

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;
```

```
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 8.0.44 MySQL Community Server - GPL
```

```
mysql> CREATE DATABASE bookdb;
Query OK, 1 row affected (0.67 sec)

mysql> _
```

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&serverTimezone=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:

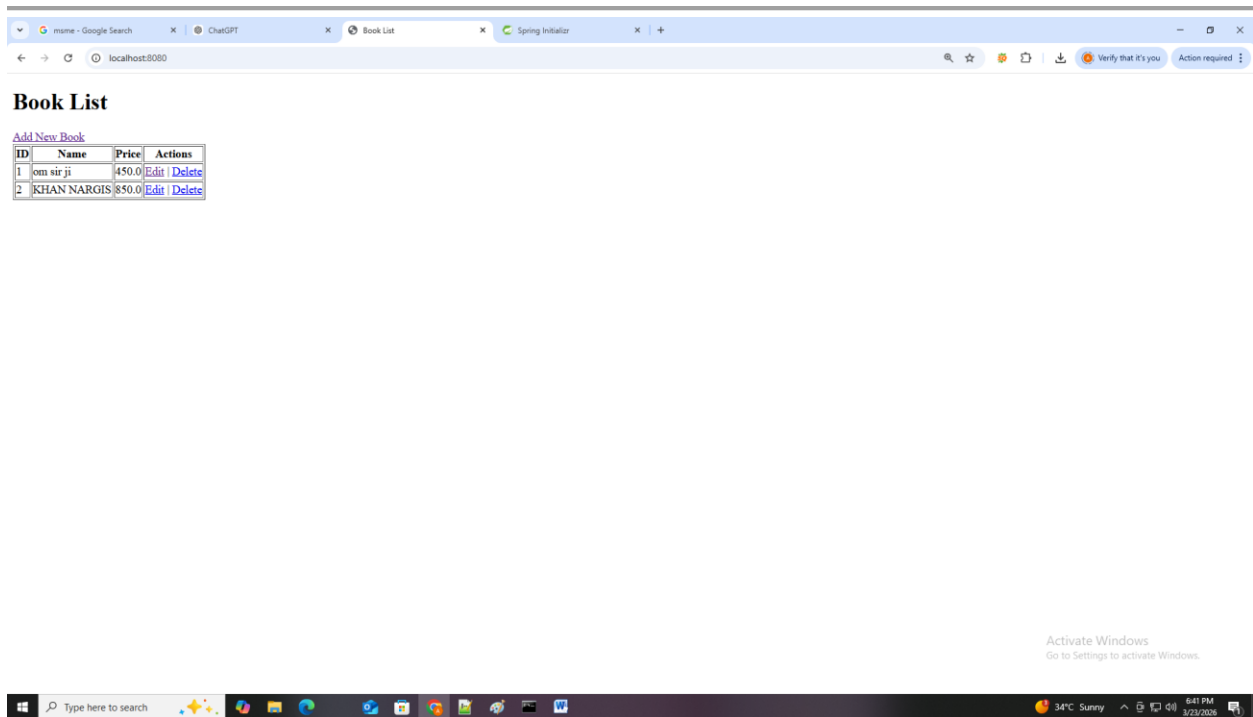
```
mvn spring-boot:run
```

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:-

```
D:\>cd java-spring-boot
D:\java-spring-boot>cd bookcrud
D:\java-spring-boot\bookcrud>mvn spring-boot:run
[INFO] Scanning for projects...
[INFO]
```

And

```

C:\WINDOWS\system32\cmd.exe -m spring-bootrun
[1] Compiling 1 source file with javac [debug parameters release 17] to target\test-classes
[2] <<< spring-boot:4.0.0:run (default-cli) < test-compile @ bookcrud <<<
[3]
[4] --- spring-boot:4.0.0:run (default-cli) @ bookcrud ---
[5] Attaching agents: []

Spring Boot (v4.0.4)

