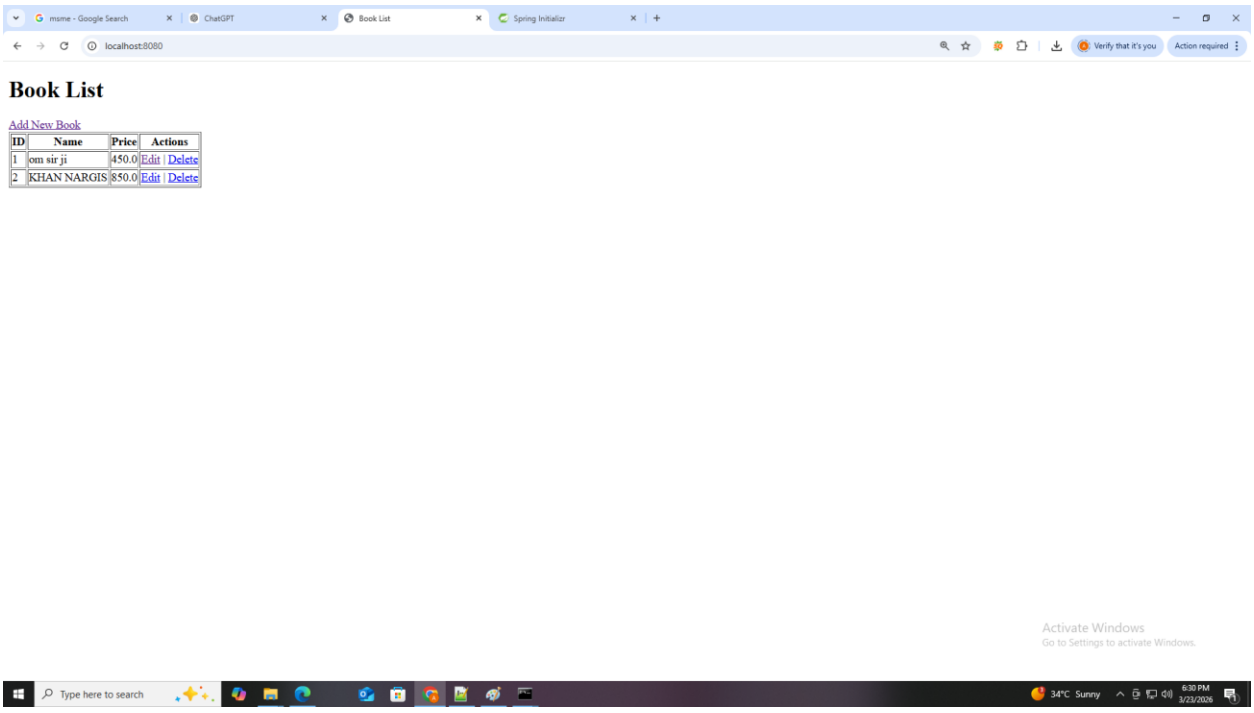



create a **Spring Boot CRUD project** with **MySQL** for a `Book` table, including HTML pages using **Thymeleaf**. Here's a complete setup.

---

# 1. Project Setup

You can use **Spring Initializr** or your IDE (IntelliJ/Eclipse) to create a Maven project with the following dependencies:

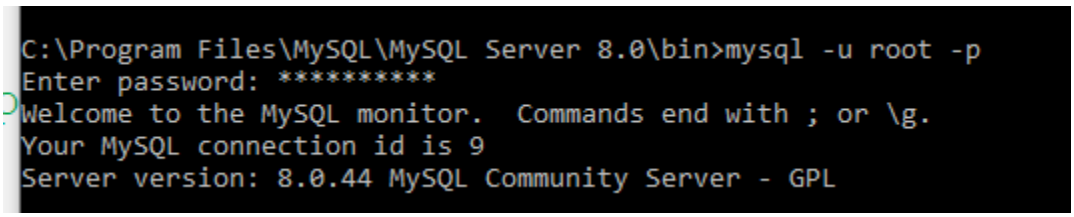
- Spring Web
- Spring Data JPA
- MySQL Driver
- Thymeleaf

