UE20CS352(OOAD with Java)

Self Learning Hands-on Assignment: MVC Framework

V Himadhith	PES1UG20CS478	Section : H	Roll no : 45

MVC Architecture Pattern:

Model-View-Controller, or MVC, is a software architecture paradigm that divides an application into three interdependent parts. This design pattern is popular in web application development because it makes it easier to divide an application's concerns into discrete modules. The Controller functions as a mediator between the Model and View, handling user input and changing the Model and View appropriately. The Model represents the data and business logic of the application, the View represents the presentation layer.

Advantages of MVC pattern:

1. Separation of Concerns: Using the MVC pattern, an application is divided into various modules, each of which oversees a different concern. Developers can work separately on various components of the programme without interfering with each other's work because to this easier management of the codebase.
2. Testability: Writing unit tests for each application module is made simpler by the separation of concerns. This makes it possible for developers to find flaws early on and repair them.
3. Code Reusability: The MVC pattern encourages modular design, which makes it simpler to reuse code across various components of the programme. This shortens the development process and lessens the possibility that problems may be added because of redundant code.
4. Scalability: Because each module may be scaled individually, the MVC approach makes it simpler to scale an application. This enables the programme to accommodate growing user demand and traffic without compromising performance.

Features of MVC Framework Chosen:

1. Inversion of Control (IoC) Container: Spring provides an IoC container that manages the creation and lifecycle of objects in the application. This allows developers to focus on the business logic of the application rather than the mechanics of object creation and management.

2. Dependency Injection (DI): Spring supports DI, which allows objects to be loosely coupled and enables easy testing and maintenance of code. This means that objects can be easily replaced or modified without affecting other parts of the application.

3. Spring MVC: Spring includes its own implementation of the MVC architecture pattern, which provides a robust framework for building web applications. It includes features such as request mapping, view resolution, and form handling, as well as support for RESTful web services.

4. Aspect-Oriented Programming (AOP): Spring provides support for AOP, which enables cross-cutting concerns such as logging, security, and transaction management to be modularized and applied across multiple modules of the application.

5. Integration with other technologies: Spring provides integration with a wide range of other technologies, including Hibernate, JPA, Struts, and JSF. This allows developers to use the best tools for each part of the application, while still maintaining a consistent programming model.

6. Testing: Spring provides support for testing through its testing framework, which includes features such as JUnit integration, mock objects, and test context management.

Problem definition with description of the chosen scenarios

The Library Management System is a web application for managing a library built using Spring Boot, following the MVC (Model-View-Controller) design pattern, and uses mongoDB as its database. The application allows users to add and view books in the library's collection.

Users can view all books in the collection. They can also add new books to the collection.

The project aims to provide a user-friendly interface for managing a library collection, making it easier for librarians to keep track of books, and allowing patrons to search for and view books in the collection.

Code:

Model class

Book.java

src > main > java > com > example > mvcbooktracker > Book.java > ...

```java
package com.example.mvcbooktracker;

import org.springframework.data.mongodb.core.mapping.Document;

@Document("books")
public class Book {
    private String name;
    private String description;
    private String author;
    private int yearOfRelease;
    private int rating;

    //Getter Functions
    public String getName() {
        return name;
    }
    public String getDescription() {
        return description;
    }
    public String getAuthor() {
        return author;
    }
    public int getYearOfRelease() {
        return yearOfRelease;
    }
    public int getRating() {
        return rating;
    }

    //Setter Functions
    public void setName(String name) {
        this.name = name;
    }
    public void setDescription(String description) {
        this.description = description;
    }
    public void setAuthor(String author) {
        this.author = author;
    }
    public void setYearOfRelease(int yearOfRelease) {
        this.yearOfRelease = yearOfRelease;
    }
    public void setRating(int rating) {
        this.rating = rating;
    }

}
```

