Project Report: Spring Boot Application

Overview

This project involves the development of a Spring Boot application featuring two entities, **Books** and **Authors**. The application implements key CRUD operations—Create, Read, and Update—allowing users to manage book and author records. Additionally, JSP pages facilitate user interaction, providing a visually appealing interface.

1. Entity Design

1.1. Entities

- **Book Entity**:
  - Attributes: `id`, `title`, `author` (Many-to-One relationship with Author).
- **Author Entity**:
  - Attributes: `id`, `name`, `books` (One-to-Many relationship with Book).

1.2. JPA Annotations

Entities are annotated with JPA annotations to define relationships:

- `@Entity` for entity classes.
- `@Id` for primary keys.
- `@ManyToOne` and `@OneToMany` for defining relationships.

1.3. Database Mapping

Entities are accurately mapped to the database, ensuring data integrity and proper relationship handling.

```java
@Entity
@Table(name = "authors")
public class Author {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "name", nullable = false)
    private String name;

    @OneToMany(mappedBy = "author", cascade = CascadeType.ALL)
    @JsonManagedReference
    private List<Book> books;
```

```java
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "title", nullable = false)
    private String title;

    @ManyToOne
    @JoinColumn(name = "author_id", nullable = false)
    @JsonBackReference
    private Author author;
```

2. CRUD Functionality

2.1. Implementation

The application allows users to:

- **Create**: Add new Books and Authors.
- **Read**: Display a list of Books and Authors with an option to view details.
- **Update**: Modify existing records for Books and Authors.

```java
    // Create
    public Author save(Author author) {
        return authorRepository.save(author);
    }

    // Read
    public List<Author> findAll() {
        return authorRepository.findAll();
    }

    public Author findById(Long id) {
        Optional<Author> author = authorRepository.findById(id);
        return author.orElse(other:null); // Return null or throw an exception if not found
    }

    // Update
    public Author update(Long id, Author authorDetails) {
        Author author = findById(id);
        if (author != null) {
            author.setName(authorDetails.getName());
            return authorRepository.save(author);
        }
        return null; // Or throw an exception
    }

    // Delete
    public void delete(Long id) {
        authorRepository.deleteById(id);
    }
```

2.2. Custom Queries

A custom query method is implemented to perform an inner join between Books and Authors, enabling efficient data retrieval.

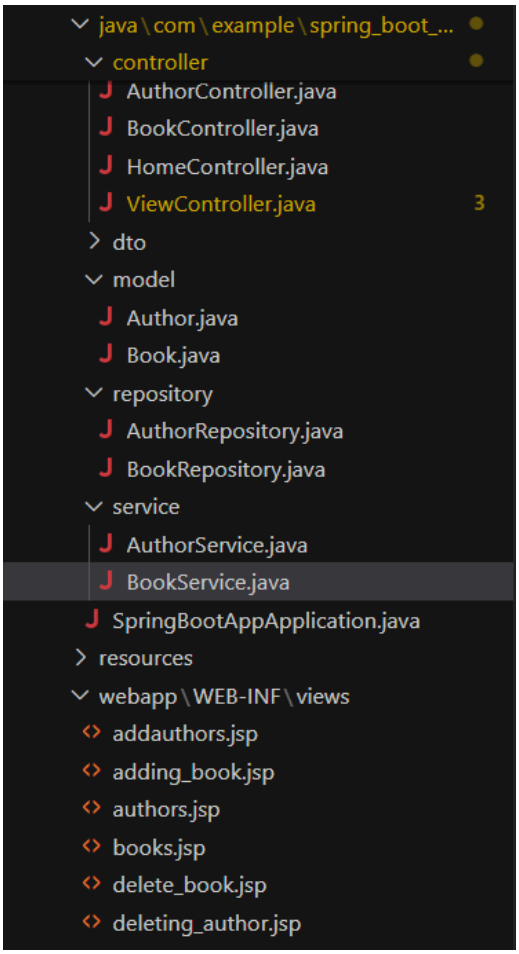
```java
// Read
public List<BookDto> findAll() {
    List<Book> books = bookRepository.findAll();
    return books.stream()
            .map(book -> new BookDto(book.getId(), book.getTitle(), book.getAuthor().getName()))
            .collect(Collectors.toList());
}
```

3. Spring Boot Component Integration

3.1. Layer Organization

The application follows a clear separation of concerns:

- **Repository Layer**: Handles database interactions.
- **Service Layer**: Implements business logic and integrates repository calls.
- **Controller Layer**: Manages HTTP requests and responses, binding data to JSP views.



3.2. HTTP Request Handling

Controller methods effectively route user requests to the appropriate service methods and prepare data for display on JSP pages.

```
<div class="authors-grid">
    <%
        // Initialize an empty string to hold authors' data
        String apiUrl = "http://localhost:8080/api/authors";
        StringBuilder result = new StringBuilder();

        try {
            // Create a URL object
            URL url = new URL(apiUrl);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setRequestMethod("GET");
            conn.setRequestProperty("Accept", "application/json");

            // Read the response
            BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
            String inputLine;
            while ((inputLine = in.readLine()) != null) {
                result.append(inputLine);
            }
            in.close();
```

4. User Interface

4.1. JSP Design

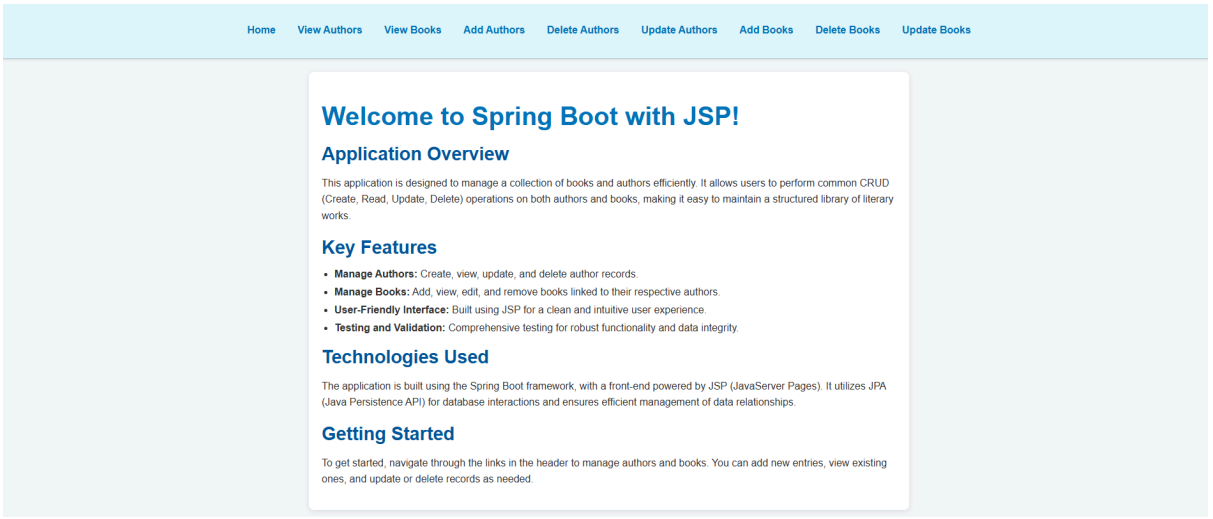
The JSP pages are designed to be user-friendly, featuring:

- Intuitive forms for data entry.
- Clear and organized displays of data.
- CSS styling to enhance visual appeal.

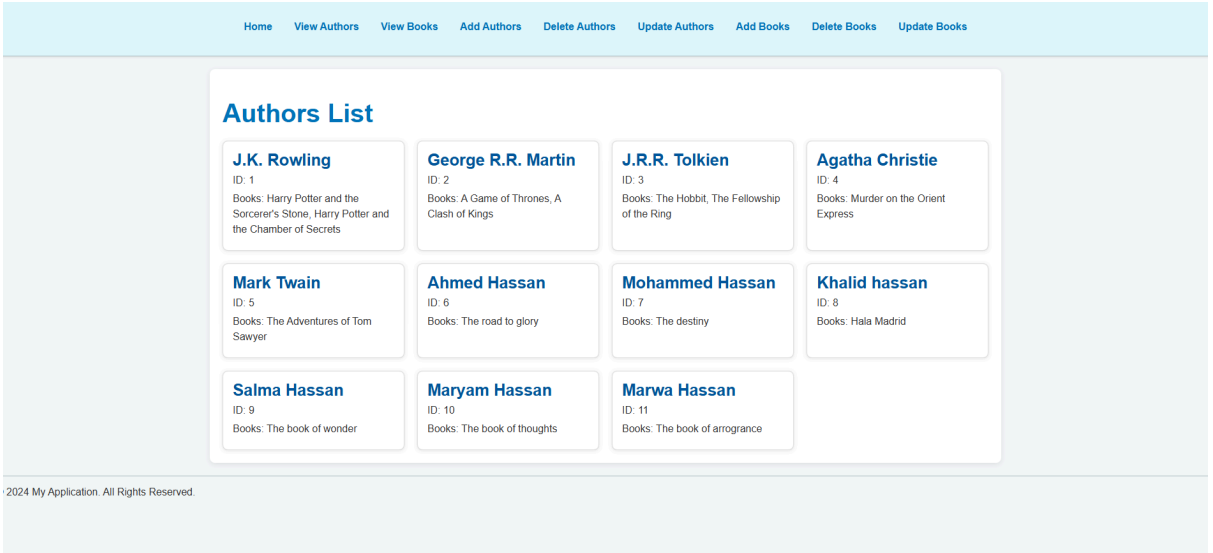
4.2. User Experience

The application is evaluated based on ease of navigation and overall user experience, ensuring that users can interact seamlessly with the system.

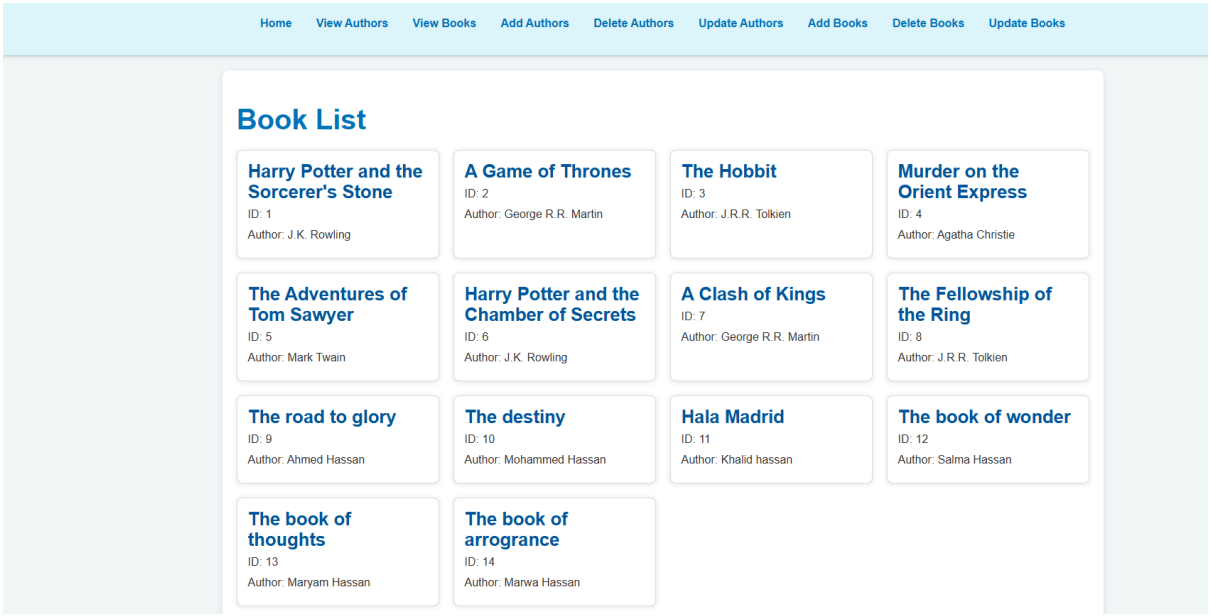
4.2.3 Homepage



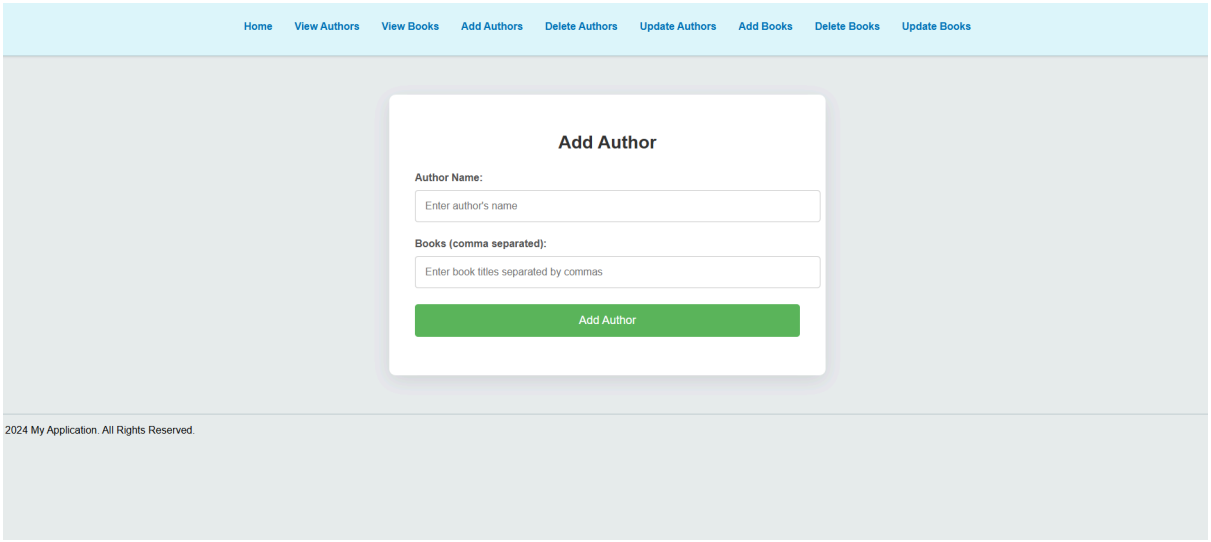
4.2.3 Authors Page



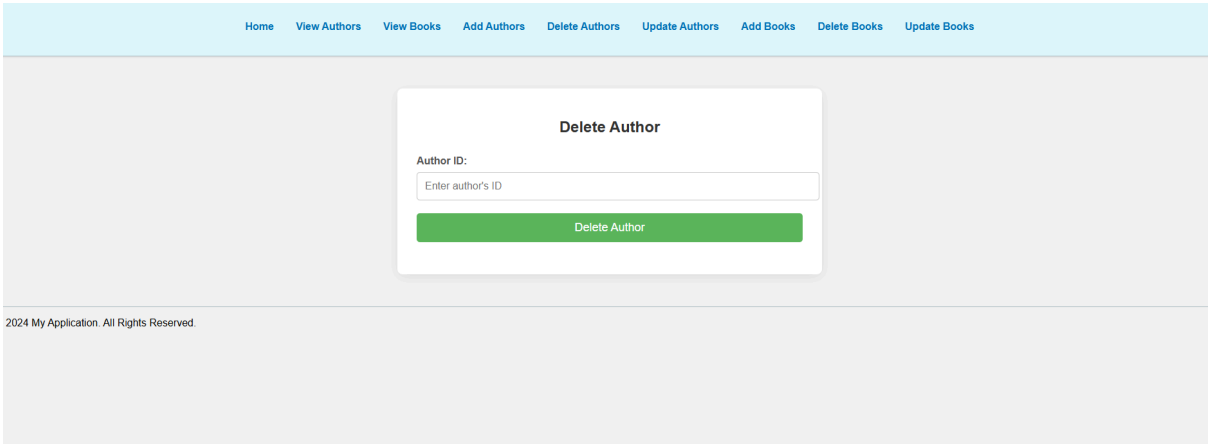
4.2.3 Books Page



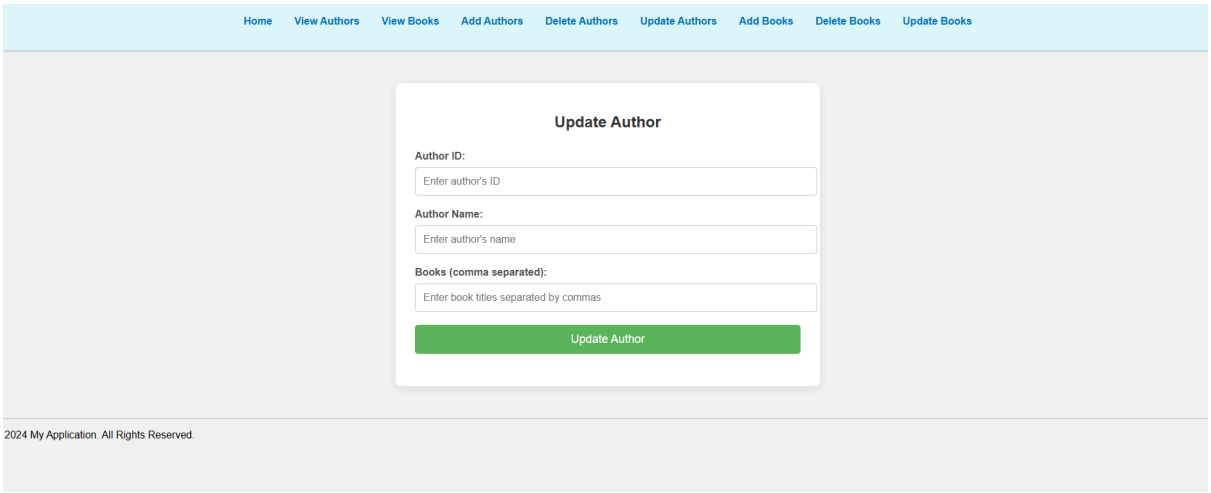
4.2.3 Add Authors Page



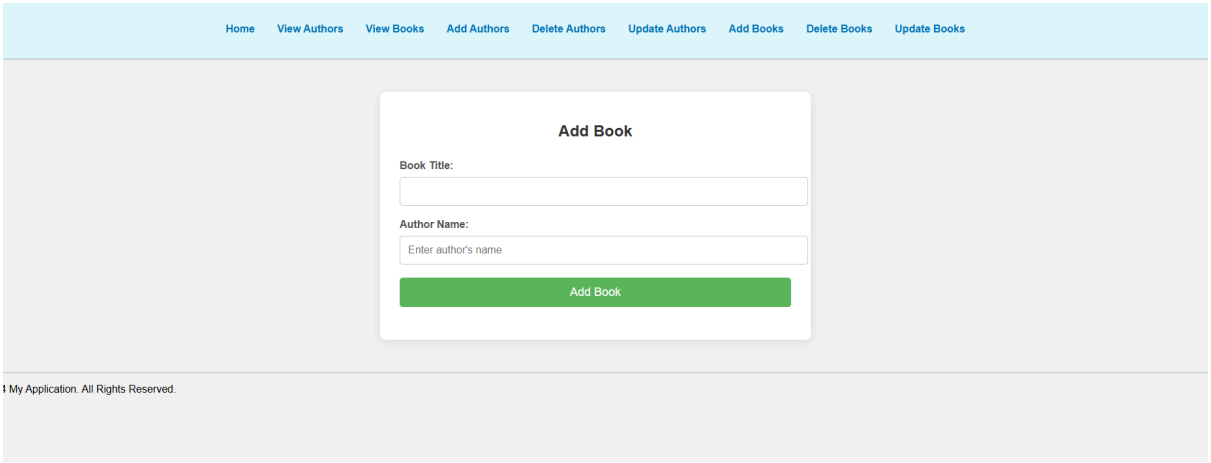
4.2.3 Delete Authors Page



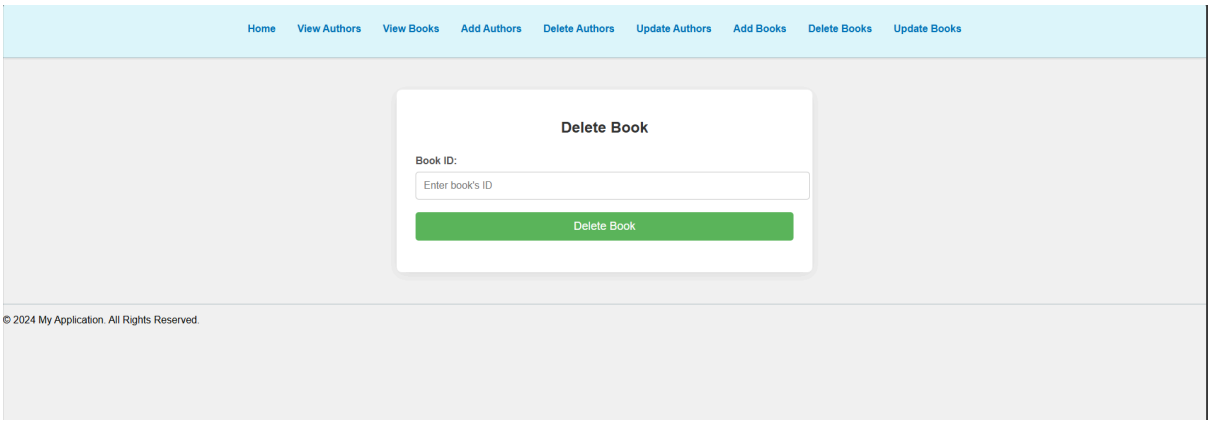
4.2.3 Update Authors Page



4.2.3 Add Books Page



4.2.3 Delete Books Page



4.2.3 Update Books Page