**Online Bookstore Inventory Management**

**🎯 Objective:**

Build a simplified backend service for managing books in an **online bookstore**, applying Spring Boot’s **Dependency Injection** (DI) features for better modularity and testability.

**📂 Scenario Overview:**

You’ve joined a team building an online bookstore platform. Your task is to build a microservice that manages the **inventory** of books. This involves creating, updating, and retrieving book information.

**Core Requirements:**

**📦 Functional Features:**

1. Add a new book to inventory
2. Get all books
3. Search for a book by ID

**Data Model:**

public class Book {

private Long id;

private String title;

private String author;

private Double price;

}

**Technical Requirements:**

**🧩 Components:**

| **Layer** | **Responsibility** |
| --- | --- |
| BookService | Business logic, injected into controller |
| BookRepository | Simulated data store, injected into service |
| BookController | REST controller for API endpoints |

**Spring Concepts to Apply:**

* @Component, @Service, @Repository
* @Autowired (field, constructor)
* Constructor-based DI
* Spring Boot application class with @SpringBootApplication

**Task Breakdown:**

**✅ 1. Create a Book model class**

public class Book {

private Long id;

private String title;

private String author;

private Double price;

// Constructors, Getters, Setters

}

✅ 2. Create a BookRepository with in-memory store

@Repository

public class BookRepository {

private Map<Long, Book> bookMap = new HashMap<>();

public void save(Book book) {

bookMap.put(book.getId(), book);

}

public Collection<Book> findAll() {

return bookMap.values();

}

public Optional<Book> findById(Long id) {

return Optional.ofNullable(bookMap.get(id));

}

}

✅ 3. Create a BookService and inject the repository

@Service

public class BookService {

private final BookRepository bookRepository;

// Constructor-based dependency injection

@Autowired

public BookService(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(Book book) {

bookRepository.save(book);

}

public Collection<Book> getAllBooks() {

return bookRepository.findAll();

}

public Book getBookById(Long id) {

return bookRepository.findById(id)

.orElseThrow(() -> new RuntimeException("Book not found"));

}

}

✅ 4. Create a BookController and inject the service

@RestController

@RequestMapping("/api/books")

public class BookController {

private final BookService bookService;

// Constructor injection

@Autowired

public BookController(BookService bookService) {

this.bookService = bookService;

}

@PostMapping

public ResponseEntity<String> addBook(@RequestBody Book book) {

bookService.addBook(book);

return ResponseEntity.ok("Book added");

}

@GetMapping

public Collection<Book> getAllBooks() {

return bookService.getAllBooks();

}

@GetMapping("/{id}")

public Book getBookById(@PathVariable Long id) {

return bookService.getBookById(id);

}

}

✅ 5. Main Application Class

@SpringBootApplication

public class BookstoreApplication {

public static void main(String[] args) {

SpringApplication.run(BookstoreApplication.class, args);

}

}

**🔁 Testing Your DI:**

* Use Postman or cURL to test endpoints.
* Try switching from constructor to field injection.
* Add logging to BookService constructor to see when Spring injects the dependency.

**🧠 Learning Goals:**

* Understand how Spring Boot injects dependencies behind the scenes
* Learn difference between constructor, field, and setter injection
* Practice proper layering with separation of concerns