HAMK
Hämeen ammatti-
korkeakoulu

LABORATORY SAMPLE HANDLING APPLICATION USING REACT.JS AND JAVA SPRING BOOT

Bachelor’s thesis
Information and Communication Technology
Autumn 2023
Laura Byman

HAMK
Hämeen ammatti-korkeakoulu

Tieto- ja viestintätekniikka, insinööri (AMK) Tiivistelmä
Tekijä Laura Byman Vuosi 2023
Työn nimi Sovellus laboratorionäytteiden käsittelyyn Reactia ja Java Spring Boot:ia käyttäen
Ohjaaja Petri Kuittinen

Opinnäytetyön ensisijaisia tavoitteina olivat: kehittää sovellus laboratorionäytteiden käsittelyyn ”proof-of-concept” näkökulmasta, sekä laajentaa omaa osaamistani web-sovelluskehitysteknologioissa. Sovellus toteutettiin responsiivisia web tekniikoita ja REST-rajapintaa hyödyntäen. Tarkoituksena oli tutkia, miten moderneja web-kehitysteknologioita voidaan hyödyntää terveydenhuollon sovellusten kehittämisessä.

Teoria osiossa esitellään teknologiat, kehykset ja kirjastot, joita käytetään ohjelmistokehitysprojektissa. Tässä osiossa käsitellään myös näiden teknologioiden ja kehysten valintaperusteita tähän projektiin ja lisäksi pohditaan vaihtoehtoisia menetelmiä.

Tutkielman käytännön osuudessa esitellään laboratorionäytteiden käsittelyyn soveltuvan sovelluksen kehittämisprosessia. Kuvaan yksityiskohtaisesti eri vaiheita sovelluskehityksessä, ja erilaisia ratkaisuja, mihin projektin aikana päädyttiin, sekä lopputulosta. Projektissa käytettiin muun muassa React, Java Spring Boot, Hibernate, Tanstack Query, ja Material UI kehikoita ja kirjastoja.

Kehitysprojektin tuloksena syntyi toimiva websovellus, joka täytti suurimmaksi osaksi sille alussa asetetut tavoitteet. Lisäksi tämän projektin aikana kerätty kokemus edisti merkittävästi henkilökohtaista oppimistani ja osaamistani käytetyistä teknologioista.

Avainsanat React, Java Sping Boot, websovellus, laboratorio
Sivut 28 sivua

HAMK
Hämeen ammatti-
korkeakoulu

Information and Communication Technology
Author Laura Byman
Abstract
Year 2023
Subject Laboratory sample handling application using React.js and Java Spring Boot
Supervisors Petri Kuittinen

The primary goals of this thesis were: to develop a laboratory blood sample handling application in a proof-of-concept fashion and to expand my own proficiency in web software development technologies. The application was designed using responsive web technologies and a REST API. The aim was to investigate how modern web development technologies can be effectively applied in the development of healthcare applications.

In the theoretical section of this thesis, I introduce the technologies, frameworks, and libraries used in the software development project. This section also provides background information on how these technologies are utilized. Furthermore, it offers an explanation of why these particular technologies were selected for this project. Additionally, I will compare them to some widely used alternatives. This approach aims to provide an understanding of the technology choices made for this project.

The practical section of the thesis consists of making a simple blood sample handling application using chosen technologies and frameworks. Some frameworks and technologies used in the project includes React, Java Spring Boot, Hibernate Tanstack Query, Node.js and Material UI. In this thesis a detailed explanation of this development process is explained.

The result of the development project was a functional web program, which met the goals set for it in the beginning. Additionally, throughout the course of this project, I gained programming experience that significantly contributed to my personal growth and learning.

Keywords React, java Spring Boot, web-application, laboratory
Pages 28 pages

Contents

1 Introduction 1

2 Project Background 1

3 App Concept 2

3.1 Users 3

3.2 Project Goals 3

4 Technologies and Frameworks 4

4.1 Backend 4

4.1.1 Java Spring Boot 5

4.1.2 Hibernate 5

4.1.3 PostgreSQL 6

4.2 Frontend 6

4.2.1 React.js 7

4.2.2 Typescript 7

4.2.3 TanStack Query 8

4.2.4 Node.js 9

4.2.5 Material UI 9

5 Laboratory Sample Handling Application 10

5.1 Database Planning 10

5.2 Backend Implementation 11

5.2.1 PostgreSQL database creation 11

5.2.2 Creating Java Spring Project 12

5.2.3 Object Oriented Mapping with Hibernate 14

5.2.4 Data Transfer Objects and Mapping 14

5.2.5 Repositories, Services and Controllers 16

5.2.6 Handling user verification 18

5.3 Frontend Implementation 19

5.3.1 Creating a Node.js project 19

5.3.2 Components and Routes 19

5.3.3 Managing State with Tanstack Query Cache 22

5.3.4 Mutating data with Tanstack query 24

6 Conclusion 25

References 27

1 Introduction

This thesis involves a practical software development project aimed at building a healthcare application using modern web development technologies. The resulting application is a responsive web application. The main objective of this thesis project was to enhance my programming skills and deepen my knowledge off Java Spring and React frameworks, thereby contributing to my own professional development. Additionally, a secondary objective was to create a fully functional application in a proof-of-concept fashion. I chose the subject of the application based on my personal interests and background.

In the theoretical section of this thesis, the programming languages, frameworks, and tools employed in the project are introduced. Additionally, I outline the project's objectives and provide some background context. The practical part consists of the software development process and planning of the application. In the conclusion, I evaluate the overall success of the project, and assess the different stages. I analyze how effectively the project goals were accomplished and which areas require further development.

2 Project Background

I have years of experience working in healthcare as a medical laboratory scientist and that is the reason I have a special interest in that field. I recall my first experience working in a medical laboratory back in the beginning of 2010s. The software used was an old Unix program with very basic text-based user interface, that did not even accept mouse inputs. As a novice medical laboratory scientist, I found it neither user-friendly nor easy to learn.

Matters have improved since then, but there is still a significant amount of outdated software being used in the healthcare industry. The healthcare industry has much potential for software development and modernization, and that is why I find it an interesting field to take as a subject for my application. For that reason, I chose to create a simple clinical laboratory sample handling application with the primary goal of developing a proof of concept. This proof of concept will serve to showcase the potential benefits of using modern frameworks in a healthcare application.

3 App Concept

In this chapter I introduce the concept and objectives that I had for the application before starting the application development project. The main concept for the application was that it could be used for monitoring and altering the state of clinical laboratory samples. Depending on their roles users could create new tests, change the state of the tests and examine the results of the completed tests.

Proper handling of samples is crucial in a clinical laboratory to maintain sample integrity and to ensure that patients receive accurate results. The different states of the sample should be recorded, and the history of the sample should be trackable. Since patient health information is confidential, access to various types of information should be carefully controlled For that reason, I decided the following objectives: The application should have different user roles with different permissions. It should also offer some basic user authentication to differentiate between users. The application should have database for storing data, and the users of the application could make different queries and mutations to data depending on their role and permissions.

Important objective in this development project was also to ensure seamless integration between the database, backend, and responsive frontend components in order to achieve accurate and efficient data handling and updating. This thesis project primarily emphasized functionality, and aspects such as visual design, information security, and rapid loading times were considered secondary due to limited time available for project.

Additionally, there were some limitations that had to be set when determining the goals for this project. Clinical laboratories manage a wide variety of samples, analyses, and tests. Due to there being a wide number of sample types and tests available commercially, limitations had to be set for this project. Consequently, I decided to develop a blood sample app and offer only a limited number of the most common blood tests.

Also in real life situations, blood samples are analyzed using specialized blood analyzers. In a commercial app, a plugin for each type of analyzer would need to be developed to transfer the results from the analyzer to the sample handling application. Since real analyzers could not be used while developing this application, an algorithm was created to produce mock data which served as the result data.

3.1 Users

The user roles in the application are as follows: the application includes three user roles - doctor, medical laboratory scientist, and customer. Each of these roles should have distinct permissions within the application. Doctors are responsible for patient care and should have permission to make test referrals and to inspect the test results. The medical laboratory scientists are responsible for the sample taking and analysis and should be able to see the referred tests and change the state of the samples. The patient or customer should have permission to see their own sample data, and they should be able to see their referred tests and the results of tests that have been analyzed. Customers should not have permissions to manipulate any sample data or to see any other data than their own..