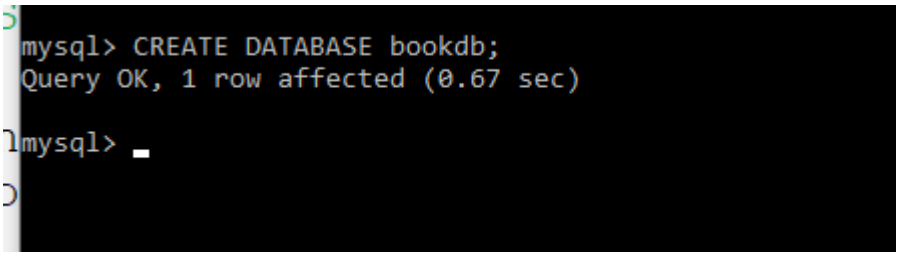
---

# 2. Configure MySQL Database

1. Create a database in MySQL:

```
CREATE DATABASE bookdb;
```




2. Add the following to `application.properties`:

```
# Application name

spring.application.name=person-crud



# MySQL datasource

spring.datasource.url=jdbc:mysql://localhost:3306/bookdb?useSSL=false&serverT
imezone=UTC

spring.datasource.username=root

spring.datasource.password=Rscoe@1989

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver



# JPA / Hibernate

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
```

```
# Server port

server.port=8080
```

Replace `yourpassword` with your MySQL root password.

---

## 3. Create Entity Class

`Book.java`:

```
package com.example.bookcrud.model;

import jakarta.persistence.*;

@Entity
public class Book {

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

    private String name;
    private Double price;

    // Constructors
    public Book() {}

    public Book(String name, Double price) {
        this.name = name;
        this.price = price;
    }

    // Getters & Setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public Double getPrice() { return price; }
    public void setPrice(Double price) { this.price = price; }
}
```

---

# 4. Create Repository

BookRepository.java:

```
package com.example.bookcrud.repository;

import com.example.bookcrud.model.Book;
import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {
}
```

---

# 5. Create Controller

BookController.java:

```
package com.example.bookcrud.controller;

import com.example.bookcrud.model.Book;
import com.example.bookcrud.repository.BookRepository;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

@Controller
public class BookController {

    private final BookRepository bookRepository;

    public BookController(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }

    // List all books
    @GetMapping("/")
    public String home(Model model) {
        model.addAttribute("books", bookRepository.findAll());
        return "index";
    }

    // Show form to add a book
    @GetMapping("/add")
    public String addBookForm(Book book) {
        return "add-book";
    }

    // Save book
    @PostMapping("/add")
    public String addBook(Book book) {
        bookRepository.save(book);
        return "redirect:/";
    }
```

```
    // Show form to update book
    @GetMapping("/edit/{id}")
    public String editBookForm(@PathVariable("id") Long id, Model model) {
        Book book = bookRepository.findById(id).orElseThrow(() -> new
IllegalArgumentException("Invalid book Id:" + id));
        model.addAttribute("book", book);
        return "edit-book";
    }

    // Update book
    @PostMapping("/update/{id}")
    public String updateBook(@PathVariable("id") Long id, Book book) {
        book.setId(id);
        bookRepository.save(book);
        return "redirect:/";
    }

    // Delete book
    @GetMapping("/delete/{id}")
    public String deleteBook(@PathVariable("id") Long id) {
        Book book = bookRepository.findById(id).orElseThrow(() -> new
IllegalArgumentException("Invalid book Id:" + id));
        bookRepository.delete(book);
        return "redirect:/";
    }
}
```

---

# 6. Create HTML Pages (Thymeleaf)

**src/main/resources/templates/index.html:**

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Book List</title>
</head>
<body>
<h1>Book List</h1>
<a href="/add">Add New Book</a>
<table border="1">
    <tr>
        <th>ID</th>
        <th>Name</th>
        <th>Price</th>
        <th>Actions</th>
    </tr>
    <tr th:each="book : ${books}">
        <td th:text="${book.id}"></td>
        <td th:text="${book.name}"></td>
        <td th:text="${book.price}"></td>
        <td>
            <a th:href="@{/edit/{id}(id=${book.id})}">Edit</a> |
            <a th:href="@{/delete/{id}(id=${book.id})}" onclick="return
```

```
confirm('Are you sure?')">Delete</a>
        </td>
    </tr>
</table>
</body>
</html>
```

**src/main/resources/templates/add-book.html:**

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Add Book</title>
</head>
<body>
<h1>Add Book</h1>
<form th:action="@{/add}" th:object="${book}" method="post">
    <p>Name: <input type="text" th:field="*{name}" /></p>
    <p>Price: <input type="number" step="0.01" th:field="*{price}" /></p>
    <p><button type="submit">Save</button></p>
</form>
<a href="/">Back to List</a>
</body>
</html>
```

