MIDDLE EAST TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

SUMMER PRACTICE REPORT
CENG 300

STUDENT NAME

Burak Metehan Tunçel

ORGANIZATION NAME

OBSS Teknoloji A.Ş.

ADDRESS

Teknopark İstanbul Sanayi Mah. Teknopark Bulvarı Blok:8A Kat
No:3 34906 Pendik/İstanbul

DATE

18.07.2022 - 26.08.2022

TOTAL WORKING DAYS

30

STUDENT'S SIGNATURE **ORGANIZATION APPROVAL**

Contents

1 Introduction

I have done my first summer practice at OBSS as a software engineer intern for 30 working days between the above dates. Since the intern programs of the OBSS are remote, and my department accepts remote practices, I have done the practice online/remotely.

I had an informative internship and learned much about Java and web development. After learning the basics of Java, Web Development, Spring, and React, our mentor asked us to develop a middle-scaled example project. Since my internship included lots of learning, my report contains two main parts: knowledge sharing and developing a project.

Due to the structure of my internship, I will mention what I learned in the knowledge-sharing part until the Final Project section, and then move into the main project.

2 About Company

OBSS is a company established in 2005 as a software and consulting company, and today, it is one of Turkey's most powerful corporate technology consulting companies. The company has three offices located in İstanbul, Ankara, and Amsterdam. OBSS is the first and the leading Atlassian Partner in Turkey.

From software architecture to coding, from application development to analysis and software testing, the company tries to include all interactive areas of corporate technology in its service range with the aim of providing the most appropriate solutions to its customers.

OBSS has 6+ different internship programs with 100+ interns every year. OBSS shares in 5+ broad expertise sectors with 17+ years of sector experience. 750+ people work at OBSS, and they have built 700+ projects. Additionally, OBSS has several spin-off products and companies. Two of them are Witwiser, which focuses on online assessment solutions and products in international markets, and intouch, a flexible mobile application to meet communities' communication and interaction needs.

3 Orientation

On the first day of my internship, the company introduced itself by 1-hour orientation. The orientation provided general information about the company and internship programs. Information about management systems, personal data protection law (KVKK), and occupational health and safety (OHS) are also given.

After the orientation, I met with my team. Firstly, our mentor introduced himself, and then everyone did so. Our mentor shared the general internship schedule and stated our responsibilities in our internship. Our primary responsibility was to attend the programs/classes on time.

4 Core `Java`

4.1 Environment Setup and First Program

Since we have been using `Java` during our internship, necessary development environments should be introduced and installed. After the setup of `Java SDK` was shown, we discussed the coding environment. Our mentor suggested `IntelliJ IDEA` and led its installation.

After we discussed what software is and how it works on a computer and programming languages, we talked about `Java`'s history and working structure. Then, we continued by creating the first project on `IntelliJ IDEA`, and we coded the first program, the classic *Hello World!*.

4.2 Introduction to `Java`

We started with the basics learned/taught while ordinarily learning a new programming language. The first week's four days were about `Java`'s basics and core concepts. Since syntax and basic concepts of `Java` are pretty similar to `C` and `C++`, understanding the topics was not that hard for me at the beginning of the week. Even though I was learning some tricks about `Java` and coding, generally, it was like I was learning the syntax of `Java`.

While learning the concepts and basics, we usually practiced them. The usual process was that after we first coded the exercise, we examined the sample solutions other team members coded. If there was an error in someone's code or someone could not understand a part of the example, we learned by looking at those codes together. This approach has been constructive for me at times because examining the codes of other people helps the learning phase.

4.2.1 Basics of `Java`

We talked about comments, identifiers, escape sequences, data types, operators, basic input and output, conditions, and loops. While learning these, data types were, most probably, the most complicated part. Although I was familiar with the terms '`call-by-value`' or '`call-by-reference`', I believe that to gain a deep understanding of data types in `Java` language, the time needed to be devoted to them. Strings in `Java`, for example, can be considered as both primitive and reference data types.

Then, arrays whose preferred syntax in `Java` differs from `C++`, type casting, and variable length argument lists came up. Arrays and type casting work almost identically in both `Java` and `C++`. Even though other programming languages include that, it was the first time I heard of the '*variable length argument lists*', which are used for an unspecified number of arguments.

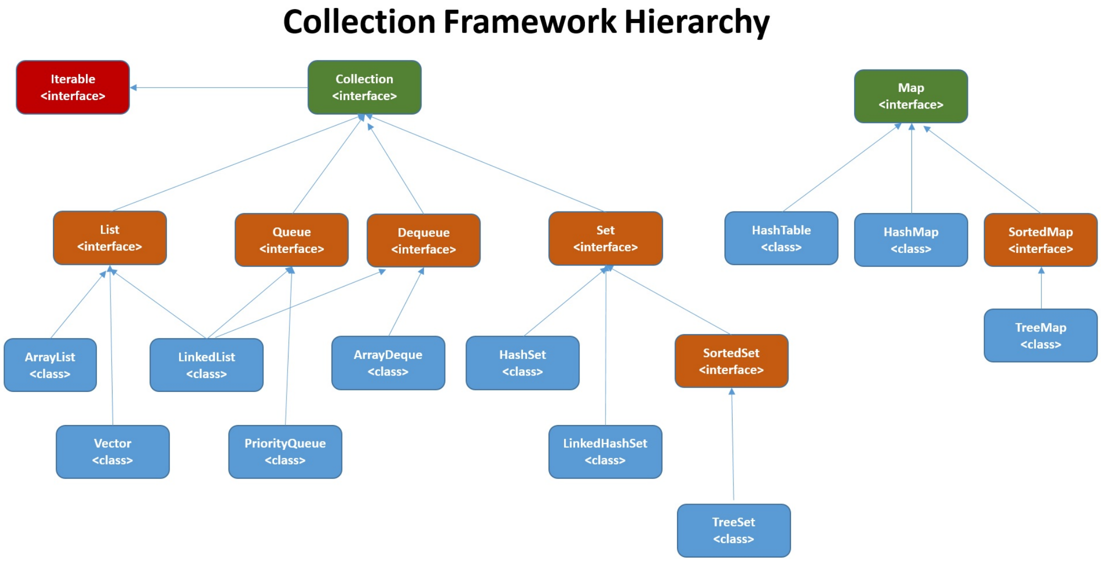
Strings have particular positions in some languages, and I believe `Java` is one of them. Strings can be confused because they can behave like both primitive and reference types. Also, I learned about `StringBuffer` and `StringBuilder`, which are basic classes to construct strings and add some functionality to them. Even though I knew what exceptions were, I did not practice much. In fact, I can say that I learned how it works and its actual use in this internship.

We also talked about advanced `Java IO` and the concept of '`serialization`', the conversion of the state of an object into a byte stream. I believe that although it is a little bit harder than other languages I use, such as `Python`, file handling is a little bit more flexible in `Java`, thanks to variety. There are several classes to make file reading and writing easier.

After these basics, some advanced features such as enumerations, interfaces, abstract, and generic classes are discussed. Although these were not new to me, I barely knew what they were and why we use them, but thanks to the exercises, I understood these. Interfaces were a little bit tricky for me because it was the first time I had encountered such a thing. Then, an unfamiliar topic arose: wildcards about which I did not know anything but practical.

4.2.2 Collections in Java

From the courses I have taken, I know some data structures. In this part, I learned that `Java` provides a variety of collections whose implementations are different under the hood. The collections that are in the figure below are discussed.



Source:
`https://facingissuesonit.com/2019/10/15/java-collection-framework-hierarchy/`

Also, `hashCode()` and `equals()`, important methods for these collections to work properly, were discussed. It was shown what they are and why we use them.

4.2.3 OOP

This part covered the basics and principles of Object Oriented Programming (OOP).

We started by discussing the basics, such as scopes, constructors, access modifiers, setter and getter methods, method overloading, and static methods. Since I was familiar with concepts from CENG 242: Programming Language Concepts, I did not have a hard time with these concepts.

After the basics, we moved into the principles. We started with inheritance and class hierarchy. Here, I learned what superclasses and subclasses are, and that `Java` does not support multiple inheritances. Furthermore, behaviors of some concepts, such as access modifiers and constructors, in inheritance are discussed. Also, other principles of abstraction, encapsulation, and polymorphism are covered, but I did not learn new information.

4.2.4 JavaBeans and JDBC

The last day of the week, when we got out of the basics of `Java` and learned databases and database connections by using `Java`, was a day that I learned new information and was challenged for the first time this week.

We started by talking about the `JavaBeans`, which is a standard. By learning and reading about `JavaBeans`, I realized that many things are standardized in `Java`, and implementations are competed instead of the standards. This makes the reason why `Java` is exceptionally preferred in corporate areas clear.

Since we would have needed to add some dependencies and Maven would be used in our projects, we talked about Maven, a build automation tool, and installed it in addition to MySQL, which was going to be used as a database. After the installations and the environment adjustments, we dived into the `Java` Database Connectivity (JDBC), `Java` API that mainly manages to connect to a database. Connecting to a database, executing queries, using of “`Statement`”, “`ResultSet`” and “`PreparedStatement`” which are basically used to sending SQL statements to databases, and transaction management.

4.3 Git & Bitbucket

To our use during the internship, we were provided a Bitbucket account. We were asked to use Git and `push` our codes to Bitbucket. Since we were asked to use Git, the basics and primary usage of Git and Bitbucket were shown.

5 Introduction to Web Development

The first week of my internship was about the Java Core, whereas we focused on the enterprise version of Java, Java EE, or Jakarta EE, and web development in the second week.

5.1 Introduction to Java EE Platform

After learning the Java versions and their primary concern, we discussed enterprise-level applications and their needs. Then, we focused on how Java values developers, vendors, and businesses. In this part, I again realized why Java is still so popular. Developers and companies can focus on development instead of losing time thinking about what they will use and learning completely different technologies.

Our mentor mentioned that the specifications are determined and defined concurringly, and vendors only compete at their implementation level in the Java world so that developers can use any J2EE implementation for development and deployment. Additionally, I learned about layers that are the foundation of software architecture (presentation, application/business, data, and service layers) and types of software architectures (one-tier, two-tier, three-tier, and N-tier architectures) and their advantages and disadvantages.