3.2 Project Goals

The goal in this thesis project was to produce an app which will have the basic functionalities needed in the process of handling a blood sample. To achieve the goal and to align with the concepts described earlier it was decided that the application should have at least the following functionalities:

- The user should be able to login with their user credentials.
- There should be three roles created within the application: doctor, medical laboratory scientist and customer. Each of these roles have different rights within the program.
- The doctor should be able to create tests for the customers. He should be able to create a referral which consists of one or multiple tests. He should be able to see customers' referrals, tests and results.
- There should be search functionality to search customers by social security number, and then to see their tests ordered by creation date.
- The medical laboratory scientist should be able to search patients and their tests, and to change their state.
- The tests should have states of referred, taken, and analyzed.
- When the state of the test is set to analyzed, a result for it should be generated and visible.
- A mock data is created to function as a result.
- The customer can inspect his own prescribed tests and their results.

Additionally following features were considered as possible features which could be included in the application if the schedule permitted:

- More advanced search and filter mechanisms, tests could be searched by date, type, customer, or state. Also, the customers could be searched by their full name.
- When inspecting test results, thy could be compared to the customers earlier results.
- Implementing Keycloak or similar for more secure user management and using HTTPS protocol.

Due to limited time, there were some aspects that I chose intentionally to leave out from the application. The ability to create new users with GUI was excluded and the app uses readymade test users. Also, the password changing functionality was left out of this projects scope. Furthermore, in real life the sample analysis or handling can fail, and therefore no result for that particular test can be generated. There will always be a certain percent of failing tests, but the possibility of sample errors and their handling was left out of this project's scope. When creating the mock test results within this application, the tests always succeed, and a result will be created.

As mentioned before, applications in healthcare industry are often handling very sensitive information such as people's health information and history. The information security in the app should be high to protect sensitive information from unauthorized parties. Developing a high level of information security can however be very time consuming. Since my aim in this project was to provide a program to demonstrate the functionalities needed in blood sample handling process, the information security was implemented only at a basic level.

4 Technologies and Frameworks

The significant part of software development is based on using different frameworks and ready-made libraries. Using them is almost compulsory nowadays since it would be highly ineffective to produce all functionalities and components from scratch. Here I will introduce the different technologies and libraries chosen for this development project.

4.1 Backend

For the backend Java Spring Boot was used. For data handling Hibernate was used to provide object related mapping. Database is a PostgreSQL relational database.

4.1.1 Java Spring Boot

Java Spring Boot is an open-source application framework built on top of the Java Spring Framework. Java Spring framework is an application framework, offering lightweight alternative to Java Enterprise Edition (Java EE). It provides features to help with the software development such as dependency injection and aspect-oriented programming. The configuration of Spring Framework was traditionally done using XML, which can be laborious at times. To simplify configuration process of Java Spring framework an extension called Java Spring Boot was made. Java Spring Boot could, among other features, provide an automatic configuration for most common Java Spring applications. (Walls, 2016, pp. 1-4)

There are alternatives for Java Spring Boot, such as Play-framework. According to the 2022 State of Developer Ecosystem Survey by JetBrains Java Spring is however significantly more commonly used than any other Java framework, with over 67% of respondent Java developers using it. (JetBrains 2022) This means that more documentation and instructions for Java Spring is available than for other frameworks. My familiarity with Java Spring Boot, and the fact that Java Spring is the most used Java framework, made me choose it as backend framework for this project.

4.1.2 Hibernate

Hibernate is an open-source Object-Relational Mapping (ORM) framework designed for Java applications. The term Object/Relational Mapping refers to the process of converting data between an object model representation and a relational data model representation, and vice-versa. ORM adds an abstraction layer over the database, allowing developers to work with objects and classes instead of tables and columns. Hibernate provides a high performance and scalable solution for mapping Java objects to SQL tables and mapping from Java data types to SQL data types. It allows developers to interact with databases using object-oriented programming concepts and Hibernates data query methods instead of writing traditional SQL queries. (Hibernate n.d.)

In my previous projects, I have found ORM to be a simpler and more flexible option than traditional SQL and that is why I chose to use an ORM framework in this project as well. With an ORM, implementing a CRUD layer for interacting with the database and even more complex data queries can be made simpler.

The Hibernate is widely used, popular ORM-framework for Java. There are alternatives such as OpenJpa, TopLink and EclipseLink. These are all Java Persistence API frameworks as is Hibernate. However, as I am most familiar with Hibernate, I have chosen to use it rather than familiarizing myself with new framework.

4.1.3 PostgreSQL

PostgreSQL is an open-source object-relational database system. Object-relational database system is an extension to traditional related database management system. It stores data in tables and uses SQL for querying, but also has support for some object-oriented characteristics. (Rouse 2013) PostgreSQL supports various functionalities such as custom data types and user defined functions, making it more flexible than pure relational database system. It still supports most features required by SQL standard. As of version 15, released in October 2022, PostgreSQL complies with at least 170 out of the 179 mandatory features for SQL:2016 Core conformance, demonstrating its compatibility to the SQL standard. (PostgreSQL, n.d.)

Alternatives to PostgreSQL include other relational database systems such as MariaDB and MySQL. Both are quite popular and could potentially be used for this project. However, I prefer the flexibility of PostgreSQL, as MySQL, for example, does not support custom data types and other customizations that PostgreSQL offers. While these features may not be crucial during the current thesis project, they could potentially become important if the project were to be further developed.

4.2 Frontend

For the frontend I decided to use React, since that is the frontend framework, I have the most experience with. Typescript was additionally used to enable types and type error checking. TanstacktQuery is a useful library for React which offers easy data synchronization and state management. To help in the creation of graphic user interface, the Material UI was used.

4.2.1 React.js

React is a popular and widely used framework to build web applications. In the ”State of JavaScript 2022”-survey, with over 39 000 web developer respondents, React was the most popular front-end framework(State Of Js, 2022) It is also the framework used in many widely popular websites, namely Facebook, Instagram, and Netflix. (Arancio, 2021)

React utilizes component-based architecture to produce easily maintainable and reusable code. Components can be used throughout the application reducing code duplication and making the application more manageable and scalable. (Nandaniya, 2023) Component based approach also allows developers to work with individual components without affecting other developers work.

Another significant benefit of React is its efficient handling of the Document Object Model (DOM). React employs a virtual DOM, virtual representation of the actual DOM. Manipulating the actual DOM is slow, but manipulating virtual DOM is much faster. By comparing virtual DOM objects to objects in actual DOM, React can decide which of the React components have been updated and update only those to actual DOM. The result is enhanced performance. (Okoro 2023).

4.2.2 Typescript

Typescript has grown increasingly popular and in the State of JavaScript survey 68% of web developers who answered said they have used Typescript. Compared to year 2016 where only 21% of the developers had used Typescript, that is a quite large increase. (State Of Js, 2022)

JavaScript does not provide a way to specify the types of variables or function arguments, making it difficult to catch type-related errors. This can make the bug fixing and debugging of run-time errors hard. TypeScript was created as an answer for this situation, and it allows developers to assign types to variables when developing JavaScript applications. Typed variables also improve code readability. (McKenzie, 2023) Typescript also supports more advanced type features, such as creating of interfaces or making union types making the use of types very flexible. As a result, it enhances code readability, maintainability and makes debugging easier.

Some of the competitors for Typescript include Flow, developed by Facebook, and Dart, created by Google. Flow is a static type checker for JavaScript. It aims to provide better type safety and catch potential type-errors during the development with static type annotations. (Flow, n.d.) Dart on the other hand is entirely separate language. Dart comes with its own libraries, tools, and syntax and static typing (Dart, n.d.) Due to my familiarity with Typescript and its easy compatibility with React I will be using Typescript in this project.

4.2.3 TanStack Query

TanStack Query, also known as React Query is a library for React to efficiently handle data fetching and synchronization. Tanstack Query will automatically create cache for fetched data, and provides a set of hooks and other functionalities, which can be used to manage and manipulate data cache.

Tanstack Query guide states that “A query is a declarative dependency on an asynchronous source of data that is tied to a unique key.” To use the useQuery hook, an unique key, and a Promise-based function to retrieve data from the server are provided. TanStack Query then manages fetching, caching, and sharing queries throughout the application (TanStack Query, n.d..-a). The useMutation hook can be used to perform mutations in the database by providing a Promise-based function for the desired data manipulation.

TanStack Query also offers other useful tools such as setQuery, invalidateQueries and removeQueries to manage the cached data after the data in the database has been changed. SetQuery can be used to manually edit the local query state. InvalidateQueries can be used to invalidate a cached query and command TanStack Query to refetch data with corresponding queryKey. (TanStack Query, n.d.-b). RemoveQueries, on the other hand, can be used to remove queries from cache, which is particularly helpful with deletions.

