Software Specifications and Tools

Final Report Part I

for

E-Lib

Version 1.1 approved

Prepared by Ionuț – Dragoș Neremzoiu

ACE UCV

17.01.2022

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

This document is meant to serve as an introduction to the main specifications and tools used for E-Library software application. In this document the reader should be able to have a general idea on what kind of specifications the app will have and what kind of tools will be used in order to develop the app, after having an in-depth analysis on the content of the document.

1.2 Intended Audience

The application will target 3 types of users: the customer, the employee, and the administrator. Thus, when logging in, the application will be able to redirect the users based on their role and they will be provided access to content limited to what their authorization grants them.

1.3 Product Scope

E-Lib is a library management software product, which is meant to simplify real life library services, such as:

- Searching for a book an user is interested in borrowing.
- Managing the books in the library and the list of records of people who are currently borrowing a certain book in a more efficient way.
- Processing the reservations as an employee.

Such simplifications allow customers, employees, and administrators to interact remote and keep them updated about library-related notifications.

1.4 References

http://maven.apache.org/guides/
https://dev.mysql.com/doc/
https://hibernate.org
https://spring.io
https://spring.io/projects/spring-boot
https://spring.io/projects/spring-security
https://spring.io/projects/spring-data-jpa
https://www.baeldung.com/spring-security-registration-password-encoding-bcrypt
https://www.baeldung.com/spring-boot-security-autoconfiguration
https://en.wikipedia.org/wiki/Apache_Maven
https://mvnrepository.com
https://maven.apache.org
https://www.geeksforgeeks.org/difference-between-spring-and-spring-boot/
srs template iee

2. Overall Description

2.1 Product Perspective

The context for the creation of this application is the need for a more efficient and less time-consuming way of providing quality library services mentioned above in 1.1 for all three types of users, which will benefit from using this kind of application. Moreover, considering the COVID – 19 pandemic, such digital platform will reduce human contact, when it comes to the need for borrowing a book, by appointing

2.2 Product Functions

Customer:

- Make reservation for a book or multiple books.
- Extend reservation for up to 3 weeks without being fined.
- Search and check for available books.
- Check remaining time of reservations.

Employee:

- Confirm reservations
- Cancel reservations
- Search and check for available books

Administrator:

- Disable and enable accounts
- Update users' role
- Manage or keep track of the books (such as inserting a new book, editing information about the book, or deleting the record of a certain book)

2.3 Assumptions and Dependencies

Since Apache Maven will be used as a build automation tool, certain dependencies on external modules, components, and other required plug-ins will be configured in order to simplify the process of software development. Some components can be found and retrieved from Maven Repository and configured in a **pom.xml** file and others are configured in a Spring file usually called **application.properties**, where usually details such as database access credentials are configured.

3. Software Tools

3.1 Eclipse

Eclipse is an IDE used in computer programming. It contains a workspace and an extensible plug-in system for customizing the environment. Since its primary use is for Java applications, it is mostly used for developing mobile, desktop GUI, web-based, enterprise, scientific, business, distributed and cloud-based applications.

3.2 Spring Tool Suite 4

Spring Tool Suite 4 is the next generation of Spring tooling for your favorite coding environment. Largely rebuilt from scratch, it provides world-class support for developing Spring-based enterprise applications, whether you prefer Eclipse, Visual Studio or Theia IDE.
In this particular case, it will provide support for Eclipse IDE.

3.3 Spring Framework

Spring Framework is an open-source application framework and inversion of control container for the Java platform. The framework’s core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE (Enterprise Edition) platform. Although the framework does not impose any specific programming model, it has become popular in the Java community as an addition to the Enterprise JavaBeans (EJB) model.

The Spring framework can be considered a collection of sub-frameworks, also called layers, such as **Spring AOP**, **Spring ORM** (Object-Relational Mapping), **Spring Web Flow**, and **Spring Web MVC**. You can use any of these modules separately while constructing a Web application, as well as together to provide better functionalities in a Web application.

3.3.1 Spring Boot

Spring Boot is built on top of the conventional Spring Framework. So, it provides all the features of Spring and is yet easier to use than Spring. Spring Boot is a microservice-based framework, making a production-ready application in very less time. In Spring Boot everything is auto-configured. We just need to user proper configuration for utilizing a particular functionality. Spring Boot is very useful if we want to develop REST API.

Difference between **Spring** and **Spring Boot**:

Spring	Spring Boot
Spring is an open-source lightweight framework widely used to develop enterprise applications.	Spring Boot is built on top of the conventional Spring Framework, widely used to develop REST APIs.
The most important feature of the Spring Framework is dependency injection.	The most important feature of the Spring Boot is Autoconfiguration.

It helps to create a loosely coupled application.	It helps to create a stand-alone application.
To run the Spring application, we need to set the server explicitly.	Spring Boot provides embedded servers such as **Tomcat**, **Jetty** etc.
To run the Spring application, a deployment descriptor is required.	There is no requirement for a deployment descriptor.
To create a Spring application, the developers write a lot of code.	It reduces the lines of code need to write a Spring-based application.
It doesn't provide support for the in-memory database.	It provides support for the in-memory database such as H2.

3.3.2 Spring Security

Spring Security is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based applications.

Spring Security is a framework that focuses on providing both authentication and authorization to Java applications. Like all Spring projects, the real power of Spring Security is found in how easily it can be extended to meet custom requirements.

Features

- Comprehensive and extensible support for both Authentication and Authorization
- Protection against session fixation, clickjacking, cross site request forgery, etc.
- Servlet API integration
- Optional integration with **Spring Web MVC**.
- Java Configuration support.
- Much more…

Spring Security framework supports wide range of authentication models. These models either provided by third parties or framework itself. Spring Security supports integration with all of these technologies:

- HTTP BASIC authentication headers
- HTTP Digest authentication headers
- HTTP X.509 client certificate exchange
- LDAP (Lightweight Directory Access Protocol)
- Form-based authentication
- OpenID authentication
- Automatic remember-me authentication
- Kerberos
- JOSSO (Java Open Source Single Sign-On)
- AppFuse
- AndroMDA
- Mule ESB
- DWR(Direct Web Request)

3.3.3 Spring Data JPA

Spring Data JPA, part of the larger Spring Data family, makes it easy to easily implement JPA based repositories. This module deals with enhanced support for JPA based data access layers. It makes it easier to build Spring-powered applications that use data access technologies.

Implementing a data access layer of an application has been cumbersome for quite a while. Too much boilerplate code has to be written to execute simple queries as well as perform pagination, and auditing. Spring Data JPA aims to significantly improve the implementation of data access layers by reducing the effort to the amount that’s actually needed. As a developer you write your repository interfaces, including custom finder methods, and Spring will provide the implementation automatically.

Features:

- Sophisticated support to build repositories based on Spring and JPA
- Support for Querydsl predicates and thus type-safe JPA queries
- Transparent auditing of domain class
- Pagination support, dynamic query execution, ability to integrate custom data access code
- Validation of @Query annotated queries at bootstrap time
- Support for XML based entity mapping
- JavaConfig based repository configuration by introducing @EnableJpaRepositories

3.4 Hibernate

Hibernate ORM (or simply Hibernate) is an object-relational mapping tool for the Java programming language. It provides a framework for mapping an object-oriented domain model to a relational database. Hibernate handles object-relational impedance mismatch problems by replacing direct, persistent database accesses with high-level object handling functions. Hibernate's primary feature is mapping from Java classes to database tables, and mapping from Java data types to SQL data types. Hibernate also provides data query and retrieval facilities. It generates SQL calls and relieves the developer from the manual handling and object conversion of the result.

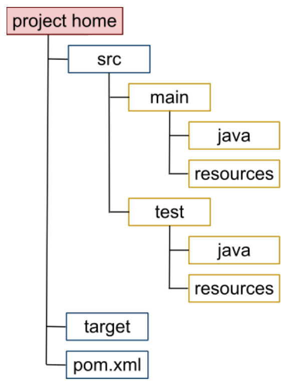
3.5 MySQL and MySQL Workbench

MySQL is a popular open-source relational database management system (RDBMS).

MySQL Workbench is the integrated environment for MySQL. It enables users to graphically administer MySQL databases and visually design database structures. It is available in two editions, the regular free and open-source *Community Edition* which may be downloaded from the MySQL website, and the proprietary *Standard Edition* which extends and improves the feature set of the *Community Edition*. In this particular case, I will use the Community Edition.

3.6 Apache Maven

Maven is a build automation tool used primarily for Java projects. It addresses two aspects of building software: how software is built and its dependencies. An **XML** file describes the software project being built, its dependencies on other external modules and components, the build order, directories, and required plug-ins. It comes with pre-defined targets for performing certain well-defined tasks such as compilation of code and its packaging. Maven dynamically downloads Java libraries and Maven plug-ins from one or more repositories such as the **Maven Repository: Central** and stores them in a local cache. This local cache of downloaded artifacts can also be updated with artifacts created by local projects. Public repositories can also be updated.



A directory structure for a Java project auto-generated by Maven

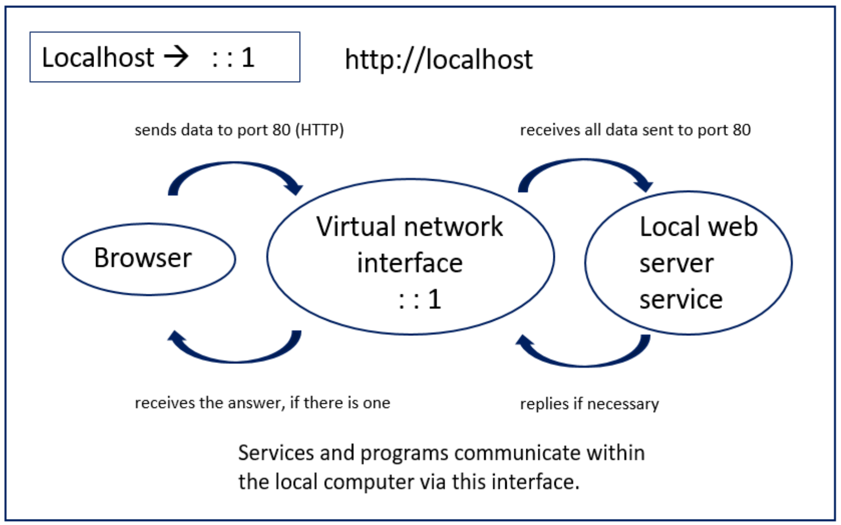
3.7 Security Specifications and Communications Interfaces

3.7.1 Security Specifications

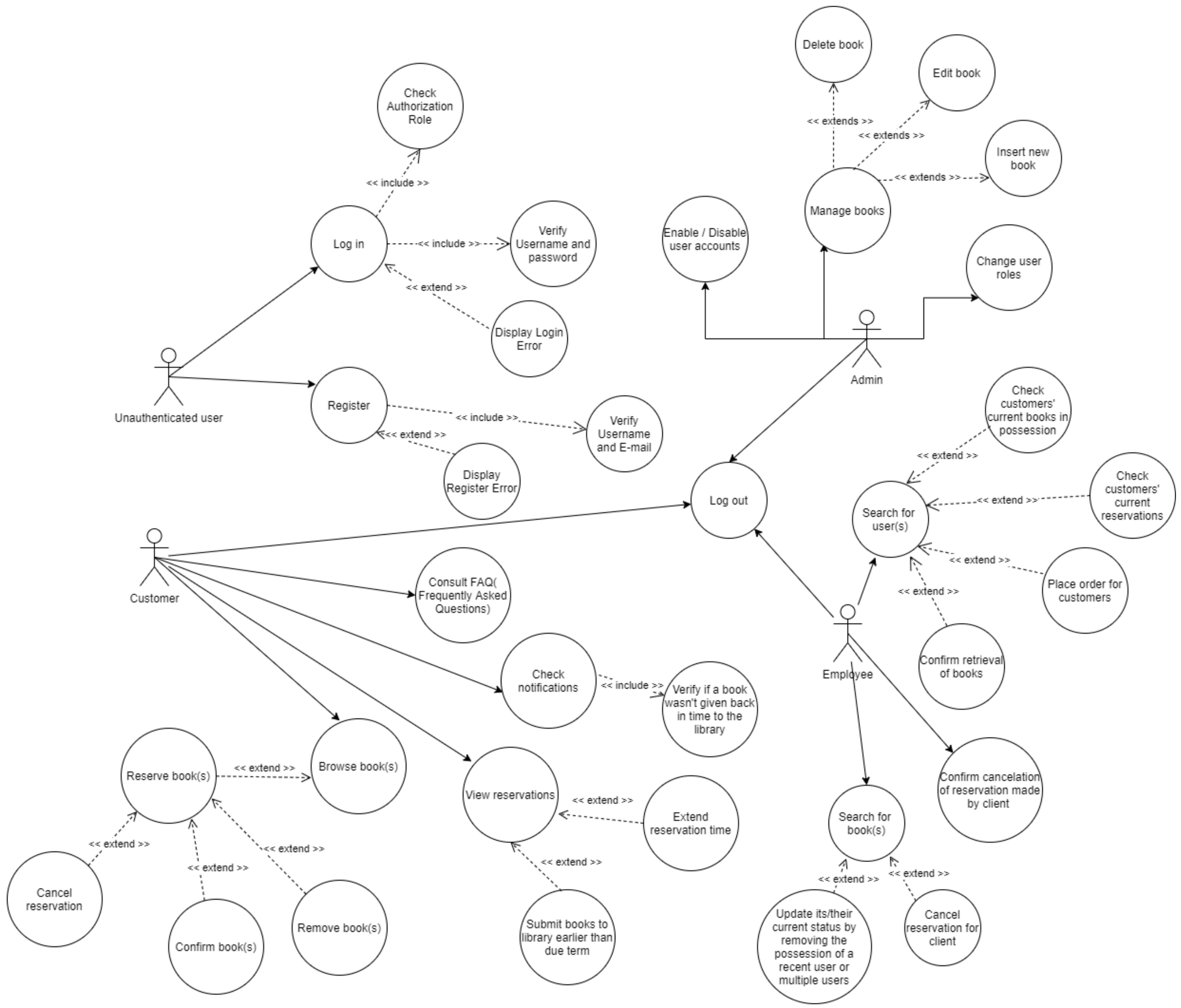
Security-wise, BCrypt, an encoding mechanism supported by Spring Security, will be used, since most of other encoding mechanisms, such as *MD5PasswordEncoder* (Message Digest) and *ShaPasswordEncoder* (Secure Hash Algorithm) use weaker algorithms and are now deprecated.

3.7.2 Communication Interfaces

Developers use the local host to test web applications and programs and for local development. During development, it is important to find out whether the application works as developed once it has internet access, because releasing a product without testing doesn't make sense. So, loopback is used by developers to test it. Developers can thus simulate a connection while also avoiding network errors. The connection just stays completely inside their own system.


Localhost → : : 1
http://localhost
sends data to port 80 (HTTP)
receives all data sent to port 80
Browser
Virtual network interface : : 1
Local web server service
receives the answer, if there is one
replies if necessary
Services and programs communicate within the local computer via this interface.

4. Initial System architecture



Specifications on the Use Case Diagram

- The human like drawings represent actors, which are meant to represent the types of users (**Customer**, **Employee**, **Admin**, **Unsigned User**) for this application.
- The circles which have an arrow pointed to represent the main functionalities (such as **Log in**, **Register**, **Consult FAQ**, etc.) of the application.

- The circles which have a dot-pointed arrow pointed to with **<< extend >>** mark represent the extended functionalities from certain functionalities (e.g.: **Extend reservation time** can only be performed if the user performs firstly **View Reservation** functionality).

- The circles which have a dot-pointed arrow pointed to with **<< include >>** mark represent the included server-side functionalities which take place in parallel after performing certain functionalities (e.g.: when a user tries to **Log in**, authentication and authorization processes begin check if the **Log in** credentials are valid. If they are validated, the user will be redirected to the home page assigned based on user's role. If the credentials are invalid, an error display message will be shown.).

Final Report Part II

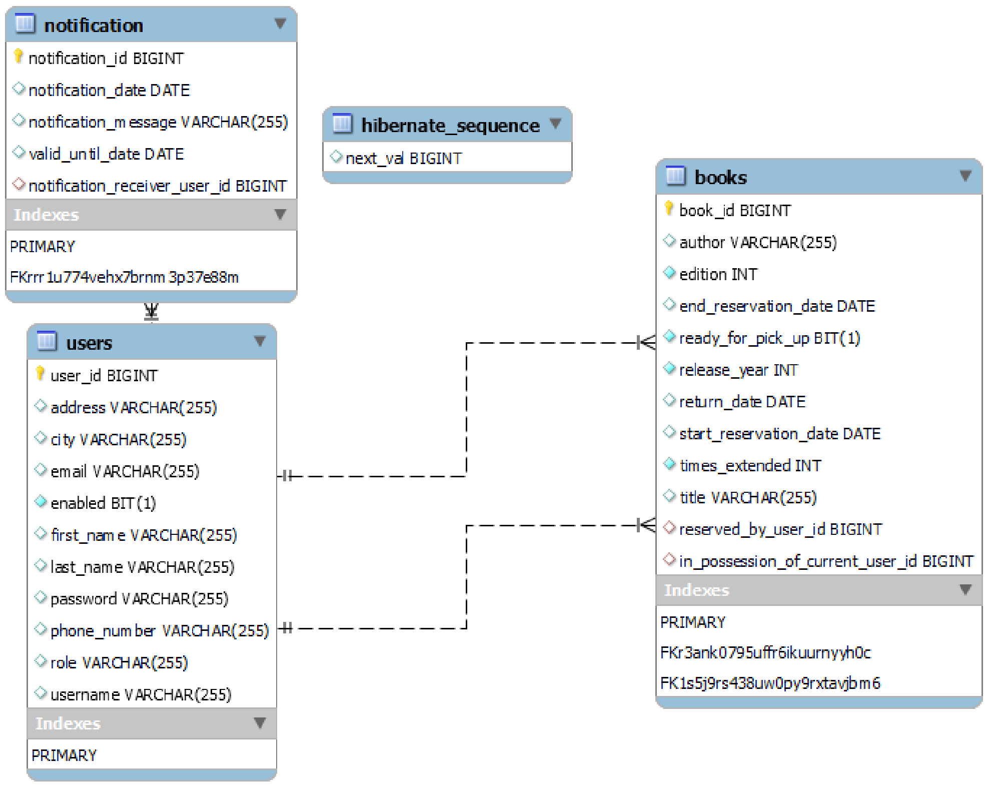
Student: Neremzoiu Ionuț-Dragoș

ACE UCV

Introduction

This part II of the report is meant to present and explain the overall design and implementation of this particular Spring MVC Web application.

Database Design



As it can be seen in the database diagram above, there are only 4 tables, i.e.: **notification**, **users**, **books** and **hibernate_sequence**. In order to clarify any possible doubts or possible misunderstandings, the following observations will be mentioned:

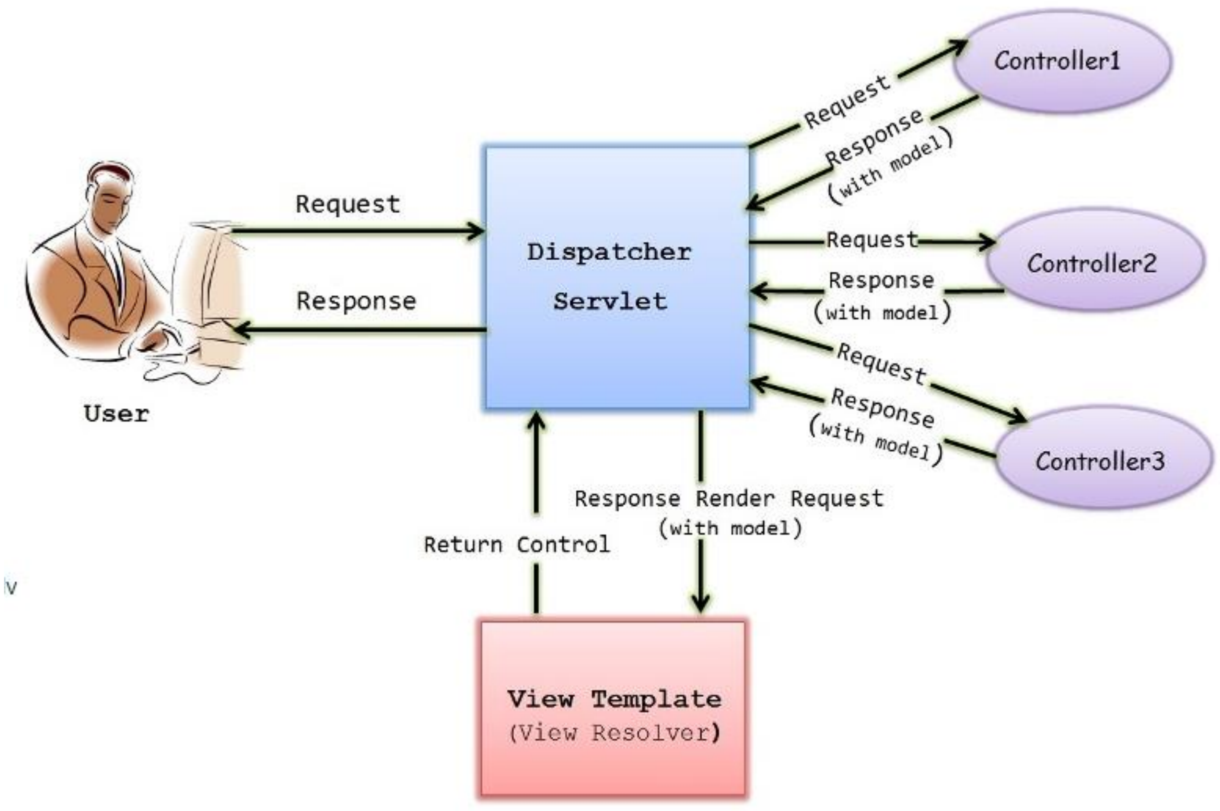
- **Users** and **notification** tables have a *one-to-many* relationship, meaning that a user can have multiple notifications.

- **Users** and **books** tables have a *one-to-many* relationship, meaning that a user can reserve or currently have in possession multiple books.

- The **reserved_by_user_id** field is meant to specify which user has currently reserved a book, whereas **in_possession_of_user_id** is meant to reveal the fact that even though the reservation period has expired for an user, the book hasn't been returned to the library by that user.

- **Hibernate_sequence** table is a table that is generated the moment the **strategy** for a *@GenereatedValue* of a field (usually an id) is set to *GenerationType.AUTO*. The way Hibernate interprets *AUTO* generation type has changed starting with Hibernate **version 5.0**. When using Hibernate **v 4.0** and *Generation Type* as *AUTO*, specifically for MySql, Hibernate would choose the *IDENTITY* strategy (and thus use the *AUTO_INCREMENT* feature) for generating IDs for the table in question. Starting with **version 5.0** when *Generation Type* is selected as *AUTO*, Hibernate uses *SequenceStyleGenerator* regardless of the database. In case of MySql Hibernate emulates a sequence using a table, because MySql doesn't support the standard sequence type natively. Hence, that's why if strategy *AUTO* is used, Hibernate will generate a table called hibernate_sequence to provide the next number for the ID sequence. (Such details will also be found in the code)

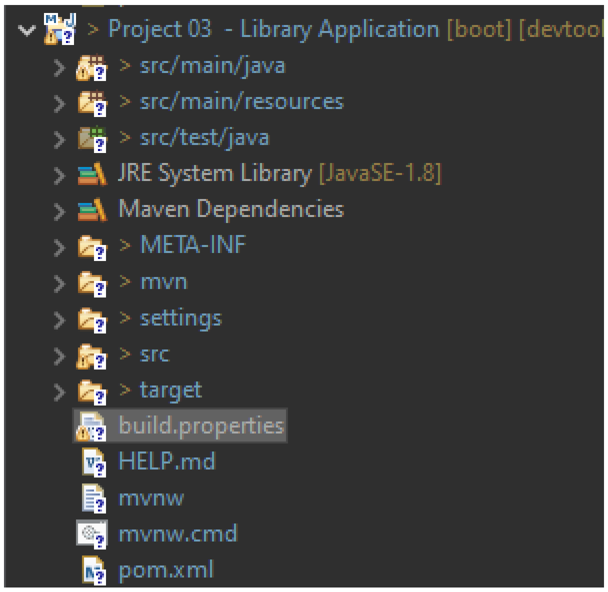
Spring MVC Architecture



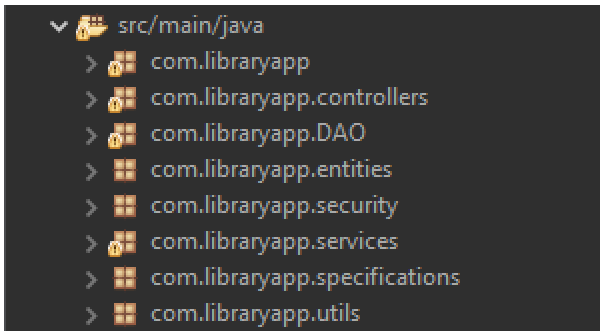
Flow of a Spring MVC application:

1. User makes a request through an URL.
2. URL is passed to dispatcher servlet.
3. Dispatcher servlet passes the request to the corresponding controller based on URL mapping.
4. Controller performs the task and returns the model and view.
5. Dispatcher servlet maps the view name to the corresponding jsp (any view technology) using View Resolver.
6. View renders the model and displays it.

