**Exercise 14: Online Bookstore - Integration Testing for REST Services**

Business Scenario:

Write integration tests for your bookstore's RESTful services.

**Spring Test Setup:**

To set up Spring Test for integration testing, we use the @SpringBootTest annotation. This annotation loads the full application context and provides a real environment for testing the components as they would work in a production setting.

* **Key Point:** Use @SpringBootTest to initialize the application context and enable integration testing with Spring.

**MockMvc Integration:**

MockMvc is used to perform end-to-end testing of the REST endpoints without starting the actual server. It allows you to simulate HTTP requests and verify responses as if interacting with a real RESTful service.

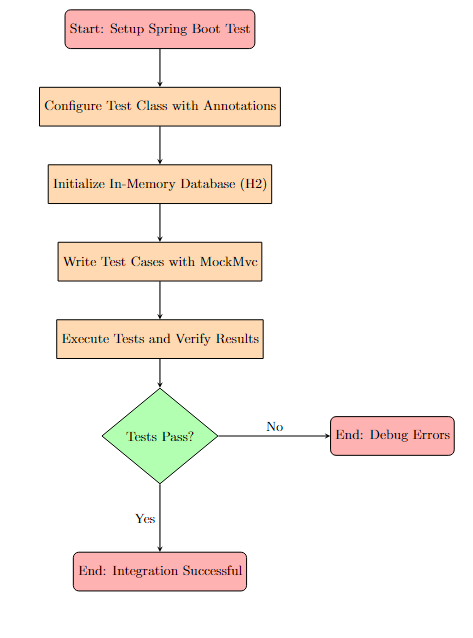
* **Key Point:** Inject MockMvc and use it to perform HTTP requests (e.g., GET, POST, PUT, DELETE) and verify responses.

**Database Integration:**

For integration testing, include a database using an in-memory database like H2. This ensures that your tests involve actual database interactions, making them more robust.

* **Key Point:** Configure an in-memory H2 database in application-test.properties file for testing.

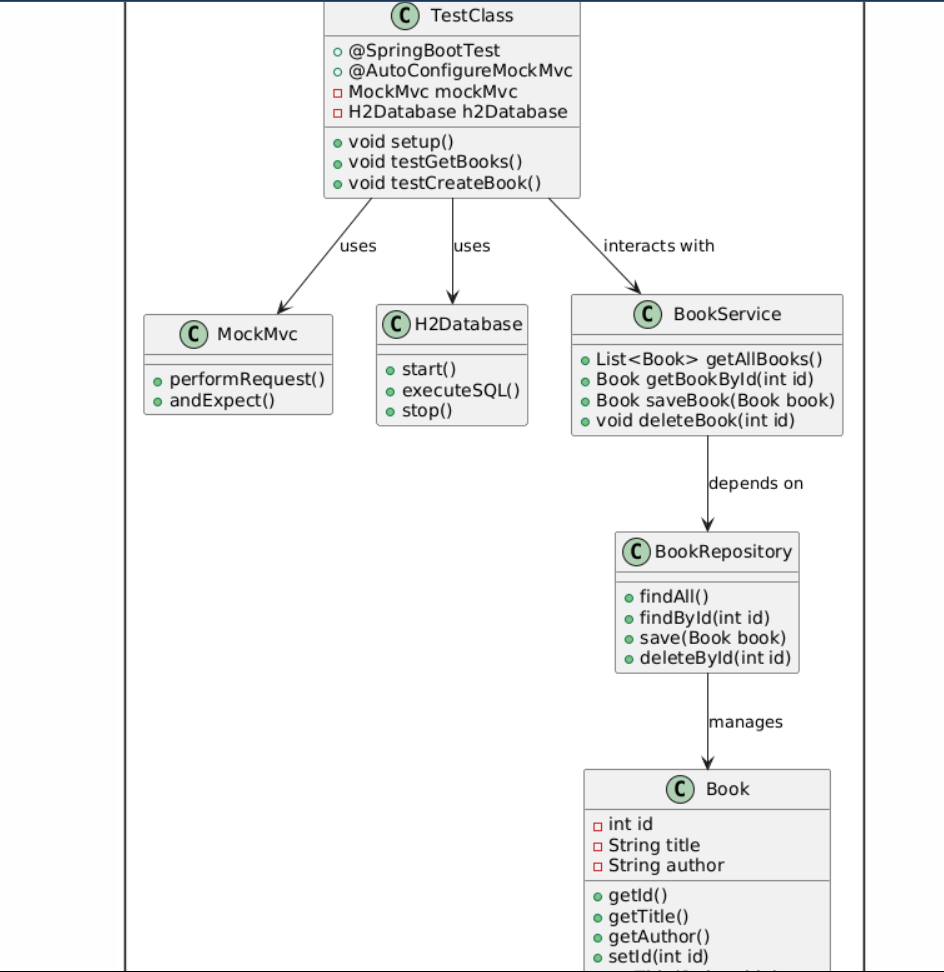
**FLOWCHART :**



**Explanation:**

1. **Start: Setup Spring Boot Test** - This is the starting point where the Spring Boot test environment is initialized.
2. **Configure Test Class with Annotations** - Annotate the test class with necessary annotations like @SpringBootTest, @AutoConfigureMockMvc, and others needed for integration testing.
3. **Initialize In-Memory Database (H2)** - Set up the H2 database for the test environment, ensuring that the database schema is initialized correctly.
4. **Write Test Cases with MockMvc** - Write the test cases using MockMvc to simulate HTTP requests and verify the responses.
5. **Execute Tests and Verify Results** - Run the tests, and ensure the results match the expected outcomes.
6. **Decision: Tests Pass?** - A decision point to check if the tests have passed.
   * **Yes:** If the tests pass, the integration is successful.
   * **No:** If the tests fail, debug and fix errors

**CLASS DIAGRAM :**



**Explanation:**

1. **TestClass:**
   * This is the primary test class where the integration tests are defined.
   * Annotated with @SpringBootTest to load the full application context.
   * Uses @AutoConfigureMockMvc to set up MockMvc for testing the web layer.
   * Contains methods like setup(), testGetBooks(), and testCreateBook() to handle the initialization and execution of test cases.
2. **MockMvc:**
   * This class is used for simulating HTTP requests and verifying responses.
   * Methods like performRequest() and andExpect() are used to perform and validate mock requests in the tests.
3. **H2Database:**
   * Represents the in-memory H2 database used in integration testing.
   * Methods like start(), executeSQL(), and stop() manage the database lifecycle during the tests.
4. **BookService:**
   * Represents the service layer of the application.
   * Methods like getAllBooks(), getBookById(int id), saveBook(Book book), and deleteBook(int id) handle business logic related to the Book entity.
5. **BookRepository:**
   * Represents the data access layer, typically extending JpaRepository in Spring Data JPA.
   * Methods like findAll(), findById(int id), save(Book book), and deleteById(int id) interact with the database to manage Book entities.
6. **Book:**
   * Represents the entity class with attributes id, title, and author.
   * Contains getter and setter methods for these attributes.