**src/main/resources/templates/edit-book.html:**

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Edit Book</title>
</head>
<body>
<h1>Edit Book</h1>
<form th:action="@{/update/{id}(id=${book.id})}" th:object="${book}"
method="post">
    <p>Name: <input type="text" th:field="*{name}" /></p>
    <p>Price: <input type="number" step="0.01" th:field="*{price}" /></p>
    <p><button type="submit">Update</button></p>
</form>
<a href="/">Back to List</a>
</body>
</html>
```

---

# 7. Run the Application

- Run `BookCrudApplication.java`:

```
package com.example.bookcrud;

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

@SpringBootApplication
public class BookCrudApplication {
    public static void main(String[] args) {
        SpringApplication.run(BookCrudApplication.class, args);
    }
}
```

- Open browser at `http://localhost:8080/`
- You can **add, edit, and delete books**.

---

# Open Your Project

Open your project in **IntelliJ IDEA**, **Eclipse**, or **VS Code**.

---

# 3. Build the Project

### Using Maven (via CMD)

1. Navigate to your project directory (where `pom.xml` is):

```
cd C:\path\to\your\project
```

2. Run:

```
mvn clean install
```

- This will compile your project and download dependencies.

---

# Run the Spring Boot Application

### Option A: Using IDE

- Open `BookCrudApplication.java`
- Click **Run → Run 'BookCrudApplication'**

## Option B: Using CMD

- Run:

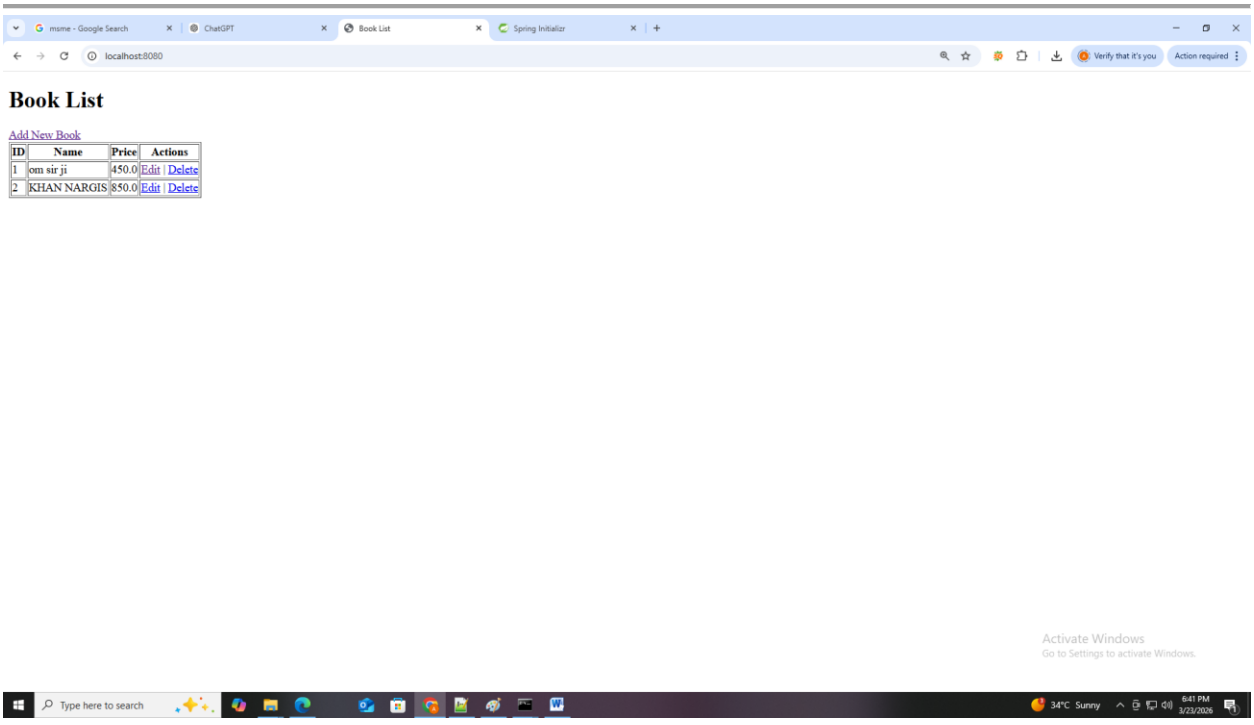
```
mvn spring-boot:run
```

# 5. Access the Application in Browser

- Open your browser and go to:

```
http://localhost:8080/
```

- You should see the **Book List page**.
- You can **Add, Edit, Delete** books.



Right now, your `BookController` is directly talking to the `BookRepository`.

Now By creating a `BookService`, you separate **business logic** from **controller logic**, making your code cleaner, more maintainable, and testable.

Here's how you can do it step by step.

---

# 1. Create the Service Class

**`BookService.java` in `com.example.bookcrud.service`**

```
package com.example.bookcrud.service;

import com.example.bookcrud.model.Book;
import com.example.bookcrud.repository.BookRepository;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class BookService {

    private final BookRepository bookRepository;

    public BookService(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }

    // List all books
    public List<Book> getAllBooks() {
        return bookRepository.findAll();
    }

    // Get book by ID
    public Book getBookById(Long id) {
        return bookRepository.findById(id)
                .orElseThrow(() -> new IllegalArgumentException("Invalid book
Id: " + id));
    }

    // Save new or updated book
    public void saveBook(Book book) {
        bookRepository.save(book);
    }

    // Delete a book
    public void deleteBook(Long id) {
        Book book = getBookById(id);
        bookRepository.delete(book);
```

```
    }
}
```

---

# 2. Update the Controller to Use the Service

### `BookController.java` (refactored)

```
package com.example.bookcrud.controller;

import com.example.bookcrud.model.Book;
import com.example.bookcrud.service.BookService;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

@Controller
public class BookController {

    private final BookService bookService;

    public BookController(BookService bookService) {
        this.bookService = bookService;
    }

    // List all books
    @GetMapping("/")
    public String home(Model model) {
        model.addAttribute("books", bookService.getAllBooks());
        return "index";
    }

    // Show form to add a book
    @GetMapping("/add")
    public String addBookForm(Book book) {
        return "add-book";
    }

    // Save book
    @PostMapping("/add")
    public String addBook(Book book) {
        bookService.saveBook(book);
        return "redirect:/";
    }

    // Show form to update book
    @GetMapping("/edit/{id}")
    public String editBookForm(@PathVariable("id") Long id, Model model) {
        model.addAttribute("book", bookService.getBookById(id));
        return "edit-book";
    }

    // Update book
    @PostMapping("/update/{id}")
    public String updateBook(@PathVariable("id") Long id, Book book) {
```

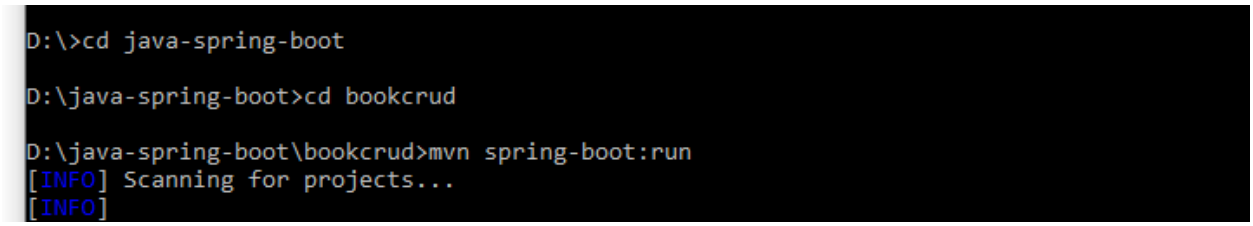
```
        book.setId(id);
        bookService.saveBook(book);
        return "redirect:/";
    }