5.2 Introduction to Web Development on Java Platform

After learning what websites and web applications are and their differences, we discussed web servers and installing one of the preferred web servers on Java World: 'Tomcat'. After discussing what Tomcat is and the installation process, we focused on the basics of the web world: HTTP and WWW, Request and Response, URL, Domain Names, DNS, and Static and Dynamic Content.

After these basics, we learned the 'Servlets', the basic concept and tool for Web development on Java world and JSP, which enables mixing static HTML content with unique code that produces the dynamic content. Then, we talked about Model-1/Model-2, MVC, and Servlet Container.

5.3 Developing `Java` Web Applications

We learned the basic anatomy of a web application and web module structure. I believe these topics are pretty beneficial because knowing the web module's structure is essential in portability's development and deployment process. Also, if the structure of a web module is not suitable for the general convention, it is hard to maintain the module.

After that, we dived into HTTP, a pretty important protocol in the world of the web since many things are shaped based on it. Mainly, I was surprised when I learned the greatness of the contents carried by a request and response. Requests and responses include a lot of headers and their fields. After HTTP, we focused on Servlets and learned what they do, their life cycle, and their methods. I can tell that Servlets are a massive part of `Java` web development in their first days.

Then we discussed the '*Threading Issues*' caused by the multi-threaded approach. I learned that the Multi-threaded Approach is an approach to solving the problem of how one servlet object serves many clients. In this approach, instead of creating multiple things, the servlet container creates a separate thread for each invocation of `service()` method, and that thread runs all servlet methods. It can be said that the multi-threaded approach makes servlets much faster. Also, we discussed the possible problem and solutions of this approach.

We also discussed information-sharing techniques among servlets (ServletContext object, HttpSession object, Request attributes). Before learning these techniques, I used incredible nonsensical methods such as different web pages to carry information. However, these techniques are beneficial and make it easier to carry information between servlets.

5.4 More on `Java` Web Applications

We learned what a developer should do for an exception and error management by configuring a web descriptor, namely `web.xml`. This week, we generally used XML files for configurations; however, following weeks, we used annotation-based configurations. Then we discussed the HTML forms, the basic data sender structure in HTML, how they are used in web development, and why they are important. I learned that forms are simple and reliable user interface control tools and that the data collected by forms are sent to the server. We also talked about `GET` and `POST` methods.

We focused on session management and filters on the week's last days. We learned session tracking techniques in detail, which of them are tracking via IP address, user authentication, hidden form fields, URL rewriting, and cookies. In the filters part, we learned how filters actually work with other parts of the server. Then, we practiced how requests are used for authentication and security purposes.

Until this week, my knowledge about web development was pretty limited, and I was a novice to the web at this point. Therefore, these topics were confusing but enthusiastic for me, and I barely imagine the next level of web development discussed next week.

6 `Spring` Framework

After learning `Java` and the basics of web development, I was amused. In web development, our next step was a framework. In the internship program, `Spring`, the world's most popular `Java` framework, is chosen for that purpose. `Spring` is an open-source framework developed for `Java`. Generally, it is used for developing web apps with the `Java` Enterprise platform.

6.1 Environment Setup

Firstly, we prepared the environment by installing Apache Tomcat -`Java` servlet container- and Maven -software project management and comprehension tool.

Since we will have been using the Maven, our mentor talked about `Maven`. We discussed what Maven is and how we can use it. Then, we briefly introduced `Spring` by mentioning what it is and why we use it.

This week was mainly about the `Spring` framework. I learned a lot of information about `Spring` this week. For ease of reading, I will dive into the subsections and briefly explain what I learned this week and what they are.

6.2 Introduction to `Spring`

By using `Spring initializr`, we created our first spring project. While creating a project, our mentor addressed the features of `Spring initilizr` and the dependencies part of it. After we opened the project, our mentor firstly focused on the dependencies and their management. While discussing the dependencies, we found a problem with versions; therefore, our mentor talked about the general versioning methods. When we ran the app for the first time, some team members encountered a problem with the port of the application; hence, our mentor needed to show us how we could change the settings of the projects by using the application.yaml file of the project, configuration file of the `Spring`.

6.2.1 Controllers

We started with the controllers. Controllers are part of the Spring Web model-view-controller (MVC) framework and meet the requests. By using the mapping annotations, requests are mapped to related functions inside the controller class. The functions usually return a response after necessary processes are done according to the request. In our projects, we usually used "ResponseEntity" classes because we returned an entity.

6.2.2 Filter and Interceptor

We continued with the filters. The filters are objects used to intercept the HTTP requests and responses of the application. Basically, they are the layer before sending the request to the controller and before sending a response to the client. Some common usages are logging requests and responses, logging request processing time, formatting request body or header, verifying authentication tokens, or compressing responses.

Interceptors are pretty similar to filters, but they act in `Spring Context`, so they are powerful enough to manage HTTP Request and Response but can implement more sophisticated behavior because they can access all `Spring` contexts. In other words, we can not use filters outside the web context, while `Spring` interceptors can be used anywhere because they are defined in the application context.

6.2.3 Exception

Then we dived into the exceptions. Firstly, we examined the default answer of `Spring` for errors. An example from my final project is given below.

```
{
  "timestamp": "2022-09-17T10:03:08.250+00:00",
  "status": 500,
  "error": "Internal Server Error",
  "trace":
↪   "tr.com.obss.jip.springfinal.exception.UserNotFoundException:
↪   User, whose id number is 99, is not found! ...",
  "message": "User, whose id number is 99, is not found!",
  "path": "/users/99"
}
```

It can be said that this type of answer constitutes a security vulnerability because of the information the error provides an error message and codes. For example, someone can tell that this system uses the `Spring` framework. After examining the default answer of `Spring`, we realized errors are also needed to be handled due to security as well as providing a meaningful message. We discussed how we could handle errors and change the behaviors and messages when an error occurs.

6.2.4 Model, Entity, and Database

After the error handling, we learned the Data Transfer Objects (DTOs), which are used to encapsulate data and carry this data between processes and their usages. I extensively used models in the final project to transfer data between processes and layers.

Since we will have been using the database, we had to make the database connection. `Spring` provides database connection capability with hibernate, which is integrated inside `Spring` and makes the database connection and necessary database operation.

Hibernate can also create the tables in the database by using entities provided by entity classes. Therefore, we used entity classes to tell the hibernate to create the table in the database.

6.2.5 Services, Repositories, and Configuration

Services are generally used for business logic, such as creating, storing, or changing data. I extensively used the services in my final project. After the controller met the request, the proper function was called, and the necessary business was done in the project.

Repositories are the mechanism for encapsulating storage, retrieval, and search behavior that emulates a collection of objects. They are used for the access layer for accessing and making necessary operations in the database. We chose to use `Spring Data JPA` for this purpose. After making the required settings from the application.yaml, `Spring` automatically connects itself to the database and gets ready to execute queries with little effort. In repositories, there is almost no need to write SQL queries executed in the database, even if it is allowed. By using the specific combination of some keys in the function names inside the repository interface, we provide necessary information to `Spring` so that `Spring` can create the proper queries and execute them by connecting to the database.

After the basics, we dived into the configuration of our apps, such as security, password encryption, or data loading when it is started.

7 `React`

On the last day of the third week, we started discussing `React`, a user interface library created by Facebook on `JavaScript`. I had almost no experience with both `React` and `JavaScript`.

7.1 MPA and SPA

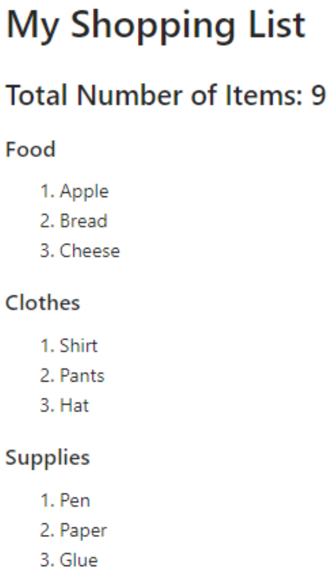
We started the discussion with MPA (multiple page applications) and SPA (single page applications) and why SPA should be chosen over MPA. `React` helps developers to develop SPA by using JavaScript and virtual DOM. Since our primary concern will have been learning `React`, we talked about its advantages and limitations and compared `React` with other frameworks and libraries by looking at the trends to have a better perspective.

We started with the `React` elements and continued with how we could render them. After learning the basics, we talked about JSX, a syntax extension to JavaScript that allows `React` elements to be written inside JavaScript using HTML tags. Since JSX has many different features, such as styling the elements and inserting JavaScript variables, we had to spend much time on it solving mini coding challenges.

7.2 Functional and Class Components

We moved into components. A `React` component can be defined as an independent, reusable component that outputs the `React` element. We compared the class and functional components and discussed their usage differences. I realized that I tended to use class components because of the OOP and class familiarities, although functional components are excessively used in the sector.

After these, we solved an example, which can be called Shopping List App. It was a simple shopping list without adding or removing options, but it only showed the provided list. After we coded the exercise, our mentor also coded the exercise to show us another perspective.



My Shopping List

Total Number of Items: 9

Food

1. Apple
2. Bread
3. Cheese

Clothes

1. Shirt
2. Pants
3. Hat

Supplies

1. Pen
2. Paper
3. Glue

Figure 1: Basic Shopping List App

7.3 States

Then we discussed a crucial topic and feature of `React: States`. We talked about what a state is and how it is used. We then jumped into another critical issue in states: setting/updating the state. We realized and learned how to fix the different behaviors of setting states.

7.4 Life Cycle Methods

Another essential topic was the life cycle methods of class components. While rendering and mounting the component, `React` runs various methods on the components at multiple phases. Life cycle methods can be used for specific purposes according to phases.

7.5 Event Handlers

Event handling is pretty similar to HTML events. We discussed that we could achieve some effects by using event handlers. For example, sending a request when a button is clicked or changing the state of a variable when a key is pressed can be achieved by event handlers.

