**Payal Joshi**

# CS 510 Full stack Project Report

I have implemented Library Management System as a part of this project. Let us try to understand the project and its implementation details.

## Implementation Details:

### Using Spring Boot and Spring MVC:

For this project, I wanted to make use of a java framework, partly because I felt that way I could arrange my web application in a layered fashion. Hence, I chose to use Spring Boot which supports Spring MVC framework. Here, MVC stands for Model, view and controller. This way I could achieve my primary purpose of using a layered architecture. Prior to this, I had used Spring MVC briefly while working in professional environment. My initial study on Spring boot helped me understand that Spring Boot supports Spring related frameworks while at the same time it eases the configuration process for you. It is indeed true. Internally, Spring Boot uses maven to download all the dependencies required for your project.

For example,One of the entry in my project's pom.xml is

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Here, I am instructing the Spring Boot to download JPA related dependencies for my project.

### Login:

An advantage of using Spring Boot is the security related configurations offered by it's WebSecurityConfigurerAdapter class. By extending this class and overriding its configure method enables the developer to make sure that requests to certain URLs are always granted to authorized users and are authenticated by ensuring that such user exists in the table with the given role. For example,

```
http.authorizeRequests().antMatchers("/register")
.permitAll()
    .antMatchers("/home").hasAnyRole("USER")
.anyRequest().authenticated().and().formLogin().loginPage("/logi
n").permitAll()
    .and()

.logout().logoutUrl("/login?logout=1").logoutSuccessHandler(cust
omizeLogoutSuccessHandler)
```

```
        .permitAll();
```

In the above code snippet, it is clearly seen that any request to url '/register' is permitted to all the users. However, request to url '/home' needs the user to log into the system. Hence, the user is redirected to the login page. The logouturl() handles the logout request which ultimately invalidates the user session.

Adapting to Spring Boot framework was trickier than I expected it to be. However, online video tutorials by Tech primers [2] and Java in Use [3] helped me understand the details in a classroom like environment. Spring Boots official documentation [1] also helped to understand the basics of it. I started with Spring Security Example given by Tech Primers [4]. This example was a good starting point for the project. It mainly helped me build the login functionality.

**Entity classes used:**

We declare persistent Plain Old Java Object (POJO) as an entity class using javax.Persistence annotation '@Entity'. '@Id' is used to declare identifier property of the class[6]. In simple words, we map POJO object to a database table. The table name is specified using '@Table' annotation. Whenever the table's column name is different than the java class's attribute name, we specify column name as '@Column(name = "bookname")'. I have created 3 entity classes which are - Books, Users and Role which are mapped to books, users and role table respectively. In 'User' entity, @ManyToMany mapping is declared to map Role entity as a Collection in the parent object i.e. it creates join table named 'user_role', that has columns 'user_id' (from user table) and 'role_id' (from role table). This way one role (ROLE_USER) can be mapped to many users. Even though I have used these entity relations in the past, I reviewed them using Hibernate Video tutorials by Java Brains[7].

**Tables Used:**

As discussed earlier, there are 3 main tables in the database.

1. books
    - Stores details related to book such as book name, author, location, no of copies, etc
2. users
    - Stores details related to users such as username, password.
3. Role
    - Stores details related to roles such as role_id and its description

User_role table is created as a result of the mapping entity.

**Register new user:**

For User registration process,I used Spring Data Repository's CrudRepository.As the name suggests, this repository provides CRUD functionality [5]. The repository implemented for this purpose is UserRepository which takes as input 'Users' class as domain to manage and id type of domain class as Integer. On the Registration form, user fills in all the required details and submits the form, on form submission, the controller class, "LoginController" handles the request. In the input, it receives the 'UserRegistration' bean submitted on form submission. It fetches all the required values from this bean and sets them in 'Users' object which is eventually persisted in database using CrudRepository's save() method [5]. Successful registration of User inserts user's entry in 'users' table as well 'user_role' table using role id assigned to 'USER' role.

**Search a book:**

This functionality allows user to search a book by its name. It is triggered once the user types a book name in the search box and clicks on search button. Similar to Registration, this functionality is also implemented using Spring Data Repository. The 'BookRepository' class extends the JpaRepository with 'Books' as its domain class. Once the controller receives a request for '/displayBookDetails' mapping, it calls the findByBookName method of BookRepository.As a result, it receives the result in the form of list. This list is as an object to thhe ModelAndView and is returned to the user. At the jsp end, the result is handled using jstl core [8].Jstl examples explained on [9] helped me display the results on the front end successfully.

**Browse Books:**

This functionality is implemented similar to the Search book functionality. However, the only difference is that I don't pass the book name and the url mapping is '/displayBookList'. The front end implementation is as explained above.

**Database :**

For database, I have used MySQL server and its client mySQL Workbench to access the data. I have created a separate database named 'library' in my local server. Its configuration details are specified in the project in application.properties file as shown in tutorial [2]. The properties include datasource url, username and password.

spring.datasource.url=jdbc:mysql://localhost:3306/library?useSSL=false

spring.datasource.username=root

spring.datasource.password=password

[Note: all the queries are auto-generated with the help of Hibernate and JPA]

**Front End:**

For front end, I used jsp, bootstrap, javascript and css. The User Login and Registration forms are implemented with the help of examples explained on bootstrap official site[10]. The validations are performed using bootstrap as well jsp's attribute like 'required'. Other than this, the home page is implemented using jumbotron[12] and container classes applied to divs. Also, the contents on this page are taken from PSU's library page [11]. The navigation bar i.e the default menu is implemented using bootstrap's navbar [13]. I have also used symbols from font-awesome for fields like user name, email, etc [14].

**Conclusion:**

During the midterm project pitch, I committed to create functionalities Login/ Registration, Browse Books, Checking their availability and locating them.I have succeeded in implementing them with few minor defects. I believe as a future work to this project, goal would be to improvise UI to suit today's UI design goals, improvising the functionalities, validations for forms, etc.

**Reference:**


[1] https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/
[2] https://www.youtube.com/watch?v=egXtoL5Kg08
[3] http://www.javainuse.com/spring/sprboot_sec
[4] https://github.com/TechPrimers/spring-security-db-example
[5]https://docs.spring.io/spring-data/data-commons/docs/1.6.1.RELEASE/reference/html/repositories.html
[6]https://docs.jboss.org/hibernate/stable/annotations/reference/en/html/entity.html#entity-hibspec-entity
[7] https://www.youtube.com/watch?v=Yv2xctJxE-w&list=PL4AFF701184976B25
[8] https://docs.oracle.com/javaee/5/jstl/1.1/docs/tlddocs/c/tld-summary.html
[9] https://www.tutorialspoint.com/jsp/jsp_standard_tag_library.htm
[10] https://getbootstrap.com/docs/4.0/components/forms/
[11] https://www.pdx.edu/floorplans/buildings/ml
[12] https://getbootstrap.com/docs/4.0/components/jumbotron/
[13] https://getbootstrap.com/docs/4.0/components/navbar/
[14] https://fontawesome.com/v4.7.0/icons/