These examples highlight only a few of TanStack Query's basic functionalities. The library also supports more complex scenarios and provides tools for handling pagination and infinite scrolling. (TanStack Query, Query Client, n.d. -c). Having used TanStackQuery in a previous project, I have found it especially useful for managing and sharing data between the database and React components. For that reason, I have chosen to use it in this project.

4.2.4 Node.js

Node.js is an environment based on Google Chrome's JavaScript engine that enables developers to run JavaScript code on the server-side. Typically, developers have used different languages for frontend development (e.g., JavaScript) and backend development (e.g., Java, Python). However, Node.js makes it possible to use one language (JavaScript) in both client and server side. (Wexler, 2019, p.5)

As I have chosen to use Java and Spring Boot in the server-side of this project, my reasons for including Node.js to this project are different. Node.js includes npm package manager, a powerful tool which enables easy installation and management of packages and dependencies in the project. Node.js is highly popular and has large community which means vast repository of available packages. Npm also serves as a command line tool which you can use for easy dependency management and easy project initialization.

4.2.5 Material UI

The Material UI is an open source React component library with collection of prebuilt components ready for use. The components are easily customizable, and a theming system allows developers to define global styles and color schemes that can be applied consistently across the entire application. (Mui, n.d.).

Material UIs adheres to Google's Material Design guidelines ensures that the components and styles provided are consistent, intuitive, and visually appealing. MUI also adheres to Web Content Accessibility Guidelines (WCAG) and components are designed to be accessible. (Manik, 2022) It would be very time consuming to develop professionally looking and accessible visual components from scratch and that is why using MUI to implement visual components in my project will be beneficial.

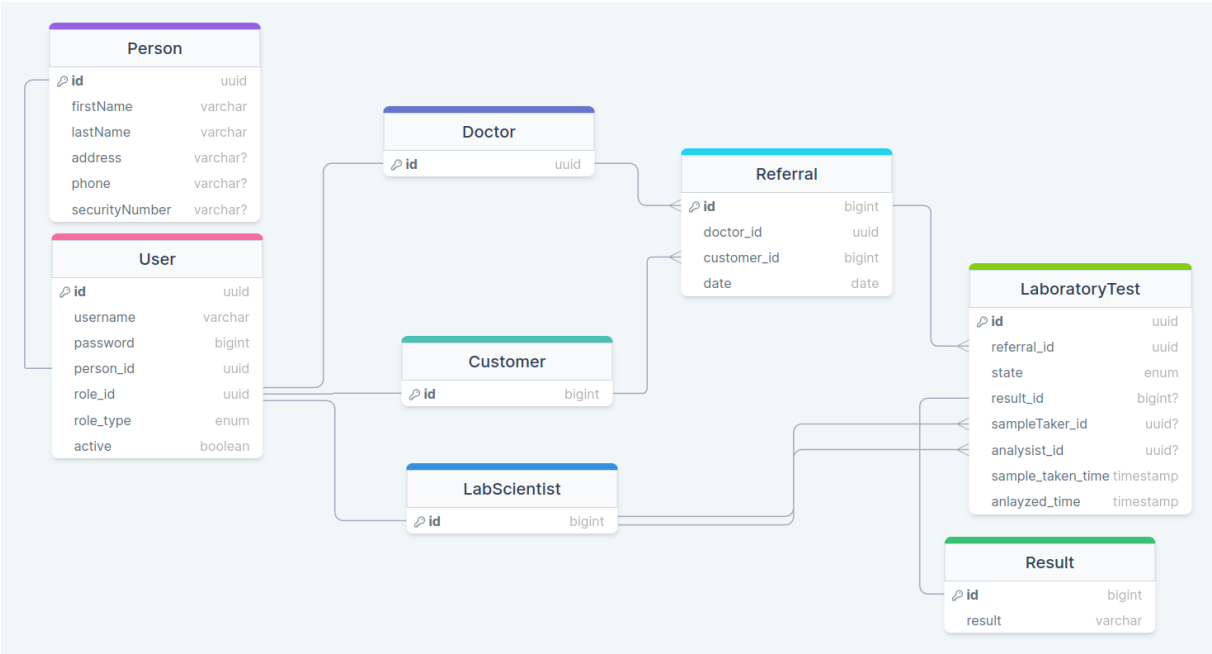
5 Laboratory Sample Handling Application

In this chapter I will describe the process of planning and implementing the app project associated with this thesis. I will begin by describing planning and creation of database and then continue with configuring and implementation of database. Lastly, I will describe the frontend, its components and communication with database and data handling.

5.1 Database Planning

To plan the database structure an entity relationship diagram (ERD) was created. ERD is a visual representation used to model and describe the relationships between the different entities. All the attributes that belong to different entities are listed there. The ERD diagram is represented in Figure1.

Figure 1 ERD



In the ERD there is entity called User which contains information about the user of the application, such as their username and role. User has a one-to-one relationship with the Person entity, which holds personal information about the user, such as their name, address, and phone number. Each username is associated with one role, and roles regulate the user's access rights within the application.

The user can only have one role. In real life, it might be possible that same person would have separate roles within the application, for example the same person could be a doctor, but also a customer. But to ensure information security, multiple roles cannot be connected to the same user. If a person needs different access rights within the application, multiple accounts with different roles will have to be created. The Person entity could be connected to multiple User entities in real life scenario, but for simplicity and this project's scope, here each Person will only have one User connected to it.

Referral table contains all the test referrals, along with information about the doctor who made the referral and the customer for whom the referral was made. A referral also has one-to many relationships to LaboratoryTest entity, which means that under one referral there can be multiple tests.

The LaboratoryTest entity includes a state attribute that controls the state of the test, such as the times the test is processed and whether the results are visible to the customer. Additionally, the LaboratoryTest entity has information of the laboratory scientist who took the sample and the analyst who analyzed it. Finally, there is a one-to-one relationship between the LaboratoryTest entity and the Result entity. The result entity holds information about the result of the prescribed test.

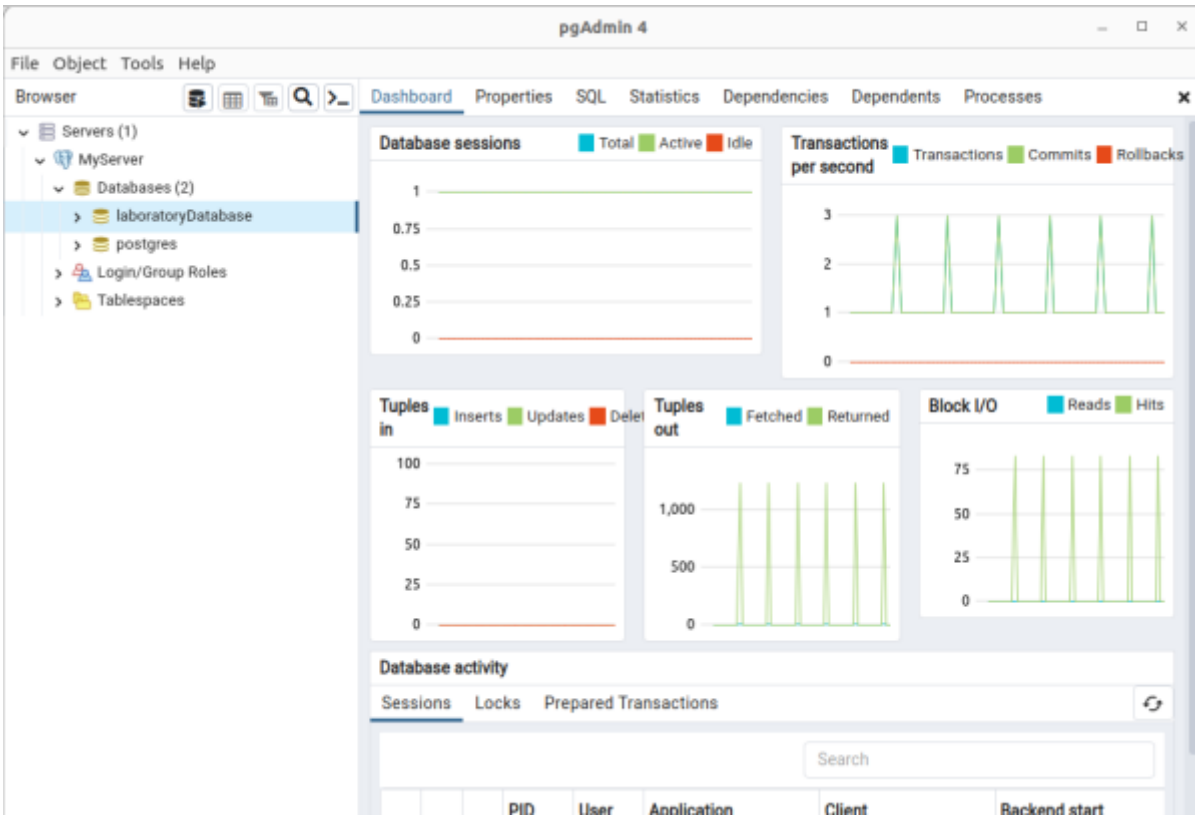
5.2 Backend Implementation

In this chapter I will describe in detail the creation and implementation of backend for my sample handling app.