2026-03-23T18:48:39.247405:30 INFO 8900 --- [book-crud] [ restartedMain] c.example.bookcrud.BookcrudApplication : Starting BookcrudApplication using Java 25.0.2 with PID 8900 (D:\java-spring-boot\bookcrud\target\classes started by B
0 Data in D:\java-spring-boot\bookcrud)
2026-03-23T18:48:39.252405:30 INFO 8900 --- [book-crud] [ restartedMain] c.example.bookcrud.BookcrudApplication : No active profile set, falling back to 1 default profile: "default"
2026-03-23T18:48:39.332405:30 INFO 8900 --- [book-crud] [ restartedMain] e.DevToolsPropertyDefaultPostProcessor : DevTools property defaults activated. Set 'spring.devtools.add-properties' to 'false' to disable
2026-03-23T18:48:39.332405:30 INFO 8900 --- [book-crud] [ restartedMain] e.DevToolsPropertyDefaultPostProcessor : For additional web related logging consider setting the 'logging.level.web' property to 'DEBUG'
2026-03-23T18:48:40.330405:30 INFO 8900 --- [book-crud] [ restartedMain] s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2026-03-23T18:48:40.399405:30 INFO 8900 --- [book-crud] [ restartedMain] s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 57 ms. Found 1 JPA repository interface.
2026-03-23T18:48:41.015405:30 INFO 8900 --- [book-crud] [ restartedMain] o.s.boot.tomcat.TomcatWebServer : Tomcat: initialized with port 8080 (http)
2026-03-23T18:48:41.066405:30 INFO 8900 --- [book-crud] [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2026-03-23T18:48:41.069405:30 INFO 8900 --- [book-crud] [ restartedMain] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/11.0.18]
2026-03-23T18:48:41.179405:30 INFO 8900 --- [book-crud] [ restartedMain] b.w.c.a.WebApplicationContextInitializer : Boot WebApplicationContext: initialization completed in 1845 ms
2026-03-23T18:48:41.455405:30 INFO 8900 --- [book-crud] [ restartedMain] org.hibernate.orm.jpa : HHH000540: Processing PersistenceUnitInfo [name: default]
2026-03-23T18:48:41.547405:30 INFO 8900 --- [book-crud] [ restartedMain] org.hibernate.orm.core : HHH000001: Hibernate ORM core version 7.2.7.Final
2026-03-23T18:48:42.326405:30 INFO 8900 --- [book-crud] [ restartedMain] o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup; ignoring JPA class transformer
2026-03-23T18:48:42.475405:30 INFO 8900 --- [book-crud] [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2026-03-23T18:48:42.790405:30 INFO 8900 --- [book-crud] [ restartedMain] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Added connection com.mysql.cj.jdbc.ConnectionImpl@36870a68
2026-03-23T18:48:43.800405:30 INFO 8900 --- [book-crud] [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2026-03-23T18:48:42.882405:30 WARN 8900 --- [book-crud] [ restartedMain] org.hibernate.orm.deprecation : HHH0000025: MySQLDialect does not need to be specified explicitly using 'hibernate.dialect' (remove the property setti
ng and it will be selected by default)
2026-03-23T18:48:42.915405:30 INFO 8900 --- [book-crud] [ restartedMain] org.hibernate.orm.connections.pooling : HHH0001005: Database info:
Database JDBC URL [jdbc:mysql://localhost:3306/bookdb?useSSL=false&serverTimezone=UTC]
Database driver: MySQL Connector/J
Database dialect: MySQLDialect
Database version: 8.0.44
Default catalog/schema: bookdb/undefined
Autocommit mode: undefined/unknown
Isolation level: REPEATABLE_READ [default REPEATABLE_READ]
JDBC fetch size: none
Pool: DataSourceConnectionProvider
Minimum pool size: undefined/unknown
Maximum pool size: undefined/unknown
2026-03-23T18:48:43.870405:30 INFO 8900 --- [book-crud] [ restartedMain] org.hibernate.orm.core : HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform integration)
2026-03-23T18:48:44.065405:30 INFO 8900 --- [book-crud] [ restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2026-03-23T18:48:44.107405:30 INFO 8900 --- [book-crud] [ restartedMain] o.s.d.j.o.query.QueryHintsConfigurer : Hibernate is in classpath; if applicable, HQS parser will be used.
2026-03-23T18:48:44.281405:30 WARN 8900 --- [book-crud] [ restartedMain] jpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Explic
itly configure spring.jpa.open-in-view to disable this warning
2026-03-23T18:48:44.300405:30 INFO 8900 --- [book-crud] [ restartedMain] o.s.b.w.a.WelcomePageHandlerMapping : Adding welcome page template: index
2026-03-23T18:48:44.744405:30 INFO 8900 --- [book-crud] [ restartedMain] o.s.boot.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path '/'
2026-03-23T18:48:44.771405:30 INFO 8900 --- [book-crud] [ restartedMain] c.example.bookcrud.BookcrudApplication : Started BookcrudApplication in 8.568 seconds (process running for 7.281)
2026-03-23T18:48:49.240405:30 INFO 8900 --- [nio-8080-exec-1] o.a.c.c.localhost.[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2026-03-23T18:48:49.240405:30 INFO 8900 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2026-03-23T18:48:49.242405:30 INFO 8900 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms

```

And Check Output:-

The screenshot shows a web browser window with the following content:

- Browser tabs: 'Google Search', 'ChatGPT', 'Book List', 'Spring Initializer'.
- Address bar: 'localhost:8080'.
- Page title: 'Book List'.
- Header: 'Add New Book'.
- Table:

ID	Name	Price	Actions
1	om sir ji	850.0	Edit Delete
2	raj singh	850.0	Edit Delete
- System tray: Shows '34°C Sunny' and the time '6:53 PM 3/23/2026'.

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)