7.6 Promises, Back-end Requests, and JWT

After the basics and some advanced topics, we moved into the back-end requests. Before talking about them, we discussed the promises, a feature of JavaScript. For back-end requests, we conversed about Axios. Axios provides support to different request types and configs and is easy to use.

We briefly discussed JWT tokens used for token-based authentication instead of password authentication every time.

8 Final Project

In this part of the report, I will explain the project I developed in detail. I will divide the project into two parts: Back-end and Front-end.

For the final project, we were given some time to develop. We were first given time to develop after the `Spring` framework part. It was for developing the back-end. The second time given was after the `React` part. `React` part was finished on the second day of the fourth week. Since the main internship program was four weeks at OBSS, we were expected to finish the final project by the end of the fourth week. However, I could not complete the project at this time. Since my internship period was six weeks and I could not complete the project, they asked me to complete the project within this period.

In the fifth week of my internship, I decided to start from scratch as I thought that I made mistakes at many points while developing the project and that if I continued to develop on these mistakes, it would harm the development phase. First, I tried to find out where I lacked and made mistakes.

I tried to learn `React` from the beginning for three days because I thought I had problems with the `React` part. As a result of this work, I have come to a position where I can do many things on `React` without difficulty. I spent the next two days thinking about simple things and design. In my last week, I rebuilt and coded the entire project, starting from scratch.

8.1 Project and Requirements

The final project was a basic Book Portal whose requirements were provided. We were provided the document located in the appendicies for basic requirements.

In my project, there are two main entities: User and Book. Each user has a role, admin or user role, and book lists for favorite and read books. Each book has many information about itself: name, author, type, publisher, and publication date. Both admins and users can log in to the system. Each admin is also a user and can do everything a user can, such as updating passwords, seeing books, adding books to their read or favorite lists, and searching for a book and its information. Additionally, an admin can add, update, and delete users and books.

8.2 Back-end

As mentioned in the requirements, I used `Spring` for the back-end. The main folder structure can be seen at the appendicies. `Spring Security` is active, and `JWT` tokens, which need to be inside the request's header, are used for authentication. After security is passed, the request is handled by controllers.

In the back-end part, there are nine main folders:

- `config`: contains the configurations and data loader.

- `controller`: contains request (Rest) controllers. If the requests are authorized, they are met by the related controller. When a request is met, necessary business logic is run.

- `entity`: contains entity classes. These entity classes are used in the tables of the database.

- `exception`: contains the user-defined exceptions and global exception handler.

- `filter`: contains filter classes. All requests go through the filters.

- `model`: contains model classes. These classes are used for data transfer between client, server, and database.

- `repo`: contains repositories. These interfaces are used to access the database by using `Spring JPA`.

- `service`: contains services. These classes consist of the business layer functions.

- `util`: contains utilities.

These are the main folders consisting of several files; I will explain them in detail.

8.2.1 `config`

The classes inside this folder are used for the configuration of `Spring` and data loading when `Spring` runs the app. After the app starts running, the data loader is not used, although configurations influence the app's answer.

8.2.1.1 `AuthEntryPoint.java:`

When authorization is failed in `Spring Security`, the function inside this class is called and returns a response with the `UNAUTHORIZED` status code.

8.2.1.2 `PasswordConfig.java:`

This class indicates the password encoder which is used through the app.

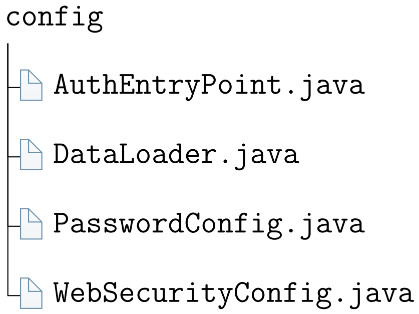


Figure 2: Structure of config folder.

8.2.1.3 `WebSecurityConfig.java`:

It is configuration of `Spring Web Security` that checks all the requests for authorization and roles. Also, if an exception occurs, it handles the exception with the help of a global exception handler.

The web security is a little bit confusing and not easy to handle. Especially, `JWT` token-based authentication challenged me. I spent almost one day understanding what `JWT` token and `JWT` token-based authentication are. By reading documents and examining examples, I implemented `JWT` token-based authentication for my security.

8.2.1.4 `DataLoader.java`:

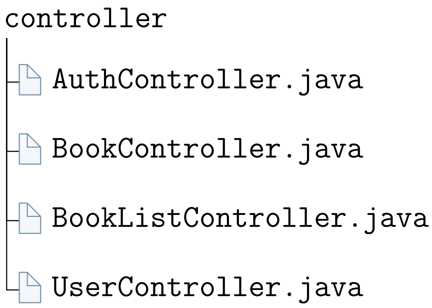
This class is run when the application is run. It checks the database for the default admin user and roles, called `ROLE_ADMIN` and `ROLE_USER`. If the database is missing, it creates admin user and roles. After learning the `ApplicationRunner` interface, it was easy to implement.

8.2.2 `controller`

Controllers are the essential elements of back-end development. Controller files are used with `@RestController` annotation, which indicates that the class is a Rest API controller. Each class has `@RequestMapping` annotation that shows the path of the controller; that is, the requests coming to the specified path are handled by that class.

The classes include several mappings for different request types such as GET, POST, PUT, or DELETE. When a request is sent to the back-end, Spring sends it to a suitable controller according to @RequestMapping. When a request arrives at the class, it handles it according to its type.

Since controllers are not tricky, I could easily code and map the requests. The most challenging part of the controllers was the authentication of functions. Since some operations, such as deleting users or books, cannot be done by regular users but admins, I needed to secure the related functions. After researching, I managed to block the regular user using some operations by using @Secured annotation. This annotation takes a string which is the privileged role, in my case ROLE_ADMIN. When this annotation is used, Spring Security checks for the additional role.



```
controller
├ AuthController.java
├ BookController.java
├ BookListController.java
└ UserController.java
```

Figure 3: Structure of controller folder.

8.2.2.1 **AuthController.java:**

This class handles the requests coming to the path '/auth' and has two POST mapping.

This class is responsible for authentication operations. It can be used to check the JWT token's validity or log in. Using the information inside the request body, I checked the necessary information and sent the response to the client. The responses include some required information for the front-end, such as validation information, admin status, or JWT token.

8.2.2.2 `BookController.java`:

This class handles the requests coming to the path '`/books`'. It has six `GET`, one `POST`, one `PUT`, and one `DELETE` mapping.

This class handles book operations such as searching, adding, updating, or deleting. I need a particular concern in this class: Pageable and List data. Since I needed pageable and list data in the front-end while searching the book, I coded two variances of each search function. This class can handle both id and name searches. In name searches, there is no need to provide the full name of the book. Instead, providing a letter or word inside the book name is enough.

8.2.2.3 `BookListController.java`:

This class handles the requests coming to the path '`/`' and has two `PUT`. Although the main path is `/`, the two functions inside this class have special path mapping for its `PUT` mapping. This class is responsible for the favorite and read list operations.

According to the traditional way, updating something is done with `PUT` request. Since adding or removing a book from read or favorite lists is updating the list, I decided to use `PUT` requests. However, there were four operations: adding or removing from the read list and adding or removing from the favorite list. Firstly, I divided all operations into four functions and realized it was not a good idea because path mapping was not nice.

So, a problem came up about how to handle these operations. I decided to have two main functions and paths for read and favorite lists. In main paths `/read` and `/fav`, I handled read and favorite list operations, respectively. A request parameter is needed as well as the book and user ids to acquire the necessary information, which book will be added or removed.

8.2.2.4 `UserController.java`:

This class handles the requests coming to the path '`/users`'. It has six `GET`, one `POST`, one `PUT`, and one `DELETE` mapping.

This class is similar to the book controller and is responsible for user operations such as searching, adding, updating, or deleting. Like in the book controller, I also need a particular concern, Pageable and List data, and I solve this problem the same way in the book controller.

8.2.3 `entity`

Entities were my main data classes. Thanks to Hibernate, I was also able to create the database tables by using `@Table` annotations, so the tables were created automatically. Around the back-end, such as between data and business layers, I used these entity classes; however, I used models while sharing and transmitting information between the client and server.

The main problem with entity classes was the mutual data types. A `User` includes `Set<Book>` inside it, and A `Book` includes `Set<User>` inside it. This was a problem when this information was sent to the client side due to this mutuality. For example, when a `User` is sent to the client, its read list is also sent. However, inside the read list, there are books that contain the `User` that is being sent to the client. Therefore, there was an infinite loop while parsing the data. I solved it by using `@ManyToMany`, and `@JsonManagedReference` annotations.

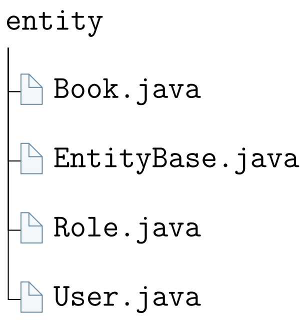


Figure 4: Structure of entity folder.

8.2.3.1 `Book.java:`

This is an entity class that extends `EntityBase` for a `Book` and is responsible for holding information of a book. It has 'name', 'author', 'page count', 'type', 'publisher', and 'publication date' fields as well as the data fields in `EntityBase`. Also, it has many-to-many relations with `User` class.

8.2.3.2 `User.java`:

This is an entity class that extends `EntityBase` for a `User` and is responsible for holding information of an user. It has ‘username’, ‘password’, ‘read list’, ‘favorite list’, and ‘roles’ fields as well as the data fields in `EntityBase`. Also, it has many-to-many relations with `Book` and `Role` classes.

8.2.3.3 `Role.java`:

This is an entity class that extends `EntityBase` for a `User` and is responsible for holding information of a role. It has the ‘name’ field and the data fields in `EntityBase`. Also, it has many-to-many relations with `User` class.

8.2.3.4 `EntityBase.java`:

This is an entity class that is the base of other entities. This holds the general information such as ‘id’, ‘creation date’, ‘update date’, or ‘activity’.