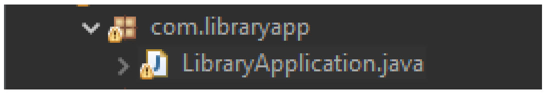
Project Structure and Parts of Implementation

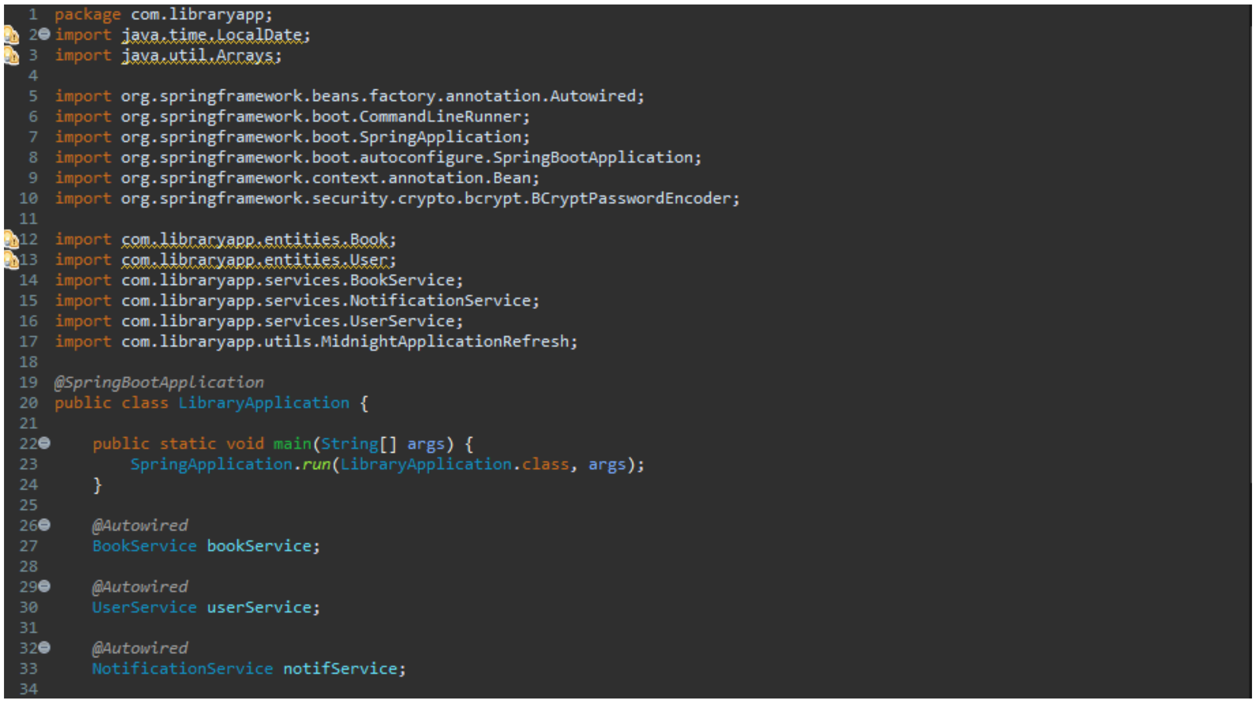

```
> Project 03  - Library Application [boot] [devtool
  > src/main/java
  > src/main/resources
  > src/test/java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  > META-INF
  > mvn
  > settings
  > src
  > target
  build.properties
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

As it can be seen in the screenshot above, the project consists of multiple files, each of them with their own content for specific purposes. In this report, however only the most important files will be discussed in more details, namely: src/main/java, src/main/resources, src/test/java, JRE System Library, Maven Dependencies, and pom.xml. And still, even for these directories not all the packages will be presented in depth, but only the most important.


```
src/main/java
  com.libraryapp
  com.libraryapp.controllers
  com.libraryapp.DAO
  com.libraryapp.entities
  com.libraryapp.security
  com.libraryapp.services
  com.libraryapp.specifications
  com.libraryapp.utils
```

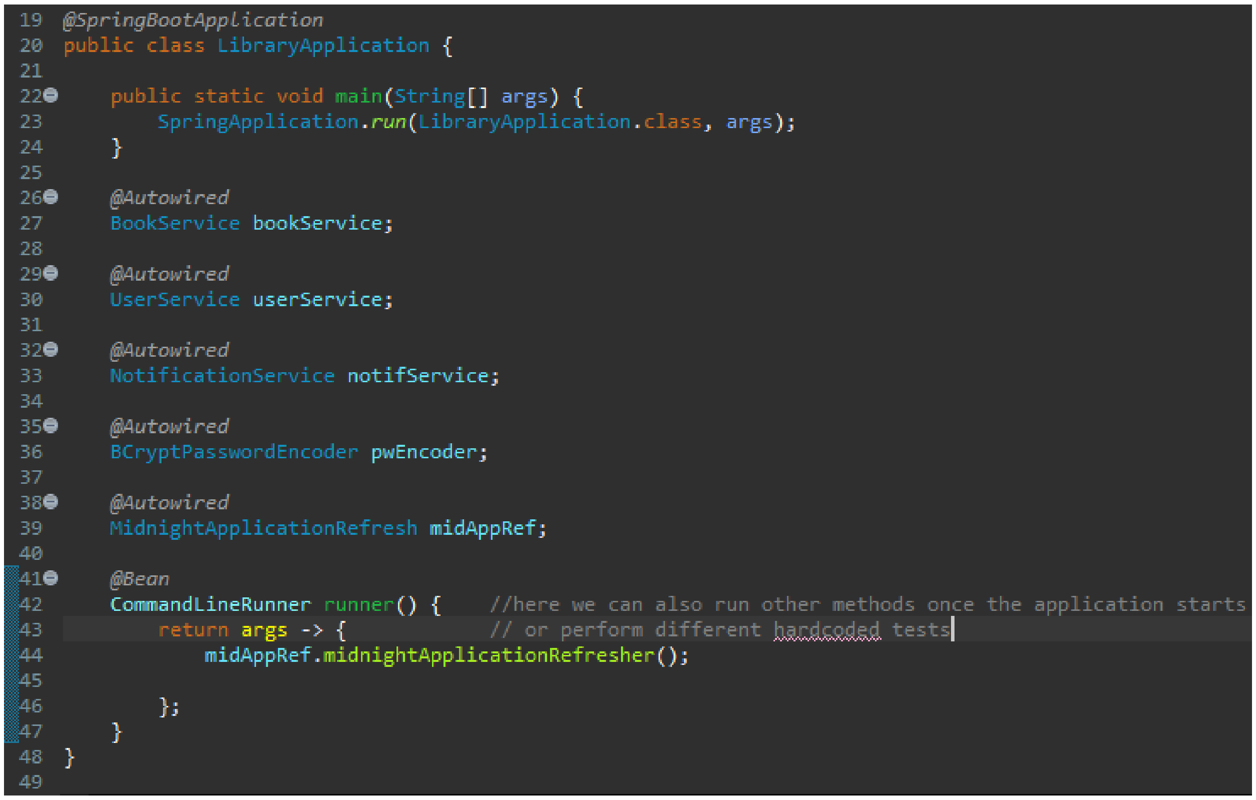
In the screenshot above, we have src/main/java, where the backend development is made. In this particular case it is modularized into 8 packages, each of them handling different tasks.



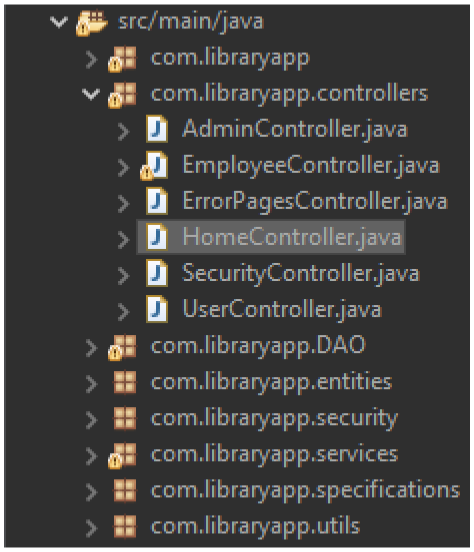


```
package com.libraryapp;
import java.time.LocalDate;
import java.util.Arrays;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import com.libraryapp.entities.Book;
import com.libraryapp.entities.User;
import com.libraryapp.services.BookService;
import com.libraryapp.services.NotificationService;
import com.libraryapp.services.UserService;
import com.libraryapp.utils.MidnightApplicationRefresh;

@SpringBootApplication
public class LibraryApplication {

    public static void main(String[] args) {
        SpringApplication.run(LibraryApplication.class, args);
    }

    @Autowired
    BookService bookService;

    @Autowired
    UserService userService;

    @Autowired
    NotificationService notifService;

```

In this particular *LibraryApplication.java* class, it’s usually the *main(String[] args)* function in which the application runs and sometimes a **@Bean** annotation such *CommandLineRunner runner()* can be added and it runs as soon as the application starts as well. Such beans are usually used to run methods once the application runs or to hardcode tests such as creating and inserting new instances of a DAO class in the database and so on.

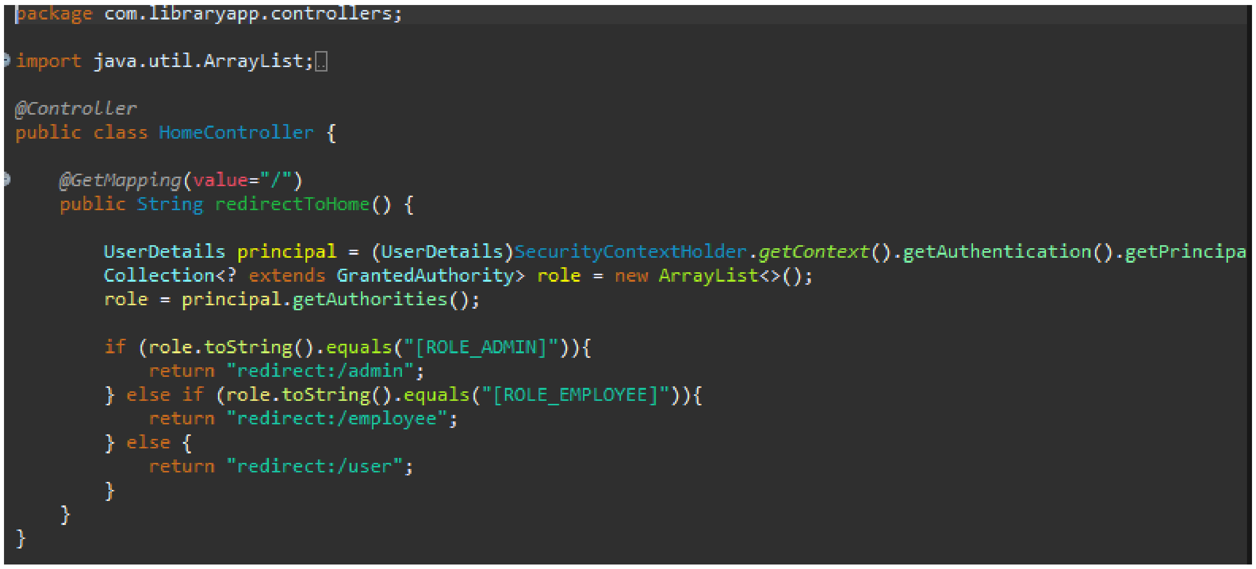


```java
@SpringBootApplication
public class LibraryApplication {

    public static void main(String[] args) {
        SpringApplication.run(LibraryApplication.class, args);
    }

    @Autowired
    BookService bookService;

    @Autowired
    UserService userService;

    @Autowired
    NotificationService notifService;

    @Autowired
    BCryptPasswordEncoder pwEncoder;

    @Autowired
    MidnightApplicationRefresh midAppRef;

    @Bean
    CommandLineRunner runner() {    //here we can also run other methods once the application starts
        return args -> {            // or perform different hardcoded tests
            midAppRef.midnightApplicationRefresher();

        };
    }
}

```



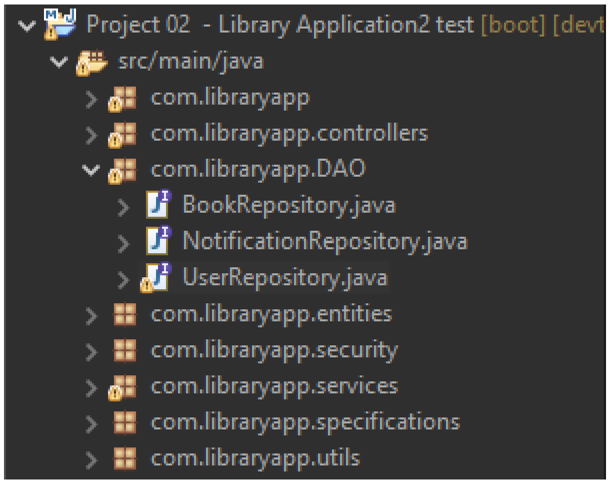
```
src/main/java
  com.libraryapp
  com.libraryapp.controllers
    AdminController.java
    EmployeeController.java
    ErrorPagesController.java
    HomeController.java
    SecurityController.java
    UserController.java
  com.libraryapp.DAO
  com.libraryapp.entities
  com.libraryapp.security
  com.libraryapp.services
  com.libraryapp.specifications
  com.libraryapp.utils
```

In this screenshot, it is revealed the way the controllers are modularized. The role of the controllers is usually to handle specific request which is mapped by its request mapping.



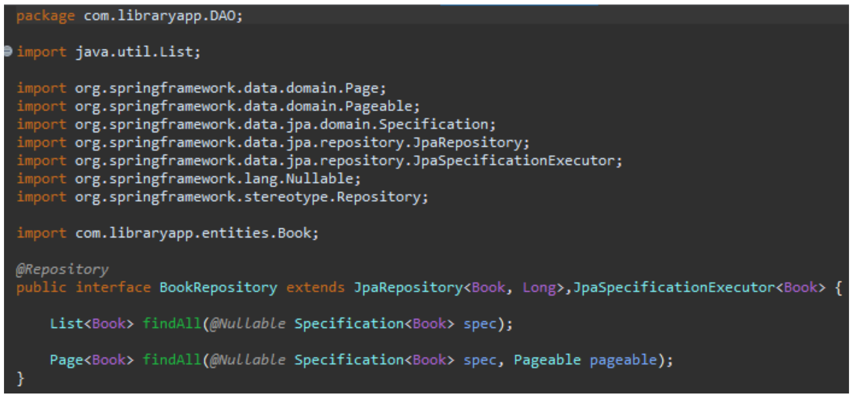
```
package com.libraryapp.controllers;

import java.util.ArrayList;

@Controller
public class HomeController {

    @GetMapping(value="/")
    public String redirectToHome() {

        UserDetails principal = (UserDetails)SecurityContextHolder.getContext().getAuthentication().getPrincipa
        Collection<? extends GrantedAuthority> role = new ArrayList<>();
        role = principal.getAuthorities();

        if (role.toString().equals("[ROLE_ADMIN]")){
            return "redirect:/admin";
        } else if (role.toString().equals("[ROLE_EMPLOYEE]")){
            return "redirect:/employee";
        } else {
            return "redirect:/user";
        }
    }
}
```

Above, we can see an example of a *HomeController* which handles the redirection of a user to a based-role home page depending on the role he/she was assigned (e.g.: admin, user, employee).



```
Project 02  - Library Application2 test [boot] [devt
  src/main/java
    com.libraryapp
    com.libraryapp.controllers
    com.libraryapp.DAO
      BookRepository.java
      NotificationRepository.java
      UserRepository.java
    com.libraryapp.entities
    com.libraryapp.security
    com.libraryapp.services
    com.libraryapp.specifications
    com.libraryapp.utils
```

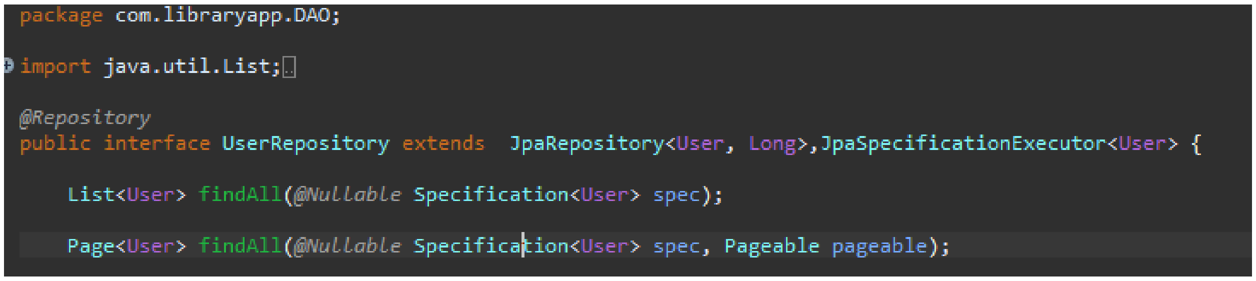
In this screenshot, it is revealed the way the reporistories or DAOs (Data Access Objects) are modularized. Their role is to provide access to a particular data resource without coupling the resource's API to the business logic.



```java
package com.libraryapp.DAO;

import java.util.List;

import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.domain.Specification;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.JpaSpecificationExecutor;
import org.springframework.lang.Nullable;
import org.springframework.stereotype.Repository;

import com.libraryapp.entities.Book;

@Repository
public interface BookRepository extends JpaRepository<Book, Long>,JpaSpecificationExecutor<Book> {

    List<Book> findAll(@Nullable Specification<Book> spec);

    Page<Book> findAll(@Nullable Specification<Book> spec, Pageable pageable);
}
```

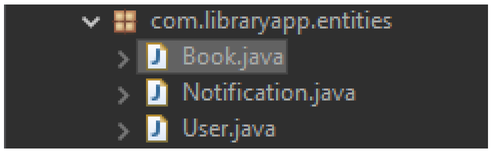
As it can be seen in the screenshot above, to make a repository for a particular database entity, it is enough to add @Repository above an interface and make it extend the JPARepository (Java Persistance Access Repository) interface. Apart from the general way a repository interface works, in this case it also extends JPASpecificationExecutor class of a generic type(in this case Book), which basically allows us to retrieve queries in a more efficient way, by providing "specifications". Specifications will be discussed in further detail in the following part of the report, where services will be presented.

The example below also extends JpaSpecificationExecutor in order to perform different search filter.



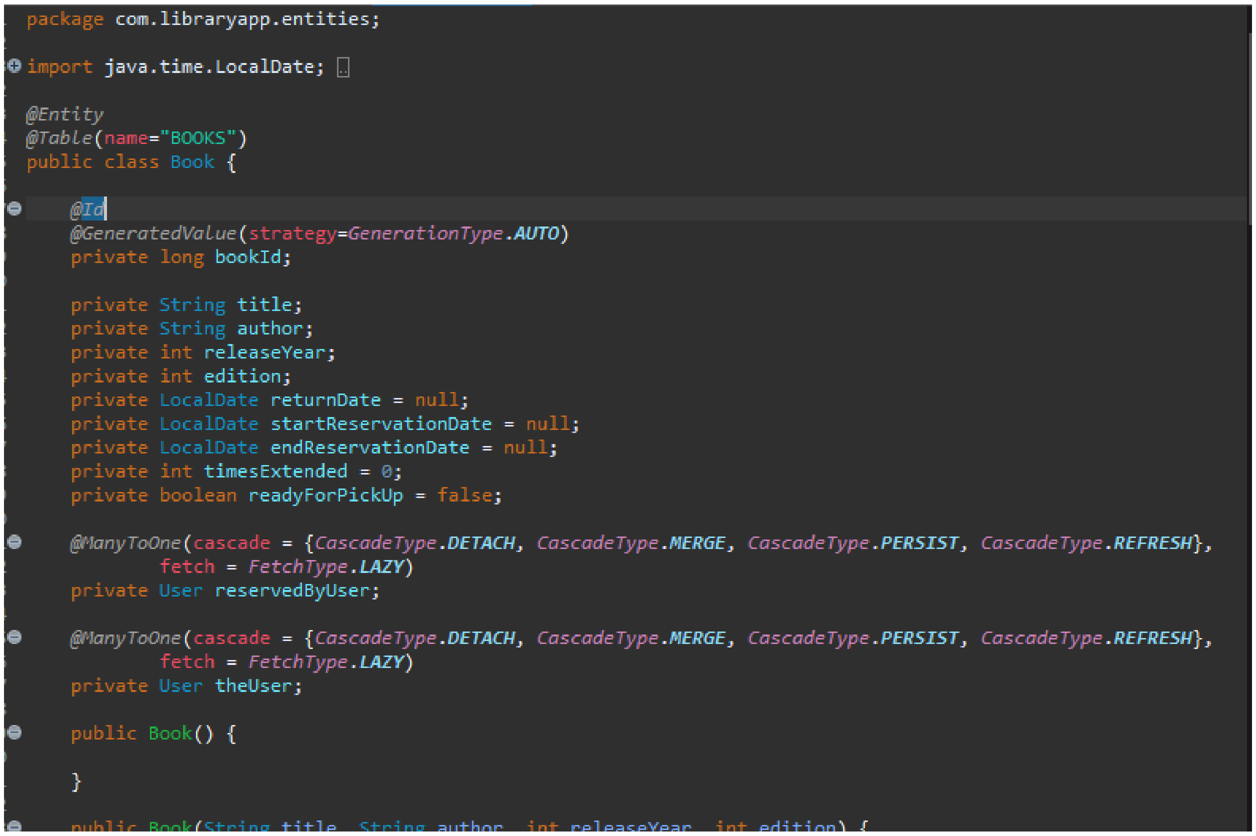
```java
package com.libraryapp.DAO;

import java.util.List;

@Repository
public interface UserRepository extends  JpaRepository<User, Long>,JpaSpecificationExecutor<User> {

    List<User> findAll(@Nullable Specification<User> spec);

    Page<User> findAll(@Nullable Specification<User> spec, Pageable pageable);
```

The following important package is the one in which all database entity classes are stored:

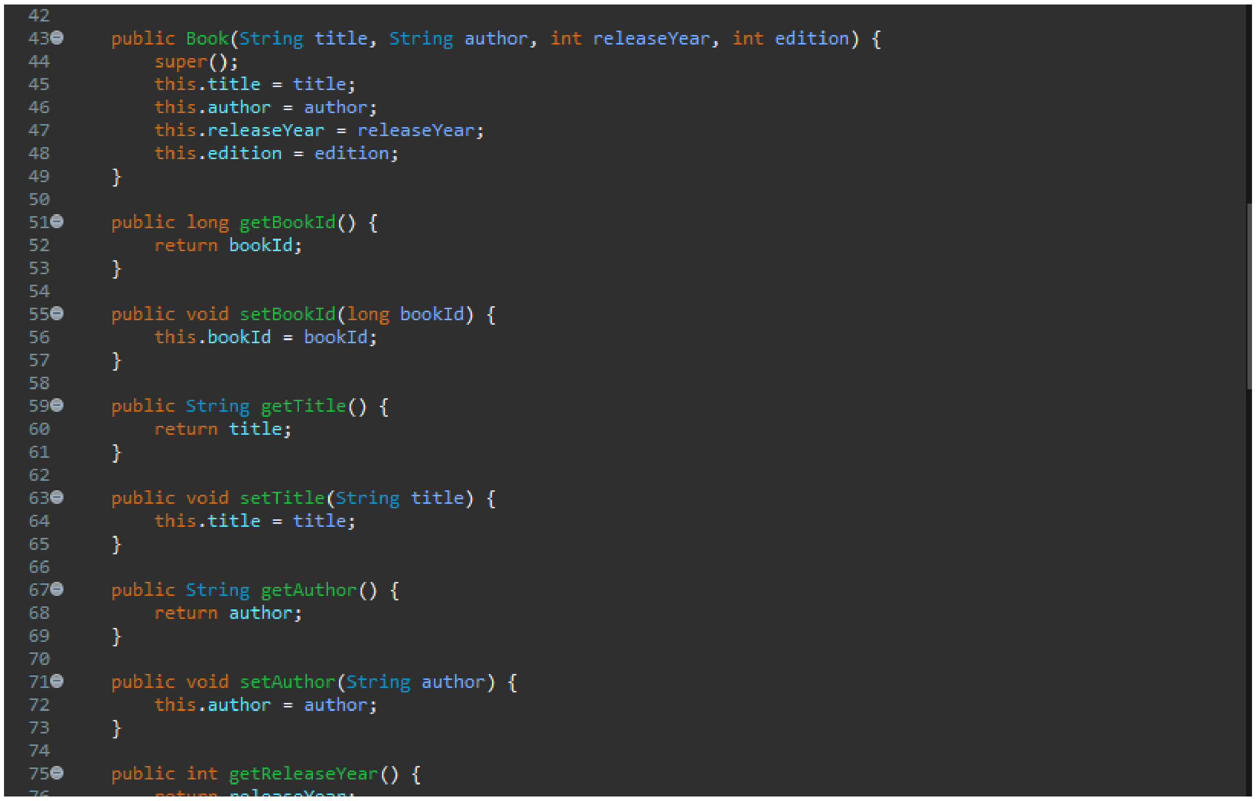