View class

MvcbooktrackerApplication.java M

src > main > java > com > example > mvcbooktracker > MvcbooktrackerApplication.java > MvcbooktrackerApplication

```java
package com.example.mvcbooktracker;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MvcbooktrackerApplication {

    Run | Debug
    public static void main(String[] args) {
        SpringApplication.run(MvcbooktrackerApplication.class, args);
    }

}
```

Controller class

BookController.java

src > main > java > com > example > mvcbooktracker > BookController.java > BookController > homepage(Model)

```java
package com.example.mvcbooktracker;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.ModelAttribute;

@Controller
public class BookController {

    @Autowired
    public MongoRepo repo;

    @GetMapping("/")
    public String homepage(Model model){
        return "homepage";
    }

    @GetMapping("/viewbooks")
    public String viewBooks(Model model) {
        List<Book> books = repo.findAll();
        model.addAttribute("allbooks",books);
        return "viewbooks";
    }

    @GetMapping("/addbook")
    public String addBook(Model model) {
        model.addAttribute("book",new Book());
```

```java
    @GetMapping("/viewbooks")
    public String viewBooks(Model model) {
        List<Book> books = repo.findAll();
        model.addAttribute("allbooks",books);
        return "viewbooks";
    }

    @GetMapping("/addbook")
    public String addBook(Model model) {
        model.addAttribute("book",new Book());
        return "addbook";
    }

    @PostMapping("/addbook")
    public String submitBook(@ModelAttribute("book") Book book) {
        repo.save(book);
        return "redirect:viewbooks";
    }
}
```

Addpage.html

```html
<!DOCTYPE HTML>
<html xmlns:th="https://www.thymeleaf.org">
<head>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f0f0f0;
            margin: 0;
            padding: 0;
        }

        .container {
            max-width: 600px;
            margin: 50px auto;
            padding: 30px;
            background-color: #ffffff;
            border-radius: 5px;
            box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
        }

        h1 {
            font-size: 24px;
            text-align: center;
            margin-bottom: 25px;
        }

        label {
            display: block;
            font-size: 14px;
            margin-bottom: 5px;
        }

        input {
            width: 100%;
            padding: 8px;
            font-size: 14px;
            border: 1px solid #cccccc;
            border-radius: 4px;
            margin-bottom: 20px;
        }

        input[type="submit"], input[type="reset"] {
            width: auto;
            background-color: #4CAF50;
            color: white;
            border: none;
            padding: 10px 20px;
            cursor: pointer;
            border-radius: 4px;
            margin-right: 10px;
        }

        input[type="submit"]:hover, input[type="reset"]:hover {
            background-color: #45a049;
        }
```

```css
        .navbar {
            background-color: #4CAF50;
            overflow: hidden;
            padding: 8px 16px;
        }

        .navbar a {
            float: left;
            display: block;
            color: white;
            text-align: center;
            padding: 14px 16px;
            text-decoration: none;
            font-size: 17px;
        }

        .navbar a:hover {
            background-color: #45a049;
        }
```

addbook.html M

src > main > resources > templates > addbook.html > html > head > style

```
</head>
<body>
    <div class="container">
        <div class="navbar">
            <a href="/">Home</a>
            <a href="viewbooks">View Books</a>
            <a href="addbook">Add Books</a>
        </div>
        <h1>Add a Book</h1>
        <form action="#" th:action="@{/addbook}" th:object="${book}" method="post">
            <label for="name">Name:</label>
            <input type="text" th:field="*{name}" id="name" />

            <label for="description">Description:</label>
            <input type="text" th:field="*{description}" id="description" />

            <label for="author">Author:</label>
            <input type="text" th:field="*{author}" id="author" />

            <label for="yearOfRelease">Year of Release:</label>
            <input type="number" th:field="*{yearOfRelease}" id="yearOfRelease" />

            <label for="rating">Rating:</label>
            <input type="number" th:field="*{rating}" id="rating" />

            <input type="submit" value="Submit" />
            <input type="reset" value="Reset" />
        </form>
    </div>
</body>
</html>
```

homepage.html

adding css for all the files will make the file pretty big so I am skipping the css part.

```
<body>
    <div class="container">
        <div class="navbar">
            <a href="/">Home</a>
            <a href="viewbooks">View Books</a>
            <a href="addbook">Add Books</a>
        </div>
        <h1>PES1UG20CS478-MVC</h1>
        <h1>Home Page</h1>
        <a href="viewbooks">View Books</a>
        <a href="addbook">Add Books</a>
    </div>
</body>
</html>
```