8.2.4 `exception`

Exceptions are used at several points to provide information to the exception handler. The exception handler catches the exceptions and sends a response to the client. The primary purpose of the exception handler is security because default `Spring` errors exploit some system information. Therefore, I used special exceptions and an exception handler to provide necessary but unimportant information outside.

Creating new exception classes was not hard. However, adjusting the exception handler is a little tough because the order of the functions can be a problem. I solved this problem using `@Order` annotations and attention.

The names of the files explain what the classes are for and what they do. These are used for specified exceptions in related positions.

8.2.5 `filter`

My app contains only one filter, which is checking the `JWT` token. When a user or admin is logged in to the system, I send a new `JWT` token. The reason for that

```
exception
├ BadRequestException.java
├ BookListException.java
├ BookNotFoundException.java
├ ConflictException.java
├ GlobalExceptionHandler.java
├ RoleNotFoundException.java
└ UserNotFoundException.java
```

Figure 5: Structure of exception folder.

is transmitting username and password in all requests. Instead, I use a `JWT` token for authentication. All requests go through the filter for `JWT` token check. If the `JWT` token does not exist in the header or it is expired, `BadRequestException` is thrown, and a response is returned with `BAD_REQUEST` status code.

This part challenged me because I did not know `JWT` and how to check it. For this filter and the `JWT` authorization, I spent almost one day. However, the main problem was not that the subject was difficult but that I misunderstood some concepts and topics and needed to change the security configuration quite a bit. By getting help from the internet and reading, I solved the problem in security configuration and filter.

```
filter
└ JwtRequestFilter.java
```

Figure 6: Structure of filter folder.

8.2.6 `model`

Models are mainly used to transmit data between client and server sides. There were two reasons. The first one is that client does not know some information, such as the creation date. The second one is that the client should not and not need to see some information, such as the user's password or the object's update date. Therefore, I used DTOs for any information transfer between the client and server sides.

In addition to DTOs, one class, called `MyUserDetails`, is not used for information transfer. It was needed for the authentication system. It helps by indicating how the necessary information is extracted from my `User` entity.

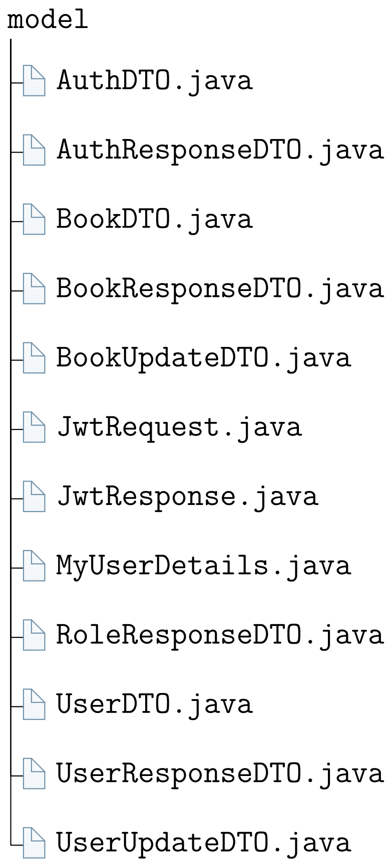


Figure 7: Structure of model folder.

8.2.7 `repo`

Repositories are used to access the databases, and they make use of the `Spring JPA` by extending `JpaRepository`. Some essential operation, such as directly adding or finding by id, is inside the JpaRepository but if I want to add something special, I write the function name by using keywords, such as findBy, or All, and Spring generates the necessary queries in the background. This is so easy to use and increases the speed of development.

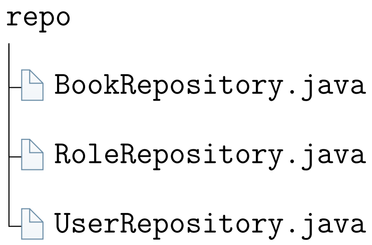


Figure 8: Structure of repo folder.

There is one for each main entity. BookRepository is used for operations on books, UserRepository is used for operations on users, and RoleRepository is used for operations on roles.

8.2.8 service

Services are the business layer of my app. All the operations, such as adding or removing the books or users, and logic are done in this layer. Controllers use the services for related operations; therefore, it can be said that there is one service for each controller.

Inside services, other services or necessary repositories are used. Accessing the database is done inside services with the help of repositories. Repositories contain several different implementations of the same operation for different needs. For example, several search functions exist for pageable and list data.

8.2.9 util

This folder contains only one class, called JwtTokenUtil. This class contains several functions applicable to the JWT token, such as getting the username or expiration date from the token. Also, it can generate a new JWT token. JWT filter and AuthController excessively used this class.

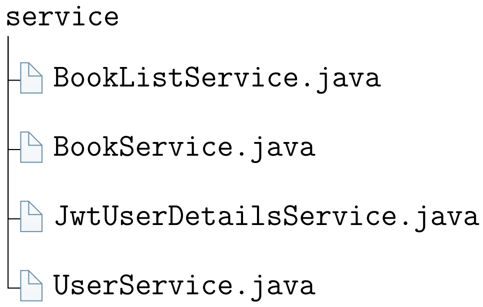


Figure 9: Structure of service folder.

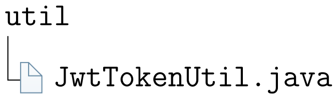


Figure 10: Structure of controller folder.

8.3 Front-end

As mentioned in the requirements, I used React for the front-end. The main folder structure can be seen in the appendices front-end-tree. I used **antd**, an enterprise-class UI design language and React UI library, for the user interface, Node.js, a JavaScript runtime built on Chrome's V8 JavaScript engine, and npm, packages manager.

The front-end part has two main folders: public and src. The folder 'public' contains the default `public.html` when npm is used to create the project, and `public.html` contains only one div whose id is '`root`' and which will be modified through entire app.

8.3.1 Folder Structure

The main project files are inside the src folder.

- `globals`: It is a folder containing the app's global variables.
- `pages`: It is a folder containing the pages called auth, book, and user, as well as the main homepage of the app.

- `service`: It is a folder containing the front-end service layer. Files inside this folder include functions to connect the back-end.

- `App.js`: It is a file and is responsible for rendering the pages and providing the main/top navigations around the app. Also, it checks the authorization token and its validation.

- `index.js`: It is a file and responsible for rendering `App.js` by using `ReactDOM`.

I will briefly explain the folder and files in detail. Then I will tell the pages.

8.3.1.1 `globals`

This folder contains just a file called `GlobalVariables`. That file includes two global variables, called `BOOK_COLUMNS` and `PAGINATION`, which are used for tables around the app.

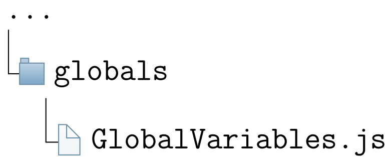


Figure 11: Structue of globals

8.3.1.2 `pages`

This folder contains the whole pages, which means users mainly see the files inside this folder. I will explain the pages in detail later.

- `Home.js`: This is the app's homepage and contains the user's general information. It can be accessed by clicking the `Home` from the top navigation.

- `auth`: This folder contains just a file called `Login`, which is responsible for the login page.

- `book`: This is the page of books. It contains five folders and one file inside it and is responsible for the operations of adding, deleting, updating, and adding/removing books to/from favorite/read lists. It uses a switch for rendering the right page of the operation.

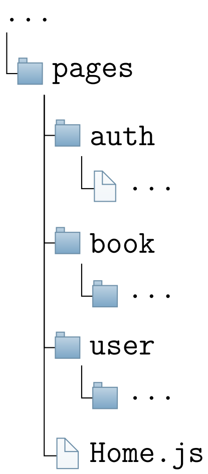


Figure 12: Structue of pages

- `user`: This is the page of users. It contains four folders and one file inside it and is responsible for adding, deleting, and updating operations. It uses a switch for rendering the right page of the operation.

8.3.1.3 `service`

This folder contains the services to access the back-end. The functions inside the service files handle the returned response and return a new object with the data inside the response. Additionally, thanks to Axios interceptor, the authorization token, which is saved to `Session Storage` or `Local Storage` while logging in, is added to the header of all requests.

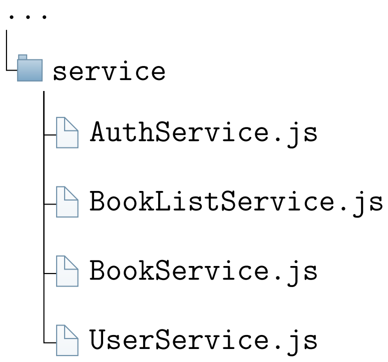


Figure 13: Structue of service

- `AuthService`: Responsible for authentication and login process.
- `BookListService`: Responsible for adding/removing books to/from read/favorite lists.
- `BookService`: Responsible for book operations such as searching, adding, removing, and updating.
- `UserService`: Responsible for user operations such as searching, adding, removing, and updating.

8.3.2 Pages

When the app is run, firstly `index.js` renders the `App.js`. `App.js` firstly checks the authorization token and its validation. If the token is valid, it directs the user to the homepage, rendering (`Home.js`). If the token is not valid, it directs the user to the login page, rendering (`Login.js`).

8.3.2.1 `Login and Logout`

This page contains a basic login form. While the login page is done by antd form, the logout item is constructed by 'Popconfirm' of antd. If the '`Remember me`' option is chosen, authorization token is saved into both `Session Storage` and `Local Storage`. However, if it is not chosen, the token is only saved into `Session Storage`.

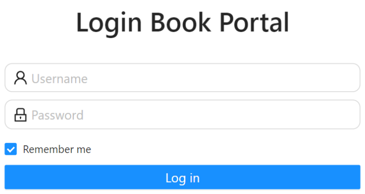
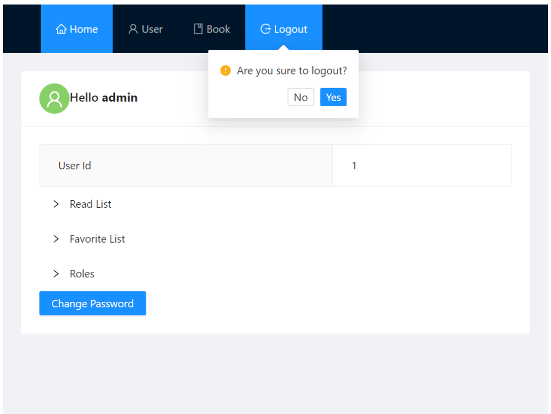


Figure 14: Login Page



Are you sure to logout?

Hello admin

User Id	1

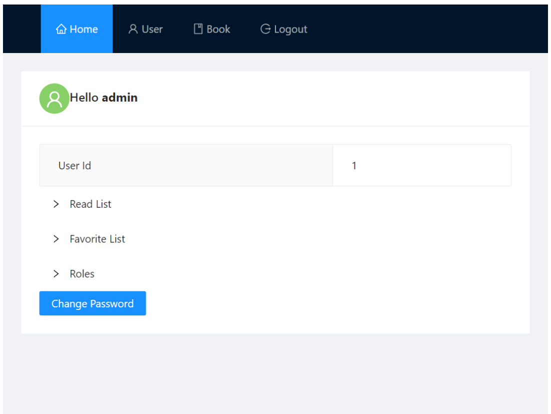
Figure 15: Logout

After logging in, users are directed to the homepage. At the top of the screen, there are four different menu items, called `Home`, `User`, `Book` and `Logout`. Home is firstly rendered, and using these navigations page can be changed.

When users click to `Logout`, a warning, which asks whether they are sure to log out, pops up. If yes is clicked, users are directed to the login page.

8.3.2.2 Home

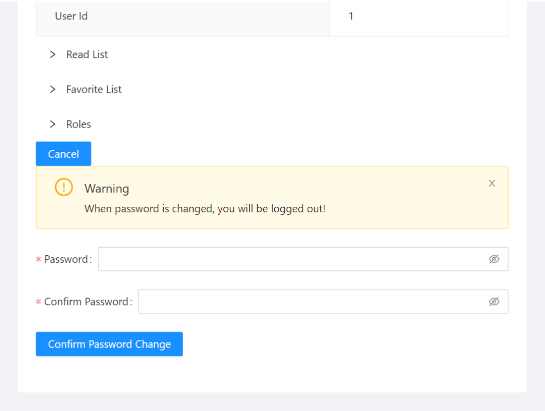
On the homepage, user information is provided. This page makes use of 'Card', 'Descriptions', 'Collapse', and 'Panel' of antd to show the information. Users can see their user ids, read lists, favorite lists, and roles on this screen. This information is acquired by sending back-end a request by username and setting the information state. By using the button "Change Password", they can also access the update form and change their passwords.



Hello admin

User Id	1

Figure 16: Home Page



User Id	1

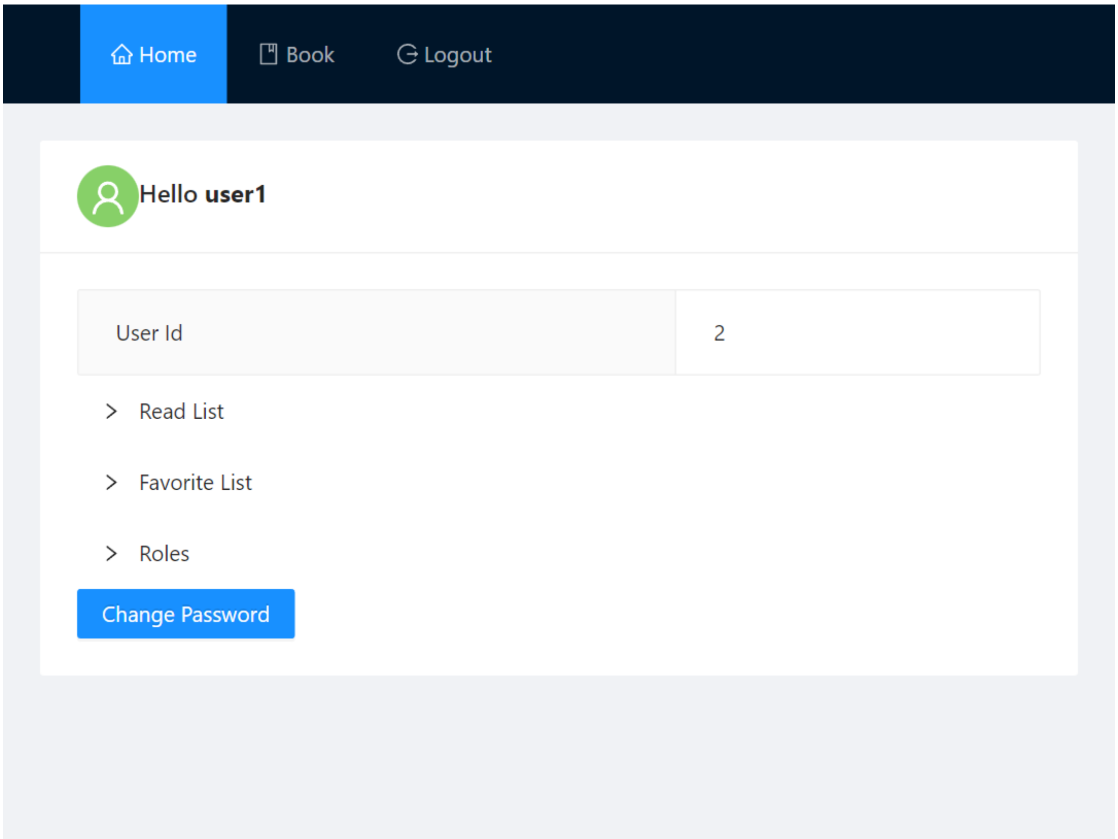
Warning
When password is changed, you will be logged out!

* Password:

* Confirm Password:

Figure 17: Password Change

Since regular users are not allowed to do user operations such as adding, deleting, or updating, `User` page is unique to admins. Therefore, `User` item is not seen when the user does not have admin roles.



Hello **user1**

User Id	2

Figure 18: User Home Page

8.3.2.3 `User`

When `User` is clicked from top navigation, `User` page is rendered. On this page, there are three operations: adding, deleting, and updating users. This page is special to admins because users are not allowed to add, delete, or update users. Therefore, this page is only accessible when the user does not have the admin role.

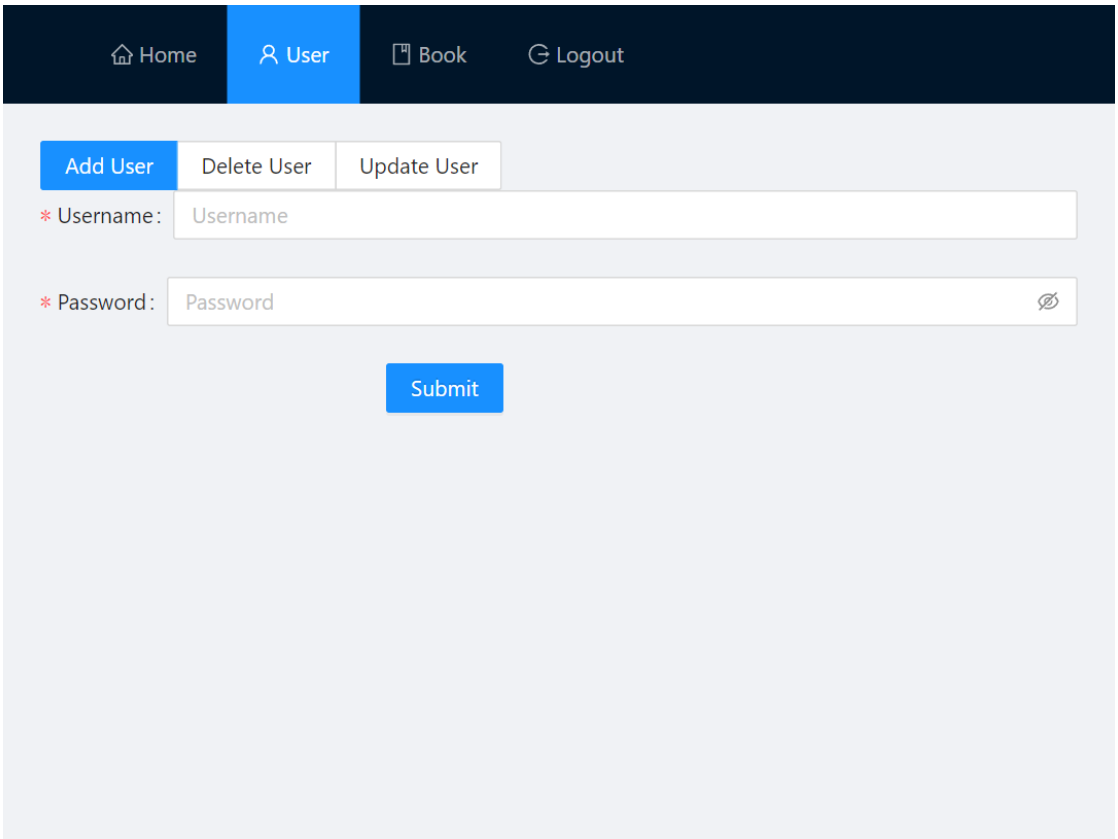


Figure 19: User Page

■ **`Add User`** By default `Add User` is rendered when `User` menu item is clicked. This page is for adding a user to the database and is unique to admins. When the page is rendered, an antd form, which includes username and password, appears.