```
com.libraryapp.entities
  Book.java
  Notification.java
  User.java
```

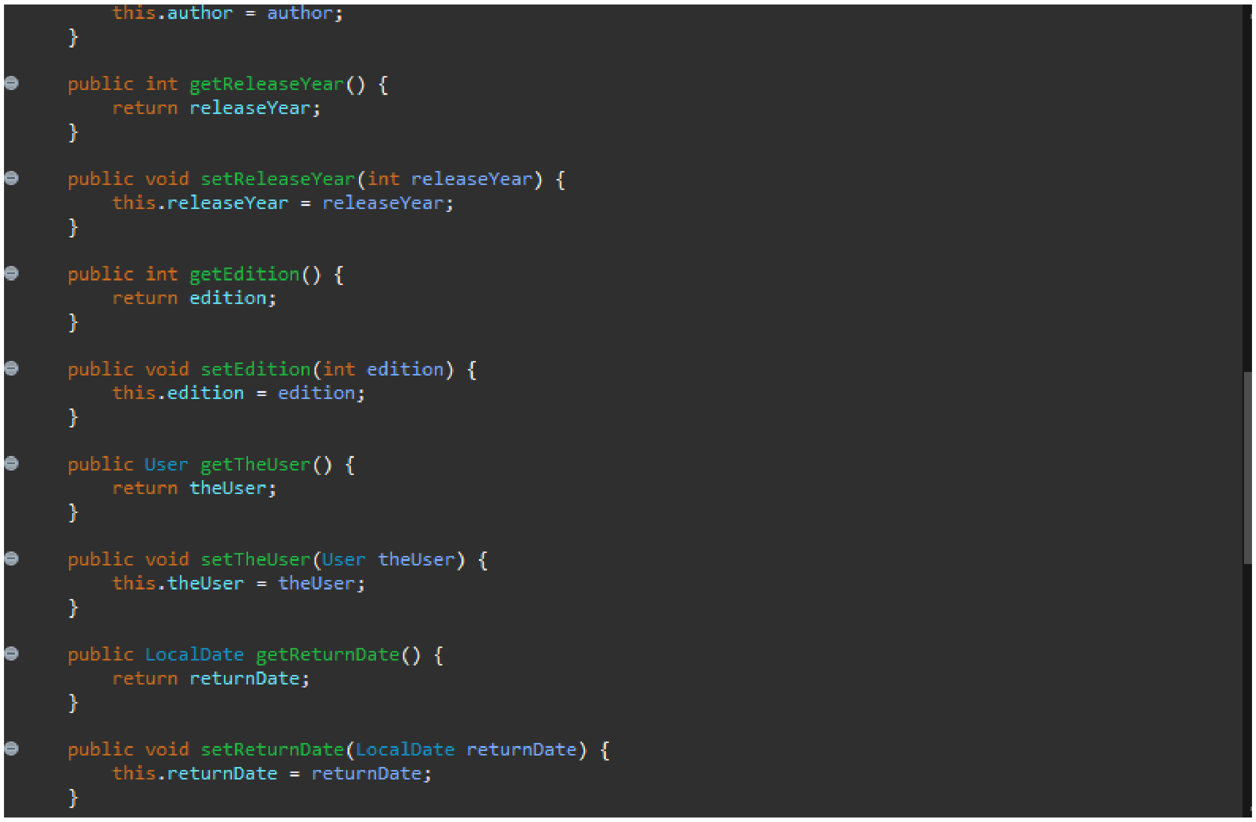
The following screenshots show the class for Book Entity:



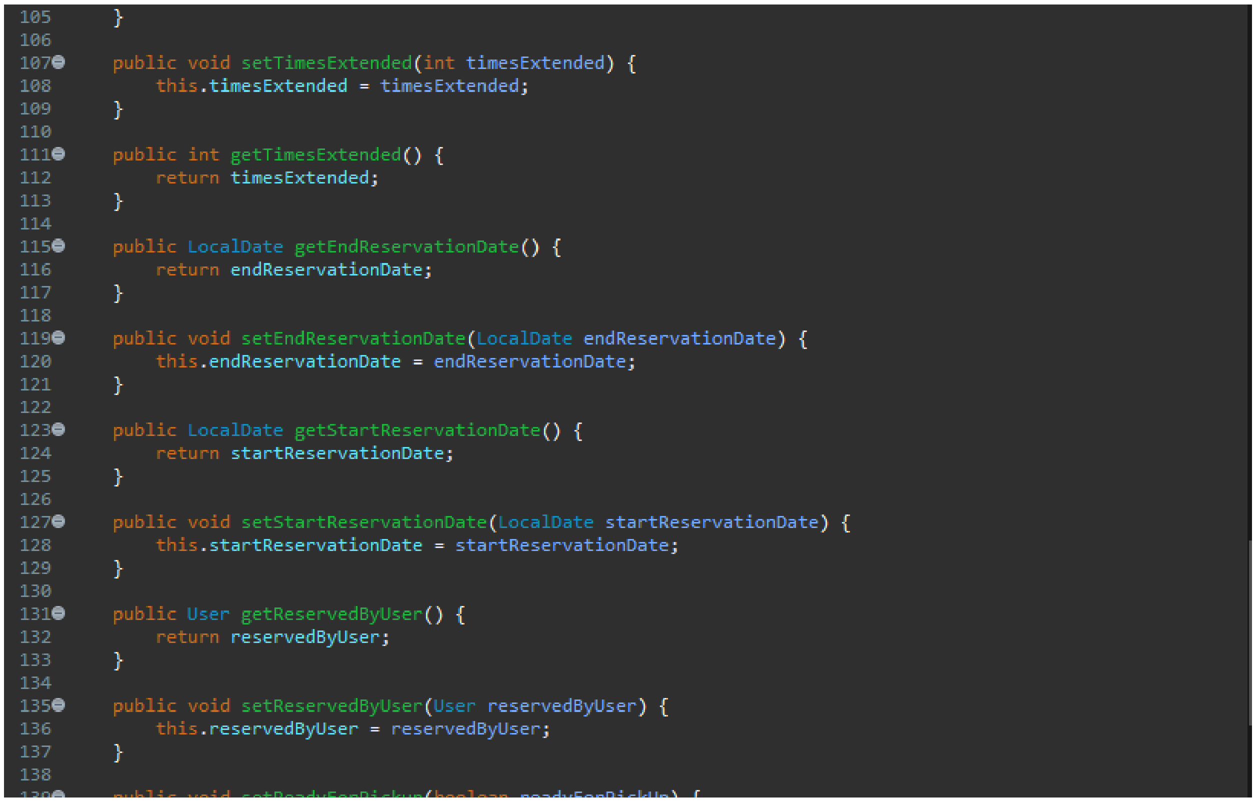
```java
package com.libraryapp.entities;

import java.time.LocalDate;

@Entity
@Table(name="BOOKS")
public class Book {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private long bookId;

    private String title;
    private String author;
    private int releaseYear;
    private int edition;
    private LocalDate returnDate = null;
    private LocalDate startReservationDate = null;
    private LocalDate endReservationDate = null;
    private int timesExtended = 0;
    private boolean readyForPickUp = false;

    @ManyToOne(cascade = {CascadeType.DETACH, CascadeType.MERGE, CascadeType.PERSIST, CascadeType.REFRESH},
            fetch = FetchType.LAZY)
    private User reservedByUser;

    @ManyToOne(cascade = {CascadeType.DETACH, CascadeType.MERGE, CascadeType.PERSIST, CascadeType.REFRESH},
            fetch = FetchType.LAZY)
    private User theUser;

    public Book() {

    }

    public Book(String title, String author, int releaseYear, int edition) {
```



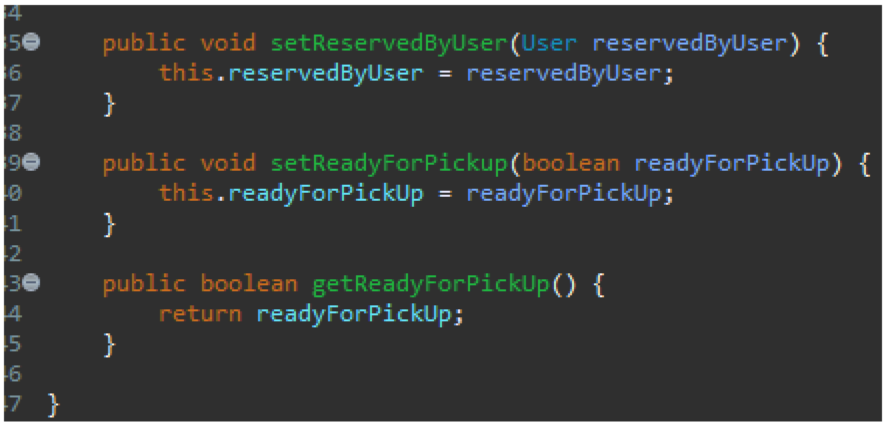
```java

    public Book(String title, String author, int releaseYear, int edition) {
        super();
        this.title = title;
        this.author = author;
        this.releaseYear = releaseYear;
        this.edition = edition;
    }

    public long getBookId() {
        return bookId;
    }

    public void setBookId(long bookId) {
        this.bookId = bookId;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getAuthor() {
        return author;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    public int getReleaseYear() {
```



```java
        this.author = author;
    }

    public int getReleaseYear() {
        return releaseYear;
    }

    public void setReleaseYear(int releaseYear) {
        this.releaseYear = releaseYear;
    }

    public int getEdition() {
        return edition;
    }

    public void setEdition(int edition) {
        this.edition = edition;
    }

    public User getTheUser() {
        return theUser;
    }

    public void setTheUser(User theUser) {
        this.theUser = theUser;
    }

    public LocalDate getReturnDate() {
        return returnDate;
    }

    public void setReturnDate(LocalDate returnDate) {
        this.returnDate = returnDate;
    }
```

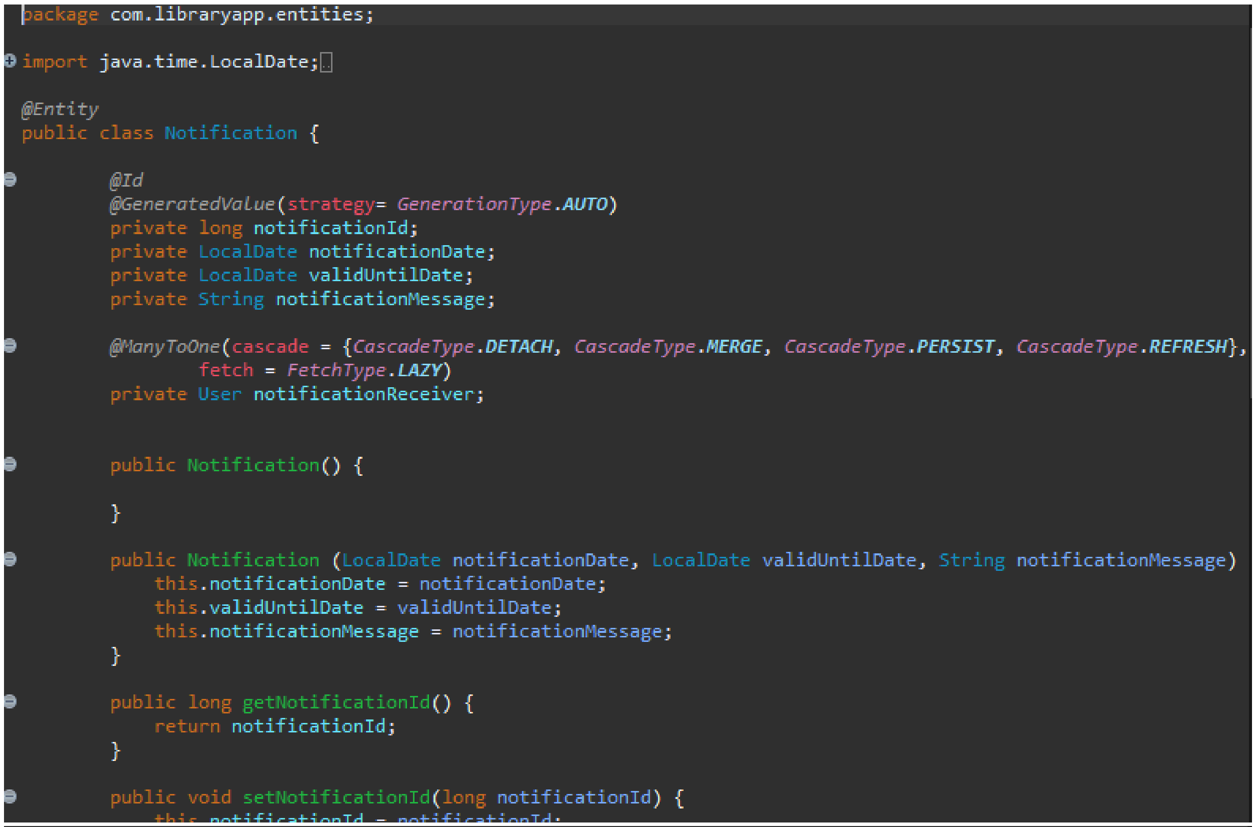


```
    }

    public void setTimesExtended(int timesExtended) {
        this.timesExtended = timesExtended;
    }

    public int getTimesExtended() {
        return timesExtended;
    }

    public LocalDate getEndReservationDate() {
        return endReservationDate;
    }

    public void setEndReservationDate(LocalDate endReservationDate) {
        this.endReservationDate = endReservationDate;
    }

    public LocalDate getStartReservationDate() {
        return startReservationDate;
    }

    public void setStartReservationDate(LocalDate startReservationDate) {
        this.startReservationDate = startReservationDate;
    }

    public User getReservedByUser() {
        return reservedByUser;
    }

    public void setReservedByUser(User reservedByUser) {
        this.reservedByUser = reservedByUser;
    }

    public void setReadyForPickup(boolean readyForPickUp) {
```

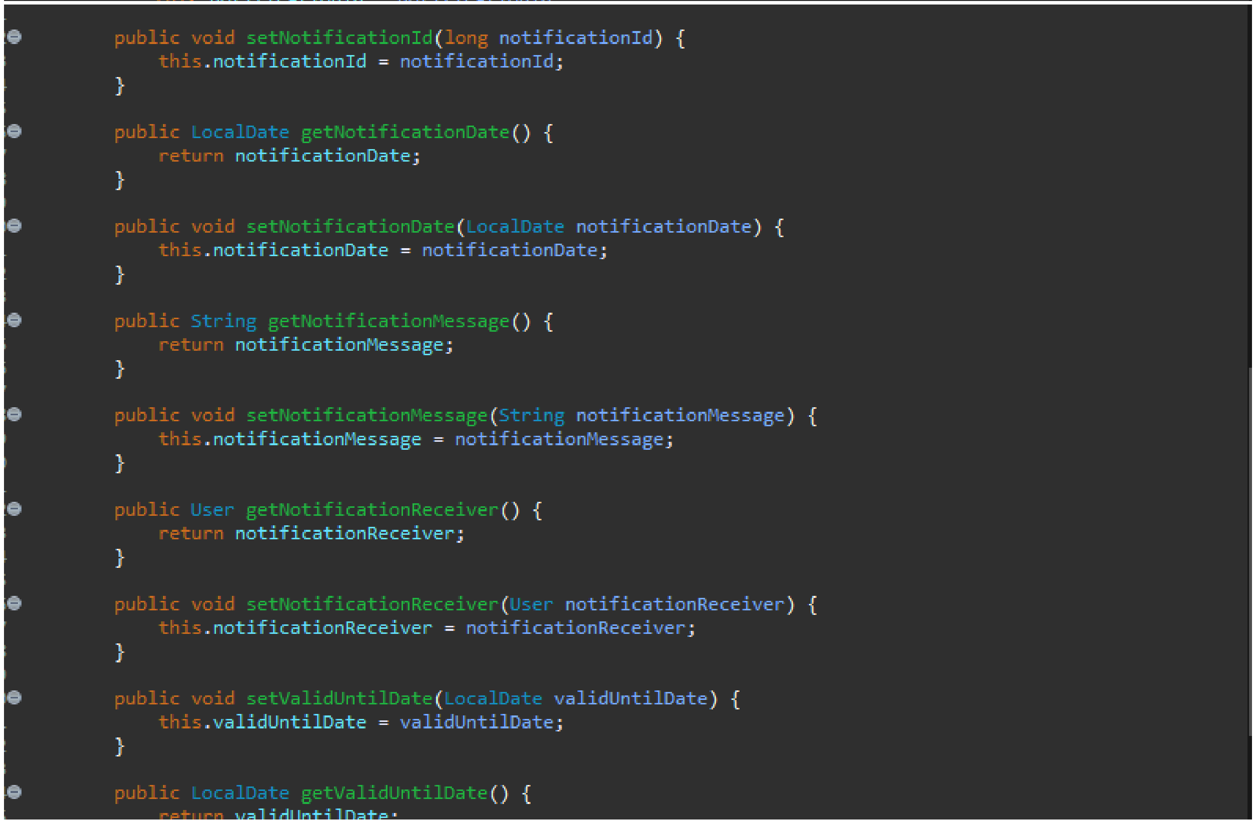


```

    public void setReservedByUser(User reservedByUser) {
        this.reservedByUser = reservedByUser;
    }

    public void setReadyForPickup(boolean readyForPickUp) {
        this.readyForPickUp = readyForPickUp;
    }

    public boolean getReadyForPickUp() {
        return readyForPickUp;
    }

}
```

The following screenshots show the class for Notification Entity:

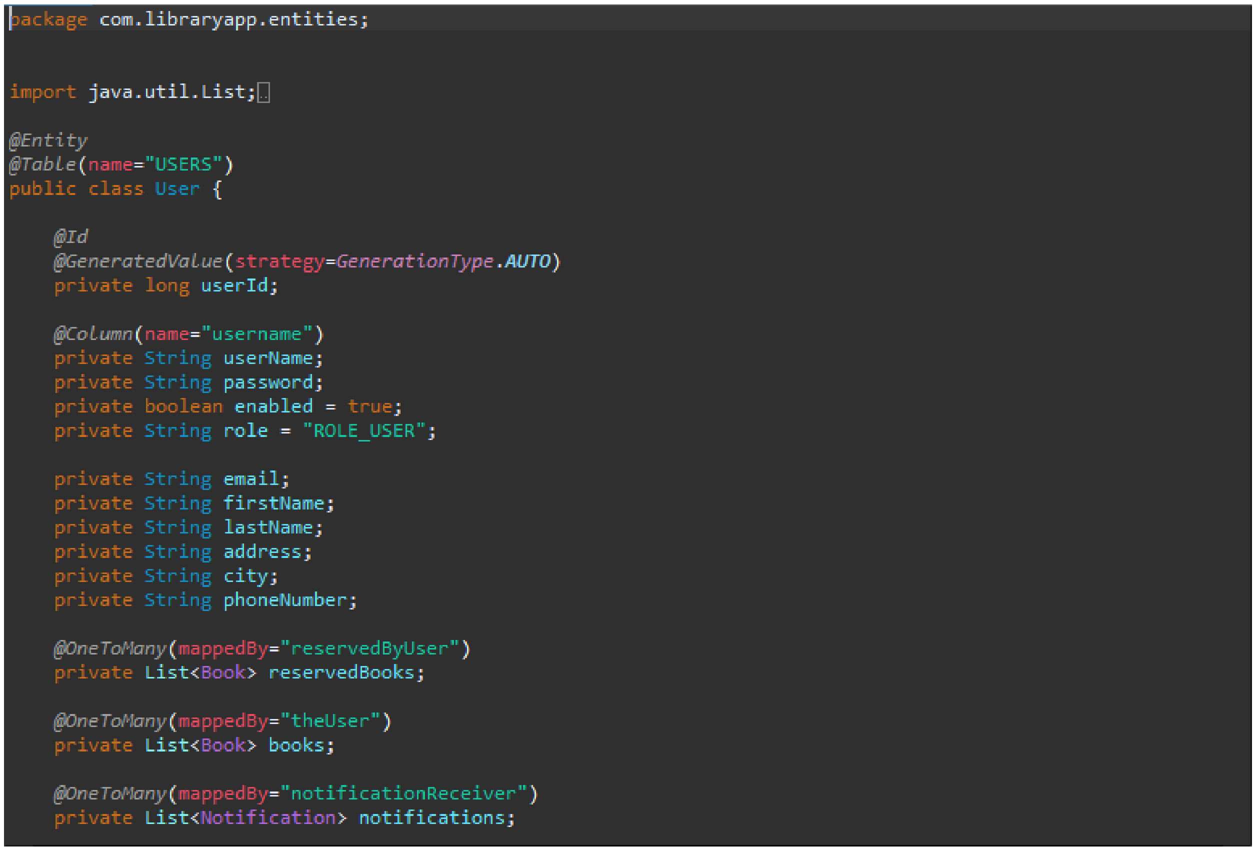


```java
package com.libraryapp.entities;

import java.time.LocalDate;

@Entity
public class Notification {

        @Id
        @GeneratedValue(strategy= GenerationType.AUTO)
        private long notificationId;
        private LocalDate notificationDate;
        private LocalDate validUntilDate;
        private String notificationMessage;

        @ManyToOne(cascade = {CascadeType.DETACH, CascadeType.MERGE, CascadeType.PERSIST, CascadeType.REFRESH},
                fetch = FetchType.LAZY)
        private User notificationReceiver;


        public Notification() {

        }

        public Notification (LocalDate notificationDate, LocalDate validUntilDate, String notificationMessage)
            this.notificationDate = notificationDate;
            this.validUntilDate = validUntilDate;
            this.notificationMessage = notificationMessage;
        }

        public long getNotificationId() {
            return notificationId;
        }

        public void setNotificationId(long notificationId) {
```

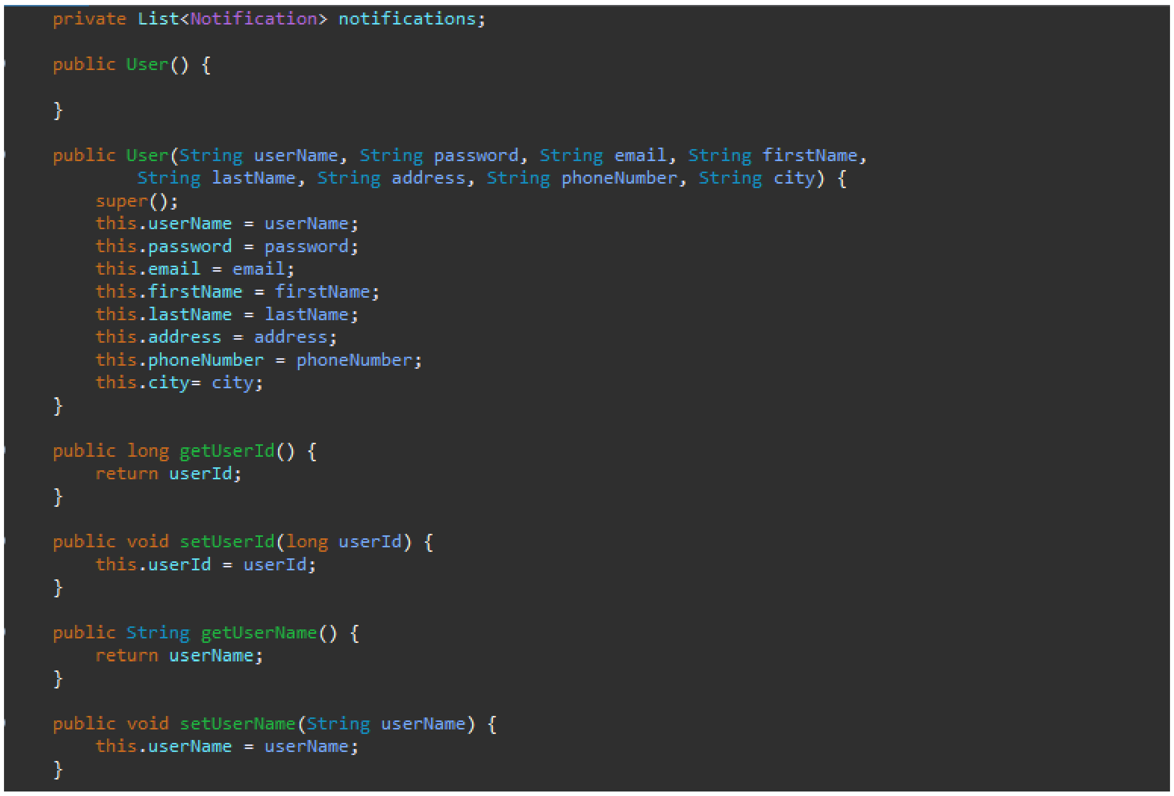