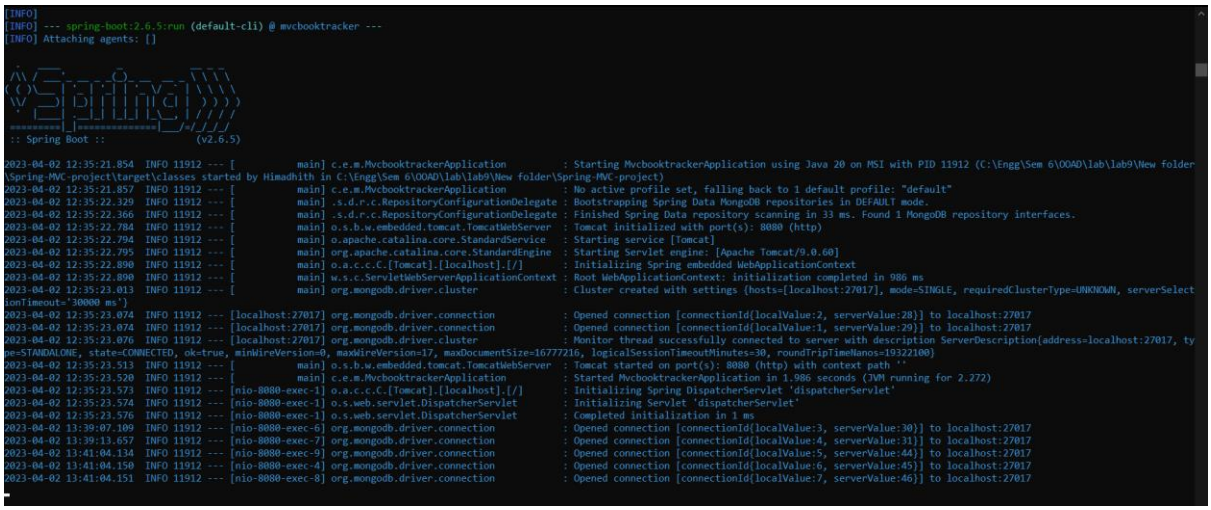
Viewbooks.html

viewbooks.html M

src > main > resources > templates > viewbooks.html > ...

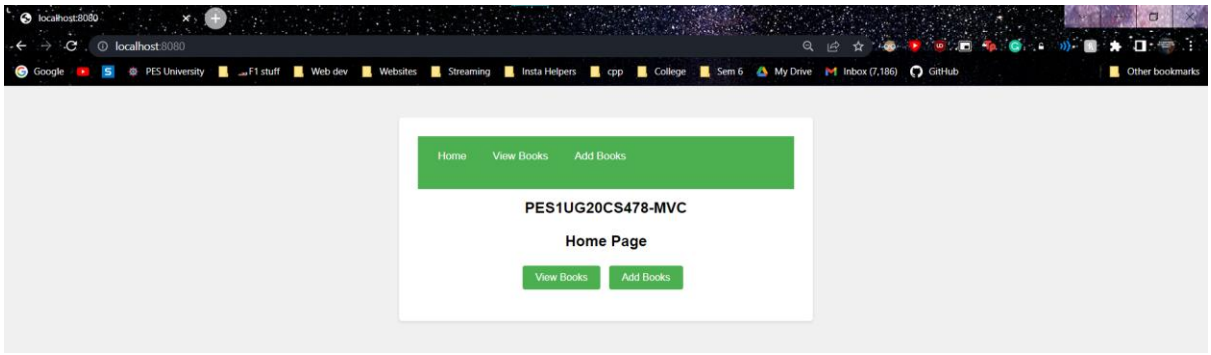
```
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="navbar">
            <a href="/">Home</a>
            <a href="viewbooks">View Books</a>
            <a href="addbook">Add Books</a>
        </div>
        <h1>All Books</h1>
        <table>
            <tr>
                <th>Name</th>
                <th>Description</th>
                <th>Author</th>
                <th>Year of Release</th>
                <th>Rating</th>
            </tr>
            <tr th:each="b : ${allbooks}">
                <td th:text="${b.name}"></td>
                <td th:text="${b.description}"></td>
                <td th:text="${b.author}"></td>
                <td th:text="${b.yearOfRelease}"></td>
                <td th:text="${b.rating}"></td>
            </tr>
        </table>
        <a href="addbook">Add Books</a>
    </div>
</body>
</html>
```

