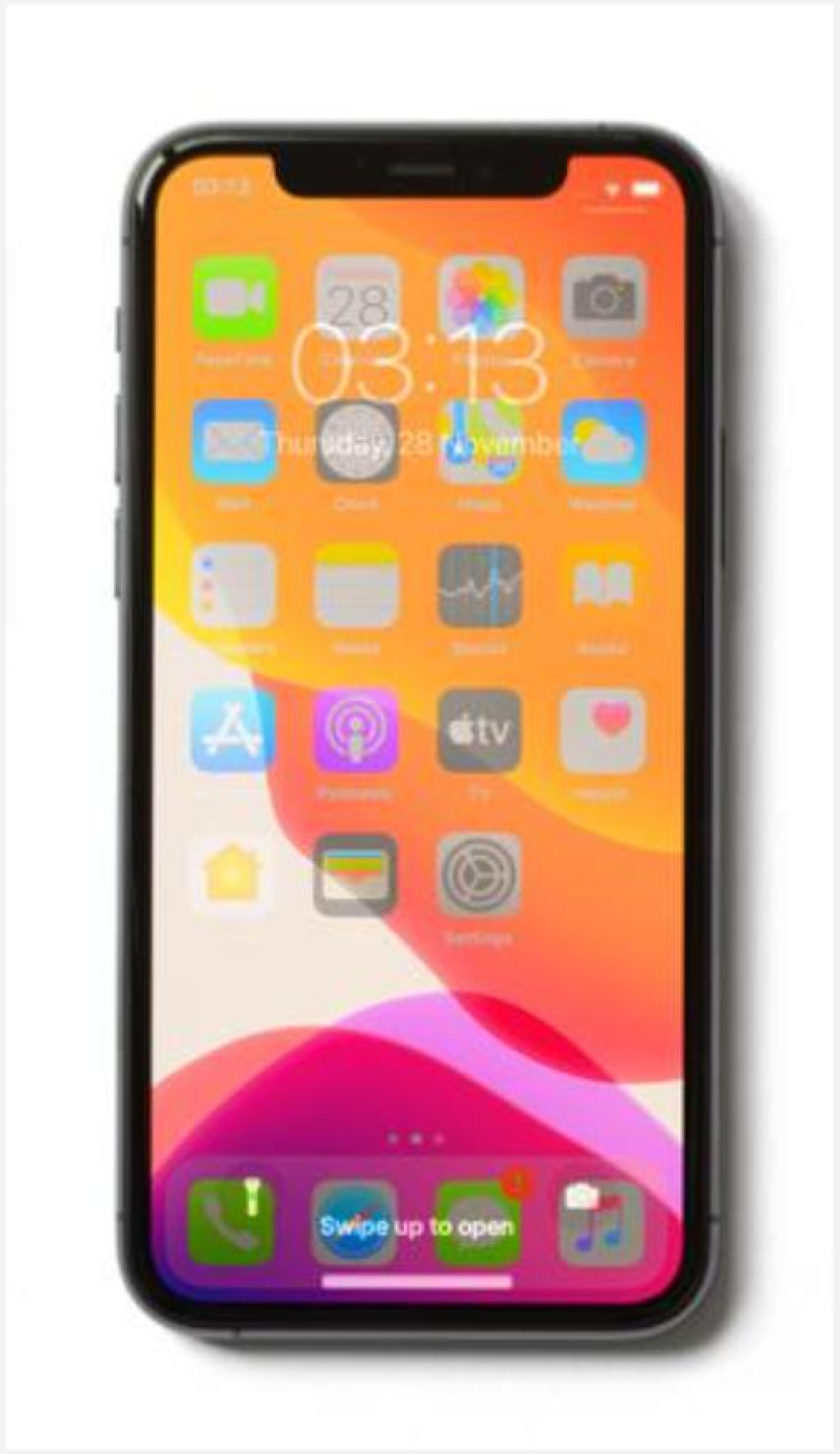
Slide Note



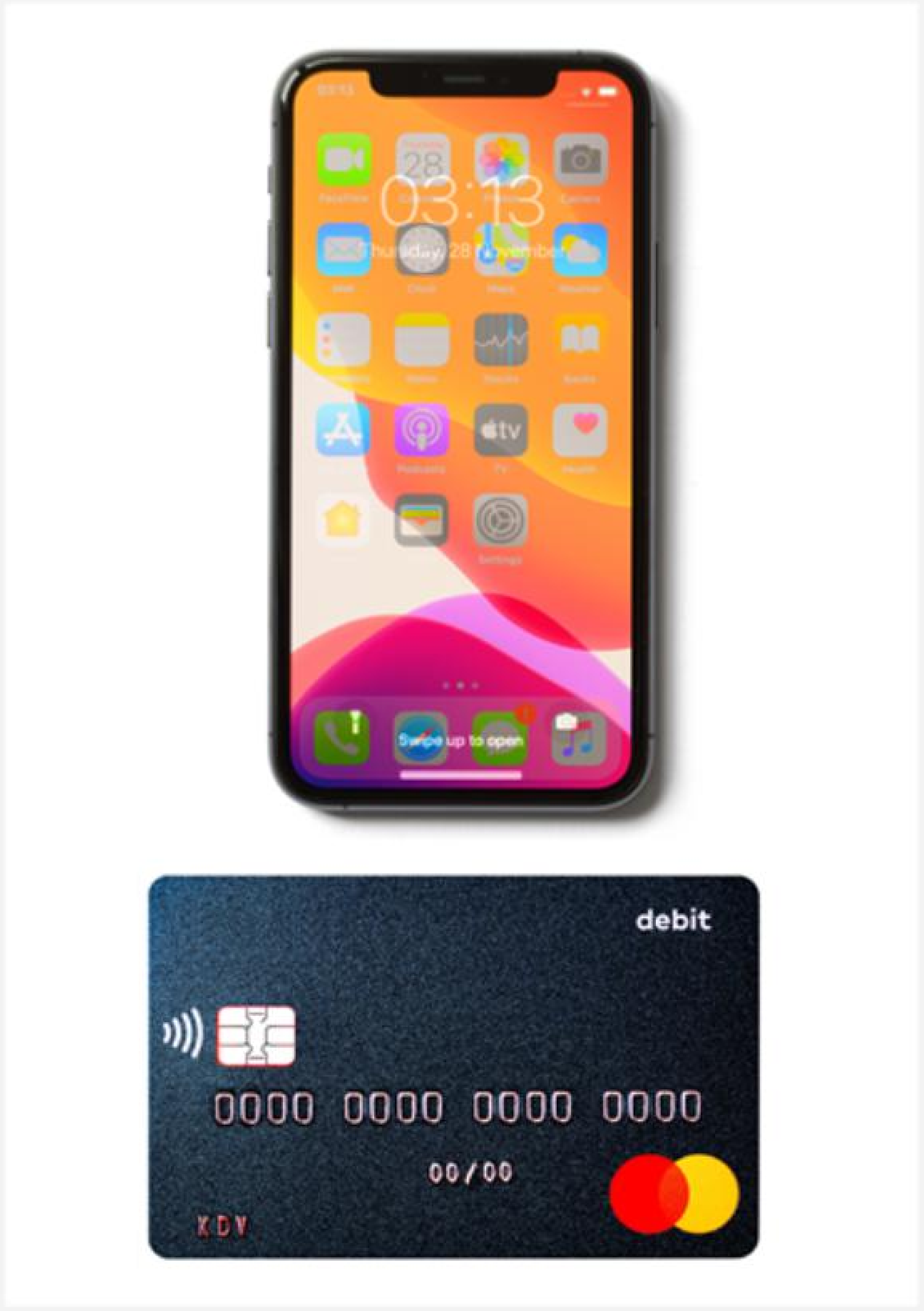
Display Screen

- Is there any problem with the display of this mobile application?

Menu

00:01 00:00:29 01/ 29

Slide Note



Think and Tell

- How do these problems impact consumers and businesses?
- When should these defects be identified?
- What steps must companies take to avoid such occurrences?