    // Delete book
    @GetMapping("/delete/{id}")
    public String deleteBook(@PathVariable("id") Long id) {
        bookService.deleteBook(id);
        return "redirect:/";
    }
}
```

---

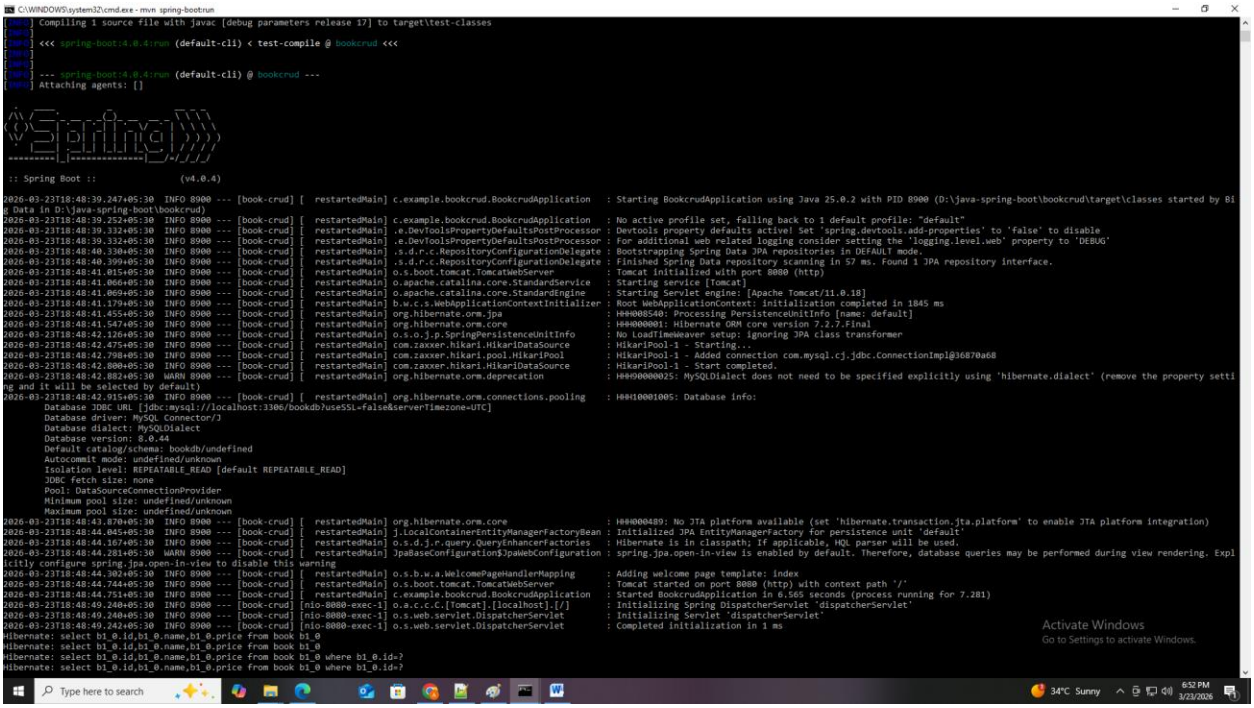
## □ Benefits of Using Service Layer

1. **Separation of concerns**: Controller handles HTTP requests, Service handles business logic.
2. **Easier to test**: You can unit test `BookService` without starting the web server.
3. **Reusability**: If you need to use book logic elsewhere (e.g., REST API), the service can be reused.
4. **Cleaner code**: Controller is simpler and just calls service methods.

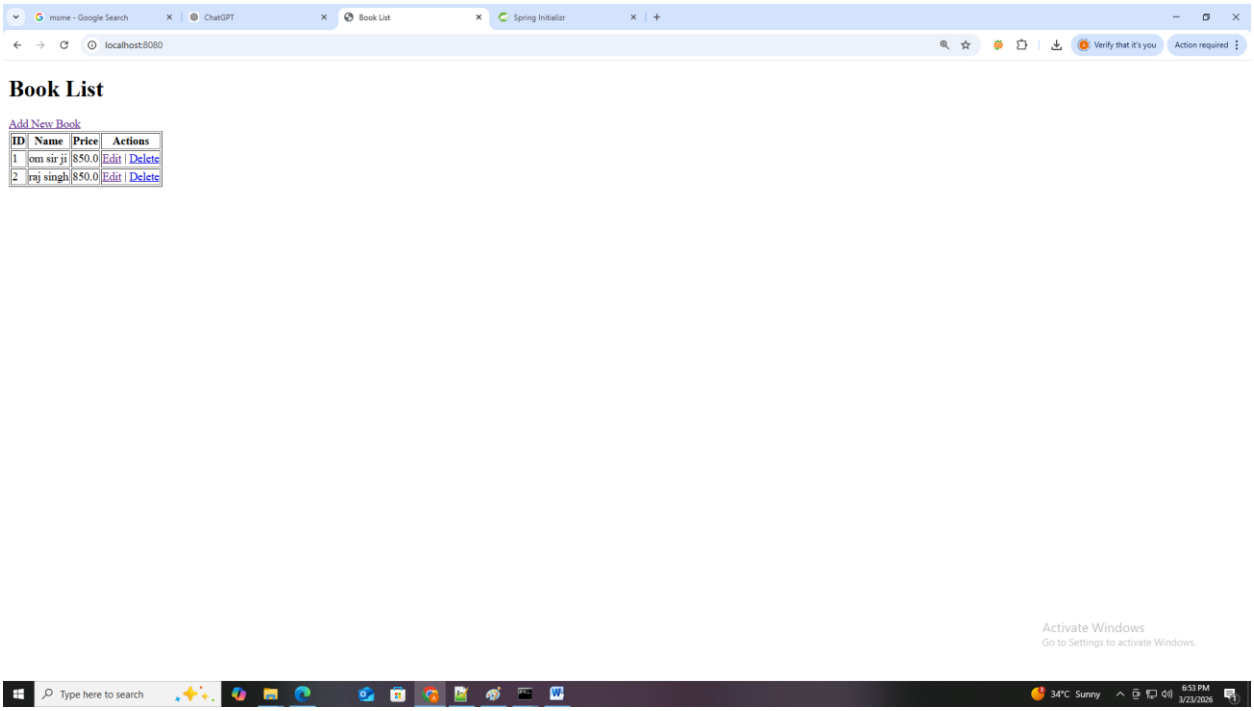
And run it:-




And




And Check Output:-



Let's go **line by line** through the `BookService.java` I provided earlier so you understand exactly what each part does.

---

```
package com.example.bookcrud.service;
```

- Defines the **package** where this class lives.
- Keeps your code organized. Here, the service layer is in `service`.

---

```
import com.example.bookcrud.model.Book;
import com.example.bookcrud.repository.BookRepository;
import org.springframework.stereotype.Service;
import java.util.List;
import java.util.Optional;
```

- **Imports** necessary classes:
  - `Book` → the entity we are managing
  - `BookRepository` → the interface that communicates with the database
  - `@Service` → tells Spring this is a **service component**
  - `List` → to return a list of books
  - `Optional` → helps handle cases where a book might not exist

---

```
@Service
public class BookService {
```

- `@Service` → Spring manages this class as a **bean**, so it can be injected into controllers.
- `public class BookService` → declares the class.

---

```
private final BookRepository bookRepository;
```

- Declares a **dependency** on the repository.
- `final` → ensures this variable is assigned once and never changed.

---

```
public BookService(BookRepository bookRepository) {
    this.bookRepository = bookRepository;
}
```

- **Constructor injection**: Spring automatically provides an instance of `BookRepository` here.
- `this.bookRepository = bookRepository;` → stores the repository in the service for use in methods.

```
// List all books
public List<Book> getAllBooks() {
    return bookRepository.findAll();
}
```

- `getAllBooks()` → method to fetch **all book records** from the database.
- `bookRepository.findAll()` → Spring Data JPA method that returns a **List of all books**.

---

```
// Get book by ID
public Book getBookById(Long id) {
    return bookRepository.findById(id)
            .orElseThrow(() -> new IllegalArgumentException("Invalid book Id:
" + id));
}
```

- `getBookById(Long id)` → fetch a **single book by its ID**.
- `bookRepository.findById(id)` → returns an `Optional<Book>`.
- `.orElseThrow(...)` → if the book is not found, throws an exception with a message.

---

```
// Save new or updated book
public void saveBook(Book book) {
    bookRepository.save(book);
}
```

- `saveBook(Book book)` → saves a new book or updates an existing one.
- `bookRepository.save(book)` → Spring Data JPA **automatically detects** if it's a new record (INSERT) or existing record (UPDATE) based on the ID.

---

```
// Delete a book
public void deleteBook(Long id) {
    Book book = getBookById(id);
    bookRepository.delete(book);
}
```

- `deleteBook(Long id)` → deletes a book.
- First, `getBookById(id)` ensures the book exists.
- `bookRepository.delete(book)` → removes it from the database.

---

## □ Summary of Responsibilities

- **Controller** → handles HTTP requests, passes calls to service
- **Service** → contains all **business logic**, talks to repository

- **Repository** → talks to the **database**

The **service layer** acts as a clean bridge between the web layer and the database.

Let's go **line by line** through the refactored `BookController.java` that uses the `BookService`. I'll explain what each line does and why it's important.

---