5.2.1 PostgreSQL database creation

The first practical step I did when starting the development of the software was to set up the database. I installed PostgreSQL and created a local server and a database using a command line and PgAdmin. PGAdmin is an administrator and management tool for PostgreSQL database, and it provides graphical user interface that allows easy creation and managment of databases (Figure 2). As Hibernate is used to handle data mapping and database table creation in the project, pgAdmin does not have a major role in the project. PgAdmin however can be very useful if any debugging is needed, as it is possible to use it to make SQL queries and inspect database.

Figure 2 Example of PgAdmin view.



5.2.2 Creating Java Spring Project

A new Java Spring project is relatively easy to set up using Spring Initializr. Configuring a new project without any helper tools can be laborious. Spring Initializr is an online tool created by Spring development team that simplifies the process of creating and configuring a Spring project (Figure 3). Using Spring Initializr user can select project type, language, build system (Maven or Gradle) and add various dependencies. Then a basic project structure with necessary dependencies based on the user's preferences can be generated. After initializing the project, a connection to local PostgreSQL database was created through IntelliJ IDE (Figure 4).

Figure 3 Example of Spring Initializr

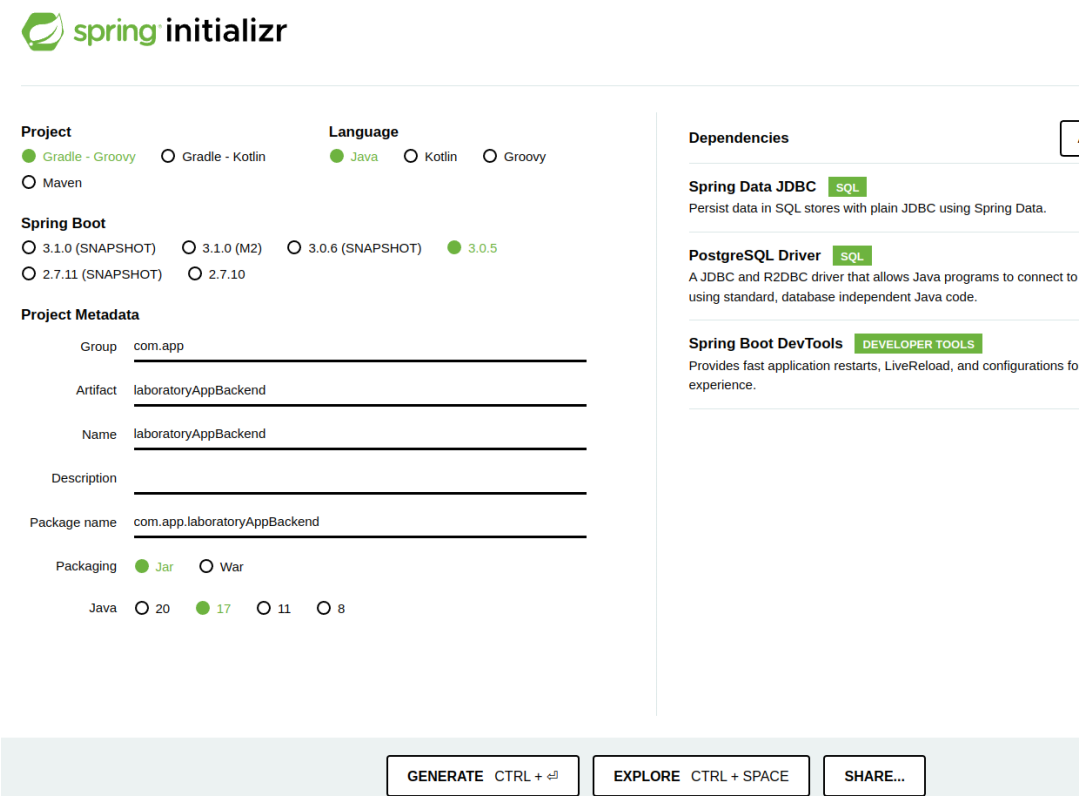
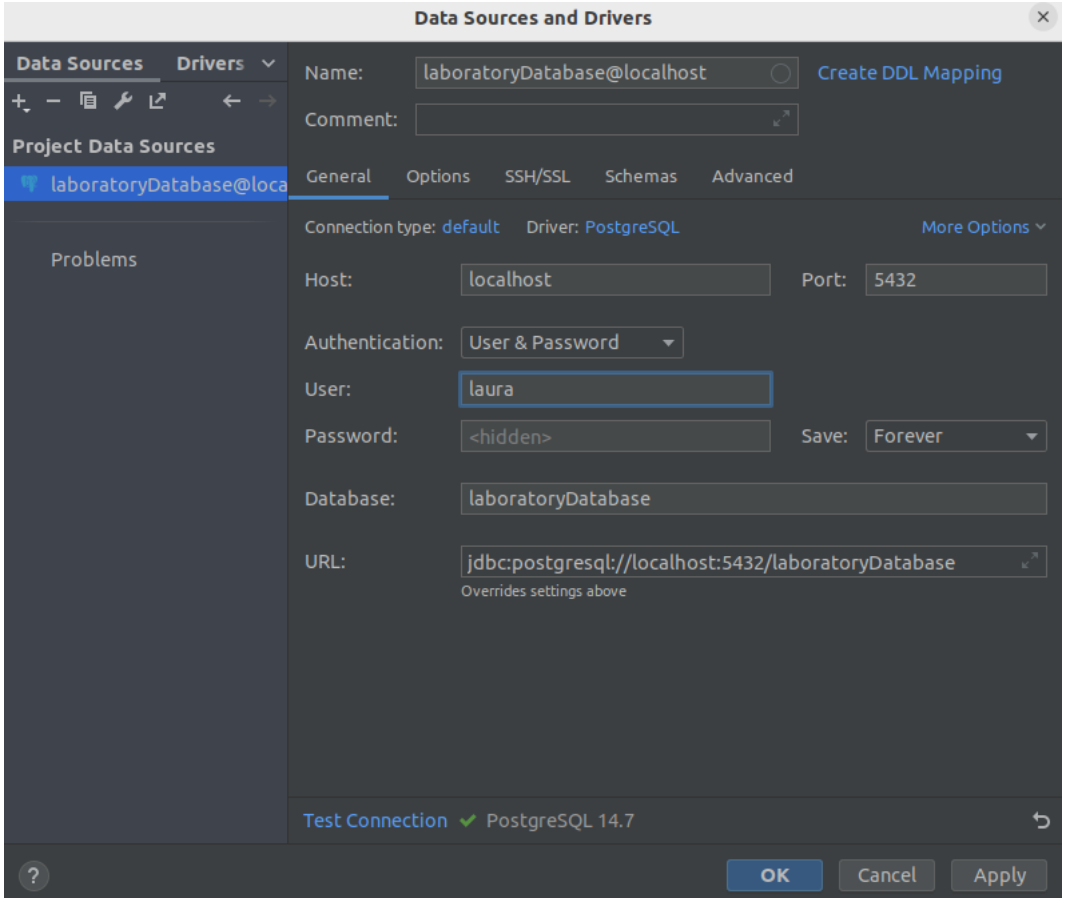


Figure 4 Creating a local database connection in intelliJ.



5.2.3 Object Oriented Mapping with Hibernate

At first the entity models and their relations were created according to the ERD diagram. I also used library called Lombok while writing entities. Lombok is a Java library that aims to reduce the amount of boilerplate code and it can be used to generate for example getters, setters, and constructors. With Hibernate and Lombok it is fairly easy to make the basic models and their relationships. Also, in the database there was no need to any many-to-many relationships which made matters simpler, since there was no need for joining tables, although Hibernate handles forming those quite well. Below is a picture of example model made with Hibernate. (Figure 5) Here we can see how all the columns in database table have corresponding variable or entity assigned inside the model object. Also, the relationships to other tables are described. After models had been made, SQL file was used to inject some test data to database.