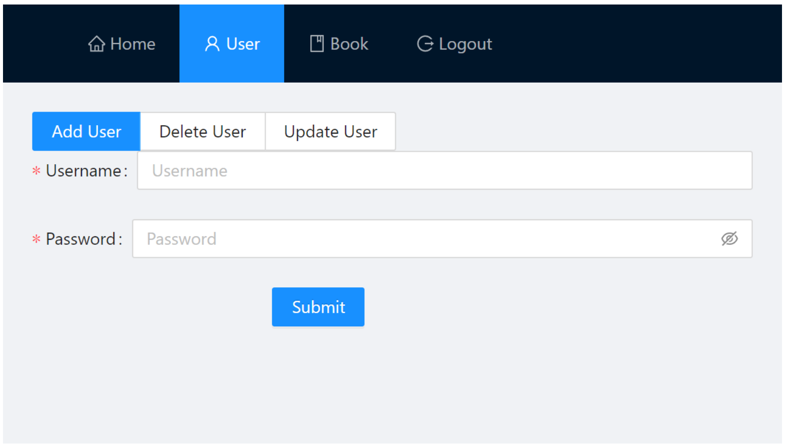
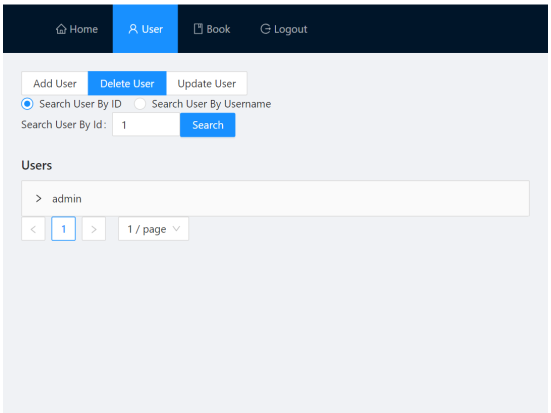


Figure 20: Add User

■ **`Delete and Update User`** `Delete User` and `Update User` contains more advanced components. When `Delete User` and `Update User` are firstly rendered, a request is sent to database to acquire users and their information by using `useEffect` of React. These pages also utilize pagination and utilities located in the util folder of the user.

`Delete User` and `Update User` are pretty similar to each other. Both pages support searching users by both id and username, and the searching option can be changed by using the radio buttons. When an id or username is provided and clicked the search button, a request is sent to the back-end through `UserService`. When the input area is cleared, the page is updated, and user data is reset. Both pages also support the pagination and pagination setting changes. Admin can change the pagination setting by using the navigation below. When pagination is changed, a new request is sent to the back-end. Additionally, the admin can see the users' information on both pages.

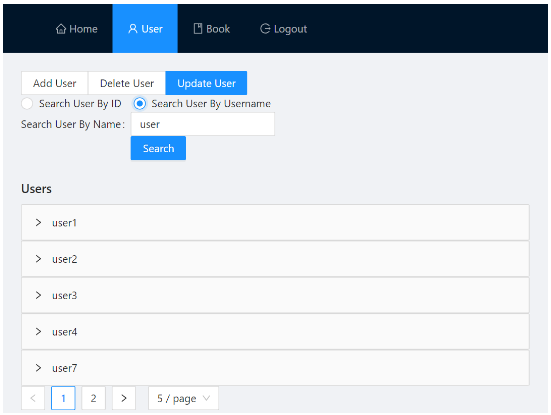


● Search User By ID ○ Search User By Username

Search User By Id : 1

admin

Figure 21: Searching by ID



○ Search User By ID ● Search User By Username

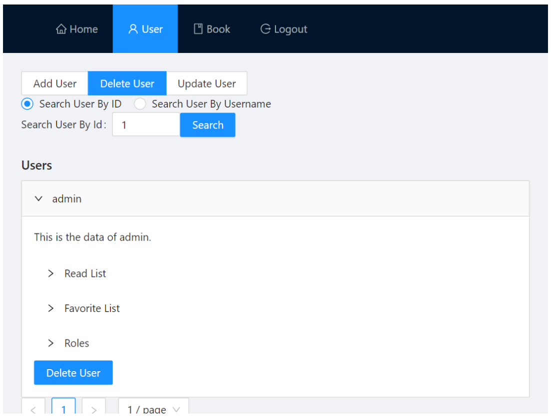
Search User By Name : user

user1
user2
user3
user4
user7

Figure 22: Searching by Name

Pagination and visualizing the data was the most challenging part for me because there were several aspects I needed to consider. I changed my visualizing method third time for the convenience of development. Because of these changes, I had to change the back-end at some points. This situation causes the loss of so much time for visualizing decisions. However, my job was not done after visualizing the decision because pagination was more challenging than expected. Since I misunderstood some parts and concepts of pagination and its callback functions, I had to spend a couple of hours searching and reading the antd documentation. In the end, I understood what I should do.

The difference between `Delete User` and `Update User` is the buttons found in all users. In `Delete User`, the name of the button is `Delete User`, and it is used for deleting the user. In `Update User`, the name of the button is `Update User`, and when it is clicked password update form appears.



● Search User By ID ○ Search User By Username

Search User By Id : 1

Users

admin

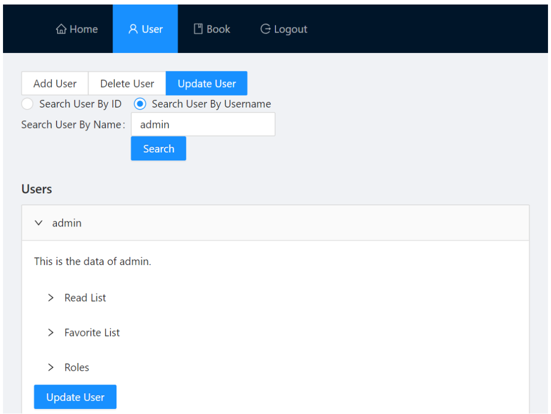
This is the data of admin.

> Read List

> Favorite List

> Roles

Figure 23: Delete User



○ Search User By ID ● Search User By Username

Search User By Name : admin

Users

admin

This is the data of admin.

> Read List

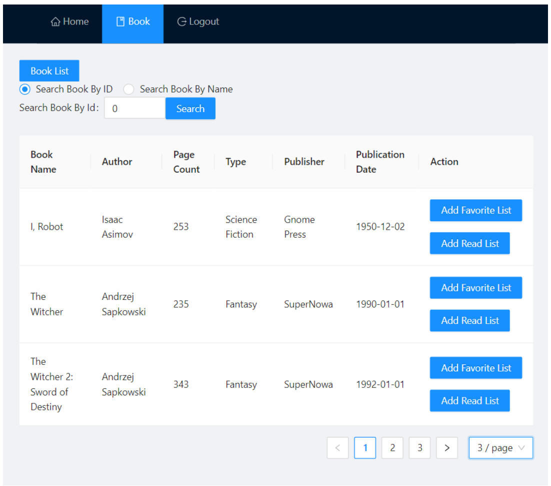
> Favorite List

> Roles

Figure 24: Update User

8.3.2.4 Book

When `Book` is clicked from top navigation, `Book` page is rendered. On this page, there are three operations: adding, deleting, updating, and listing books. By default `Book List` is rendered. Although this page is not special, some operations are not allowed to normal users. Therefore, normal users can list books and add/remove these books to/from their lists.

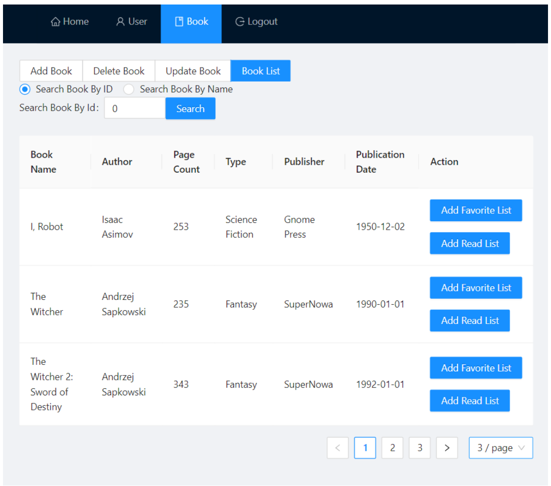


● Search Book By ID ○ Search Book By Name

Search Book By Id : 0

Book Name	Author	Page Count	Type	Publisher	Publication Date	Action
I, Robot	Isaac Asimov	253	Science Fiction	Gnome Press	1950-12-02	Add Favorite List Add Read List
The Witcher	Andrzej Sapkowski	235	Fantasy	SuperNowa	1990-01-01	Add Favorite List Add Read List
The Witcher 2: Sword of Destiny	Andrzej Sapkowski	343	Fantasy	SuperNowa	1992-01-01	Add Favorite List Add Read List

Figure 25: Book Page for Normal User



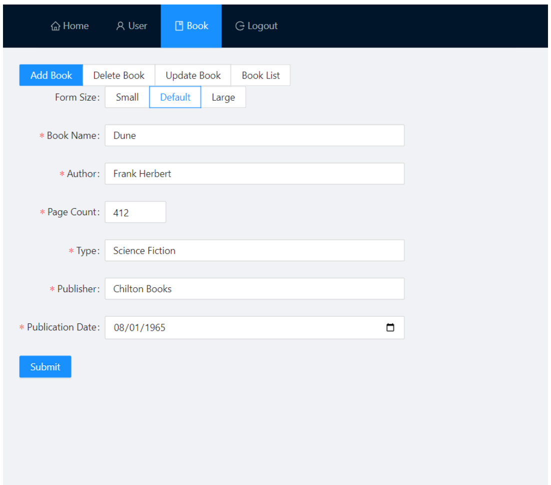
● Search Book By ID ○ Search Book By Name

Search Book By Id : 0

Book Name	Author	Page Count	Type	Publisher	Publication Date	Action
I, Robot	Isaac Asimov	253	Science Fiction	Gnome Press	1950-12-02	Add Favorite List Add Read List
The Witcher	Andrzej Sapkowski	235	Fantasy	SuperNowa	1990-01-01	Add Favorite List Add Read List
The Witcher 2: Sword of Destiny	Andrzej Sapkowski	343	Fantasy	SuperNowa	1992-01-01	Add Favorite List Add Read List

Figure 26: Book Page for Admins

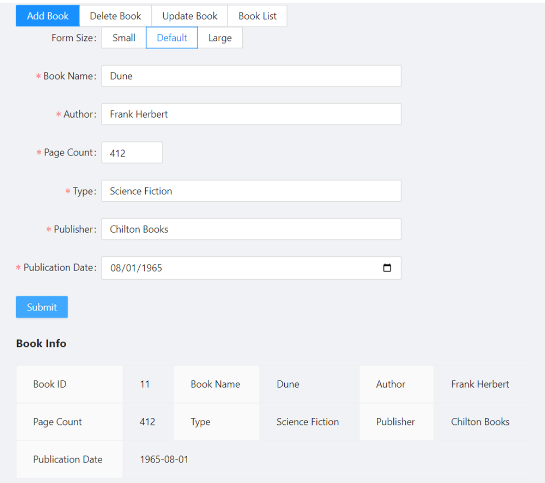
■ **`Add Book`** This page is for adding a book to the database and is unique to admins. When the option is chosen, `Add Book` is rendered and provides a form to add a book. The fields of `Book` entity: name, author, page count, type, publisher, and publication date. All the fields are required, and the date can be chosen from the calendar. When the book is submitted, a request is sent to the back-end with the help of `BookService`. When the book is saved, the saved book's information is shown below the form. Additionally, the size of the form can be changed by using the 'Form Size' radio buttons.