```java
        public void setNotificationId(long notificationId) {
            this.notificationId = notificationId;
        }

        public LocalDate getNotificationDate() {
            return notificationDate;
        }

        public void setNotificationDate(LocalDate notificationDate) {
            this.notificationDate = notificationDate;
        }

        public String getNotificationMessage() {
            return notificationMessage;
        }

        public void setNotificationMessage(String notificationMessage) {
            this.notificationMessage = notificationMessage;
        }

        public User getNotificationReceiver() {
            return notificationReceiver;
        }

        public void setNotificationReceiver(User notificationReceiver) {
            this.notificationReceiver = notificationReceiver;
        }

        public void setValidUntilDate(LocalDate validUntilDate) {
            this.validUntilDate = validUntilDate;
        }

        public LocalDate getValidUntilDate() {
```

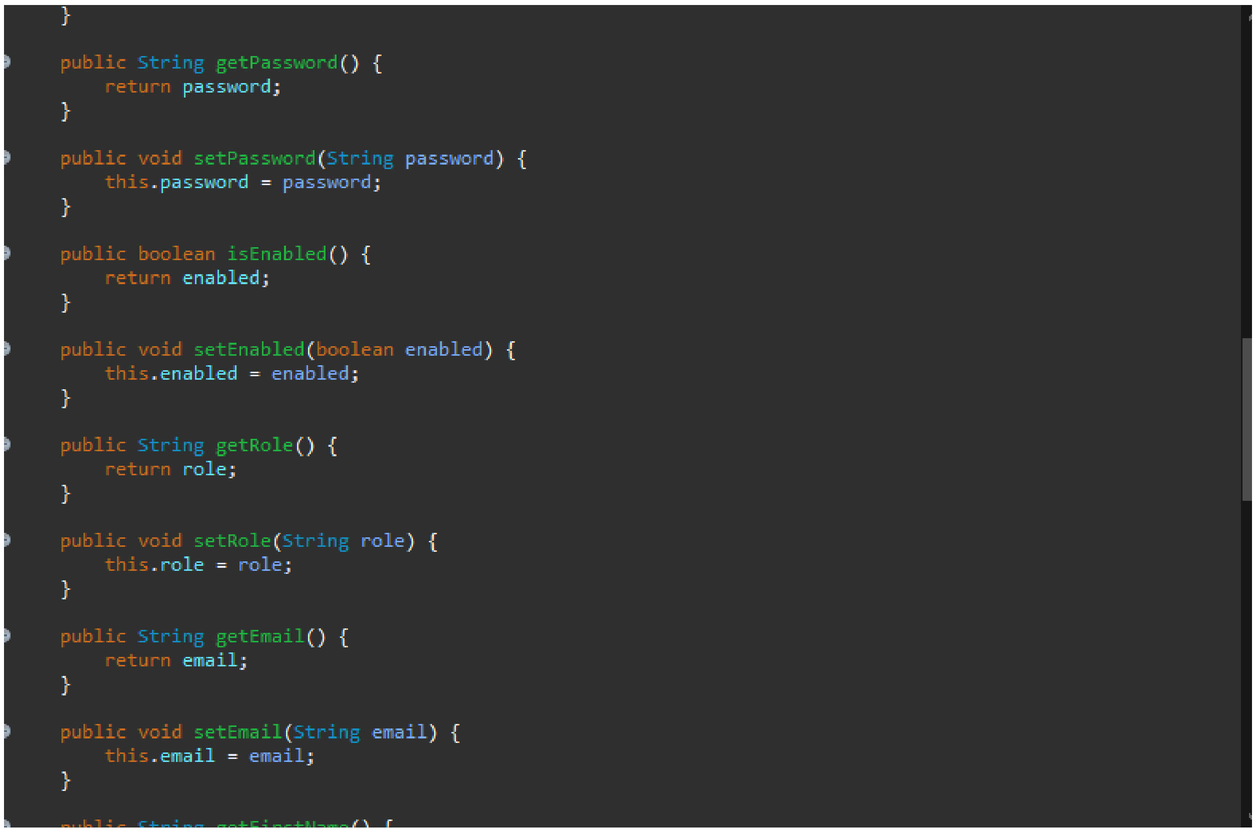
And finally, the following screenshots illustrate the User Entity class:



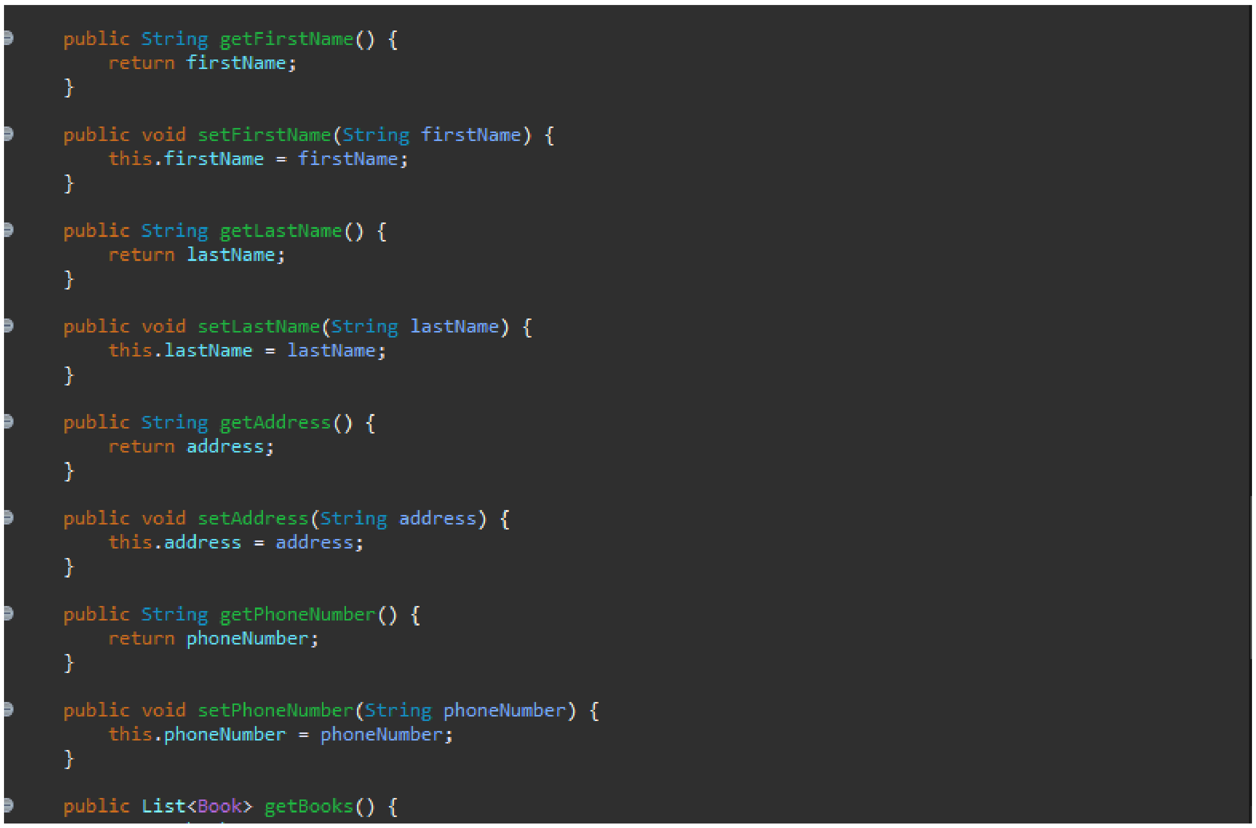
```
package com.libraryapp.entities;


import java.util.List;

@Entity
@Table(name="USERS")
public class User {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private long userId;

    @Column(name="username")
    private String userName;
    private String password;
    private boolean enabled = true;
    private String role = "ROLE_USER";

    private String email;
    private String firstName;
    private String lastName;
    private String address;
    private String city;
    private String phoneNumber;

    @OneToMany(mappedBy="reservedByUser")
    private List<Book> reservedBooks;

    @OneToMany(mappedBy="theUser")
    private List<Book> books;

    @OneToMany(mappedBy="notificationReceiver")
    private List<Notification> notifications;
```



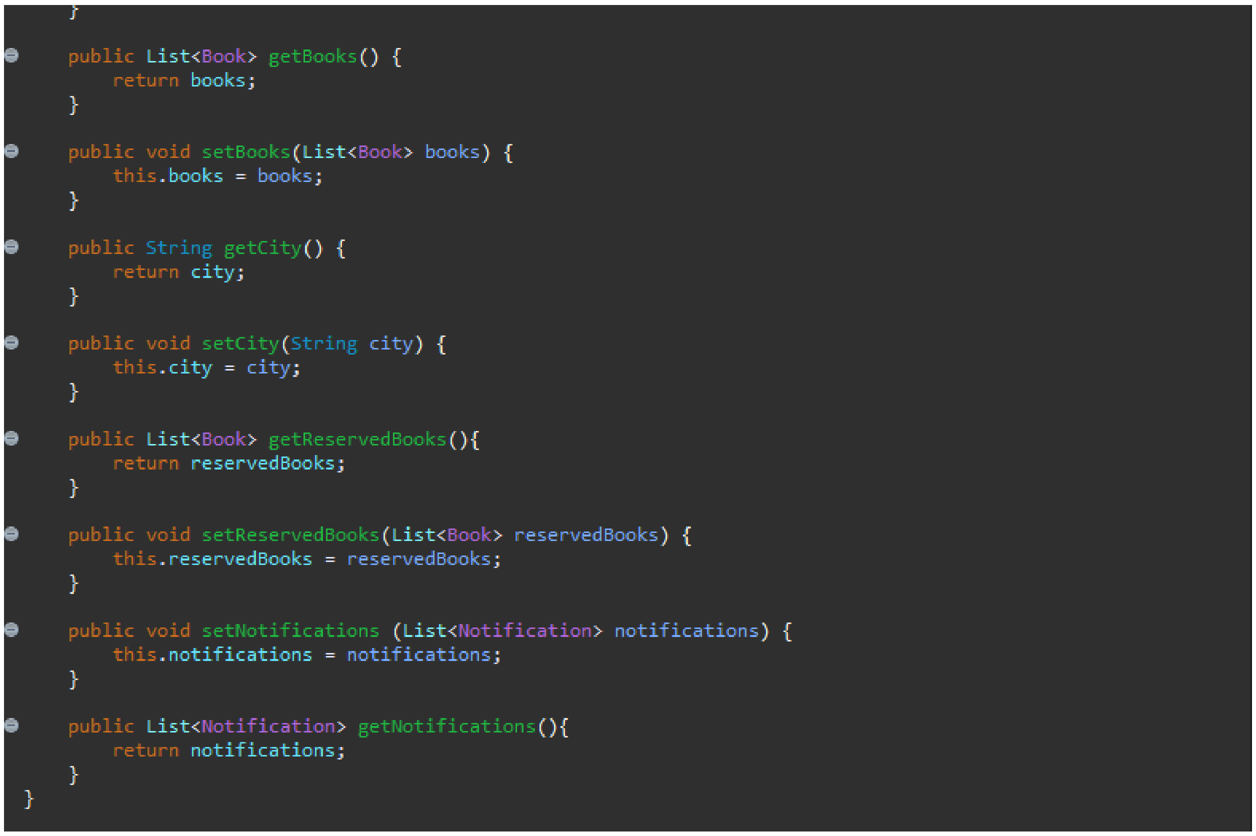
```
    private List<Notification> notifications;

    public User() {

    }

    public User(String userName, String password, String email, String firstName,
            String lastName, String address, String phoneNumber, String city) {
        super();
        this.userName = userName;
        this.password = password;
        this.email = email;
        this.firstName = firstName;
        this.lastName = lastName;
        this.address = address;
        this.phoneNumber = phoneNumber;
        this.city= city;
    }

    public long getUserId() {
        return userId;
    }

    public void setUserId(long userId) {
        this.userId = userId;
    }

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }
```



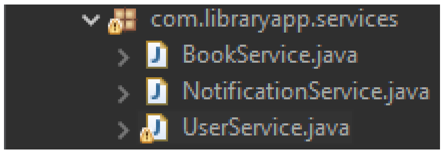
```
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public boolean isEnabled() {
        return enabled;
    }

    public void setEnabled(boolean enabled) {
        this.enabled = enabled;
    }

    public String getRole() {
        return role;
    }

    public void setRole(String role) {
        this.role = role;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
```



```
    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public String getPhoneNumber() {
        return phoneNumber;
    }

    public void setPhoneNumber(String phoneNumber) {
        this.phoneNumber = phoneNumber;
    }

    public List<Book> getBooks() {
```



```java
    }

    public List<Book> getBooks() {
        return books;
    }

    public void setBooks(List<Book> books) {
        this.books = books;
    }

    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }

    public List<Book> getReservedBooks(){
        return reservedBooks;
    }

    public void setReservedBooks(List<Book> reservedBooks) {
        this.reservedBooks = reservedBooks;
    }

    public void setNotifications (List<Notification> notifications) {
        this.notifications = notifications;
    }

    public List<Notification> getNotifications(){
        return notifications;
    }
}
```

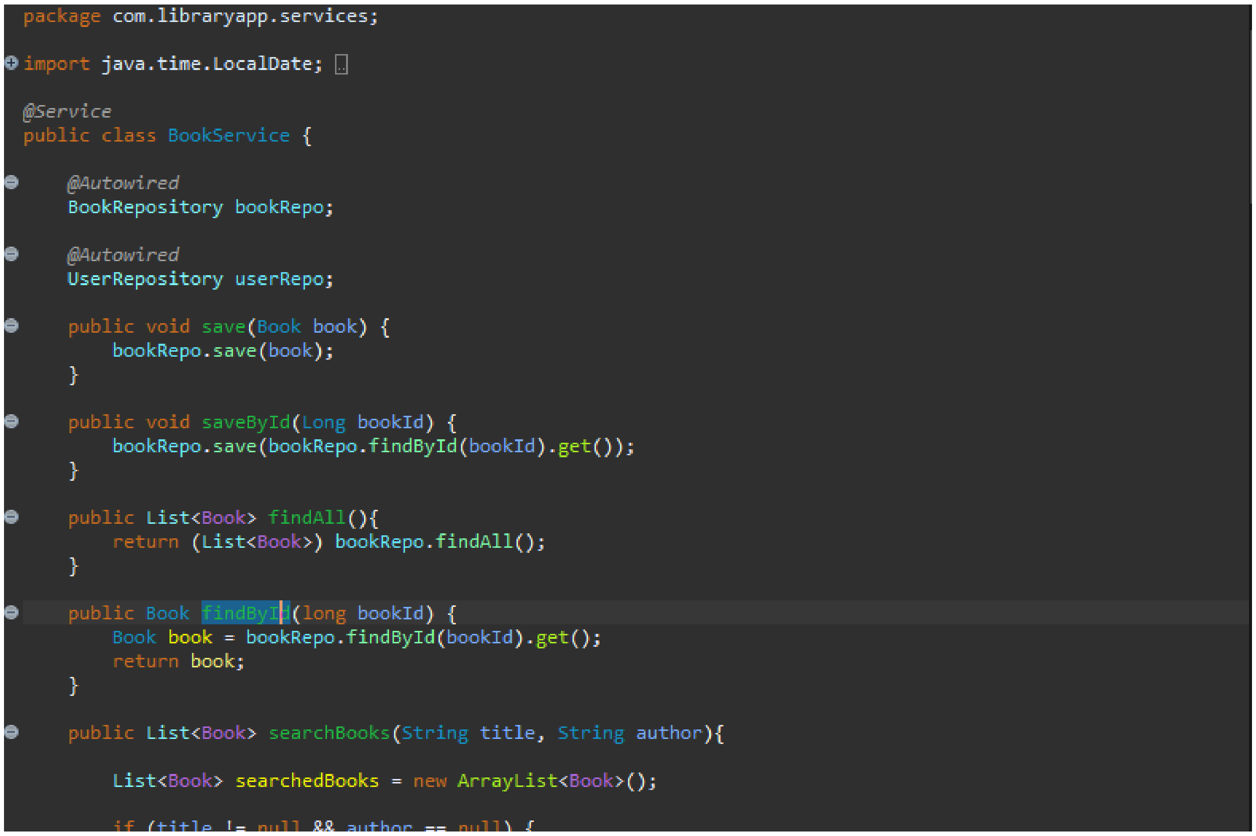


```
com.libraryapp.services
    BookService.java
    NotificationService.java
    UserService.java
```

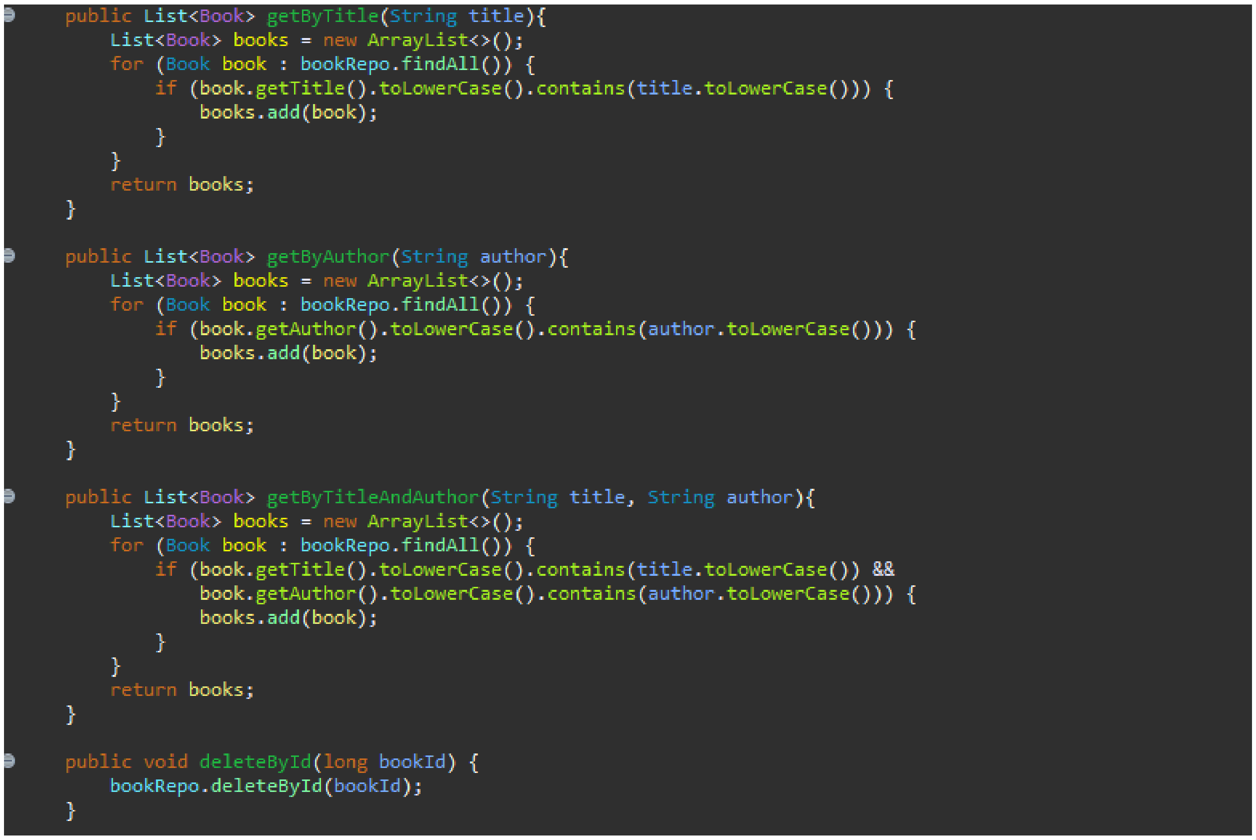
In this particular screenshot, it is revealed the package that contains the service layer of each entity the DAO package. This additional layer acts a communicator between controller and repository layer and contains business logic.

The following screenshots will reveal details about how each service class of each entity was developed thus far. Keep in mind that even though it may seem complete, I believe that some methods might need some improvements, others need to be removed and maybe replaced with completely methods, some need some code refactoring for better code reading, etc.

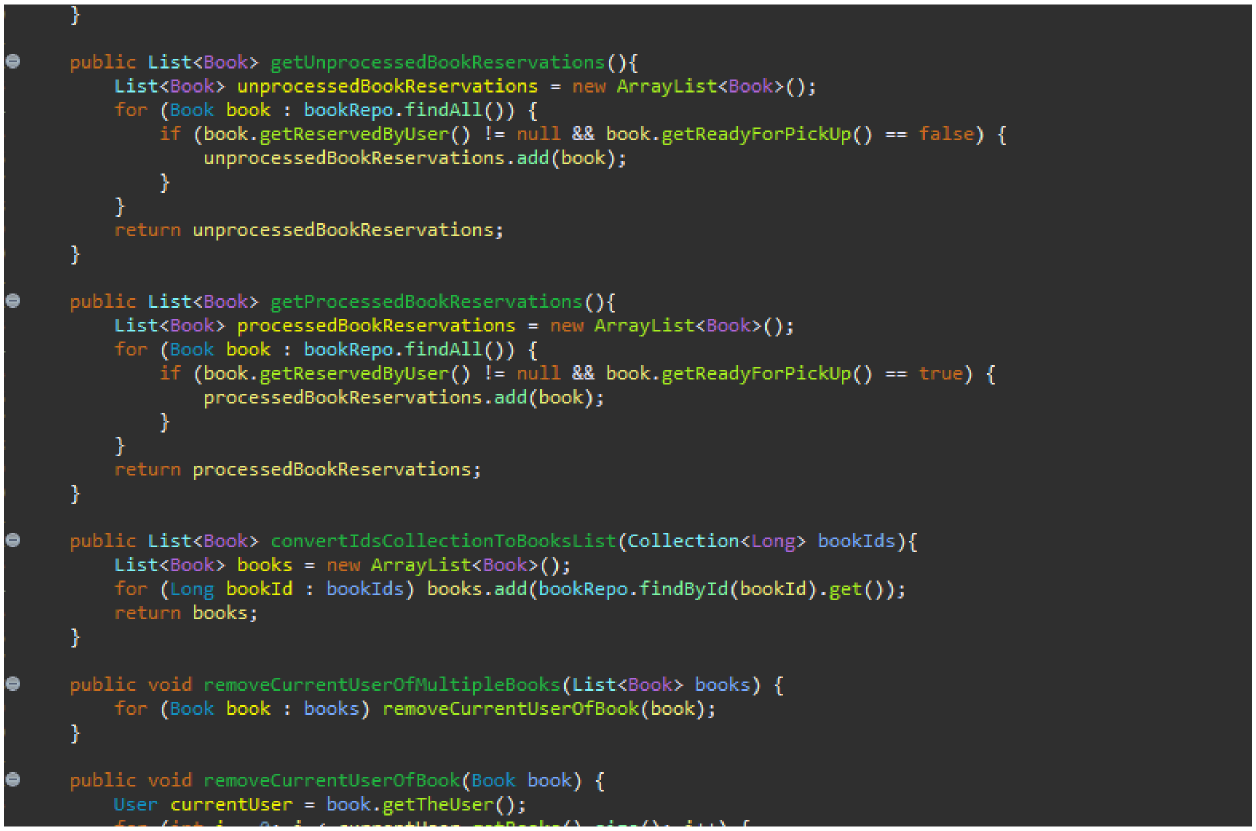
The following screenshots illustrate the services which handle books.



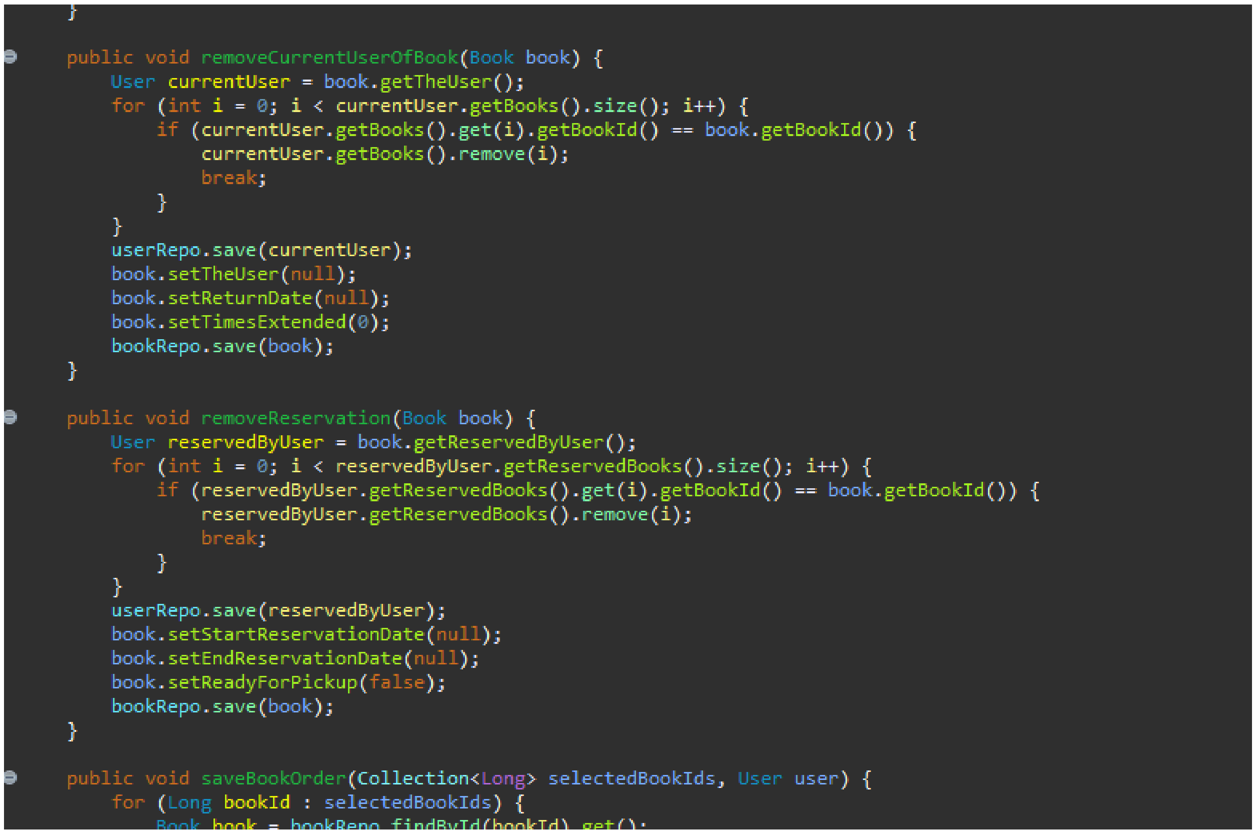
```java
package com.libraryapp.services;

import java.time.LocalDate;