Figure 5 Model for Referral

```
@Entity
@Getter
@Setter
public class Referral {
    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    private UUID id;
    @Column(nullable = false)
    private Date date;
    @ManyToOne
    @JoinColumn(name = "doctor_id",  nullable = false)
    private Doctor doctor;
    @ManyToOne
    @JoinColumn(name = "customer_id",  nullable = false)
    private Customer customer;
    @OneToMany(mappedBy = "referral")
    private Set<LaboratoryTest> tests;
}
```

5.2.4 Data Transfer Objects and Mapping

Communication between backend and frontend is done in JSON format. To map the JSON data to objects that Hibernate can understand and vice versa, separate mapping is needed. Data transfer objects (DTOs) are needed to function as a model for a mapper to describe how the conversion of entities and data should be handled when converting it to JSON and back.

JSON format consists of key-value pairs, where keys are strings and values can be strings, numbers, objects, arrays, Booleans, or null. That means that when making DTO based on certain data model the variables should be converted to these primitive types. As an example, in Figure 6 the DTO of Referral from this project is presented. Here we can see that Doctor entity for example has been replaced by doctor_id as string.

Figure 6 Data transfer object for Referral

```
@JsonInclude(JsonInclude.Include.NON_NULL)
@Data
public class ReferralDTO {
    private String id;
    private String date;
    private String doctor_name;
    private String doctor_id;
    private String customer_id;
}
```

After DTOs have been created, a mapper is needed to make the conversion between data transfer objects and Hibernate entities. There are several libraries that can be used to automate mapping process and reduce boilerplate code that is often a result of manual mapping. They also make complex mapping such as mapping nested objects etc. easier. For these reasons, I used MapStruct mapping library in this project. The instructions MapStruct needs to make the conversions are created inside a mapper and MapStruct will handle rest of the mapping. The mapper created for the ReferralDTO is presented in Figure 7.

In ReferralMapper clas several functions were implemented to convert ReferralDTO to Referral and vice versa. Above these functions instructions for MapStruct for handling conversions are provided. For simple conversion specifying only source and target is sufficient. For more complex conversions a helper function is needed to assist with the mapping. In ReferralMapper an example of helper function would be ”mapIdToDoctor” which instructs MapStruct on how to convert doctor id to Doctor entity. In this case DoctorService layer is called to find Doctor entity which corresponds the id.

Also, when making DTOs it is important to consider what data we want to pass to frontend and what data we are expecting. DTOs should only contain relevant information. In the mapping process, the fields that are not required can be marked as ignored.

Figure 7 Mapper for Referral entity.

```
@Mapper(componentModel = "spring")
public abstract class ReferralMapper {

    @Autowired
    protected DoctorService doctorService;
    @Autowired
    protected CustomerService customerService;
    @Autowired
    protected AppUserService appUserService;

    @Mapping(target = "doctor_name", source = "doctor.id", qualifiedByName =
"doctorIdToDoctorName")
    @Mapping(target = "customer_id", source = "customer.id")
    @Mapping(target = "doctor_id", source = "doctor.id")
          public abstract ReferralDTO referralToReferralDto(Referral referral);

    public abstract Collection<ReferralDTO> referralToReferralDto(Collec-
tion<Referral> referrals);

    @Mapping(target = "doctor", source = "doctor_id", qualifiedByName = "doc-
torIdToDoctor")
    @Mapping(target = "customer", source = "customer_id", qualifiedByName =
"customerIdToCustomer")
    @Mapping(target = "tests", ignore = true)
    public abstract Referral referralDtoToReferral(ReferralDTO dto);

    @Named("doctorIdToDoctor")
    Doctor mapIdToDoctor(String id) {
        return doctorService.findById(UUID.fromString(id));
    }

    @Named("doctorIdToDoctorName")
    String mapIdToDoctorName(String id) {
    Doctor doctor = doctorService.findById(UUID.fromString(id));
    String doctorName = appUserService.getUserPersonData(doctor.getAp-
pUser().getId()).getFirstName() + " " +appUserService.getUserPersonData(doc-
tor.getAppUser().getId()).getLastName();
        return doctorName;
    }

    @Named("customerIdToCustomer")
    Customer mapIdToCustomer(String id) {
        return customerService.findById(UUID.fromString(id));
    }
```

5.2.5 Repositories, Services and Controllers

To keep the code maintainable, several layers were created to separate the backend application logic. At first repository interface layer was created. A repository interface is responsible for interacting with the data access layer of the application: in this case PostgreSQL database. This interaction is handled with Hibernate and using models created previously. Spring Data offers JPA repository interface, which provides a set of methods for performing CRUD (Create, Read, Update, Delete) and other most common operations on a database table. I created repository interfaces for all the entities and made them all extend JPA repository. Below are a couple of example pictures of repositories in this project and different queries that can be made within repository layer. (Figure 8 and Figure 9)

Figure 8 CustomerRepsoitory and an example of query using @Query annotation from Spring Data JPA. This closely resembles native SQL query.

```
public interface CustomerRepository  extends JpaRepository<Customer, UUID> {
    @Query("SELECT c FROM Customer c JOIN c.appUser u JOIN u.person p WHERE
p.securityNumber = ?1")
    Customer findBySecurityNumber(String securityNumber);
}
```

Figure 9 Making queries using JPA feature called "Derived Query Methods". Wit this feature, queries can be created simply by using certain naming conventions in function names.

```
public interface LaboratoryTestRepository extends JpaRepository<Laboratory-
Test, UUID> {
    List<LaboratoryTest> findAllByReferralId(UUID referralId);
    List<LaboratoryTest> findAllByReferralIdInOrderByRefer-
ral_DateDesc(List<UUID> referralIds);

}
```

The service layer is an abstraction layer between repository and a controller. Service layer is commonly used for implementing the business logic in backend application and is used for more complex handling of data. At first a CrudService interface was created which encapsulates the common CRUD operations. Then other services were created extend the CrudService. The reason this was done is to make sure that within implementation of each service all CRUD operations will be implemented even if they are not needed at present. This makes implementing more complex queries and scaling up the project easier in the future.

Controller layer was created to be the layer responsible of communicating with the frontend. Inside controllers the endpoints that are callable from the frontend were created. With CRUD

implemented in services there exists basic functionalities to create, edit and delete data from database. Despite that the endpoints to expose these functionalities to frontend should not be created unnecessarily and will only be created as they are needed. For example, the delete operations should be used with extreme caution in the project. As this is a laboratory app, data should not be deleted without careful deliberation, and all actions must be traceable to maintain data integrity.

5.2.6 Handling user verification

The app uses token-based user verification. When a user logs in with the correct credentials, the backend generates a JSON Web Token (JWT) and returns it as a response. When logged in user makes then requests, the token is sent within Request Header. The backend then decodes the token and determines whether user has privileges to perform specific actions. For example, the backend may use the token to identify the user's roles and see if user has rights to access customer data or create new referrals (Figure 10). This approach protects the endpoints of api as access to information is limited to those with correct token.

This is however a simple user verification and by means not sufficient enough for an application which handles sensitive user information as customers' health information. Data transfer itself, for instance, lacks encryption such as SSL/TLS. The transfer of data using HTTPS protocol would be much preferred. These however were intentionally omitted from this project due to time limits and this is the implementation that will suffice in this context.

Figure 10 Example of token-based authorization. When user is adding new referral, backend will check the token by calling designed function authenticateDoctorToken(authHeader). This will return UNAUTHORIZED if user role type is not doctor.

```
 @PostMapping("/{customerId}/add")
    public ResponseEntity createReferral(@PathVariable UUID customerId, @Re-
questBody String[] testTypes,
                                                @RequestHeader(value="Authoriza-
tion", required = false) String authHeader) {


        ResponseEntity<String> authenticationResult = this.jwtTokenUtil.au-
thenticateDoctorToken(authHeader);
        if (authenticationResult != null) {
            return authenticationResult;
        }

        Referral referral = new Referral();
        referral.setDate(new Date());

        String bearerToken = jwtTokenUtil.getToken(authHeader);

        String username = jwtTokenUtil.getUsernameFromToken(bearerToken);
        UUID doctorId = appUserService.findDoctorIdByAppUse-
rUsername(username);

        if(doctorId == null){
            return ResponseEntity.status(400).body("Doctor associated with
given id was not found.");
        }

        referral.setDoctor(doctorService.findById(doctorId));
        referral.setCustomer(customerService.findById(customerId));

        referralService.add(referral);
        URI uri = URI.create("referrals/" + referral.getId());

        for (String testType : testTypes) {
            LaboratoryTest test = new LaboratoryTest();
            test.setState(State.REFERRED);
            test.setTestType(TestType.fromValue(testType));
            test.setReferral(referral);
            laboratoryTestService.add(test);
        }
        return ResponseEntity.created(uri).body(referralMapper.referralToRe-
ferralDto(referral));
```

5.3 Frontend Implementation

This chapter will describe the practical implementation of the frontend.