```
package com.example.bookcrud.controller;
```

- Defines the **package** for the controller.
- Organizes your classes logically; controllers go into `controller` package.

---

```
import com.example.bookcrud.model.Book;
import com.example.bookcrud.service.BookService;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
```

- **Imports necessary classes**:
  - `Book` → the entity representing a book.
  - `BookService` → service layer to handle business logic.
  - `@Controller` → marks this class as a Spring MVC controller.
  - `Model` → used to pass data from the controller to the view (Thymeleaf templates).
  - `@GetMapping`, `@PostMapping`, `@PathVariable` → annotations to map HTTP requests to methods.

---

```
@Controller
public class BookController {
```

- `@Controller` → tells Spring Boot this class handles **HTTP requests** and returns views.
- `public class BookController` → declares the class.

---

```
private final BookService bookService;
```

- Declares a **dependency** on the service layer.
- `final` → ensures it's assigned only once (via constructor).

```
public BookController(BookService bookService) {
    this.bookService = bookService;
}
```

- **Constructor injection**: Spring automatically provides a `BookService` instance.
- `this.bookService = bookService;` → assigns the injected service to the class variable for later use.

```
// List all books
@GetMapping("/")
public String home(Model model) {
    model.addAttribute("books", bookService.getAllBooks());
    return "index";
}
```

- `@GetMapping("/")` → maps the root URL (`http://localhost:8080/`) to this method.
- `Model model` → allows passing data to the view.
- `model.addAttribute("books", bookService.getAllBooks());` → adds all books to the template under the name `books`.
- `return "index";` → tells Spring to render `index.html` in `templates/`.

```
// Show form to add a book
@GetMapping("/add")
public String addBookForm(Book book) {
    return "add-book";
}
```

- `@GetMapping("/add")` → maps `/add` URL to show the **Add Book form**.
- `Book book` → a new empty `Book` object used by Thymeleaf form binding.
- `return "add-book";` → renders `add-book.html`.

```
// Save book
@PostMapping("/add")
public String addBook(Book book) {
    bookService.saveBook(book);
    return "redirect:/";
}
```

- `@PostMapping("/add")` → handles **form submission** from Add Book page.
- `Book book` → automatically bound from the form data (`name` and `price`).
- `bookService.saveBook(book);` → saves the book to the database.
- `return "redirect:/";` → redirects back to the **book list page**.

```
// Show form to update book
@GetMapping("/edit/{id}")
```

```
public String editBookForm(@PathVariable("id") Long id, Model model) {
    model.addAttribute("book", bookService.getBookById(id));
    return "edit-book";
}
```

- `@GetMapping("/edit/{id}")` → maps URLs like `/edit/1` to this method.
- `@PathVariable("id") Long id` → extracts the `id` from the URL.
- `model.addAttribute("book", bookService.getBookById(id));` → fetches the book by ID and sends it to the template for editing.
- `return "edit-book";` → renders `edit-book.html`.

---

```
// Update book
@PostMapping("/update/{id}")
public String updateBook(@PathVariable("id") Long id, Book book) {
    book.setId(id);
    bookService.saveBook(book);
    return "redirect:/";
}
```

- `@PostMapping("/update/{id}")` → handles the **Edit Book form submission**.
- `book.setId(id);` → ensures the book has the correct ID so JPA updates the existing record instead of inserting a new one.
- `bookService.saveBook(book);` → saves the updated book.
- `return "redirect:/";` → redirects back to the book list page.

---

```
// Delete book
@GetMapping("/delete/{id}")
public String deleteBook(@PathVariable("id") Long id) {
    bookService.deleteBook(id);
    return "redirect:/";
}
```

- `@GetMapping("/delete/{id}")` → maps URLs like `/delete/1`.
- `@PathVariable("id") Long id` → extracts the book ID from the URL.
- `bookService.deleteBook(id);` → deletes the book.
- `return "redirect:/";` → returns to the list page after deletion.

---

## □ Summary of Responsibilities

| Layer | Responsibility in This Controller |
|---|---|
| `@Controller` | Handles HTTP requests and returns views |
| `BookService` | Provides business logic and database operations |
| `Model` | Sends data to Thymeleaf templates |
| `@GetMapping` / `@PostMapping` | Maps URLs to methods for listing, adding, editing, deleting books |
| Redirects | After save/update/delete, user is sent back to `/` (book list) |