@Service
public class BookService {

    @Autowired
    BookRepository bookRepo;

    @Autowired
    UserRepository userRepo;

    public void save(Book book) {
        bookRepo.save(book);
    }

    public void saveById(Long bookId) {
        bookRepo.save(bookRepo.findById(bookId).get());
    }

    public List<Book> findAll(){
        return (List<Book>) bookRepo.findAll();
    }

    public Book findById(long bookId) {
        Book book = bookRepo.findById(bookId).get();
        return book;
    }

    public List<Book> searchBooks(String title, String author){

        List<Book> searchedBooks = new ArrayList<Book>();

        if (title != null && author == null) {
```



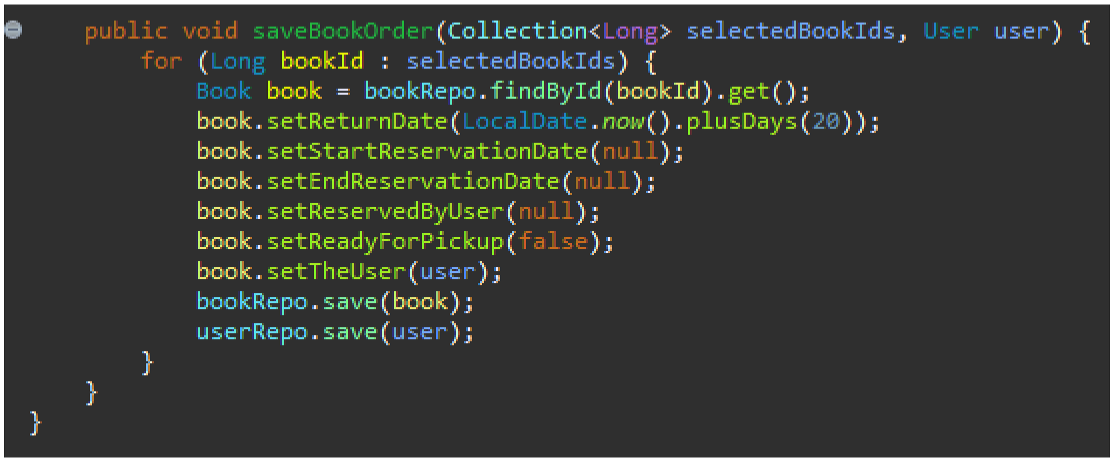
```java
    public List<Book> getByTitle(String title){
        List<Book> books = new ArrayList<>();
        for (Book book : bookRepo.findAll()) {
            if (book.getTitle().toLowerCase().contains(title.toLowerCase())) {
                books.add(book);
            }
        }
        return books;
    }

    public List<Book> getByAuthor(String author){
        List<Book> books = new ArrayList<>();
        for (Book book : bookRepo.findAll()) {
            if (book.getAuthor().toLowerCase().contains(author.toLowerCase())) {
                books.add(book);
            }
        }
        return books;
    }

    public List<Book> getByTitleAndAuthor(String title, String author){
        List<Book> books = new ArrayList<>();
        for (Book book : bookRepo.findAll()) {
            if (book.getTitle().toLowerCase().contains(title.toLowerCase()) &&
                book.getAuthor().toLowerCase().contains(author.toLowerCase())) {
                books.add(book);
            }
        }
        return books;
    }

    public void deleteById(long bookId) {
        bookRepo.deleteById(bookId);
    }
```



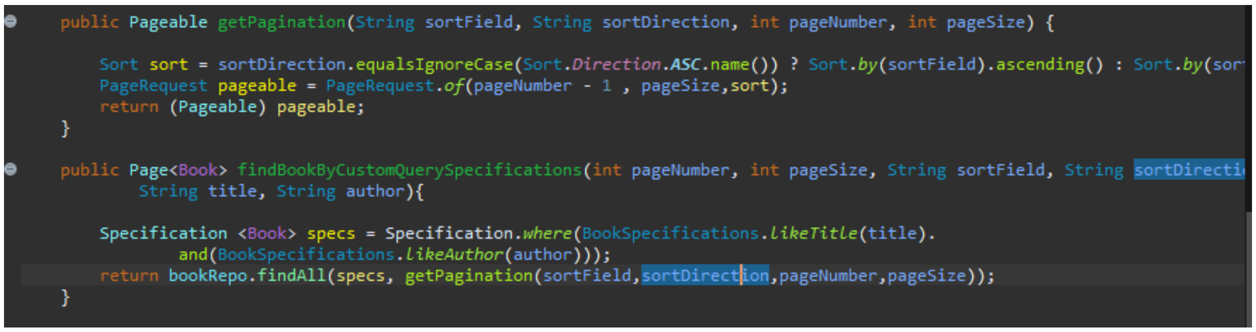
```java
    }

    public List<Book> getUnprocessedBookReservations(){
        List<Book> unprocessedBookReservations = new ArrayList<Book>();
        for (Book book : bookRepo.findAll()) {
            if (book.getReservedByUser() != null && book.getReadyForPickUp() == false) {
                unprocessedBookReservations.add(book);
            }
        }
        return unprocessedBookReservations;
    }

    public List<Book> getProcessedBookReservations(){
        List<Book> processedBookReservations = new ArrayList<Book>();
        for (Book book : bookRepo.findAll()) {
            if (book.getReservedByUser() != null && book.getReadyForPickUp() == true) {
                processedBookReservations.add(book);
            }
        }
        return processedBookReservations;
    }

    public List<Book> convertIdsCollectionToBooksList(Collection<Long> bookIds){
        List<Book> books = new ArrayList<Book>();
        for (Long bookId : bookIds) books.add(bookRepo.findById(bookId).get());
        return books;
    }

    public void removeCurrentUserOfMultipleBooks(List<Book> books) {
        for (Book book : books) removeCurrentUserOfBook(book);
    }

    public void removeCurrentUserOfBook(Book book) {
        User currentUser = book.getTheUser();
        for (int i = 0; i < currentUser.getBooks().size(); i++) {
```



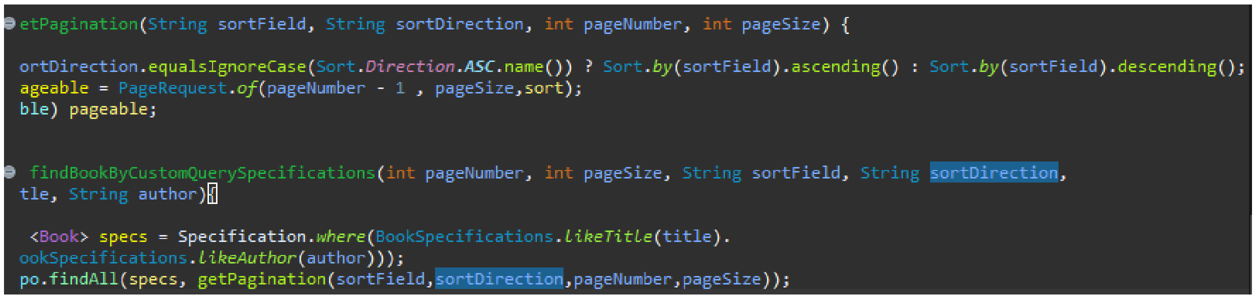
```java
    }

    public void removeCurrentUserOfBook(Book book) {
        User currentUser = book.getTheUser();
        for (int i = 0; i < currentUser.getBooks().size(); i++) {
            if (currentUser.getBooks().get(i).getBookId() == book.getBookId()) {
                currentUser.getBooks().remove(i);
                break;
            }
        }
        userRepo.save(currentUser);
        book.setTheUser(null);
        book.setReturnDate(null);
        book.setTimesExtended(0);
        bookRepo.save(book);
    }

    public void removeReservation(Book book) {
        User reservedByUser = book.getReservedByUser();
        for (int i = 0; i < reservedByUser.getReservedBooks().size(); i++) {
            if (reservedByUser.getReservedBooks().get(i).getBookId() == book.getBookId()) {
                reservedByUser.getReservedBooks().remove(i);
                break;
            }
        }
        userRepo.save(reservedByUser);
        book.setStartReservationDate(null);
        book.setEndReservationDate(null);
        book.setReadyForPickup(false);
        bookRepo.save(book);
    }

    public void saveBookOrder(Collection<Long> selectedBookIds, User user) {
        for (Long bookId : selectedBookIds) {
            Book book = bookRepo.findById(bookId).get();
```



```java
    public void saveBookOrder(Collection<Long> selectedBookIds, User user) {
        for (Long bookId : selectedBookIds) {
            Book book = bookRepo.findById(bookId).get();
            book.setReturnDate(LocalDate.now().plusDays(20));
            book.setStartReservationDate(null);
            book.setEndReservationDate(null);
            book.setReservedByUser(null);
            book.setReadyForPickup(false);
            book.setTheUser(user);
            bookRepo.save(book);
            userRepo.save(user);
        }
    }
}
```



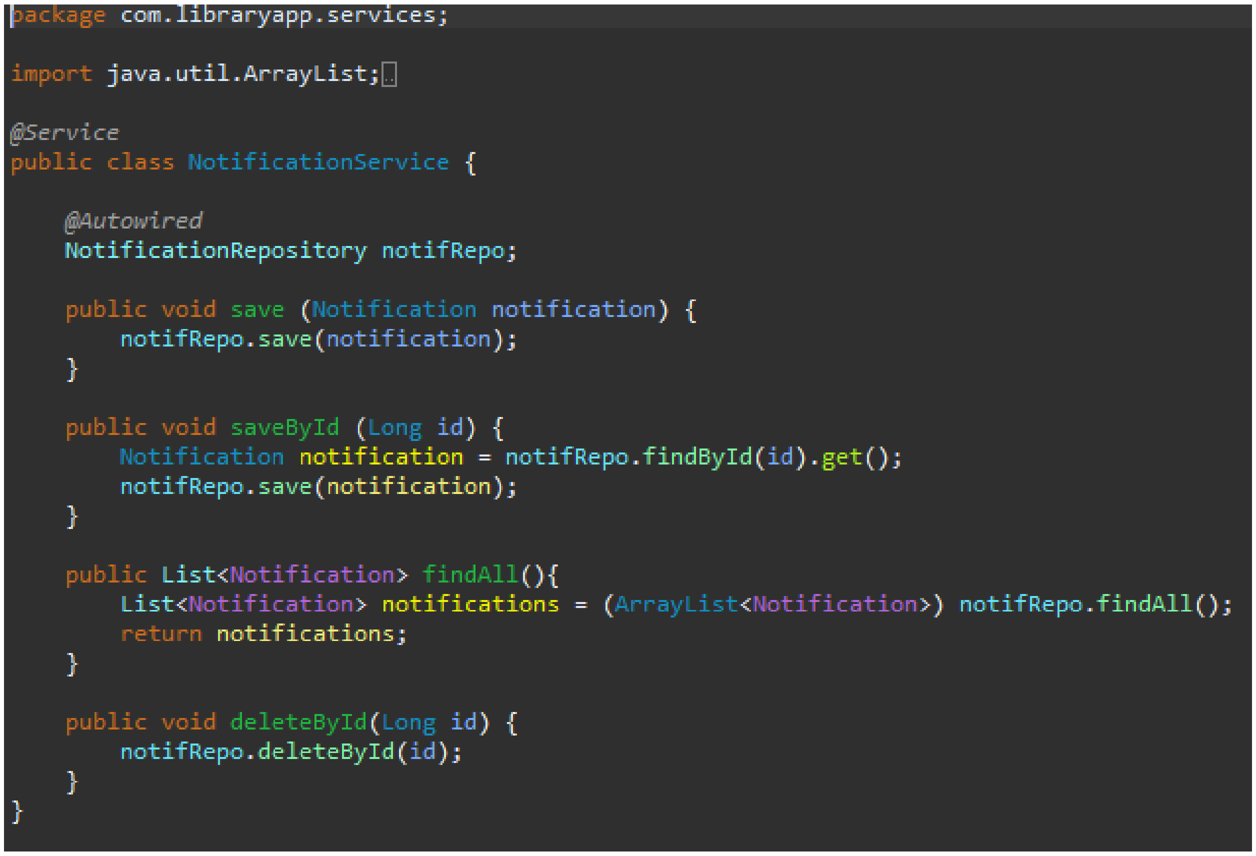
```java
    public Pageable getPagination(String sortField, String sortDirection, int pageNumber, int pageSize) {

        Sort sort = sortDirection.equalsIgnoreCase(Sort.Direction.ASC.name()) ? Sort.by(sortField).ascending() : Sort.by(sor
        PageRequest pageable = PageRequest.of(pageNumber - 1 , pageSize,sort);
        return (Pageable) pageable;
    }

    public Page<Book> findBookByCustomQuerySpecifications(int pageNumber, int pageSize, String sortField, String sortDirecti
            String title, String author){

        Specification <Book> specs = Specification.where(BookSpecifications.likeTitle(title).
                and(BookSpecifications.likeAuthor(author)));
        return bookRepo.findAll(specs, getPagination(sortField,sortDirection,pageNumber,pageSize));
    }
```



```java
etPagination(String sortField, String sortDirection, int pageNumber, int pageSize) {

ortDirection.equalsIgnoreCase(Sort.Direction.ASC.name()) ? Sort.by(sortField).ascending() : Sort.by(sortField).descending();
ageable = PageRequest.of(pageNumber - 1 , pageSize,sort);
ble) pageable;


 findBookByCustomQuerySpecifications(int pageNumber, int pageSize, String sortField, String sortDirection,
tle, String author){

 <Book> specs = Specification.where(BookSpecifications.likeTitle(title).
ookSpecifications.likeAuthor(author)));
po.findAll(specs, getPagination(sortField,sortDirection,pageNumber,pageSize));
```

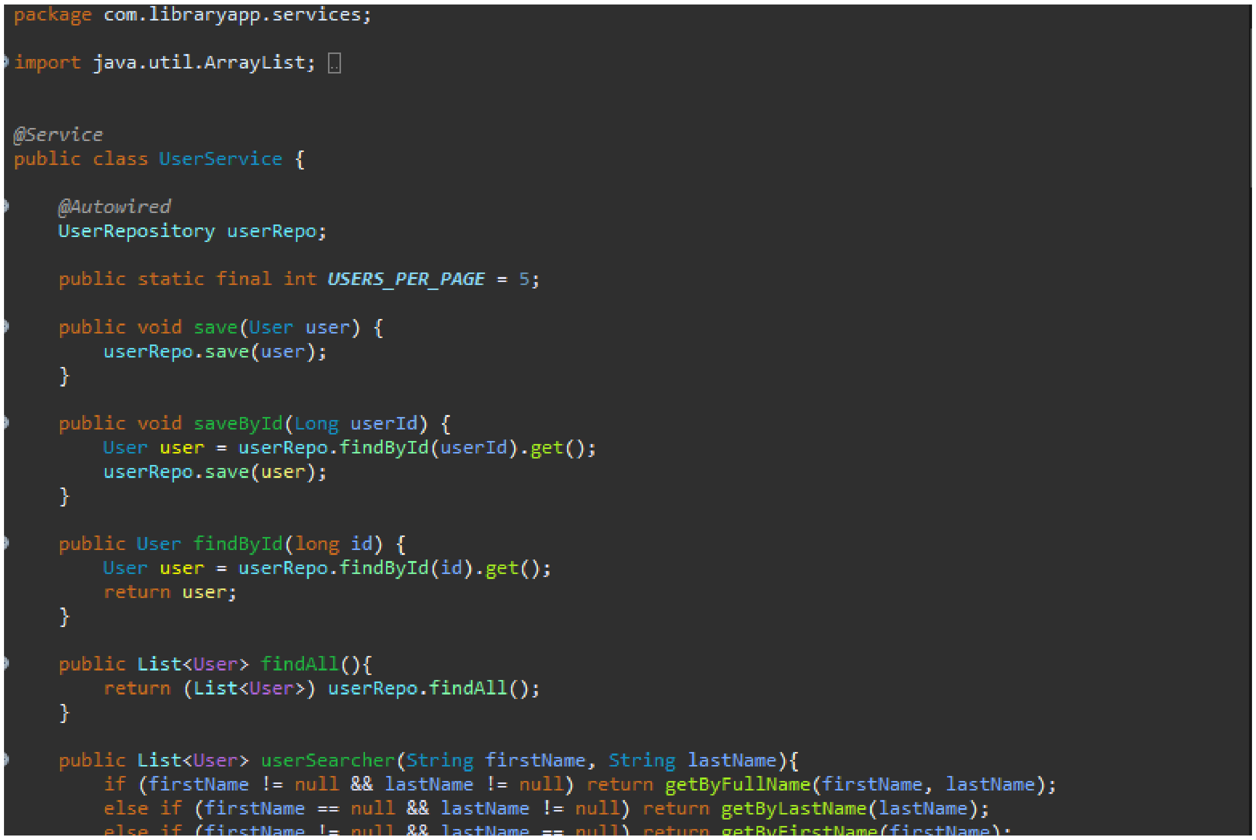
The last two methods illustrated in the last two screenshots handle the pagination after performing a search on books with different specifications such as all books starting with letters ‘c’, ’o’, ’d’ and ‘e’, the full name of an author or a part of the name of the author etc.

The following screenshots show the services of Notification entity.

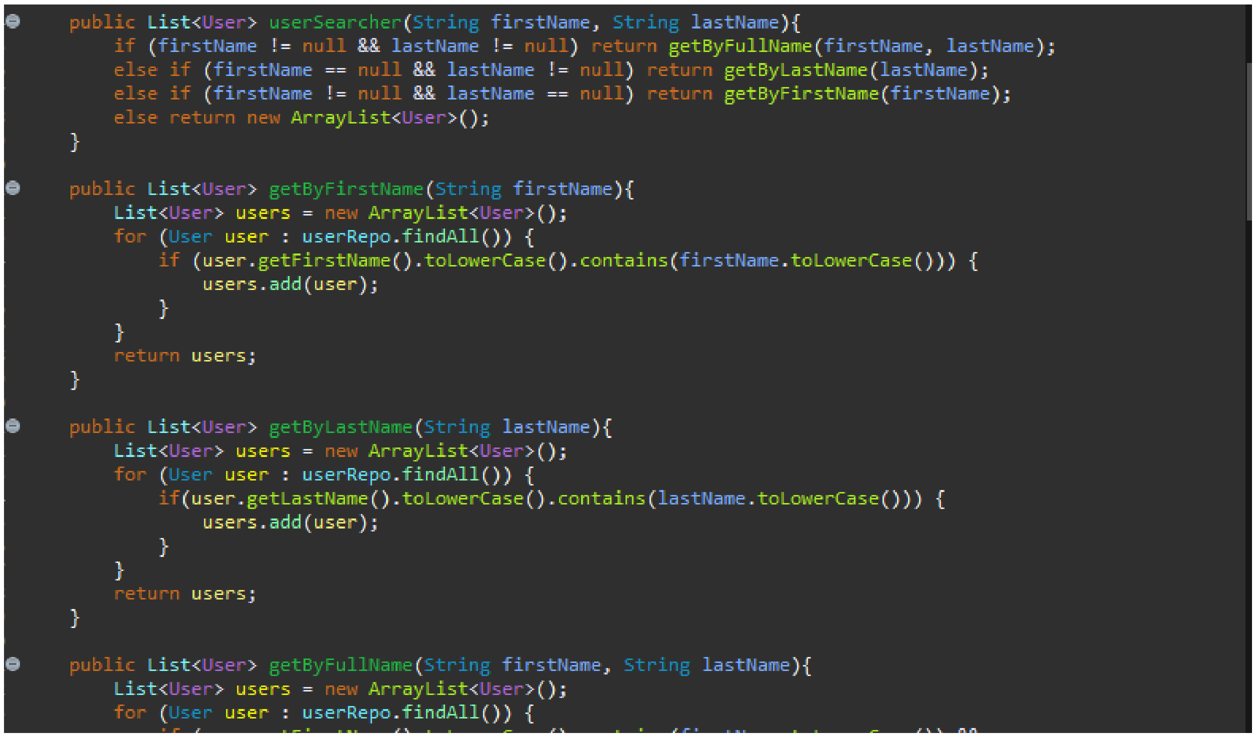


```java
package com.libraryapp.services;

import java.util.ArrayList;