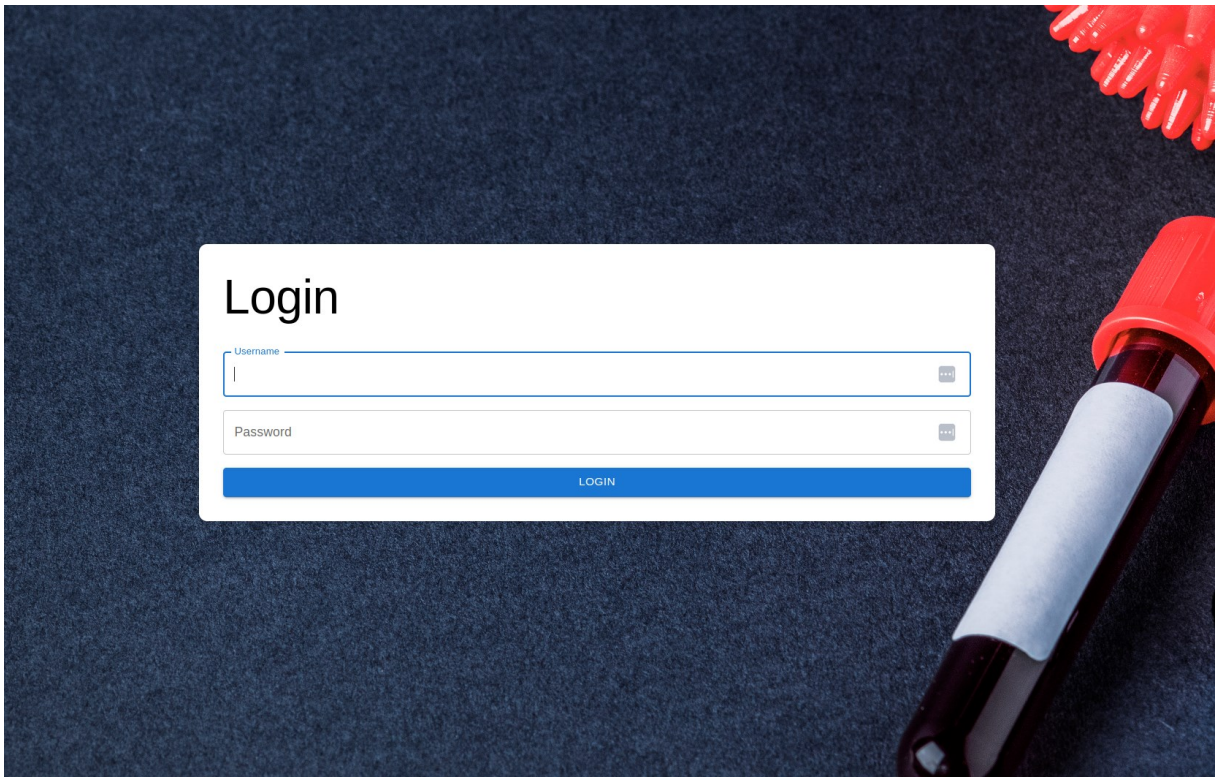
5.3.1 Creating a Node.js project

Create.react-app is a node package created by Facebook which allows easy basic initialization for Node.js project and with TypeScript template. This was used to create initial project structure and a package.json file with basic settings. After the project was initialized, the other dependencies used in the project such as React, Tanstack Query, TypeScript and MUI were installed with Node Package Manager.

5.3.2 Components and Routes

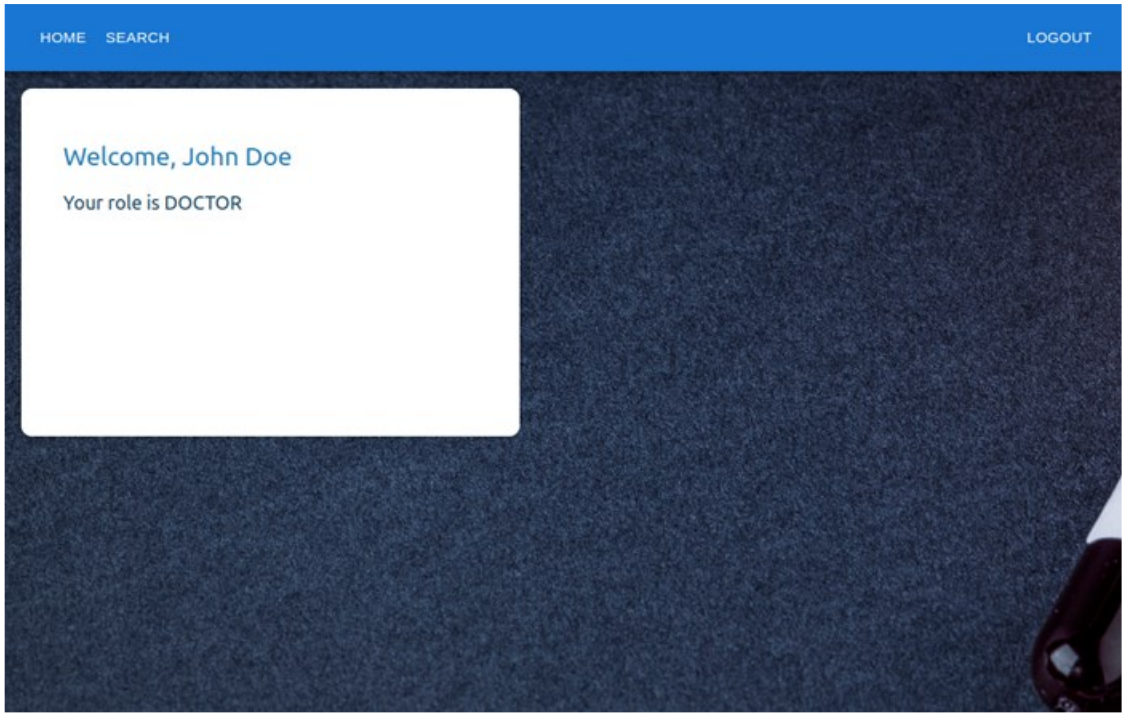
React router was used to direct user on different places inside the application. The routes available are dependent on user roles. The app has simple user login. If user is not logged in the app opens to a login screen (Figure 11).

Figure 11 Login screen



If user is logged in successfully the home screen and navigation bar will be visible. (Figure 12) With navigation user can move to different places within the app.

Figure 12 Home screen



If user is of role doctor or laboratory scientist, the customer search will be visible. Here the doctor or lab scientist can search customer based on their social security number (Figure 13). If customer is found, user can see what customers' referrals, tests and test states. Also results from completed tests can be seen. The doctor can also prescribe new tests using new referral form, and lab scientist can change test states. (Figure 14, Figure 15)

If user is logged in as customer the options will be more limited, as customer cannot access customer search. Instead, a tests view displaying tests prescribed for the customer and their states and results if available will be shown.

Figure 13 Customer search view

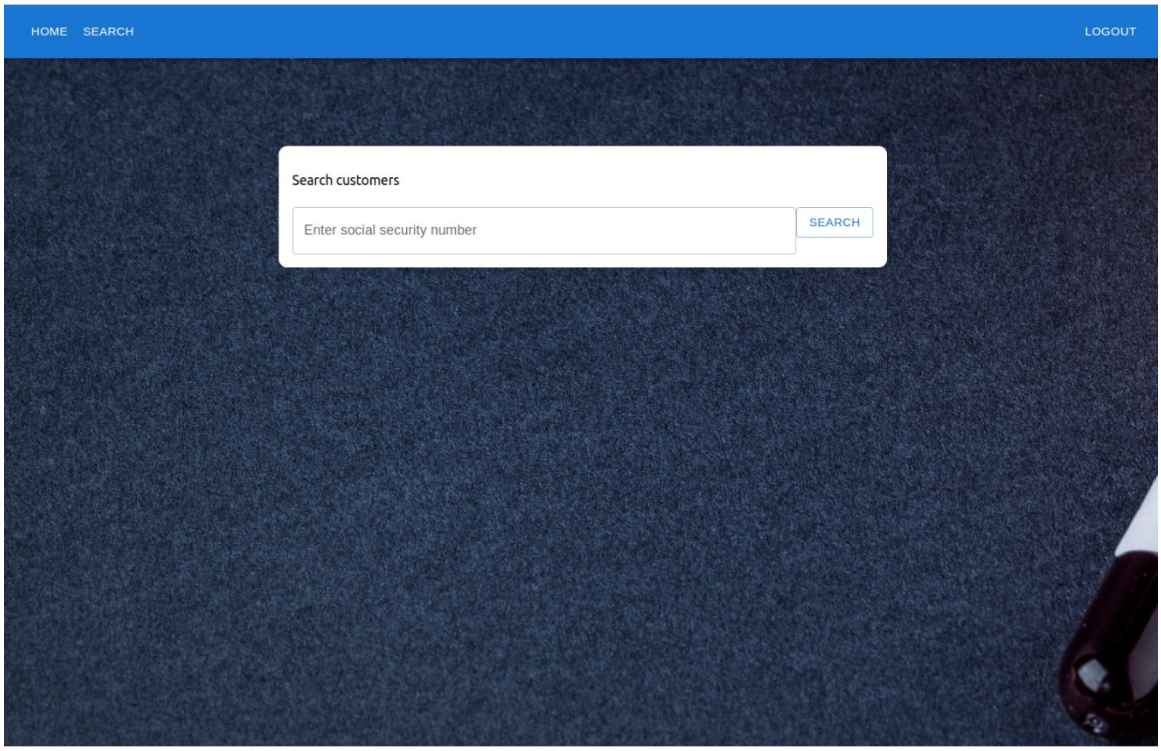
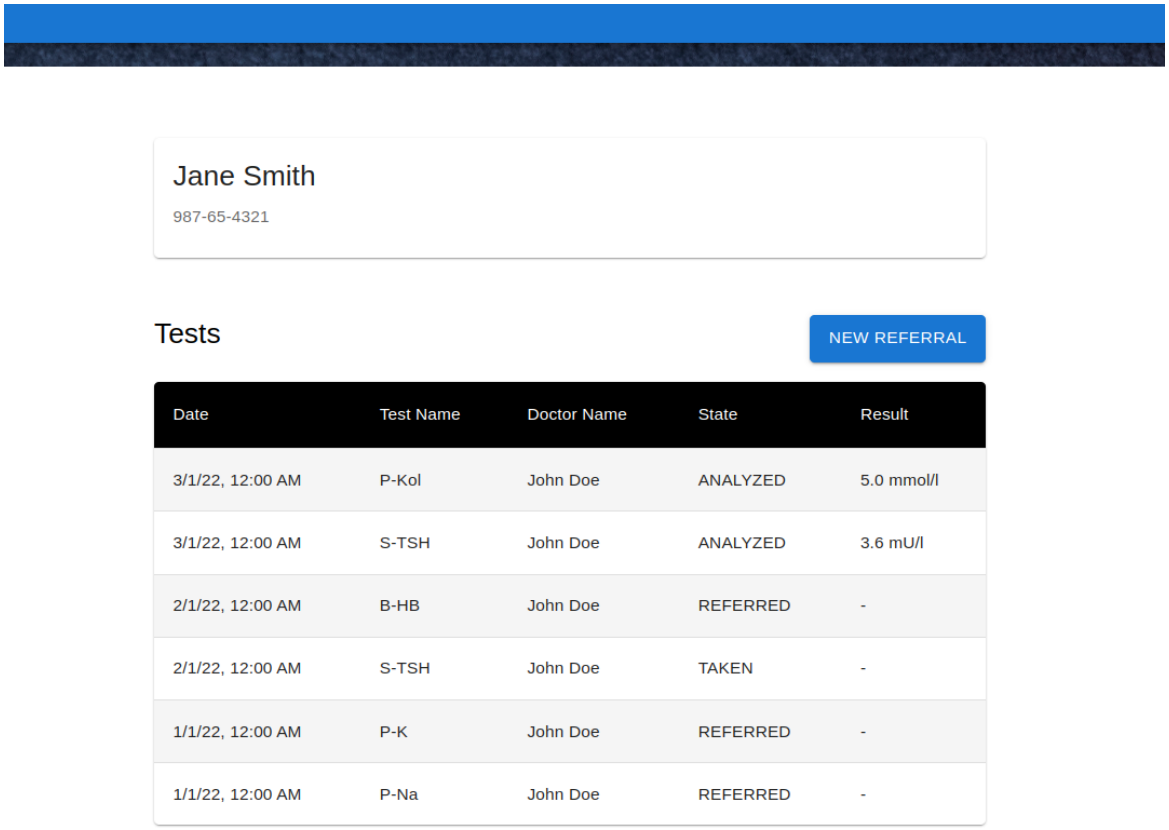
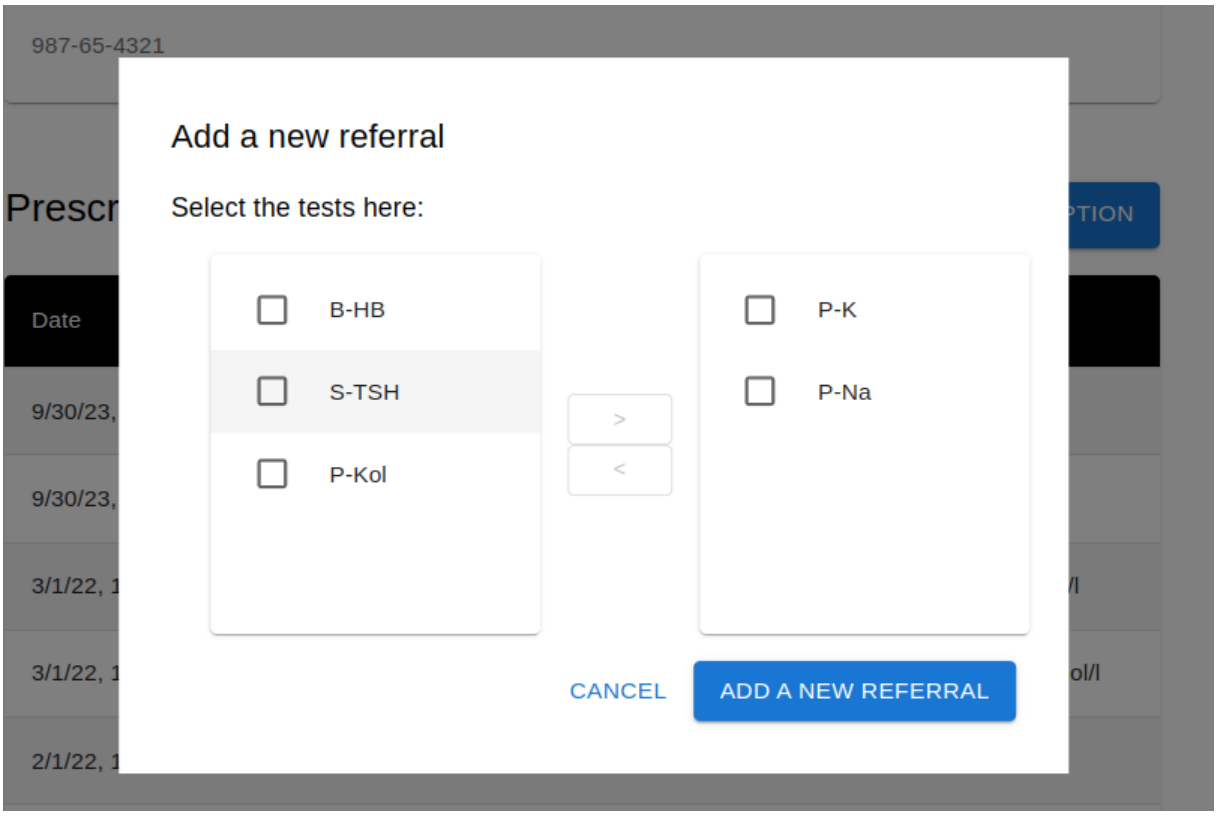


Figure 14 The tests view for customer when user is logged in as doctor.



Date	Test Name	Doctor Name	State	Result
3/1/22, 12:00 AM	P-Kol	John Doe	ANALYZED	5.0 mmol/l
3/1/22, 12:00 AM	S-TSH	John Doe	ANALYZED	3.6 mU/l
2/1/22, 12:00 AM	B-HB	John Doe	REFERRED	-
2/1/22, 12:00 AM	S-TSH	John Doe	TAKEN	-
1/1/22, 12:00 AM	P-K	John Doe	REFERRED	-
1/1/22, 12:00 AM	P-Na	John Doe	REFERRED	-

Figure 15 In new referral form a doctor can prescribe new tests for selected customer.



5.3.3 Managing State with Tanstack Query Cache

Tanstack query library was used in the project to manage the global state of the frontend application. With Tanstack, a range of keys and query caches were established, and data hooks were created to fetch and save data within the caches. Updates or refetching of data within a query cache would trigger automatic updates across all components that depended on that particular cache. Below in Figure 16 is an example data fetching hook.

In this example, we specify the queryKey, which serves as an unique identifier for Tanstack's query cache where the data will be stored. Additionally, the query function will be provided which performs the API call with Axios. (Figure 17). Tanstack useQuery also provides states isLoading and isError. These states are useful in handling situations when data is still loading, or an error occurs during the data-fetching process. These states can be used to display loaders or error messages inside components.

Figure 16 Data fetching hook to fetch customers referral data.

```
export const useReferralData = (customerId: UUID | undefined) => {
  const {
    data: referrals,
    isLoading,
    isError,
    error,
  } = useQuery({
    queryKey: ["referrals", { customerId }],
    queryFn: () => getCustomerReferrals({ customerId }),
  });

  return {
    referrals,
    isLoading,
    isError,
    error,
  };
};
```

Axios is a JavaScript library used for making HTTP requests to backend. Axios uses Javascript Promise objects to make asynchronous requests and manages the response. Axios library is used in this project to make GET, POST and PUT HTTP requests to the backend. Because this project utilizes TypeScript, each response and request is typed, so expected data structures for every API call are known. Tanstack query will then handle updating the local state as it will handle the query cache updates according to data and responses returned.

Figure 17 Axios library is used to make api call to get customers referrals.

```
export const getCustomerReferrals = async ({
  customerId,
}: ReferralRequest) => {
  return customerId === null || customerId === undefined
    ? Promise.reject(new Error("Invalid customerId"))
    : axios
        .get(`${BASE_URL}/api/v1/referrals/${customerId}`)
        .then((response) => response.data as ReferralResponse[]);
};
```

5.3.4 Mutating data with Tanstack query

When it comes to modifying data, Tanstack Query offers a tool called 'useMutation' hook. This hook allows us to specify a mutating function, which is a function call to modify data in the backend. After the call, the local query cache can then be updated to match the modified state of data.

One approach is to optimistically update query cache immediately after the call, even before the response has been received. In this scenario, the local state will be modified based on what we expect to be the successful result using Tanstack query tool called ‘setQuery’. If, however, the API call returns an error response, the cache can be invalidated. This triggers a data re-fetch and effectively erases any local alterations made. Optimistic data updates can produce smoother user experience because user can see the mutated data immediately and does not have to wait for call to finish. The backend then updates in the background.

In some cases, this behavior is not preferable. In cases of more complex mutations or when data consistency is a priority it is preferable to update the data only on success. Examples of this can be if redirect or navigation is performed after the mutation. Using the "onSuccess" mutation approach, the mutation hook is provided with a mutating function. The call is then made to the backend, but the data will only be updated locally if response is successful.

In Figure 18 an example of using Tanstack useMutation and ‘onSuccess’ approach is provided. In this example we have a hook to add new referral. AddNewReferral function is given as mutation function, and it is asynchronous function to make an API call using Axios. If response from this call is successful, the onSuccess scenario is executed updating the local query cache. But, in the event of error, onError scenario is executed and failed response from api call will cause Tanstack to invalidate the query cache leading to data refetch from backend.

.

Figure 18 UseMutation hook to add new referral for customer.

```
const { mutate: handleAddReferral } = useMutation({
  mutationFn: ({ customerId, testTypeValues }: NewReferralRequest) =>
    addNewReferral(
      {
        customerId,
        testTypeValues,
      },
      user?.token
    ),

  onSuccess: async (
    response: ReferralResponse,
    { customerId }: NewReferralRequest
  ) => {
    queryClient.setQueryData<ReferralResponse[]>(
      ["referrals", { customerId }],
      (previousValue) => {
        if (!previousValue) {
          return [response];
        }
        const updatedValue = [...previousValue, response];
        return updatedValue;
      }
    );
  },

  onError: async (_error, { customerId }: NewReferralRequest) => {
    await queryClient.invalidateQueries(["referrals", { customerId }]);
  },
});
```

6 Conclusion

The resulting application quite successfully met the initially established requirements. It had the required user roles, and all the users could use the basic actions described at the beginning of the project. The basic workflow is functional: doctors can search for customers and prescribe tests, while lab scientists can update the test statuses, mimicking the real-world processing of blood samples. After test is analyzed, a result is generated. A customer can then inspect her tests and see the result of her blood tests.

Regarding security, basic authentication was implemented successfully. Although more secure authentication with Keycloak and HTTPS had to been left out, I was able to implement the endpoint protection with tokens which I consider important improvement of security.

The frontend of application was the part of the project which left most room for further development. There could be more functionalities to enhance user experience and make the application more versatile. There could be more abilities for user to search and filter data. For instance, users should be able to search for customers by their full names and select the correct individual from the resulting list. There should also be options for filtering tests by name, creation date, or analysis date. Additionally, pagination should be implemented to handle long lists of tests. Also, the visual aspects of the app could be enhanced.

While the application may lack a broad range of functionalities, I am quite pleased with its structure and scalability. The planning of database and implementing the structure for backend with different layers and mapping took considerable amount of time and that set the time limit for frontend developing. But after the work was done to set the basis, adding new features to frontend and scaling up the project should be faster in the future.

I had experiences of some of these frameworks and libraries from previous projects. There were still many issues especially in backend development and setting up and configuring the project with which I was not remarkably familiar. I learned a great deal especially how to build backend with Java Spring Boot and using different layers, models and mappers to handle the data. I also became more adept at full stack developing and a gained better understanding of how to develop a software project as a whole.

References

Arancio, s. (5.8.2021). ReactJS: A brief history. Retrieved March 21, 2023, from https://medium.com/@sjarancio/reactjs-a-brief-history-3c1e969a477f

Dart (n.d.). Dart overview. Retrieved May 24, 2023, from https://dart.dev/overview

Flow. (n.d.) Getting Started. Retrieved May 24, 2023, from https://flow.org/en/docs/getting-started/

Hibernate. (n.d.). Hibernate Getting Started Guide. Version 6.2.0.CR4. Retrieved March 18, 2023, from https://docs.jboss.org/hibernate/orm/6.2/quickstart/html_single/.

JetBrains. (2022). The State of Developer Ecosystem 2022. https://www.jetbrains.com/lp/devecosystem-2022/

Manik, S. (2022). What is MUI and what do you need to know about it?. Retrieved March 5, 2023, from https://talent500.co/blog/what-is-mui-and-what-do-you-need-to-know-about-it/

McKenzie, C. 2023. JavaScript vs. TypeScript: What's the difference?. Retrieved May 22, 2023, from https://www.theserverside.com/tip/JavaScript-vs-TypeScript-Whats-the-difference

Mui (n.d.). Material UI – Overview. Retrieved March 5, 2023, from https://mui.com/material-ui/getting-started/overview/

Nandaniya, H. (2023). A Guide to Component-Based Architecture: Features, Benefits and more. Retrieved May 24, 2023, from https://marutitech.com/guide-to-component-based-architecture/

Okoro, A. (2023). React Virtual DOM - Using Virtual DOM in React. Retrieved March 21, 2023, from https://www.knowledgehut.com/blog/web-development/react-virtual-dom

PostgreSQL. (n.d.). About PostgreSQL. Retrieved March 21, 2023, from https://www.postgresql.org/about/

Rouse, M. (2013). Object-Relational Database. https://www.techopedia.com/definition/8714/object-relational-database-ord

State of Js. (2022). State of JavaScript 2022. https://2022.stateofjs.com/en-US/

TansTack Query. (n.d.. -a) Queries. Retrieved March 25, 2023, from https://tanstack.com/query/v4/docs/react/guides/queries

TansTack Query. (n.d.. -b) Query Invalidation. Retrieved March 25, 2023, from https://tanstack.com/query/v4/docs/react/guides/query-invalidation

TansTack Query. (n.d.. -c) Query Client. Retrieved March 25, 2023, from https://tanstack.com/query/v4/docs/react/reference/QueryClient

Walls, C. (2016). *Spring Boot in Action.* Manning.

Wexler, J. (2019). *Get Programming with Node.js*. Manning.