Slide Note

Product Testing

If a software application is developed and ready to be deployed, how do you ensure that it is defect-free?
Is software testing the solution?

Menu

00:03 00:00:29 03/ 29

Slide Note

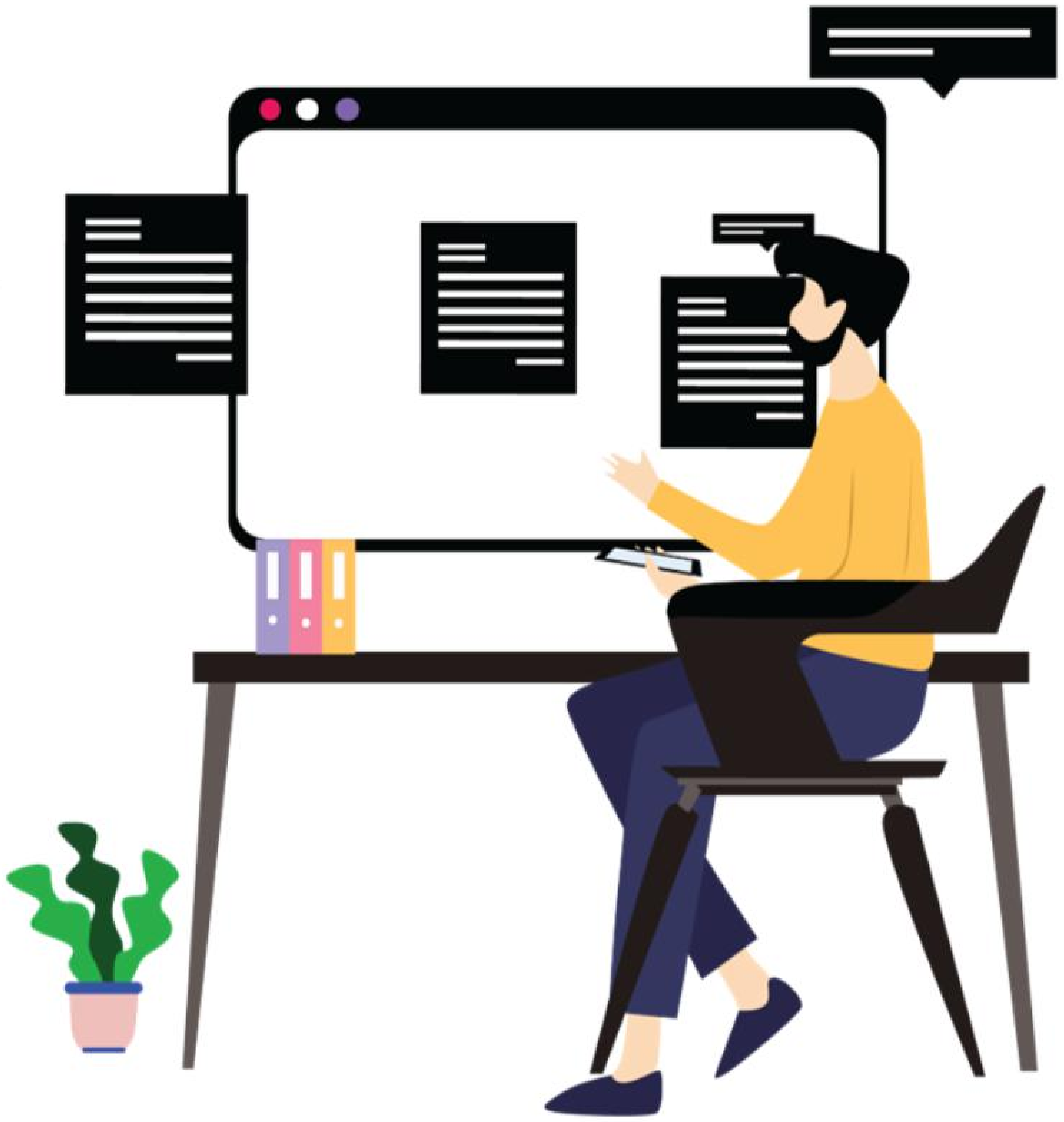
Think and Tell

- Should testing of the Spring Boot application be carried out layer by layer?
- Should the layers be dependent on one another for the testing process?
- Can you test the RESTful app in a loosely coupled manner?
- To test the service layer methods, can you use the actual object of the repository layer, or must we use a substitute?