@Service
public class NotificationService {

    @Autowired
    NotificationRepository notifRepo;

    public void save (Notification notification) {
        notifRepo.save(notification);
    }

    public void saveById (Long id) {
        Notification notification = notifRepo.findById(id).get();
        notifRepo.save(notification);
    }

    public List<Notification> findAll(){
        List<Notification> notifications = (ArrayList<Notification>) notifRepo.findAll();
        return notifications;
    }

    public void deleteById(Long id) {
        notifRepo.deleteById(id);
    }
}
```

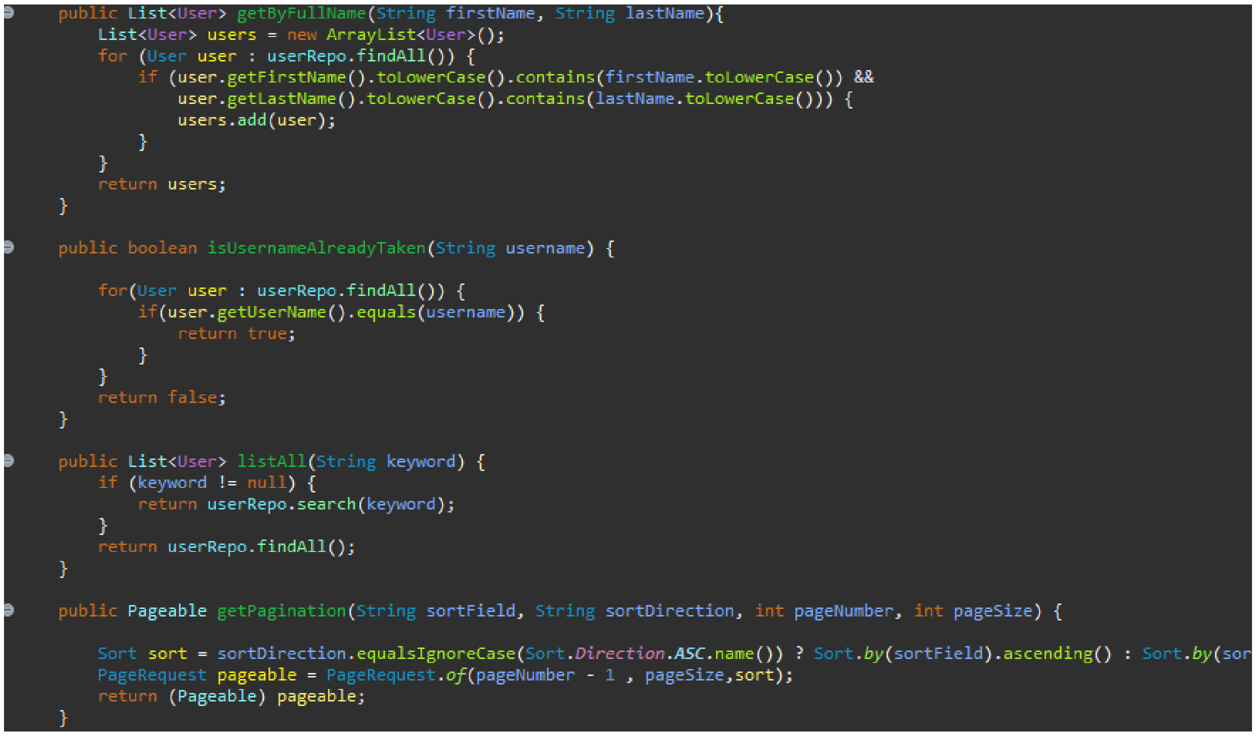
The following screenshots highlight the services of the User Entity:



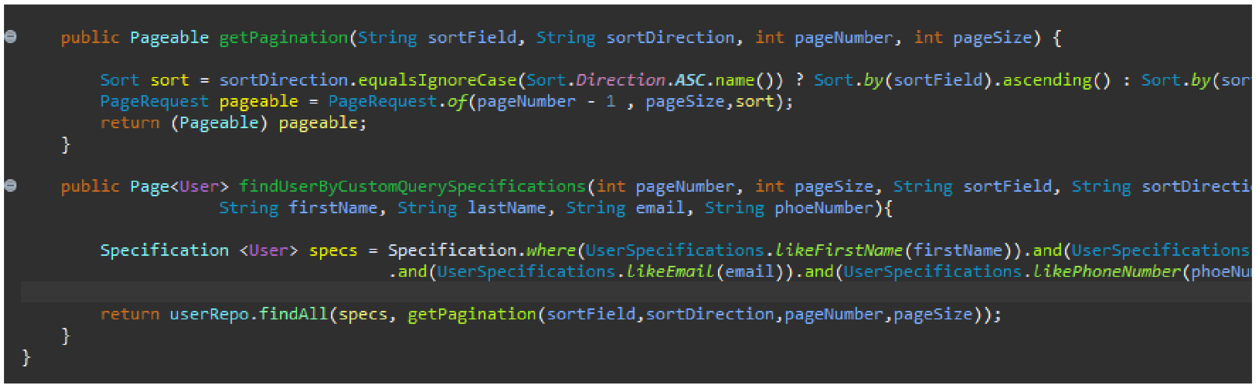
```java
package com.libraryapp.services;

import java.util.ArrayList;


@Service
public class UserService {

    @Autowired
    UserRepository userRepo;

    public static final int USERS_PER_PAGE = 5;

    public void save(User user) {
        userRepo.save(user);
    }

    public void saveById(Long userId) {
        User user = userRepo.findById(userId).get();
        userRepo.save(user);
    }

    public User findById(long id) {
        User user = userRepo.findById(id).get();
        return user;
    }

    public List<User> findAll(){
        return (List<User>) userRepo.findAll();
    }

    public List<User> userSearcher(String firstName, String lastName){
        if (firstName != null && lastName != null) return getByFullName(firstName, lastName);
        else if (firstName == null && lastName != null) return getByLastName(lastName);
```



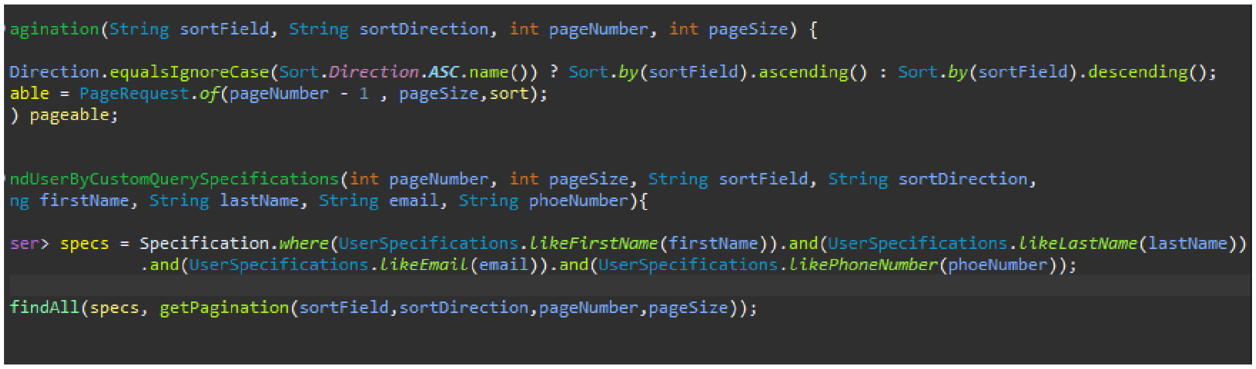
```java
    public List<User> userSearcher(String firstName, String lastName){
        if (firstName != null && lastName != null) return getByFullName(firstName, lastName);
        else if (firstName == null && lastName != null) return getByLastName(lastName);
        else if (firstName != null && lastName == null) return getByFirstName(firstName);
        else return new ArrayList<User>();
    }

    public List<User> getByFirstName(String firstName){
        List<User> users = new ArrayList<User>();
        for (User user : userRepo.findAll()) {
            if (user.getFirstName().toLowerCase().contains(firstName.toLowerCase())) {
                users.add(user);
            }
        }
        return users;
    }

    public List<User> getByLastName(String lastName){
        List<User> users = new ArrayList<User>();
        for (User user : userRepo.findAll()) {
            if(user.getLastName().toLowerCase().contains(lastName.toLowerCase())) {
                users.add(user);
            }
        }
        return users;
    }

    public List<User> getByFullName(String firstName, String lastName){
        List<User> users = new ArrayList<User>();
        for (User user : userRepo.findAll()) {
```



```
    public List<User> getByFullName(String firstName, String lastName){
        List<User> users = new ArrayList<User>();
        for (User user : userRepo.findAll()) {
            if (user.getFirstName().toLowerCase().contains(firstName.toLowerCase()) &&
                user.getLastName().toLowerCase().contains(lastName.toLowerCase())) {
                users.add(user);
            }
        }
        return users;
    }

    public boolean isUsernameAlreadyTaken(String username) {

        for(User user : userRepo.findAll()) {
            if(user.getUserName().equals(username)) {
                return true;
            }
        }
        return false;
    }

    public List<User> listAll(String keyword) {
        if (keyword != null) {
            return userRepo.search(keyword);
        }
        return userRepo.findAll();
    }

    public Pageable getPagination(String sortField, String sortDirection, int pageNumber, int pageSize) {

        Sort sort = sortDirection.equalsIgnoreCase(Sort.Direction.ASC.name()) ? Sort.by(sortField).ascending() : Sort.by(sor
        PageRequest pageable = PageRequest.of(pageNumber - 1 , pageSize,sort);
        return (Pageable) pageable;
    }
```



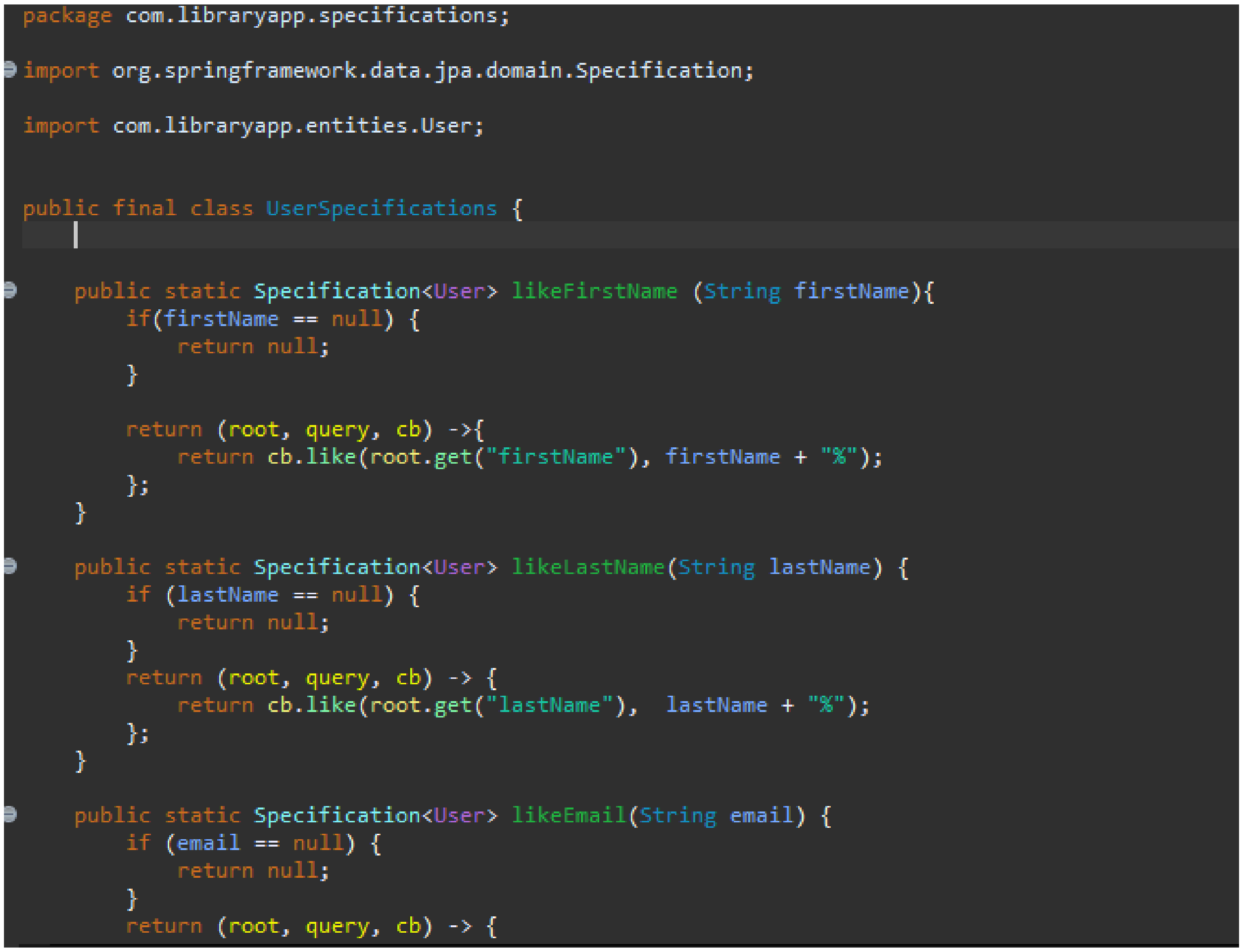
```
    public Pageable getPagination(String sortField, String sortDirection, int pageNumber, int pageSize) {

        Sort sort = sortDirection.equalsIgnoreCase(Sort.Direction.ASC.name()) ? Sort.by(sortField).ascending() : Sort.by(sor
        PageRequest pageable = PageRequest.of(pageNumber - 1 , pageSize,sort);
        return (Pageable) pageable;
    }

    public Page<User> findUserByCustomQuerySpecifications(int pageNumber, int pageSize, String sortField, String sortDirecti
                String firstName, String lastName, String email, String phoeNumber){

        Specification <User> specs = Specification.where(UserSpecifications.likeFirstName(firstName)).and(UserSpecifications
                                .and(UserSpecifications.likeEmail(email)).and(UserSpecifications.likePhoneNumber(phoeNu

        return userRepo.findAll(specs, getPagination(sortField,sortDirection,pageNumber,pageSize));
    }
}
```



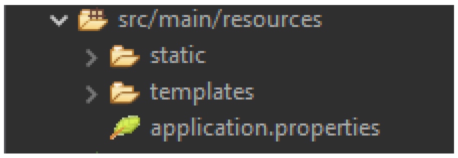
```
agination(String sortField, String sortDirection, int pageNumber, int pageSize) {

Direction.equalsIgnoreCase(Sort.Direction.ASC.name()) ? Sort.by(sortField).ascending() : Sort.by(sortField).descending();
able = PageRequest.of(pageNumber - 1 , pageSize,sort);
) pageable;


ndUserByCustomQuerySpecifications(int pageNumber, int pageSize, String sortField, String sortDirection,
ng firstName, String lastName, String email, String phoeNumber){

ser> specs = Specification.where(UserSpecifications.likeFirstName(firstName)).and(UserSpecifications.likeLastName(lastName))
              .and(UserSpecifications.likeEmail(email)).and(UserSpecifications.likePhoneNumber(phoeNumber));

findAll(specs, getPagination(sortField,sortDirection,pageNumber,pageSize));
```

The last two methods work the same way as mentioned in Book services.

In the Specifications package, there are Specifications classes in which the queries can be made in a custom way according to a regex rule (such as the string provided followed by anything that comes after that, which is also the rule by which the search is made by). The instances which play a role in making this possible are root, query and cb (criteria builder) which are used in a lambda expression, which is what is ultimately returned in a Specification method.

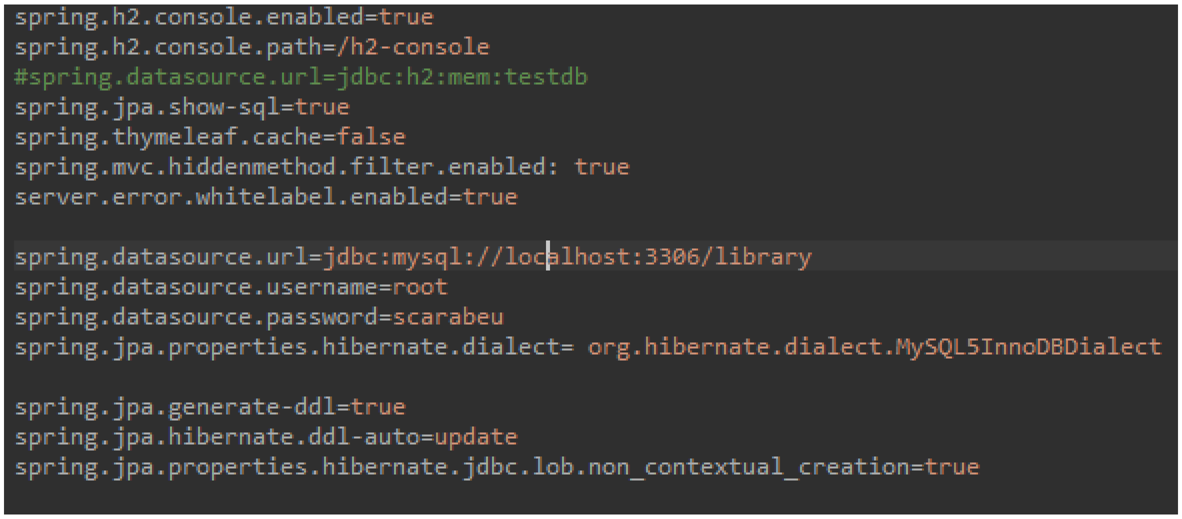


```java
package com.libraryapp.specifications;

import org.springframework.data.jpa.domain.Specification;

import com.libraryapp.entities.User;


public final class UserSpecifications {
    

    public static Specification<User> likeFirstName (String firstName){
        if(firstName == null) {
            return null;
        }

        return (root, query, cb) ->{
            return cb.like(root.get("firstName"), firstName + "%");
        };
    }

    public static Specification<User> likeLastName(String lastName) {
        if (lastName == null) {
            return null;
        }
        return (root, query, cb) -> {
            return cb.like(root.get("lastName"),  lastName + "%");
        };
    }

    public static Specification<User> likeEmail(String email) {
        if (email == null) {
            return null;
        }
        return (root, query, cb) -> {
```



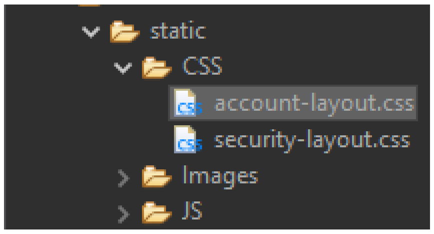
```
src/main/resources
    static
    templates
    application.properties
```

In this particular directory it is found the implementation of the views, where also models come in play. Apart from that we can also find an *application.properties* configuration file



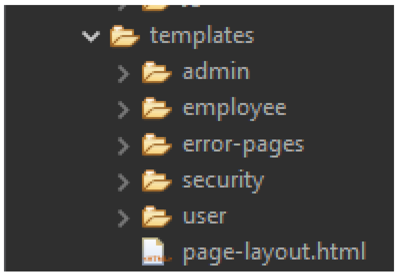
```
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
#spring.datasource.url=jdbc:h2:mem:testdb
spring.jpa.show-sql=true
spring.thymeleaf.cache=false
spring.mvc.hiddenmethod.filter.enabled: true
server.error.whitelabel.enabled=true

spring.datasource.url=jdbc:mysql://localhost:3306/library
spring.datasource.username=root
spring.datasource.password=scarabeu
spring.jpa.properties.hibernate.dialect= org.hibernate.dialect.MySQL5InnoDBDialect

spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
```

The screenshot above illustrates the way the database is configured such that the application can access its data.

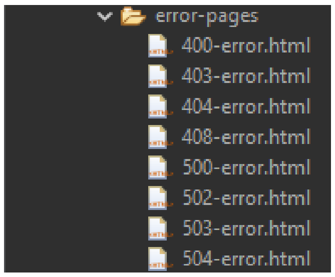


```
static
  CSS
    account-layout.css
    security-layout.css
  Images
  JS
```

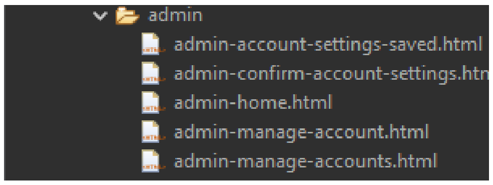
This particular screenshot shows the location of CSS files and Images used for the web design.



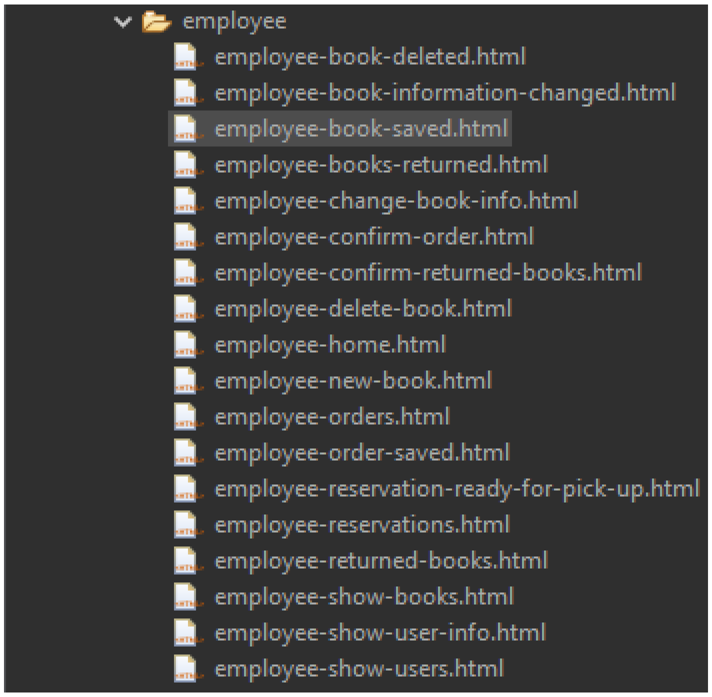
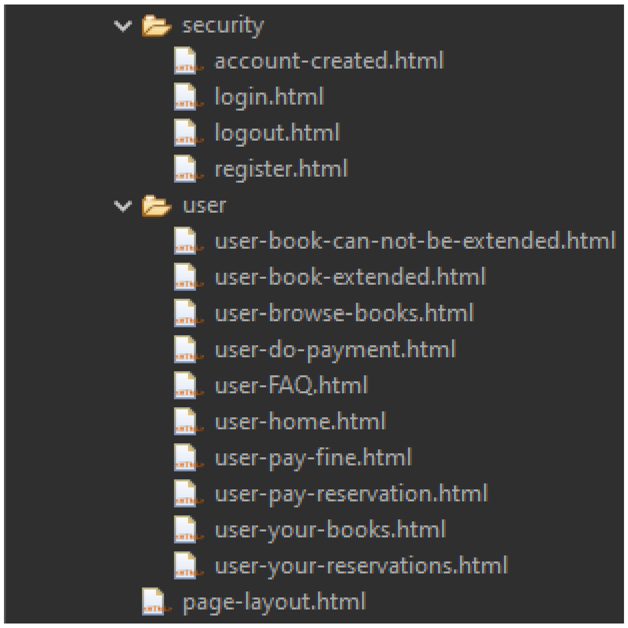
```
templates
  admin
  employee
  error-pages
  security
  user
  page-layout.html
```



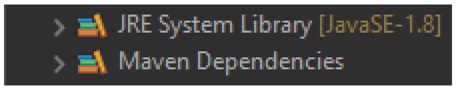
```
error-pages
  400-error.html
  403-error.html
  404-error.html
  408-error.html
  500-error.html
  502-error.html
  503-error.html
  504-error.html
```



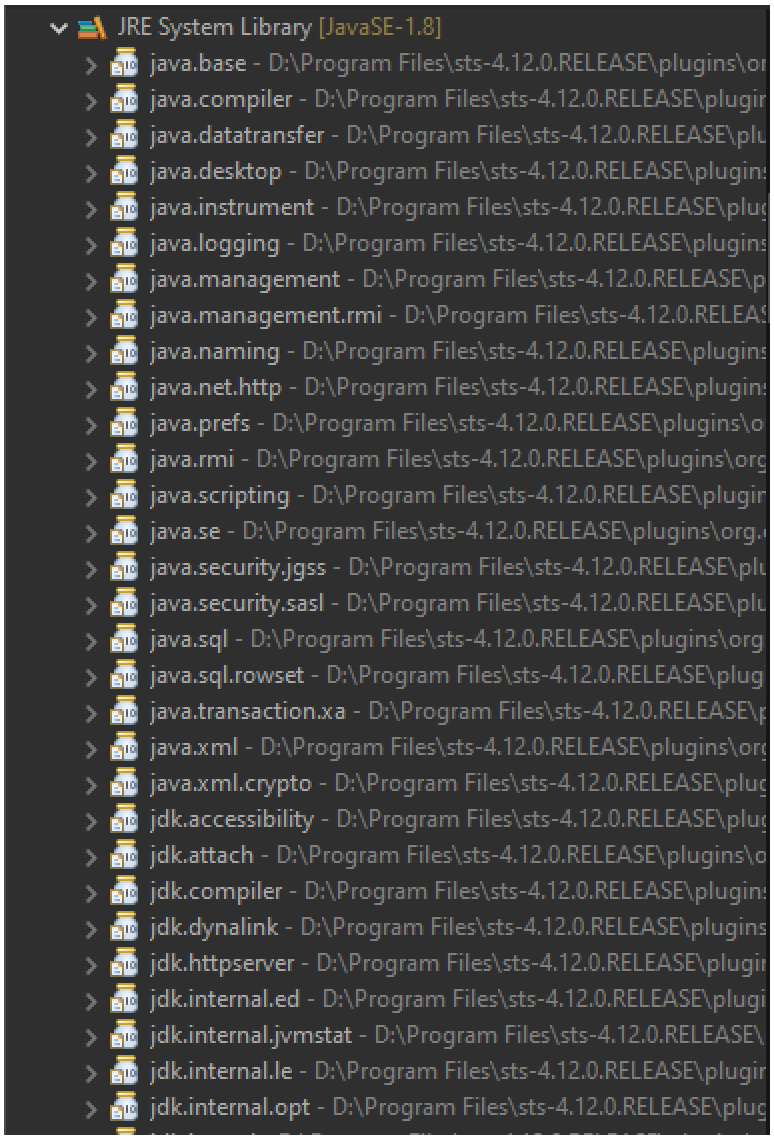
```
admin
  admin-account-settings-saved.html
  admin-confirm-account-settings.htn
  admin-home.html
  admin-manage-account.html
  admin-manage-accounts.html
```


```
employee
    employee-book-deleted.html
    employee-book-information-changed.html
    employee-book-saved.html
    employee-books-returned.html
    employee-change-book-info.html
    employee-confirm-order.html
    employee-confirm-returned-books.html
    employee-delete-book.html
    employee-home.html
    employee-new-book.html
    employee-orders.html
    employee-order-saved.html
    employee-reservation-ready-for-pick-up.html
    employee-reservations.html
    employee-returned-books.html
    employee-show-books.html
    employee-show-user-info.html
    employee-show-users.html
```

```
security
    account-created.html
    login.html
    logout.html
    register.html
user
    user-book-can-not-be-extended.html
    user-book-extended.html
    user-browse-books.html
    user-do-payment.html
    user-FAQ.html
    user-home.html
    user-pay-fine.html
    user-pay-reservation.html
    user-your-books.html
    user-your-reservations.html
page-layout.html
```

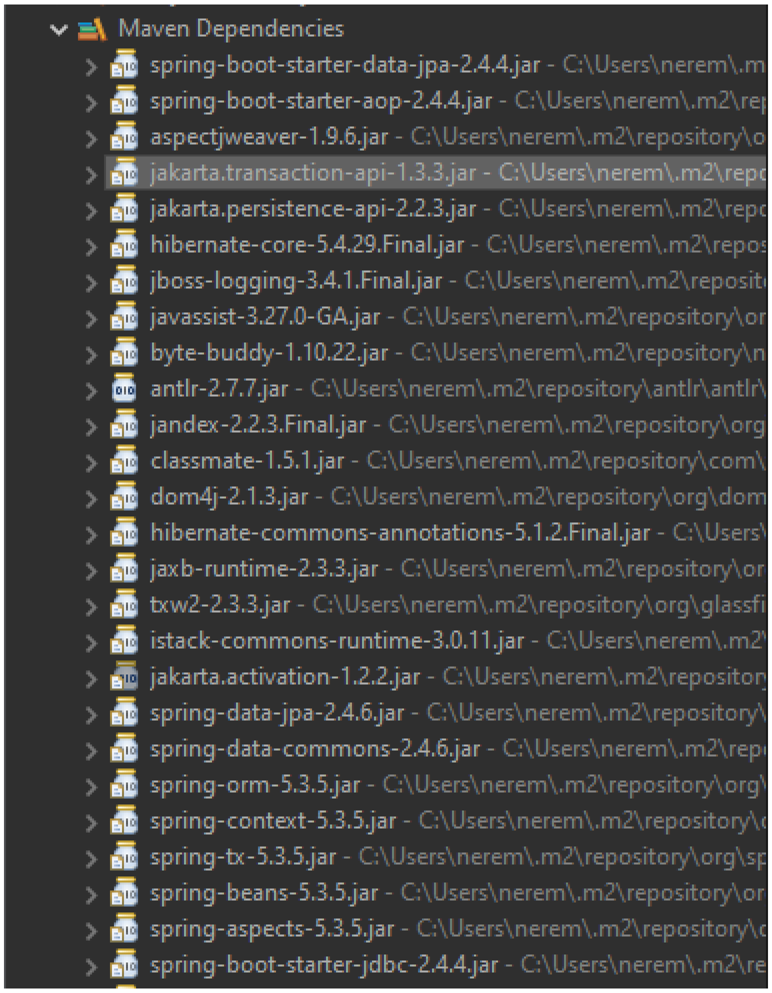
The screenshots show a template for each database entity, each possible error page, for security and a general page-layout.