Form Size: Default

* Book Name: Dune
* Author: Frank Herbert
* Page Count: 412
* Type: Science Fiction
* Publisher: Chilton Books
* Publication Date: 08/01/1965

Figure 27: Book Adding Form



Form Size: Default

* Book Name: Dune
* Author: Frank Herbert
* Page Count: 412
* Type: Science Fiction
* Publisher: Chilton Books
* Publication Date: 08/01/1965

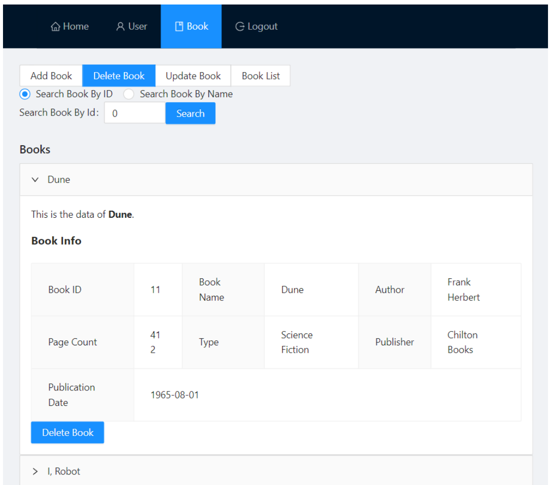
Book Info

Book ID	11	Book Name	Dune	Author	Frank Herbert
Page Count	412	Type	Science Fiction	Publisher	Chilton Books
Publication Date	1965-08-01				

Figure 28: Book Added to Database

■ **`Delete and Update Book`** The `Delete` and `Update` pages of the book are similar to user's and special to admins. Both pages support searching books by both id and username, and the searching option can be changed by using the radio buttons. Again, when the input area is cleared, the page is updated, and user data is reset. Both pages also support the pagination and pagination setting changes. Admin can change the pagination setting by using the navigation below. Additionally, the admin can see the books' information on both pages.

The only difference between `Delete Book` and `Delete User` is the visualizing data. I preferred the 'Descriptions' of antd to visualize the data of a book. The differences between `Update Book` and `Update User` are the visualizing method and updating form. Only the page count, publisher, or publication date of a book can be updated.



Search Book By Id: 0

Books

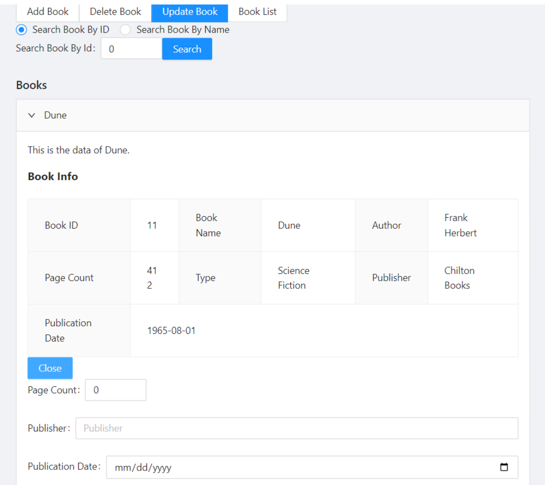
Dune

This is the data of **Dune**.

Book Info

Book ID	11	Book Name	Dune	Author	Frank Herbert
Page Count	412	Type	Science Fiction	Publisher	Chilton Books
Publication Date	1965-08-01				

I, Robot



Search Book By Id: 0

Books

Dune

This is the data of Dune.

Book Info

Book ID	11	Book Name	Dune	Author	Frank Herbert
Page Count	412	Type	Science Fiction	Publisher	Chilton Books
Publication Date	1965-08-01				

Page Count: 0

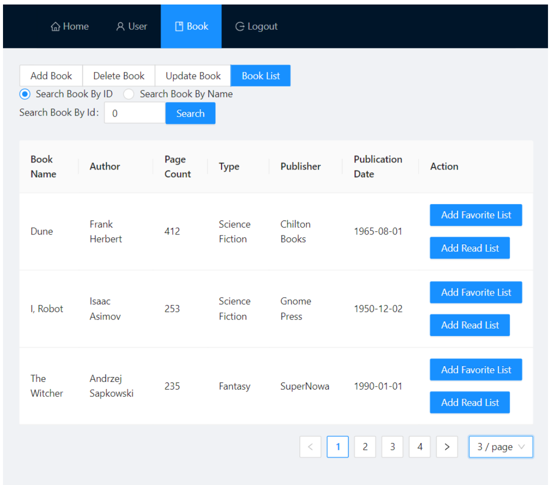
Publisher: Publisher

Publication Date: mm/dd/yyyy

Figure 29: Book Delete

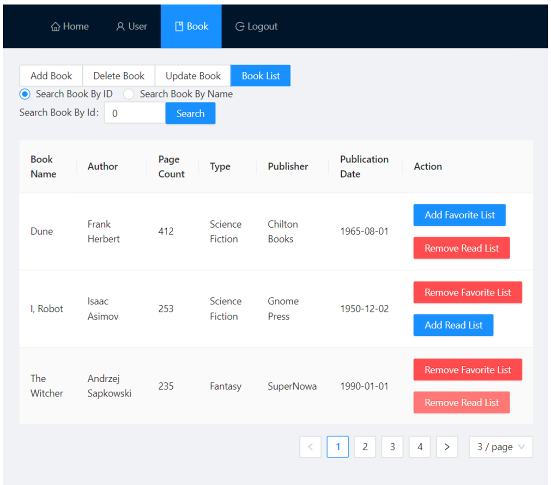
Figure 30: Book Update

■ **Book List** This page is accessible by all users and the default rendered page of Book item. 'Table' and 'Pagination' from antd are used on this page. It lists all the books in the database and provides searching by both id and name. Information about each book is provided in each row, and the actions operate on the book in that row. Users can add or remove the book to/from their read or favorite lists.



Search Book By Id: 0

Book Name	Author	Page Count	Type	Publisher	Publication Date	Action
Dune	Frank Herbert	412	Science Fiction	Chilton Books	1965-08-01	Add Favorite List Add Read List
I, Robot	Isaac Asimov	253	Science Fiction	Gnome Press	1950-12-02	Add Favorite List Add Read List
The Witcher	Andrzej Sapkowski	235	Fantasy	SuperNowa	1990-01-01	Add Favorite List Add Read List



Search Book By Id: 0

Book Name	Author	Page Count	Type	Publisher	Publication Date	Action
Dune	Frank Herbert	412	Science Fiction	Chilton Books	1965-08-01	Add Favorite List Remove Read List
I, Robot	Isaac Asimov	253	Science Fiction	Gnome Press	1950-12-02	Remove Favorite List Add Read List
The Witcher	Andrzej Sapkowski	235	Fantasy	SuperNowa	1990-01-01	Remove Favorite List Remove Read List

Figure 31: Book List

Figure 32: Book List Example

If the book is not added to the related list, the action button is blue, and the text of it specifies the adding operation. However, if the book is already added to the corresponding list, the action button is red, and the text of it specifies the removing operation. When the button is clicked, a request is sent to the database, and the book data is updated. As a result, the page is re-rendered with the new state.

This part was hard to code. Putting the action buttons and performing their operations were more complex, aside from the force of the table. It was easy to send a request and save the book to the user's list in the database; however, it was hard to update and re-render the book data and the state of the book. I thought to do this task without getting requests because of the complexity. Also, in the first place, the book information had a problem when the pagination was updated. To solve these problems, I spent several hours.

9 Conclusion

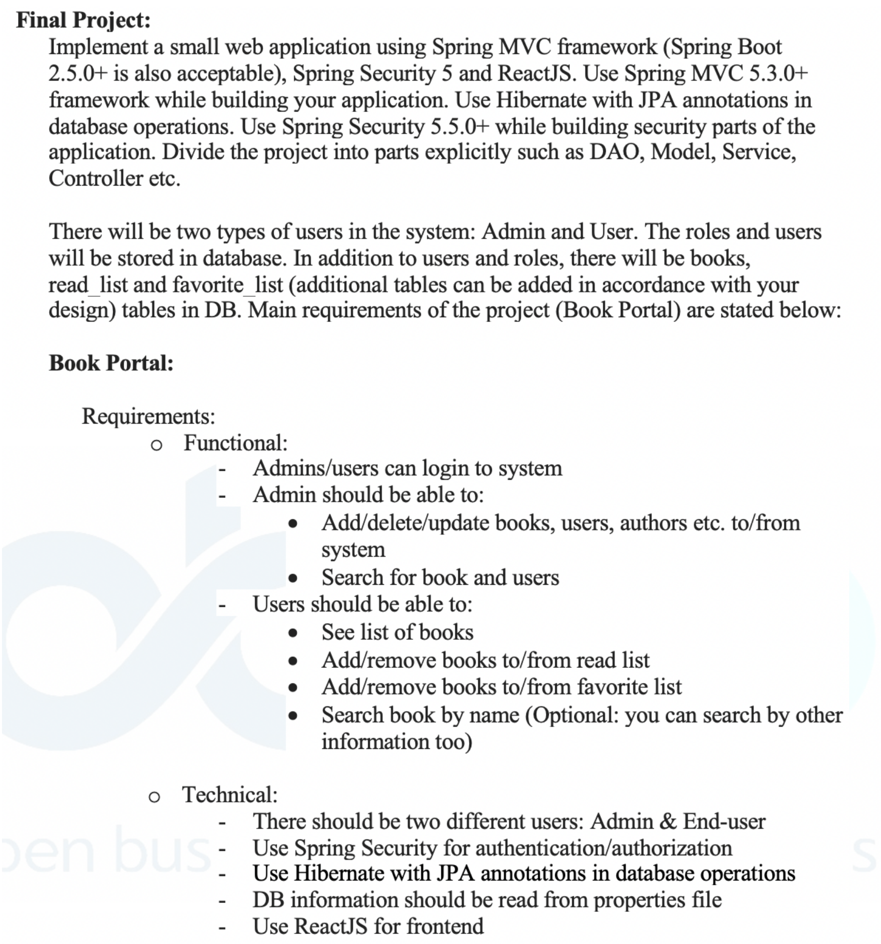
This summer practice was my first practice and the first experience in Java dn web development. The first part of my internship consisted of knowledge sharing. That part taught me so much about Java, Web Development, Spring. This knowledge sharing was quite intense and sometimes hard to follow, and the last part of it was not beneficial for me due to the intensity. I had hard times about React and I had to spend time and efforts to learn and straighten the information that I misunderstood. I believe information sharing part may be planned better by spreading the program for at least one week more, five weeks at total.

The second part of the practice consisted of developing a project from scratch. The project part was pretty beneficial because before this project, I was trying to do something exactly that needed to be done in my projects and homework at school, and I was trying to fix bugs if it happened. Although I did not have much difficulty in my previous experiences, I had a lot of difficulty in this project because I did not have time constraints or too much time in the projects and assignments I have done so far, and I have not dealt with such a comprehensive project before.

Although the requirements were given in the project, we were asked to think and develop many things such as thinking the design of the general flow. Until the end of the fourth week (last week of the normal internship program), we were asked to finish this project. Due to my limited time, I did not have time to think much, so I started developing it after a short thought process. This short-term thinking and inexperience made my job very difficult. I couldn't complete the project in the expected time due to the long time I wasted on unfocused parts such as design.

In short, I tried to do everything in the best way and move on to the next stage, instead of revealing something tangible on a part and then continuing to develop on it. But I realized too late that this wouldn't work in this type of development. Therefore, I was not able to finish my project at the end of the fourth week. However, I managed to finish my project with decent planning and developing until the end of the my internship.

In conclusion, in addition to many things I learned about Java and web development, I learned how not to develop a project from scratch and possible mistakes in a four-week period. In the last two weeks of my internship, I understood how a project should be developed and what needs to be done before it starts. Additionally, I do think I would choose to continue on web development even though I had hard time at some points.



Final Project:

Implement a small web application using Spring MVC framework (Spring Boot 2.5.0+ is also acceptable), Spring Security 5 and ReactJS. Use Spring MVC 5.3.0+ framework while building your application. Use Hibernate with JPA annotations in database operations. Use Spring Security 5.5.0+ while building security parts of the application. Divide the project into parts explicitly such as DAO, Model, Service, Controller etc.

There will be two types of users in the system: Admin and User. The roles and users will be stored in database. In addition to users and roles, there will be books, read_list and favorite_list (additional tables can be added in accordance with your design) tables in DB. Main requirements of the project (Book Portal) are stated below:

Book Portal:

Requirements:

- o Functional:
  - - Admins/users can login to system
  - - Admin should be able to:
    - Add/delete/update books, users, authors etc. to/from system
    - Search for book and users
  - - Users should be able to:
    - See list of books
    - Add/remove books to/from read list
    - Add/remove books to/from favorite list
    - Search book by name (Optional: you can search by other information too)
- o Technical:
  - - There should be two different users: Admin & End-user
  - - Use Spring Security for authentication/authorization
  - - Use Hibernate with JPA annotations in database operations
  - - DB information should be read from properties file
  - - Use ReactJS for frontend

Figure 33: Requirements of Book Portal

10 Appendicies

```
Book Portal Back-end
├ src.main
│ ├ java.tr.com.obss.jip.bookportal
│ │ ├ config
│ │ │ └ ...
│ │ ├ controller
│ │ │ └ ...
│ │ ├ entity
│ │ │ └ ...
│ │ ├ exception
│ │ │ └ ...
│ │ ├ filter
│ │ │ └ ...
│ │ ├ model
```

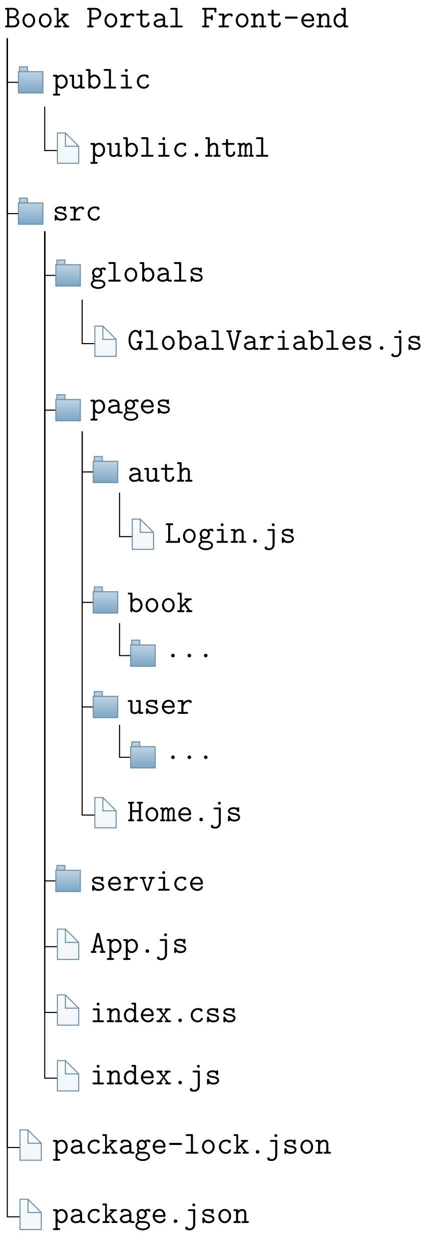
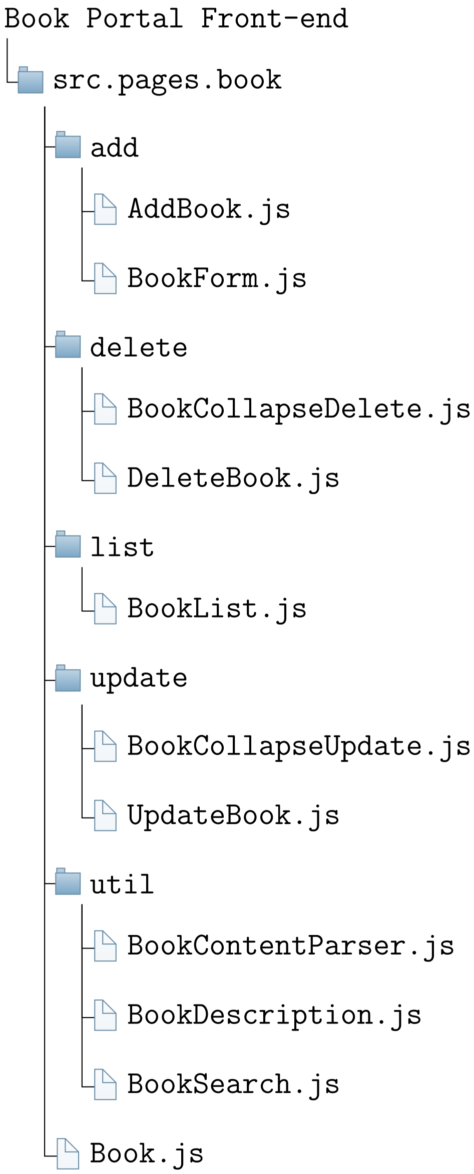


Figure 35: Front-end Directory Tree of Book Portal



```
Book Portal Front-end
└── src.pages.book
    ├── add
    │   ├── AddBook.js
    │   └── BookForm.js
    ├── delete
    │   ├── BookCollapseDelete.js
    │   └── DeleteBook.js
    ├── list
    │   └── BookList.js
    ├── update
    │   ├── BookCollapseUpdate.js
    │   └── UpdateBook.js
    ├── util
    │   ├── BookContentParser.js
    │   ├── BookDescription.js
    │   └── BookSearch.js
    └── Book.js
```

Figure 36: Directory Tree of Front-end Book

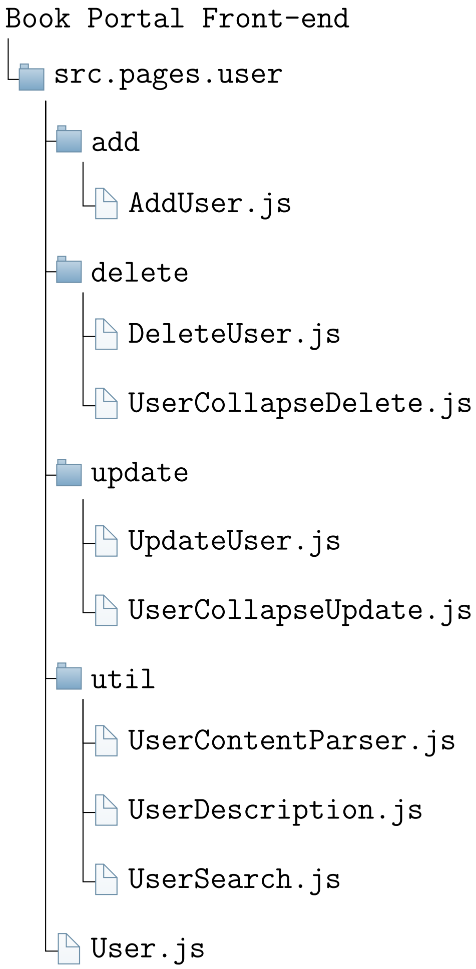


Figure 37: Directory Tree of Front-end User