Menu

Slide Note

Test RESTful Services at Service Layer and Data Layer by Using Testing Tools (JUnit, Mockito)



Menu

00:05:00:00:29 05/ 29

Slide Note

Learning Objectives

- Describe JUnit 5 Testing
- Implement repository layer testing
- Explore the Mockito framework
- Implement service layer testing

Menu

00:06 00:00:29 06/ 29

Slide Note

JUnit 5 Testing

Menu

00:07:00:00:29 07/ 29

Slide Note

JUnit 5 Testing

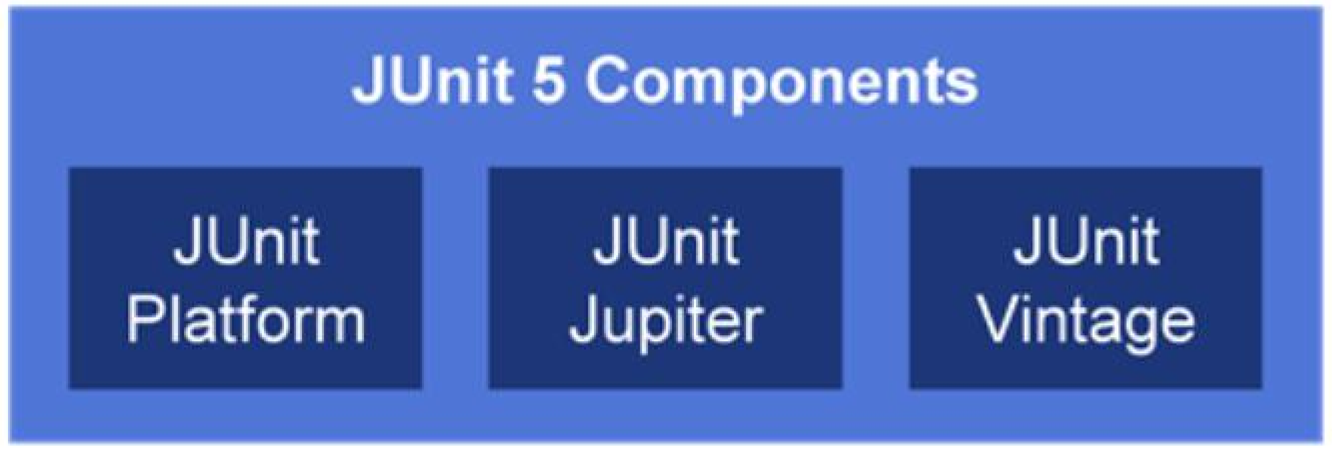
- JUnit is the easiest and most preferred testing framework for Java and Spring applications.
- Advantages of JUnit testing:
  - Improves the quality of the application
  - Avoids buggy situations

Menu

Slide Note

JUnit 5 Architecture

- `JUnit Platform`
  - Launches testing frameworks on the JVM
  - Its TestEngine API is used to build a testing framework that runs on the JUnit platform
- `JUnit Jupiter`
  - Blend of a new programming model for writing tests and an extension model for extensions
  - Contains new annotations like `@BeforeEach`, `@AfterEach`, `@AfterAll`, `@BeforeAll`, etc.
- `JUnit Vintage`
  - Provides support to execute previous JUnit version 3 and 4 tests on this new platform



Slide Note

JUnit 5 Annotations

- In JUnit 5, the test lifecycle is driven by 4 primary annotations, i.e., @BeforeEach, @AfterEach, @BeforeAll, and @AfterAll. Along with it, each test method must be marked with the @Test annotation.

JUnit 5	Description
@Test	Identifies a method as a test method.
@BeforeEach	Denotes that the annotated method should be executed *before* **each** @Test, @RepeatedTest method, etc. in the current class; analogous to JUnit 4's @Before.
@AfterEach	Denotes that the annotated method should be executed *after* **each** @Test, @RepeatedTest method, etc., in the current class; analogous to JUnit 4's @After.

Menu

00:10 00:00:29 10/ 29

Slide Note

JUnit 5 Annotations (contd.)

JUnit	Description
@Ignore or @Ignore ("Why disabled")	This is useful when the underlying code has been changed and the test case has not yet been adopted. If the execution time of this test is too long to be included, then it is best to provide an optional description of why the test is disabled.
@Test (timeout=100)	Fails if the method takes longer than 100 milliseconds.

Menu

00:11 00:00:29 11/ 29

Slide Note

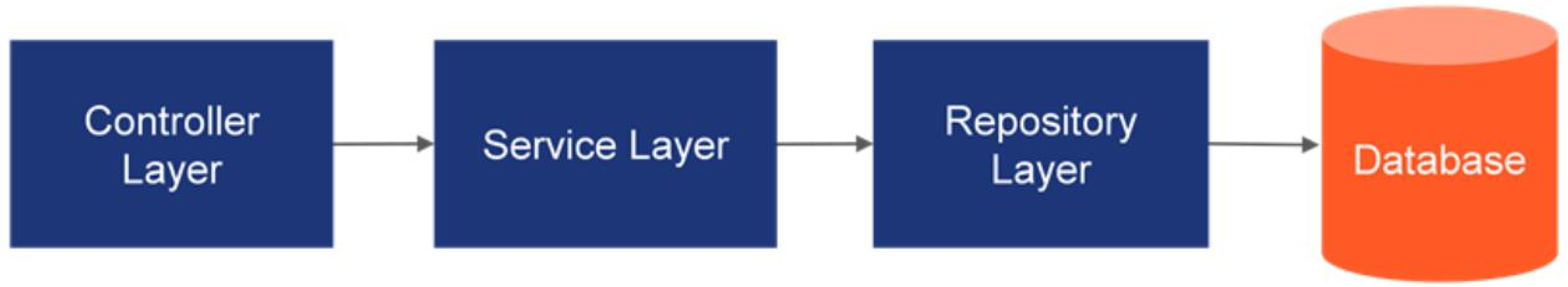
Implementing Repository Layer Testing

Menu

00:12 00:00:29 12/ 29

Slide Note

Layers in Spring Boot Application



Menu

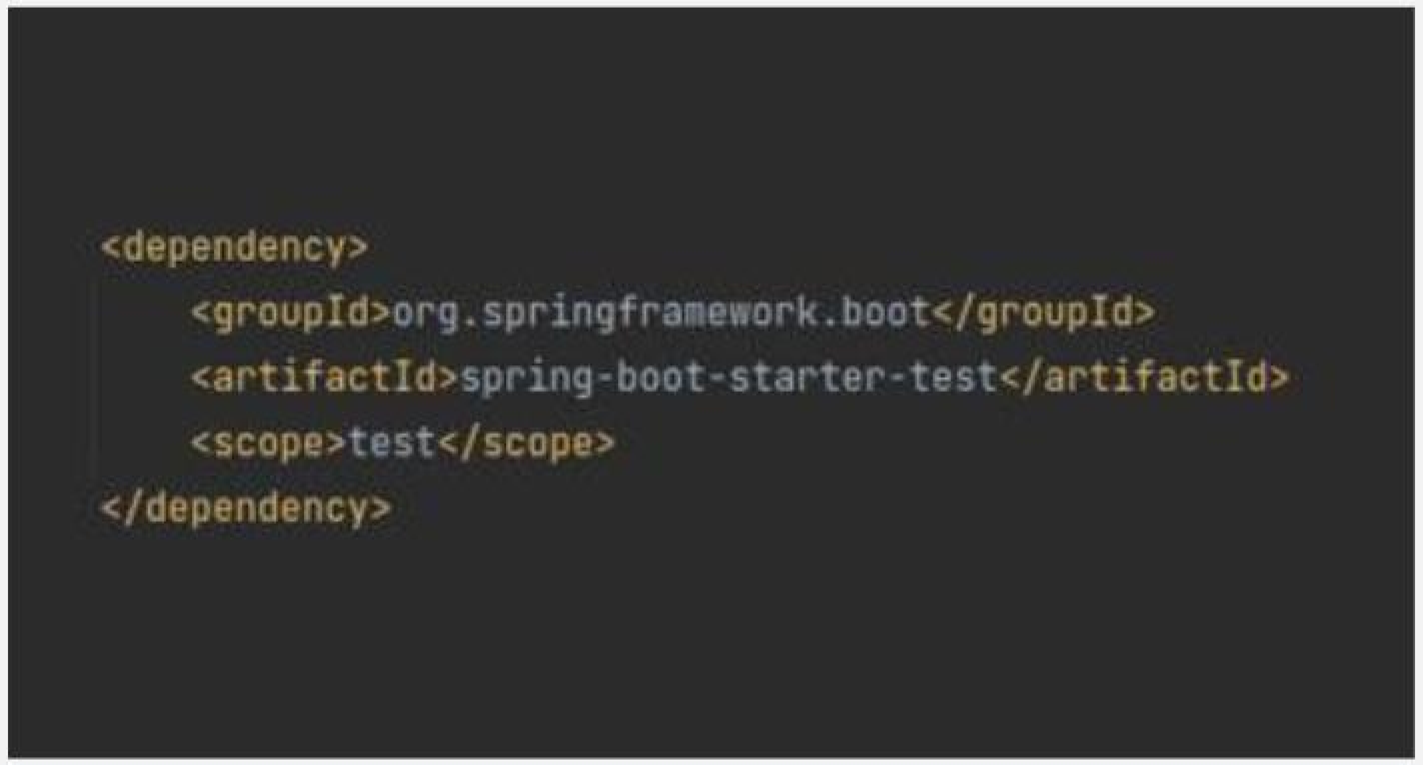
Testing RESTful Web Services

- Testing the RESTful APIs helps us keep the application bug-free.
- Testing the RESTful APIs ensures the functionality of the application is working properly when it is deployed in production.
- The purpose of REST API testing is to record the response of REST API by sending various HTTP requests and checking their responses.
- REST API testing is done for GET, POST, PUT, and DELETE methods.

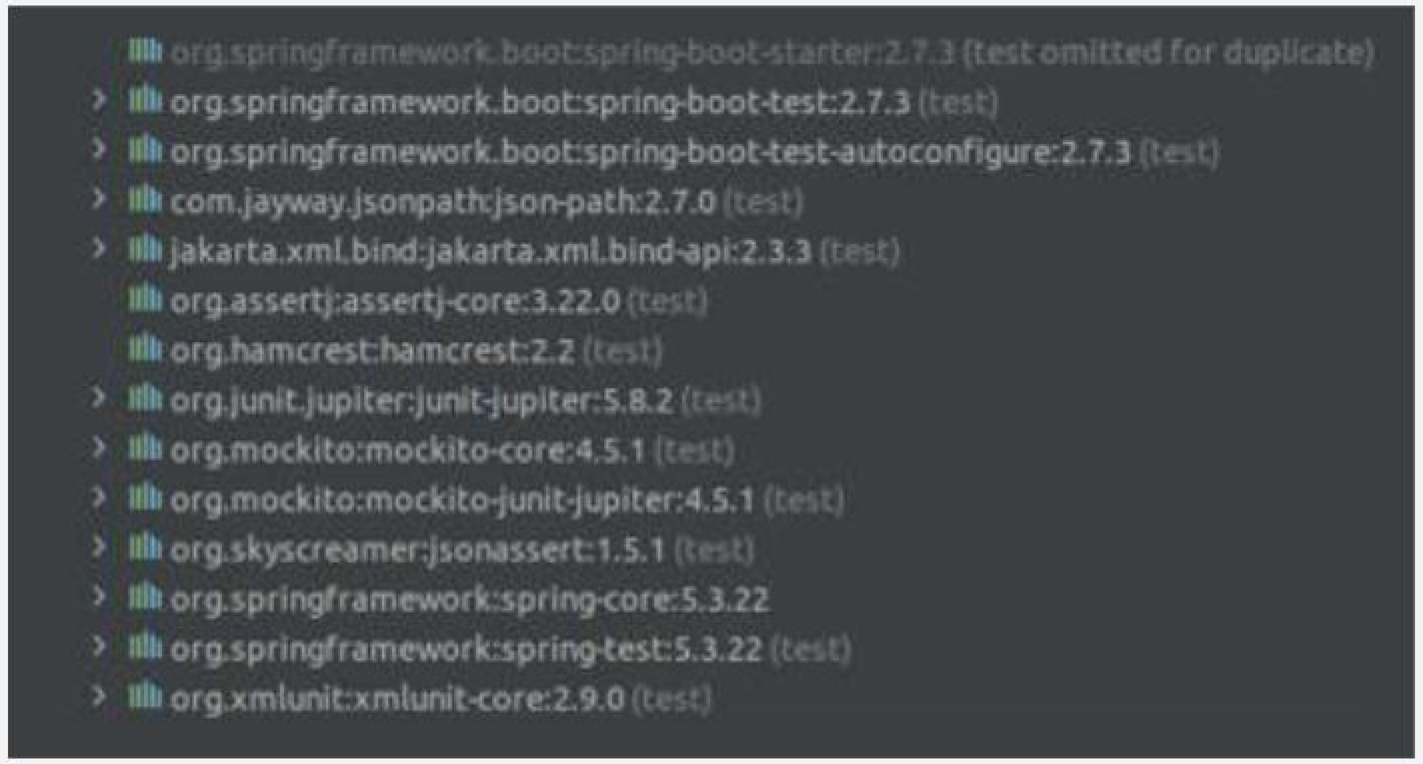
Slide Note

Spring-Boot-Starter-Test

- **Spring Boot provides us with the dependency of** `spring-boot-starter-test`**, which enables the testing capabilities of a Spring application.**
- **The dependencies required for testing, such as** `JUnit, Mockito, Hamcrest, spring-test, spring-boot-test`**, etc., are included with the** `spring-boot-starter-test`.
- `Mockito` **is a popular mock framework that can be used in conjunction with the JUnit.**
- `Hamcrest` **is commonly used with JUnit and other testing frameworks for making assertions.**
- `Spring test & Spring Boot Test` **– provides utilities for integration test support for Spring Boot applications.**



```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```



