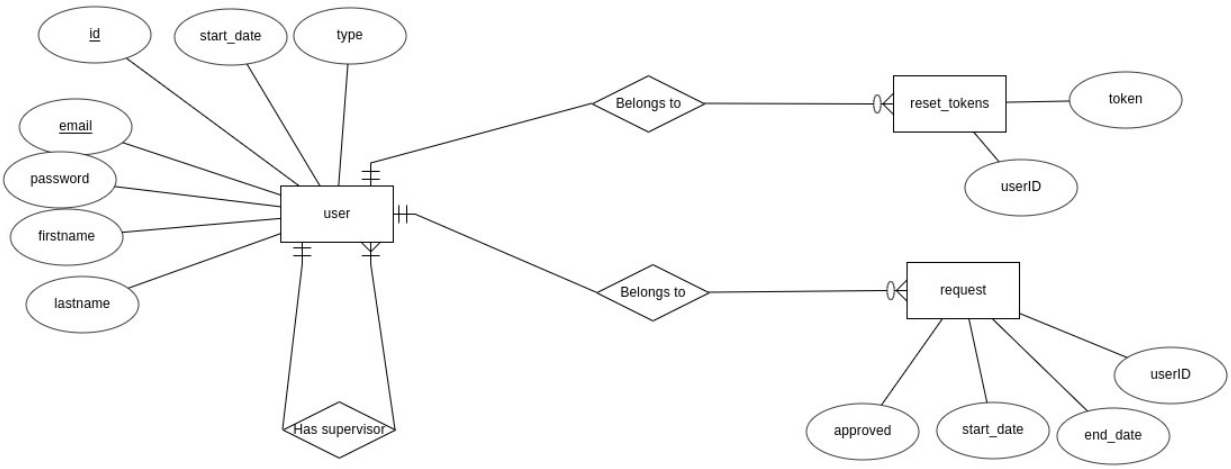
4. System implementation

In this section I will describe the implementation details of each component.

4.1 Backend API

The back end api is built in java using Spring Boot for the rest service and mysql as the relational database.

I started by first designing the Entity-Relationship diagram for the database, which looks like the following:



The ER diagram was then converted in the equivalent database schema.

The technologies and libraries used are:

- Spring MVC for the REST layer
- Spring data JPA for the persistence layer (uses Hibernate under the hood)
- Swagger for documenting the API functions (through springfox-swagger2 implementation)
- Junit, AssertJ and Mockito for testing the application.
- Spring security for configuring authentication and authorization
- Spring Boot Starter Mail for seding email notifications.
- mysql-connector-java for connecting to the database
- org.jwonwebtoken.jjwt for creating and validating json web tokens
- org.apache.poi for creating microsoft documents (excel)
- log42j for handling our logging requirements

The code structure of the application looks as following:

The explanation for each of the packages:

- **configuration –** contains spring security and spring mvc configuration classes
- **controllers –** where the rest controllers of the application are defined
- **exception –** defines some custom exception for handling authentication problems
- **filters -** defines the jwt spring security filter
- **models –** defines the database entities and some models for handling requests
- **repositories –** defines the spring data jpa repositories for each on of the entities
- **services –** defines some common services which will be used by the application
- **util –** defines some utility classes for handling various functions, such as sending an email, creating and validating json web tokens, and exporting to excel sheets.

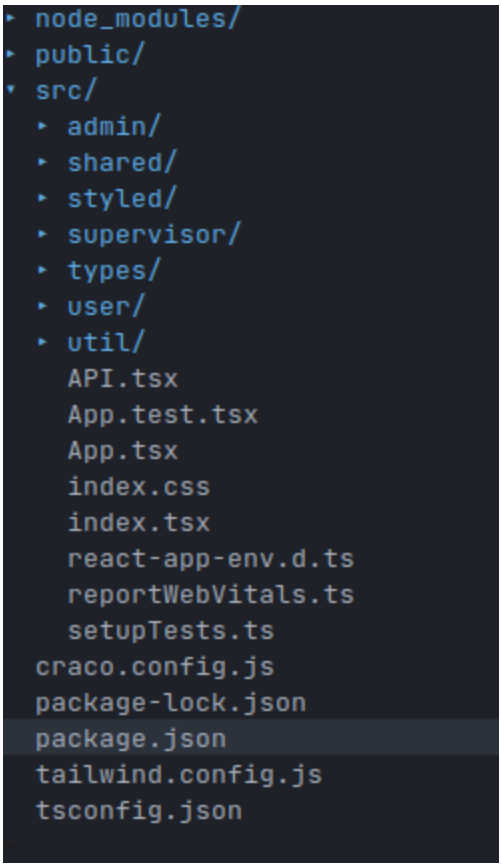
One thing to note here is the main service of our application, which is the UserService. It handles the business logic of all the operations related to users and validates them. Almost all the controllers use this service to perform their operations. I have made sure to document this and many other classes using the JavaDoc specification, so that information about methods can be available right in your IDE.

4.2 Front end client

The front end client is built using Typescript and React. This way we can use the scalability of Typescript with the componentization of React to build the front end.

The technologies and libraries used are:

- Craco – used for Create React App Configuration Override
- React – the react library
- React Router – for handling client side routing
- Typescript – a superset of javascript which provides type safety
- TailwindCSS – css framework for building complex front end components



The front end application also uses the new *fetch* api for making http requests and communicating with the api. The code structure of the application looks like the following:

- **admin –** Contains the admin page components
- **supervisor –** contains the supervisor specific components
- **user –** contains the user specific components
- **types –** contains common type definitions
- **styled –** defines the styled components the application will use
- **shared –** components shared by all three users, such as the login
- **utils –** utility functions, such as converting dates, performing calculations etc.

One thing to note in this project is that I have used the new and preferred way of React functional components with hooks.

I have also set up the authentication functionality of the application using the useContext and useState hooks. Those definitions can be found in the App.tsx file.

5. Conclusion

In this paper I have shown the analysis, design and implementation of the project assigned to me by Lufhansa industry solutions.

The system was divided into components, analyzed and built using state of the art technogies and enterprise designs. All the sources and documentation can be found over at my github account: https://github.com/Ergys2000/Lufthansa-Test-Project

Thank you for your time and consideration!