```
JRE System Library [JavaSE-1.8]
Maven Dependencies
```

The screenshot above illustrates the files for where each maven dependency file is and which java plugins are used. The maven dependencies have a .jar extensions whereas JRE files don't have a particular extension.



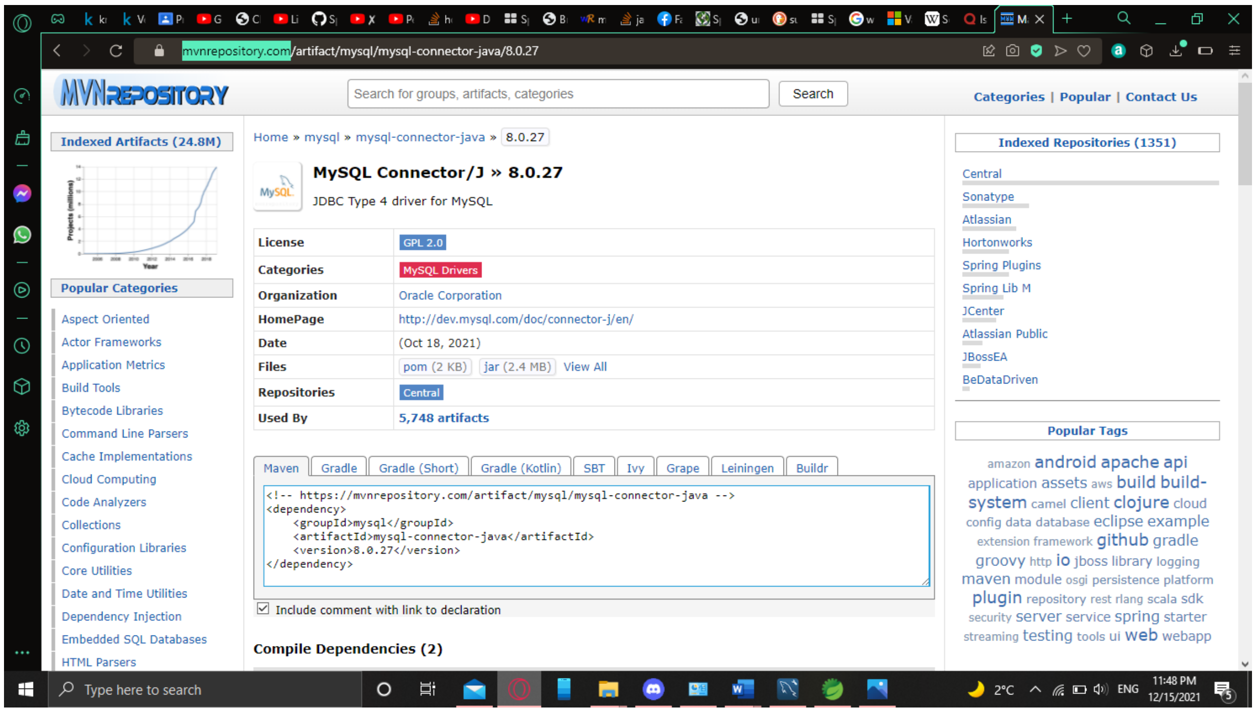
```
JRE System Library [JavaSE-1.8]
  java.base - D:\Program Files\sts-4.12.0.RELEASE\plugins\or
  java.compiler - D:\Program Files\sts-4.12.0.RELEASE\plugir
  java.datatransfer - D:\Program Files\sts-4.12.0.RELEASE\plu
  java.desktop - D:\Program Files\sts-4.12.0.RELEASE\plugins
  java.instrument - D:\Program Files\sts-4.12.0.RELEASE\plug
  java.logging - D:\Program Files\sts-4.12.0.RELEASE\plugins
  java.management - D:\Program Files\sts-4.12.0.RELEASE\p
  java.management.rmi - D:\Program Files\sts-4.12.0.RELEAS
  java.naming - D:\Program Files\sts-4.12.0.RELEASE\plugins
  java.net.http - D:\Program Files\sts-4.12.0.RELEASE\plugins
  java.prefs - D:\Program Files\sts-4.12.0.RELEASE\plugins\o
  java.rmi - D:\Program Files\sts-4.12.0.RELEASE\plugins\org
  java.scripting - D:\Program Files\sts-4.12.0.RELEASE\plugir
  java.se - D:\Program Files\sts-4.12.0.RELEASE\plugins\org.
  java.security.jgss - D:\Program Files\sts-4.12.0.RELEASE\plu
  java.security.sasl - D:\Program Files\sts-4.12.0.RELEASE\plu
  java.sql - D:\Program Files\sts-4.12.0.RELEASE\plugins\org
  java.sql.rowset - D:\Program Files\sts-4.12.0.RELEASE\plug
  java.transaction.xa - D:\Program Files\sts-4.12.0.RELEASE\p
  java.xml - D:\Program Files\sts-4.12.0.RELEASE\plugins\org
  java.xml.crypto - D:\Program Files\sts-4.12.0.RELEASE\plug
  jdk.accessibility - D:\Program Files\sts-4.12.0.RELEASE\plug
  jdk.attach - D:\Program Files\sts-4.12.0.RELEASE\plugins\o
  jdk.compiler - D:\Program Files\sts-4.12.0.RELEASE\plugins
  jdk.dynalink - D:\Program Files\sts-4.12.0.RELEASE\plugins
  jdk.httpserver - D:\Program Files\sts-4.12.0.RELEASE\plugir
  jdk.internal.ed - D:\Program Files\sts-4.12.0.RELEASE\plugi
  jdk.internal.jvmstat - D:\Program Files\sts-4.12.0.RELEASE\
  jdk.internal.le - D:\Program Files\sts-4.12.0.RELEASE\plugir
  jdk.internal.opt - D:\Program Files\sts-4.12.0.RELEASE\plug
```



```
Maven Dependencies
  spring-boot-starter-data-jpa-2.4.4.jar - C:\Users\nerem\.m
  spring-boot-starter-aop-2.4.4.jar - C:\Users\nerem\.m2\re
  aspectjweaver-1.9.6.jar - C:\Users\nerem\.m2\repository\o
  jakarta.transaction-api-1.3.3.jar - C:\Users\nerem\.m2\rep
  jakarta.persistence-api-2.2.3.jar - C:\Users\nerem\.m2\rep
  hibernate-core-5.4.29.Final.jar - C:\Users\nerem\.m2\repos
  jboss-logging-3.4.1.Final.jar - C:\Users\nerem\.m2\reposit
  javassist-3.27.0-GA.jar - C:\Users\nerem\.m2\repository\or
  byte-buddy-1.10.22.jar - C:\Users\nerem\.m2\repository\n
  antlr-2.7.7.jar - C:\Users\nerem\.m2\repository\antlr\antlr\
  jandex-2.2.3.Final.jar - C:\Users\nerem\.m2\repository\org
  classmate-1.5.1.jar - C:\Users\nerem\.m2\repository\com\
  dom4j-2.1.3.jar - C:\Users\nerem\.m2\repository\org\dom
  hibernate-commons-annotations-5.1.2.Final.jar - C:\Users\
  jaxb-runtime-2.3.3.jar - C:\Users\nerem\.m2\repository\or
  txw2-2.3.3.jar - C:\Users\nerem\.m2\repository\org\glassfi
  istack-commons-runtime-3.0.11.jar - C:\Users\nerem\.m2\
  jakarta.activation-1.2.2.jar - C:\Users\nerem\.m2\repositor
  spring-data-jpa-2.4.6.jar - C:\Users\nerem\.m2\repository\
  spring-data-commons-2.4.6.jar - C:\Users\nerem\.m2\rep
  spring-orm-5.3.5.jar - C:\Users\nerem\.m2\repository\org\
  spring-context-5.3.5.jar - C:\Users\nerem\.m2\repository\
  spring-tx-5.3.5.jar - C:\Users\nerem\.m2\repository\org\sp
  spring-beans-5.3.5.jar - C:\Users\nerem\.m2\repository\or
  spring-aspects-5.3.5.jar - C:\Users\nerem\.m2\repository\
  spring-boot-starter-jdbc-2.4.4.jar - C:\Users\nerem\.m2\re
```

And finally at last, but not least the last most important file worth mentioning is pom.xml , where basically all maven dependencies can configured/added/removed, etc by simply copy-pasting their xml from https://mvnrepository.com and once any kind of such action is done, the Maven dependencies file is updated.

Example of how a dependency can be added:


mvnrepository.com/artifact/mysql/mysql-connector-java/8.0.27

Indexed Artifacts (24.8M)

Popular Categories

- Aspect Oriented
- Actor Frameworks
- Application Metrics
- Build Tools
- Bytecode Libraries
- Command Line Parsers
- Cache Implementations
- Cloud Computing
- Code Analyzers
- Collections
- Configuration Libraries
- Core Utilities
- Date and Time Utilities
- Dependency Injection
- Embedded SQL Databases
- HTML Parsers

Home » mysql » mysql-connector-java » 8.0.27

MySQL Connector/J » 8.0.27

JDBC Type 4 driver for MySQL

License	GPL 2.0
Categories	MySQL Drivers
Organization	Oracle Corporation
HomePage	http://dev.mysql.com/doc/connector-j/en/
Date	(Oct 18, 2021)
Files	pom (2 KB) jar (2.4 MB) View All
Repositories	Central
Used By	5,748 artifacts

Maven | Gradle | Gradle (Short) | Gradle (Kotlin) | SBT | Ivy | Grape | Leiningen | Buildr

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.27</version>
</dependency>
```

☑ Include comment with link to declaration

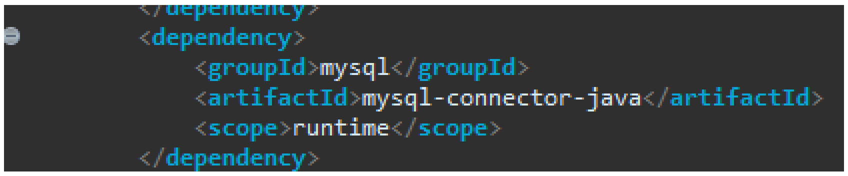
Compile Dependencies (2)

Indexed Repositories (1351)

- Central
- Sonatype
- Atlassian
- Hortonworks
- Spring Plugins
- Spring Lib M
- JCenter
- Atlassian Public
- JBossEA
- BeDataDriven

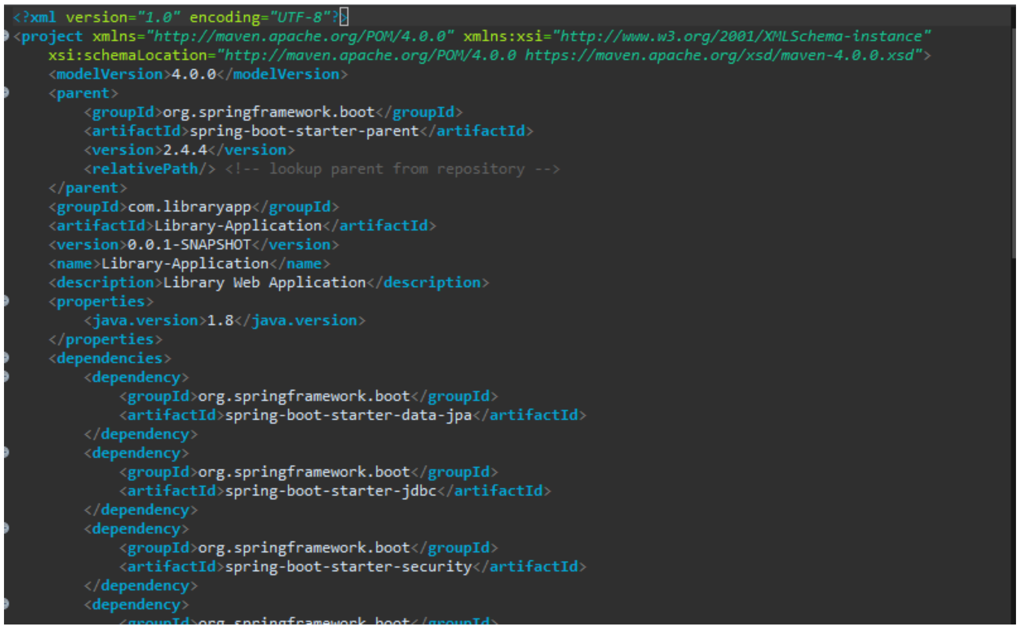
Popular Tags

amazon android apache api application assets aws build build-system camel client clojure cloud config data database eclipse example extension framework github gradle groovy http io jboss library logging maven module osgi persistence platform plugin repository rest rlang scala sdk security server service spring starter streaming testing tools ui web webapp

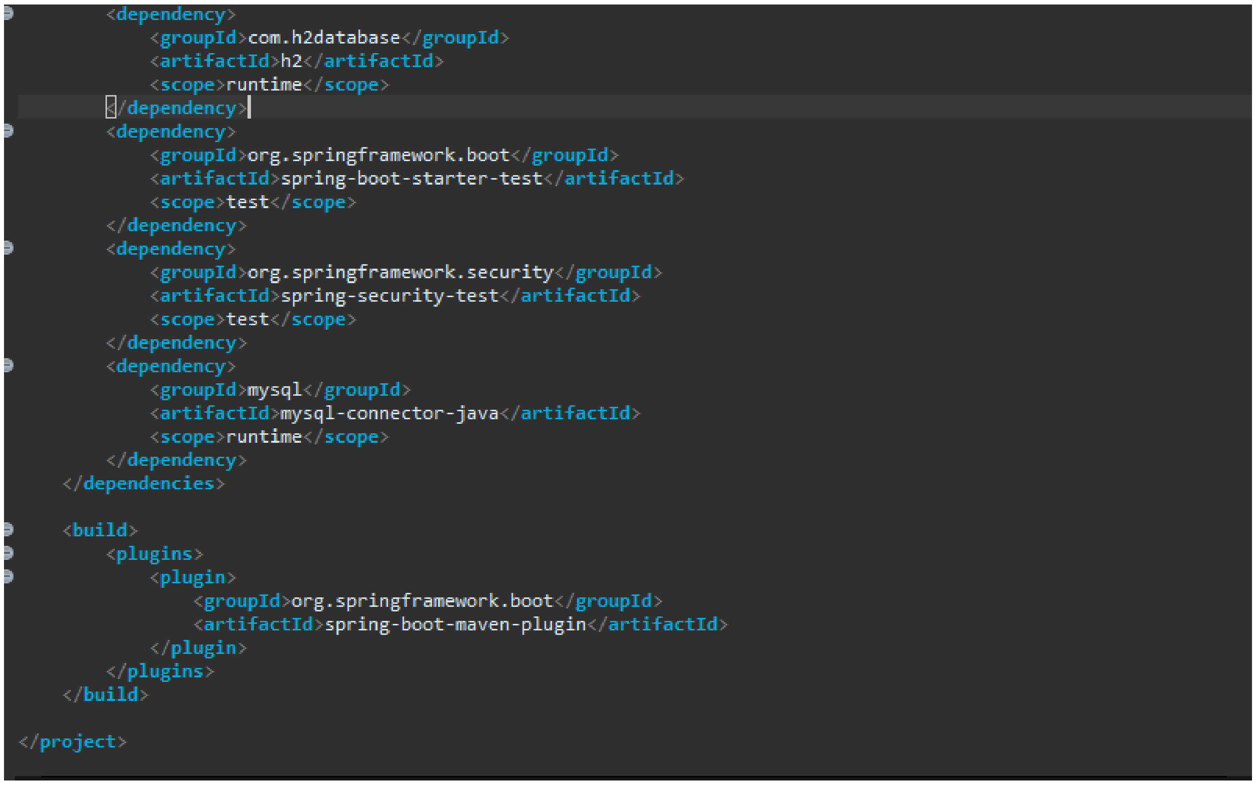


```
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
```

The following screenshots illustrate a part of how the pom.xml configuration looks like overall:



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.4.4</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.libraryapp</groupId>
    <artifactId>Library-Application</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>Library-Application</name>
    <description>Library Web Application</description>
    <properties>
        <java.version>1.8</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jdbc</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        <dependency>
```



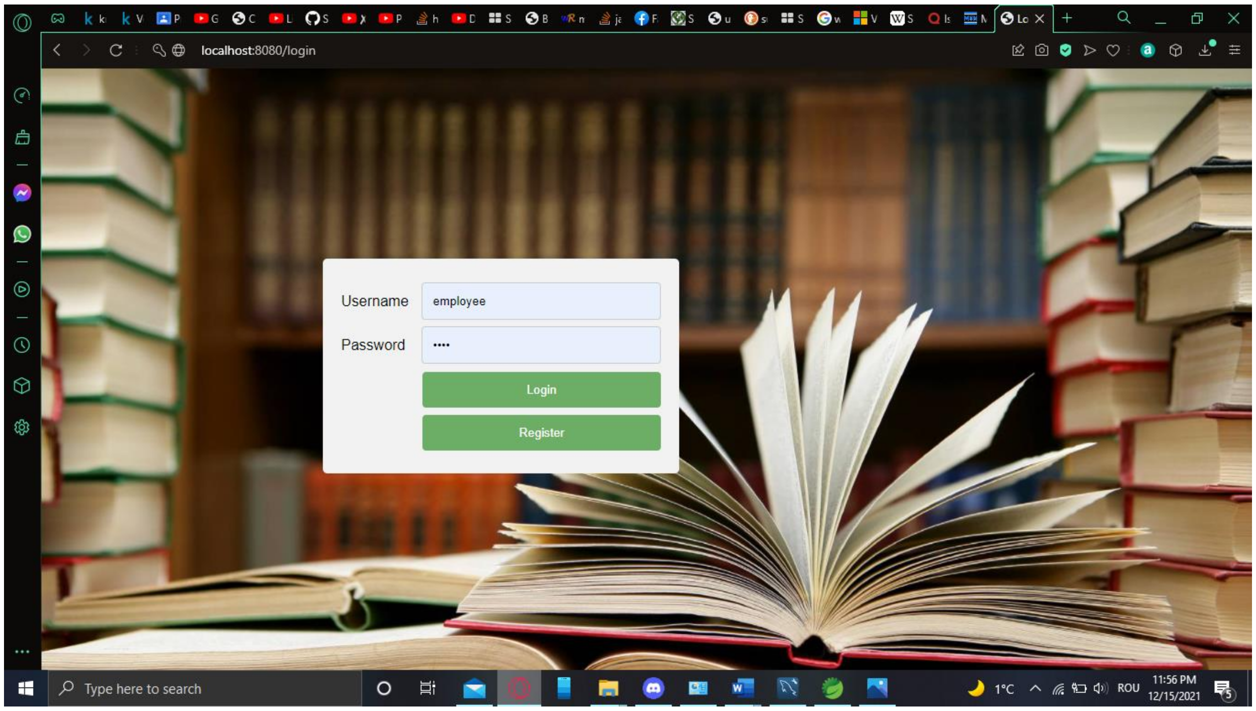
```xml
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

Frontend implementation

Here are some examples of the current user interfaces, because not all of them are 100% done or not presented yet due to other reasons (such as not making time to properly make all screenshots of all functionalities which are a lot) as I want them to be and still require some improvements:

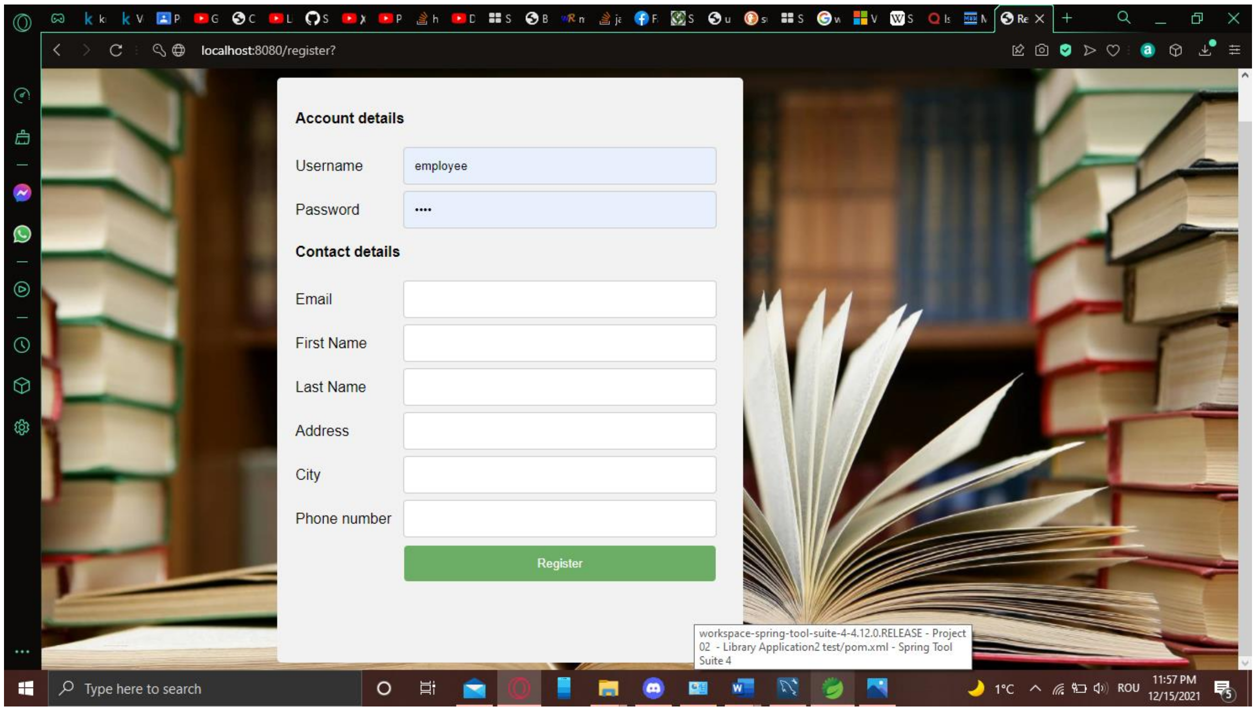
Login Form

As the interface title suggests, this is a login in form through which users can either **log in** to the application or **register** if they don't have an account already.