Console with application running:

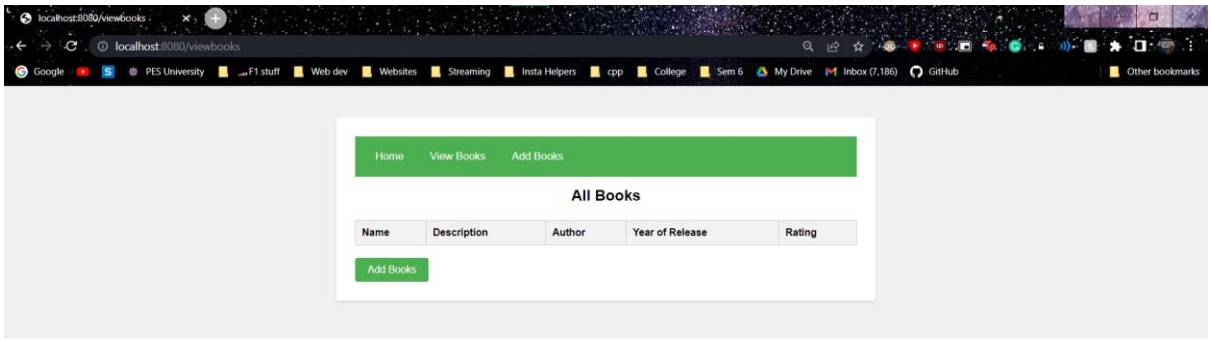


UIs related to the 2 scenarios:

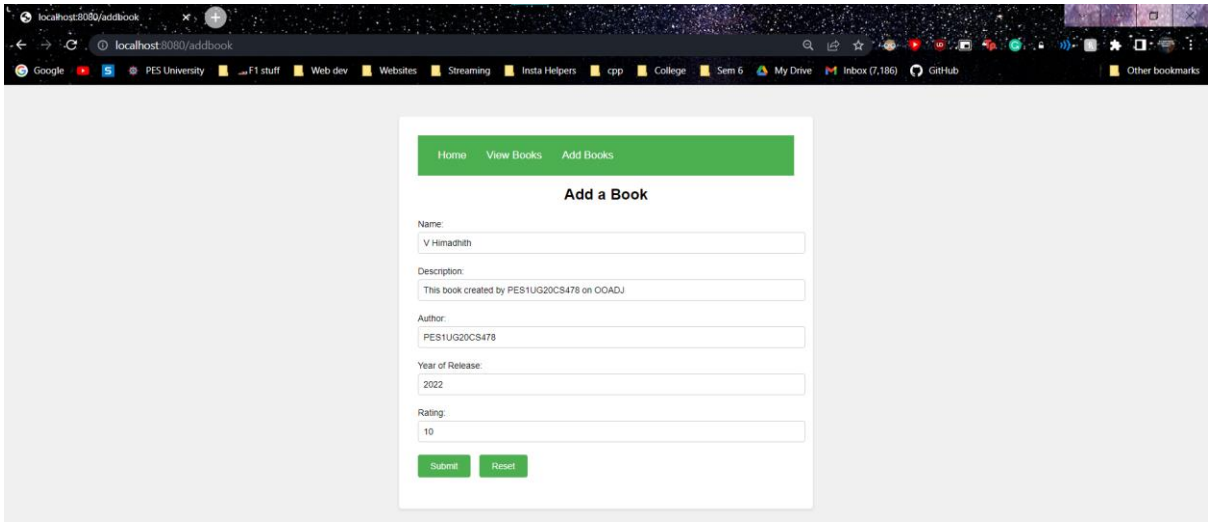
Homepage



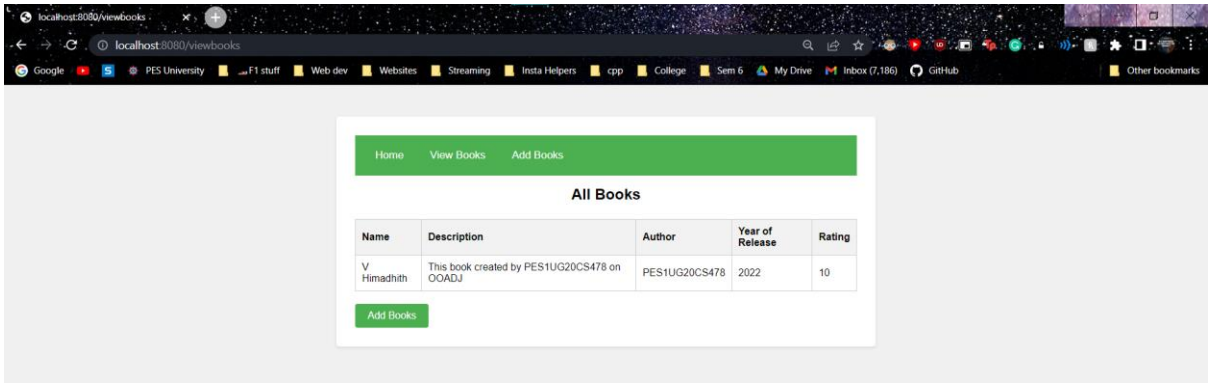
Before adding any books

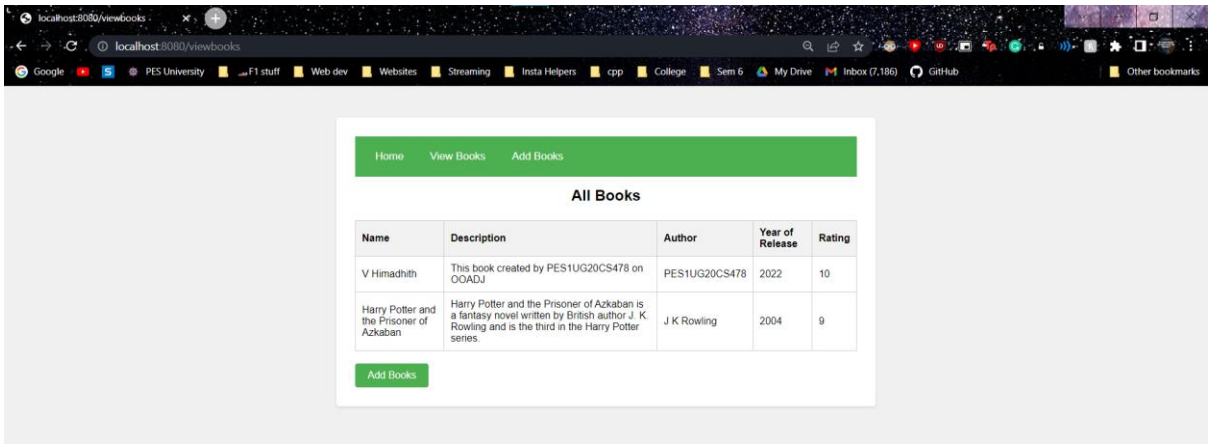


The form to add books



Results after adding books





MongoDB: