Online Bookstore

Low Level Design Doc

Work Sample Simulation – Robin Singh Rana

27 Feb, 2023

Problem Statement

We need to design an online book store to facilitate the user for the requirements as defined below. This facilitates the user to manage and search the books online and maintain a catalog of different books.

Requirements

There are the below requirements from a user point of view. The user must be able to:

1. Add a book into the book store
2. Delete a book from book store
3. Get the list of all the books
4. Search the book store for book(s)
    a. by book id
    b. by author name
    c. by book category

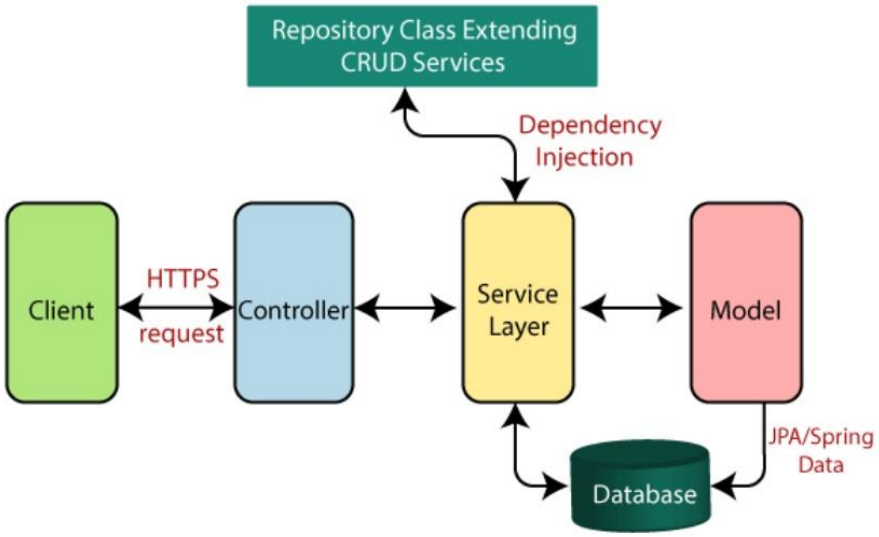
Scope of the document

This document discusses the low-level design approach of a system to serve the above-mentioned requirements.

Design and Implementation

For our design, we have used **Spring MVC framework** along with Spring boot for quick development. As the name suggests, this framework follows the Model-View-Controller design pattern. We also split our systems into different logic layers following **Controller-Service-Repository pattern**:

- Controllers
- Services
- Repository
- Models



Controllers: The classes marked with @Controller are responsible for processing incoming REST API requests, compute the result using business logic, prepare a model and render the view to the user based on prepared model. For our system, we have created three controllers based on their serving streams:

- HomePageController: for rendering homepage
- SearchController: for serving searching the books requests
- BookStoreController: for serving the request for managing the books in the book store.

```java
@Controller
public class HomePageController {

    @RequestMapping(value = "/")
    public String home() {

        return "homePage";
    }
}

@Controller
@RequestMapping("/search")
public class SearchController {}

@Controller
@RequestMapping("/book")
public class BookStoreController {}
```

Services: The classes marked with @Service annotation are responsible for applying the business logic on the requested data fetched from database via repository layer. It serves as a bridge between a controller and the repository layer. The controllers mentioned above make use of service classes. For our use-case we've created two main services one for managing the books in the database i.e. BookService and the other for computing the search results i.e. BookSearchService.

Repository: The classes marked with @Repository annotation are responsible for providing CRUD operations on database tables. Here we make use of Spring Data JPA that provides basic CRUD operations.

```java
@Repository
public interface BookRepository extends CrudRepository<Book, Integer>{}
```

Models: These are the classes that store information about the main objects in the system for example: book object in our system. These are marked with @Entity so that their instances are available to JPA. This will make it easier to store and retrieve instances from the persistent data store when needed.

```java
@jakarta.persistence.Entity
public class Book implements Serializable{

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int bookId;
    private String bookName;
    private String authorName;
    private String category;

}
```

API Structures:

Since we are using view resolution as part of MVC architecture, so response of all APIs will be rendered in a view and will be presented to the user. The main APIs of our system with request structure are:

- **Add a book:**

```
Payload:
{
  "bookName":"Core Java",
  "age":"Robin Singh Rana",
  "category":"Technical"
}

@RequestMapping(value = "/addBook", method = RequestMethod.POST)
public String addBook(@RequestParam(name = "bookName") String bookName,
                      @RequestParam(name = "authorName") String authorName,
                      @RequestParam(name = "category") String category) {}
```

- **Delete a book:**

```
Payload:
{
  "bookId":501
}

@RequestMapping(value = "/deleteBook", method = RequestMethod.GET)
public String deleteBook(@RequestParam(name = "bookId") String bookId) {}
```

- **Get all books:**

```
@RequestMapping(value = "/getAllBooks")
public String getAllBooks(Model model) {}
```

- **Search a book by book id:**

```
Payload:
{
  "bookId":501
}

@RequestMapping(value = "/searchByBookId", method = RequestMethod.GET)
public String searchByBookId(@RequestParam(name = "bookId") String bookId, Model model) {}
```

- **Search books by book category:**

```
Payload:
{
  "bookCategory":"Technical"
}

@RequestMapping(value = "/searchByCategory", method = RequestMethod.GET)
public String searchByCategory(@RequestParam(name = "bookCategory") String bookCategory) {}
```

- **Search books by author name:**

```
Payload:
{
  "authorName":"Robin Singh Rana"
}

@RequestMapping(value = "/searchByAuthorName",  method = RequestMethod.GET)
public String searchByAuthorName(@RequestParam(name = "authorName") String authorName) {}
```

Tech Stack

Backend: Java, Spring MVC, Spring Boot, Maven as build tool

Frontend: HTML, CSS, Thymeleaf as template engine, DataTables

Database: H2 In memory DB with Spring JPA