Register Form

As the title above suggests, here unregistered users can create an account by filling in the the following form with different private data (username, password, email, first name, last name, adrress, city, phone number).



localhost:8080/register?

Account details

Username	employee
Password	••••

Contact details

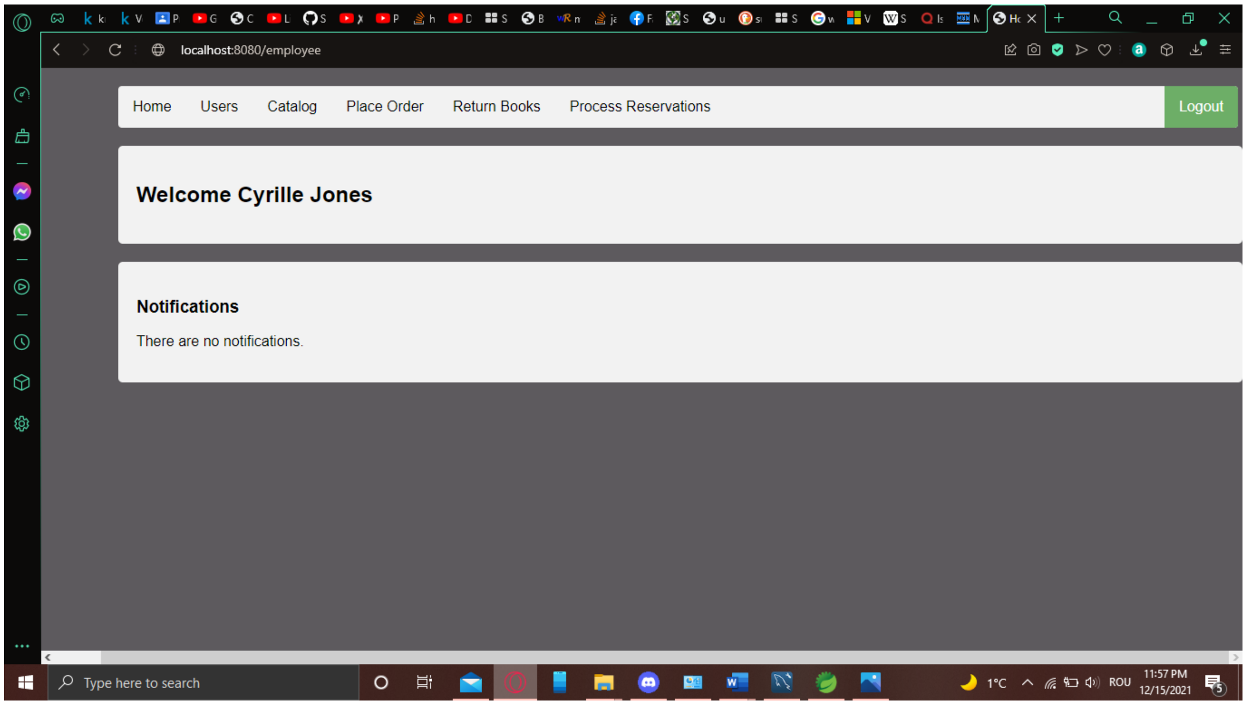
Email	
First Name	
Last Name	
Address	
City	
Phone number	

Register

workspace-spring-tool-suite-4-4.12.0.RELEASE - Project 02 - Library Application2 test/pom.xml - Spring Tool Suite 4

Home page of an employee user

If a user logs in with an employee account, he will be recognized as an actual **employee**, and will thus be redirected to a home page in which **only** employees have access to. In this home page, the employee can mainly access the **catalog**, **users** (as **customers**), place order for **customers**, **return books** and **process reservations** or come back to the **home** page.



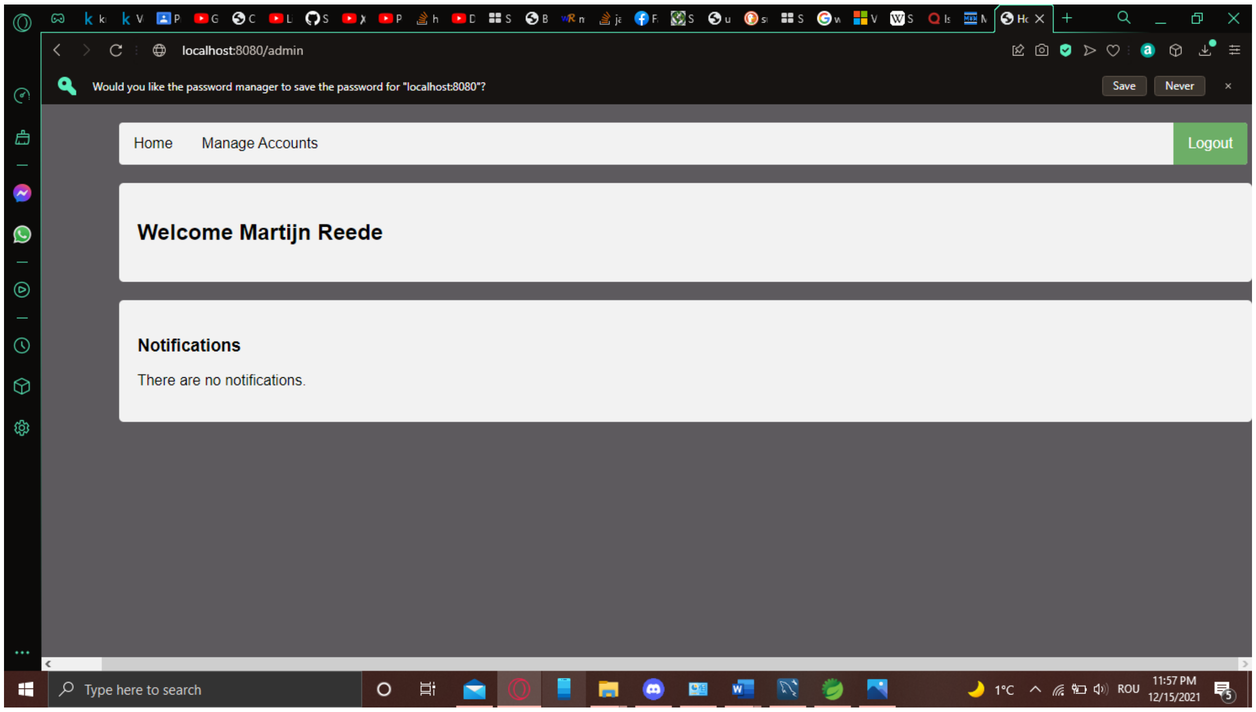
Welcome Cyrille Jones

Notifications

There are no notifications.

Home page of an admin user

If a user logs in with an employee account, he will be recognized as an actual **admin**, and will thus be redirected to a home page in which **only library managers** or **administrators** have access to. In this home page, the admin, for now, can mainly **manage accounts** or come back to **home** page.


localhost:8080/admin

Would you like the password manager to save the password for "localhost:8080"?

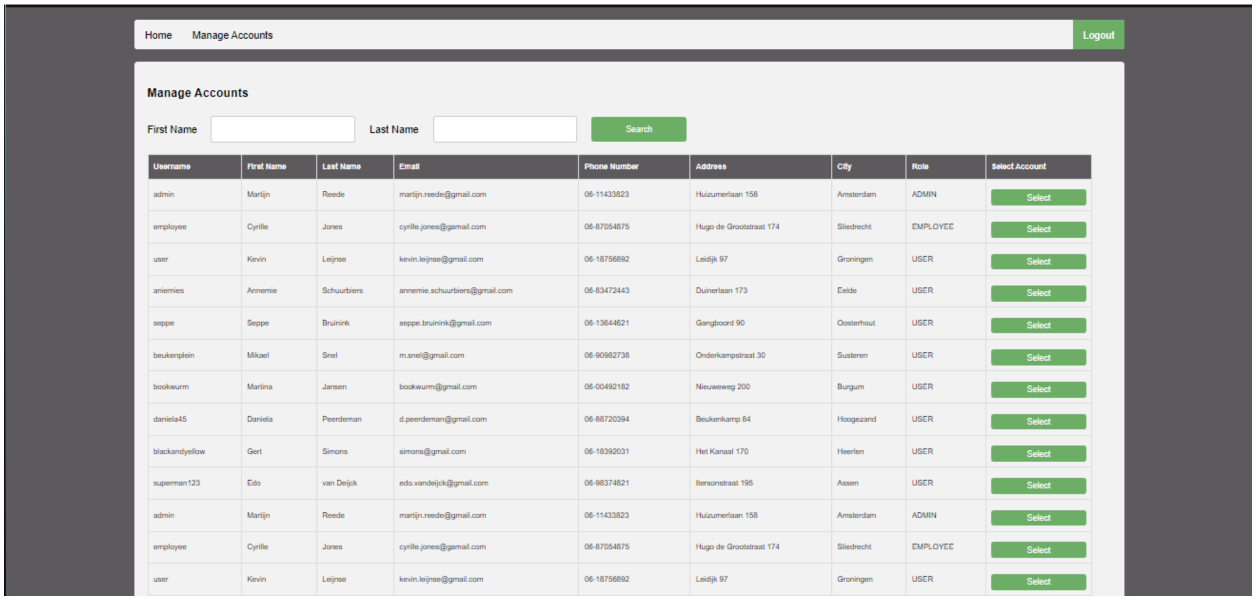
Welcome Martijn Reede

Notifications

There are no notifications.

Manage Accounts

As an admin, he can edit settings of a user account by either selecting it from the list of users or searching it either by first name or last name.



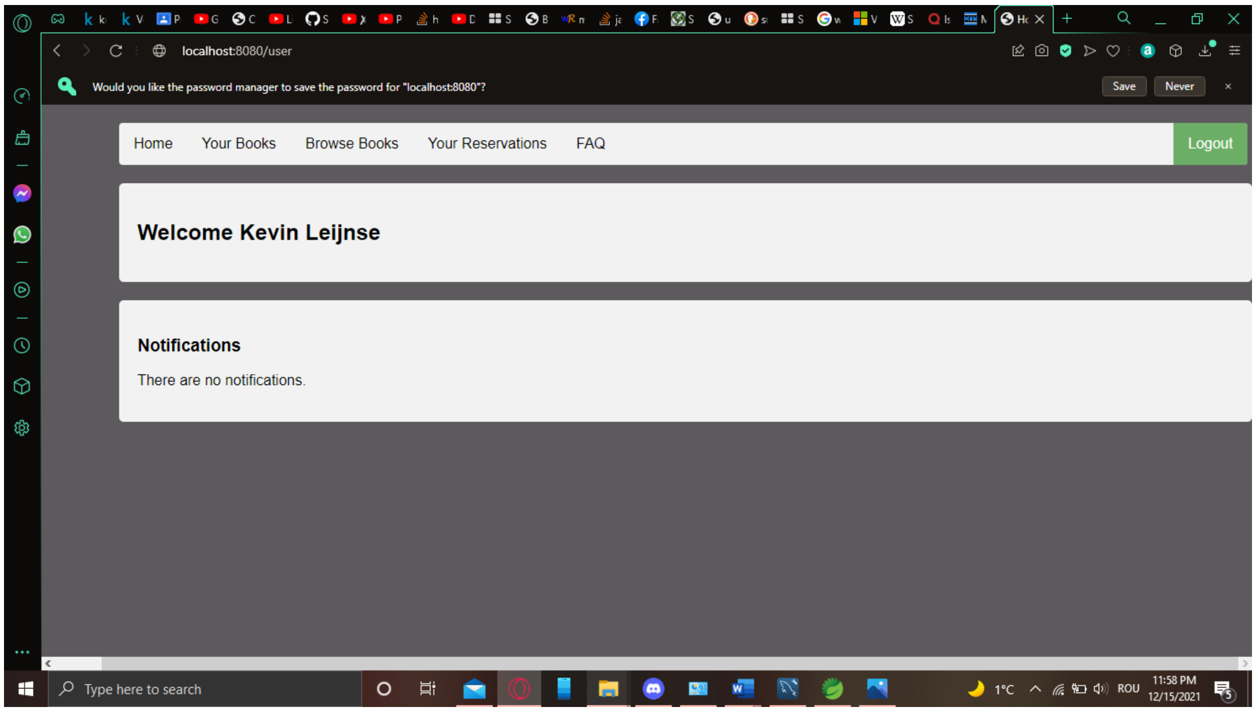
Manage Accounts

First Name | Last Name | Search

Username	First Name	Last Name	Email	Phone Number	Address	City	Role	Select Account
admin	Martijn	Reede	martijn.reede@gmail.com	06-11433823	Huizumerlaan 158	Amsterdam	ADMIN	Select
employee	Cyrille	Jones	cyrille.jones@gamail.com	06-87054875	Hugo de Grootstraat 174	Sliedrecht	EMPLOYEE	Select
user	Kevin	Leijnse	kevin.leijnse@gmail.com	06-18756892	Leidijk 97	Groningen	USER	Select
aniemies	Annemie	Schuurbiers	annemie.schuurbiers@gmail.com	06-83472443	Duinerlaan 173	Eelde	USER	Select
seppe	Seppe	Bruinink	seppe.bruinink@gmail.com	06-13644621	Gangboord 90	Oosterhout	USER	Select
beukenplein	Mikael	Snel	m.snel@gmail.com	06-90982738	Onderkampstraat 30	Susteren	USER	Select
bookwurm	Martina	Jansen	bookwurm@gmail.com	06-00492182	Nieuweweg 200	Burgum	USER	Select
daniela45	Daniela	Peerdeman	d.peerdeman@gmail.com	06-88720394	Beukenkamp 84	Hoogezand	USER	Select
blackandyellow	Gert	Simons	simons@gmail.com	06-18392031	Het Kanaal 170	Heerlen	USER	Select
superman123	Edo	van Deijck	edo.vandeijck@gmail.com	06-98374821	Itersonstraat 195	Assen	USER	Select
admin	Martijn	Reede	martijn.reede@gmail.com	06-11433823	Huizumerlaan 158	Amsterdam	ADMIN	Select
employee	Cyrille	Jones	cyrille.jones@gamail.com	06-87054875	Hugo de Grootstraat 174	Sliedrecht	EMPLOYEE	Select
user	Kevin	Leijnse	kevin.leijnse@gmail.com	06-18756892	Leidijk 97	Groningen	USER	Select

Home page of a customer user

If a user logs in with a regular registered account, he will be recognized as an actual **customer**, and will thus be redirected to a home page in which **only customers** have access to. In this home page, the customer can mainly **manage accounts** or come back to **home** page.



Would you like the password manager to save the password for "localhost:8080"?

Welcome Kevin Leijnse

Notifications

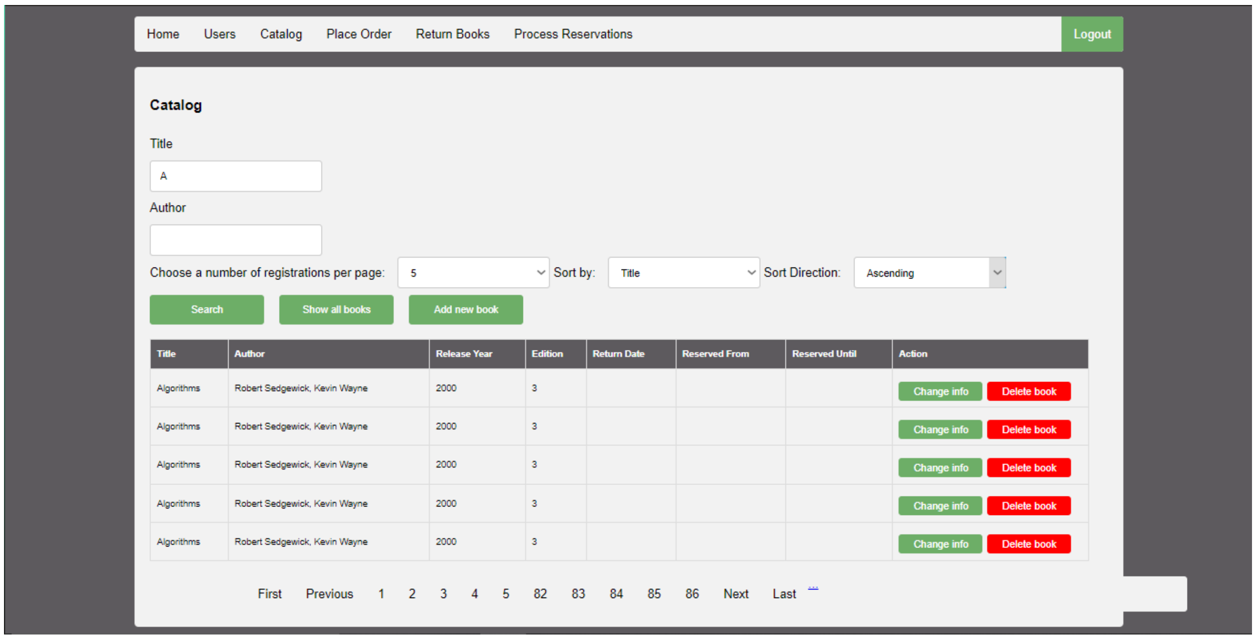
There are no notifications.

Catalog

In this section, the employee has multiple choices. He or she can :

- search for a book by **title** or **author**, choose **a number of registered books per page** from search results, sort the results by **title** or **author** or **sort the results in ascending or descending direction**.
- return all paginated books
- **add** a new book
- **change** book info

- **delete** a book



Home Users Catalog Place Order Return Books Process Reservations Logout

Catalog

Title: A

Author:

Choose a number of registrations per page: 5 Sort by: Title Sort Direction: Ascending

Search | Show all books | Add new book

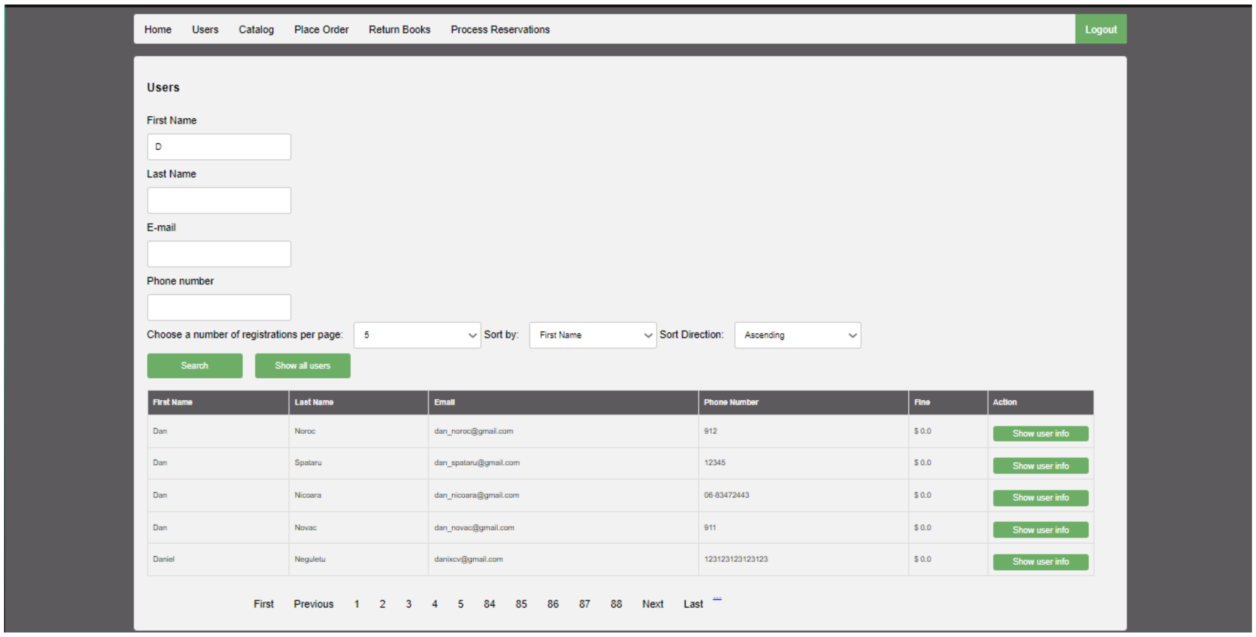
Title	Author	Release Year	Edition	Return Date	Reserved From	Reserved Until	Action
Algorithms	Robert Sedgewick, Kevin Wayne	2000	3				Change info Delete book
Algorithms	Robert Sedgewick, Kevin Wayne	2000	3				Change info Delete book
Algorithms	Robert Sedgewick, Kevin Wayne	2000	3				Change info Delete book
Algorithms	Robert Sedgewick, Kevin Wayne	2000	3				Change info Delete book
Algorithms	Robert Sedgewick, Kevin Wayne	2000	3				Change info Delete book

First Previous 1 2 3 4 5 82 83 84 85 86 Next Last

Users

In this section, the employee has multiple choices. He or she can :

- search for a user by **first name**, **last name**, **e-mail** or **phone number**, choose **a number of registered users per page** from search results, sort the results by fields previously mentioned or **sort the results in ascending or descending direction**.
- return all paginated users
- **show** user info



Users

First Name: D

Last Name:

E-mail:

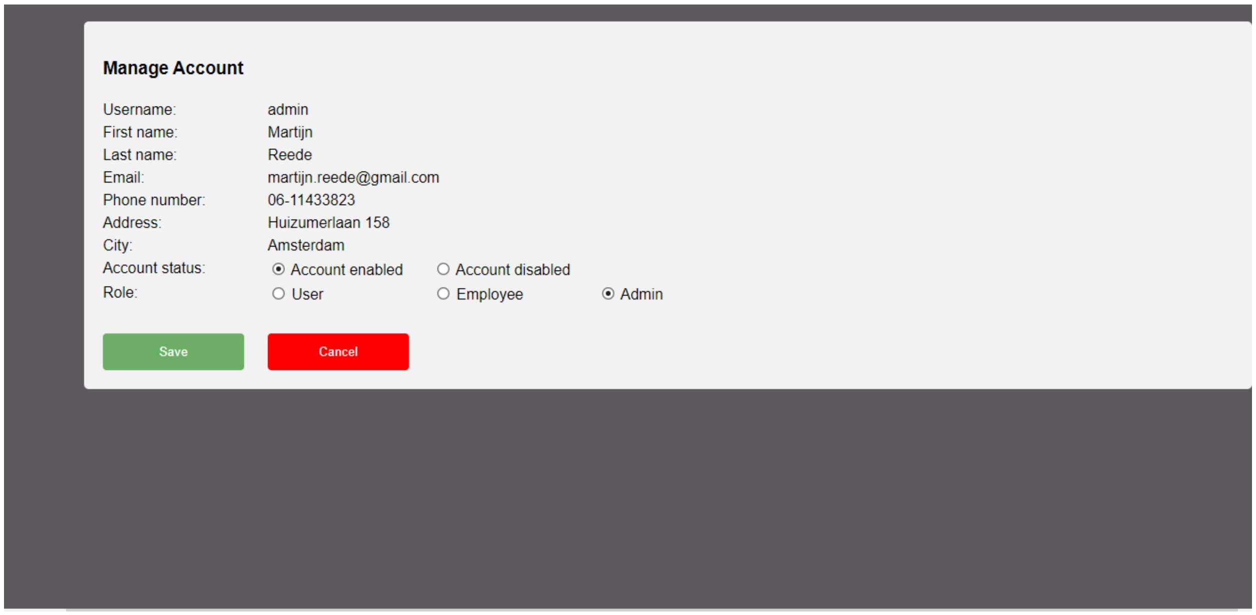
Phone number:

Choose a number of registrations per page: 5 Sort by: First Name Sort Direction: Ascending

First Name	Last Name	Email	Phone Number	Fine	Action
Dan	Noroc	dan_noroc@gmail.com	912	$ 0.0	Show user info
Dan	Spataru	dan_spataru@gmail.com	12345	$ 0.0	Show user info
Dan	Nicoara	dan_nicoara@gmail.com	06-83472443	$ 0.0	Show user info
Dan	Novac	dan_novac@gmail.com	911	$ 0.0	Show user info
Daniel	Neguletu	danixcv@gmail.com	123123123123123	$ 0.0	Show user info

Manage Account

In this page, when an admin selects a user, the admin can either enable or disable an account (a disabled account cannot log in) or change the role of a user account.



Manage Account

Username:	admin
First name:	Martijn
Last name:	Reede
Email:	martijn.reede@gmail.com
Phone number:	06-11433823
Address:	Huizumerlaan 158
City:	Amsterdam
Account status:	[✓ Account enabled] Account disabled
Role:	User Employee [✓ Admin]

Last Remark: The only thing that wasn’t fully respected in this architecture was the fact that I didn’t manage to update the admin role to have the feature to manage books instead of the employee.