**SpringBoot projects**

Contents

[ **UsersServiceSpringBoot** 1](#_Toc165480442)

[ **spring-boot-testing-reddit-clone** 5](#_Toc165480443)

* **UsersServiceSpringBoot**

IDE: IntelliJ IDEA

unit testing in SpringBoot Application

* Dependencies:

Spring Web, **Spring Security** (for **authentication**, to encrypt password), Spring Data **JPA** (H2 database)

ModelMapper (auto mapping), Spring Validation, mockito, junit, maven, Spring Boot 2.7.1, packaging jar.

Can start your project using Spring initializr: <https://start.spring.io/>

* **Steps:**

1. Main application, @SpringBootApplication, UsersServiceSpringBootApplication.java

Run main(), start server Tomcat port 8888.

Its test is: UsersServiceSpringBootApplicationTest.java

Controller, HTTP endpoints, @RestController, UsersController.java

1. Test with postman:

Post: <http://localhost:8080/users>

request:

{

    "firstName":"Zarbio",

    "lastName": "Ordoñez",

    "email": "test@test.com",

    "password": "12345678",

    "repeatPassword": "12345678"

}

response:

{

    "userId": "f5428a06-f80e-467e-a894-55d98bc7d5b5",

    "firstName": "Zarbio",

    "lastName": "Ordoñez",

    "email": "test@test.com"

}

1. Implement tests: Unit tests and Integration tests.

* **Unit Test**:

Test only one layer, mock the rest.

e.g.: for web service only the **Web Layer**

A screen shot of a computer

Description automatically generated

@Restontroller: methods, HTTP requests

@Service: business logic (sorting, filtering)

@Repository: data access objects for database.

In this project, test **web layer**, **mock** the rest layers.

Sprint Framework will create Sprint application **context** for web layer only.

Test REST Controllers: UsersControllersWebLayerTest.java: tests for UsersController.java

Unit Test for getUsers() method: involve mock Service, don’t include @Repository.

UsersControllerWebLayerTest.java

* **Integration Test**:

Involve all layers, no mocking any layer.

e.g.: **Web Layer, Service Layer, Data Layer**

A screenshot of a computer

Description automatically generated

In this project, test all layers (Web Layer, Service Layer, Data Layer), UsersController.java, UsersService.java, UsersRepository.java

UsersControllerIntegrationTest.java. Integration Test

**@SpringBootTest**: create application context run similarly as a production but by default does not start a web server.

Can configure @SpringBootTest to load a real web environment with embedded server running on specific port number.

application.properties

Authentication and Authorization with **JWT** access token

JWT (JSON Web Token) is used in API authentication and authorization, mostly to retrieve a list of objects.

**JWT access token** in authorization **header**

methods to test:

post: createUser(), create user in in-memory database (h2)

get: getUsers(), get list of users

* **Testing Data Layer only for an Entity (JPAN Entities):**

Test Data Layer of Spring Boot Application

UserEntityIntegrationTest.java, UserEntity.java

Integration tests for data layer separately from other layers

JPA Entities

@DataJpaTest: Application Context with JPA-related components only.

By default, test method is Transactional and will rollback when completes.

By default, in-memory database is used.

com.appsdeveloperblog.tutorials.junit.io

UserEntity.java

UserRepository.java

UserEntityIntegrationTest.java

* **Testing JPA Repository:**

test code that was written by us.

not to write test methods to test query methods, because if the method is incorrect, then your application will fail to start up, optionally write test for the query methods with large names.

Write test for **JPQL** Query

JPQL (Java Persistence Query Language)A screenshot of a computer

Description automatically generated

The query method name can be more complex:

A screen shot of a computer

Description automatically generated

In that case, for complex names is useful use JPQL queries, so write tests for JPQL queries:

A screenshot of a computer

Description automatically generated

UsersRepositoryTest.java, UsersRepository.java

refs:

Testing Java with JUnit 5 & Mockito

<https://tcsglobal.udemy.com/course/testing-java-code-with-junit-5-and-mockito/learn/lecture/31481676#overview>

Section 10: Spring Boot - Testign REST Controllers

* **spring-boot-testing-reddit-clone**

Spring Boot Testing Tutorial - Crash Course

<https://www.youtube.com/watch?v=aPoJPESMJBk>

* Architecture

User -> Angular <-> Spring Boot -> MySQL

Controller Layer <-> Service Layer <-> Persistence Layer

A diagram of a service layer

Description automatically generated

* Reddit Clone Application

Reddit clone with Spring boot and Angular

<https://programmingtechie.com/2020/05/14/building-a-reddit-clone-with-spring-boot-and-angular/>

See explication of development

* Backend:

Java 8, Spring Boot, **Spring Security** with JPA Authentication, **Spring Data JPA** with MySQL, Spring MVC, Token Based **Authentication** in the form of JSON Web Tokens (**JWT**).

repository: <https://github.com/SaiUpadhyayula/spring-reddit-clone>

* Frontend:

Angular 9, Bootstrap 4

repository: <https://github.com/SaiUpadhyayula/angular-reddit-clone>

* Tests

<https://github.com/SaiUpadhyayula/spring-boot-testing-reddit-clone>

JUnit 5

1. Unit Testing using Junit 5 and Mockito: **Unit** **Test for Service Layer**

<https://programmingtechie.com/2020/10/16/spring-boot-testing-tutorial-unit-testing-with-junit-5-and-mockito/>

**Unit Testing is a practice in the software development process, where you test the functionality of a component (e.g.: java class) in isolation, without depending on any external dependencies. We cannot use any Spring features.**

**A rule of thumb to remember when testing our code, is to make sure that the test we wrote actually fails when the behavior of the code changes, that is the main reason we are writing tests, to get the feedback immediately when we unintentionally changed the behavior of the method.**

CommentController.java (Controller Layer, class) -> CommentService.java (Service Layer, class) -> CommentRepository.java (Persistence Layer, interface)

CommentService.java -> CommentServiceTest.java

PostService.java -> PostServiceTest.java

JUnit 5 built-in Assertions

AssertJ library (is more readable, use Fluent API)

* We can replace the assertions methods of JUnit 5 by the AssertJ library (is more readable):

assertThat(commentService.containsSwearWords("This is a comment")).isFalse();

assertThatThrownBy(() -> {

commentService.containsSwearWords("This is a shitty comment");

}).isInstanceOf(SpringRedditException.class)

.hasMessage("Comments contains unacceptable language");

* @Transactional(readOnly = true)

used in Spring Data JPA or Spring JDBC, simplifies transaction management in data access operations, ensuring consistency and atomicity of database operations.

1. **Integration Testing** the DB Layer using Test **Containers**: Test for **Persistence Layer**

<https://programmingtechie.com/2020/10/21/spring-boot-testing-tutorial-database-testing-with-test-containers/>

Test Data Access Layer with Test Containers, test the interaction between Persistence Layer and Database

* Testing the Database layer using an embedded database:

Test using **h2 in-memory database**:

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<version>1.4.200</version>

<scope>test</scope>

</dependency>

Configuration: application-test.properties

PostRepository.java

PostRepositoryTestEmbedded.java

@ActiveProfiles("test") // get application-test.properties for h2 database

Test the database logic:

@Sql("classpath:test-data.sql")

* Testing Database Layer using **TestContainers**:

Here I use MySQL (like production database) test container instead of H2 in-memory database.

TestContainers: Java library for JUnit tests using Docker container.

Requeriments:

1. Install Docker Engine.

2. Add MySQL test container dependency in Java project. It allow to run MySql database inside a container:

<dependency>

<groupId>org.testcontainers</groupId>

<artifactId>mysql</artifactId>

<version>1.14.3</version>

<scope>test</scope>

</dependency>

MySQL database name is: spring-reddit-test-db

<https://testcontainers.com/>

<https://java.testcontainers.org/test_framework_integration/manual_lifecycle_control/>

BaseTest.java: MySQL test container

PostRepositoryTest.java

UserRepositoryTest.java

1. Testing REST APIs using Spring MockMvc: **Test for Controller Layer**

<https://programmingtechie.com/2020/10/23/spring-boot-testing-tutorial-testing-rest-apis-using-mockmvc/>

@WebMvcTest: create the Spring Context with only beans which are related to the Spring MVC components like @Controller, @RestController, @AutoconfigureWebMvc, etc.

test for web/presentation layer (@WebMvcTest)

PostController.java, PostControllerTest.java

PostService.java, getAllPosts()