```
  org.springframework.boot:spring-boot-starter:2.7.3 (test omitted for duplicate)
> org.springframework.boot:spring-boot-test:2.7.3 (test)
> org.springframework.boot:spring-boot-test-autoconfigure:2.7.3 (test)
> com.jayway.jsonpath:json-path:2.7.0 (test)
> jakarta.xml.bind:jakarta.xml.bind-api:2.3.3 (test)
  org.assertj:assertj-core:3.22.0 (test)
  org.hamcrest:hamcrest:2.2 (test)
> org.junit.jupiter:junit-jupiter:5.8.2 (test)
> org.mockito:mockito-core:4.5.1 (test)
> org.mockito:mockito-junit-jupiter:4.5.1 (test)
> org.skyscreamer:jsonassert:1.5.1 (test)
> org.springframework:spring-core:5.3.22
> org.springframework:spring-test:5.3.22 (test)
> org.xmlunit:xmlunit-core:2.9.0 (test)
```

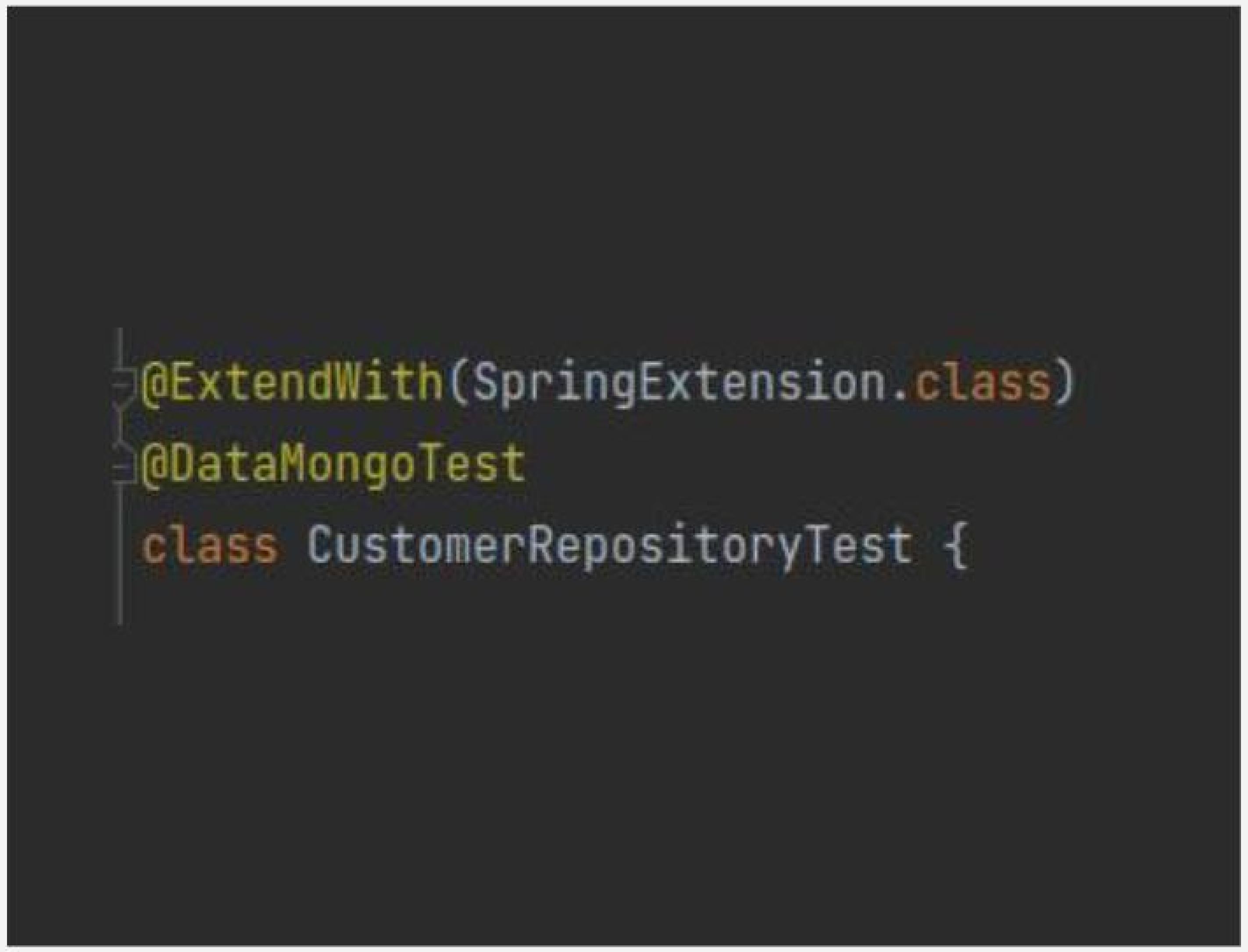
Menu

00:15 00:00:29 15/ 29

Slide Note

Testing Repository Layer

- The test class needs to be annotated with `@ExtendWith(SpringExtension.class)` and `@DataMongoTest`.
- The `@ExtendWith` annotation integrates the Spring TestContext Framework with JUnit 5's Jupiter programming model.
- The `@DataMongoTest` annotation scans the `@Document` classes and configures Spring Data repositories.

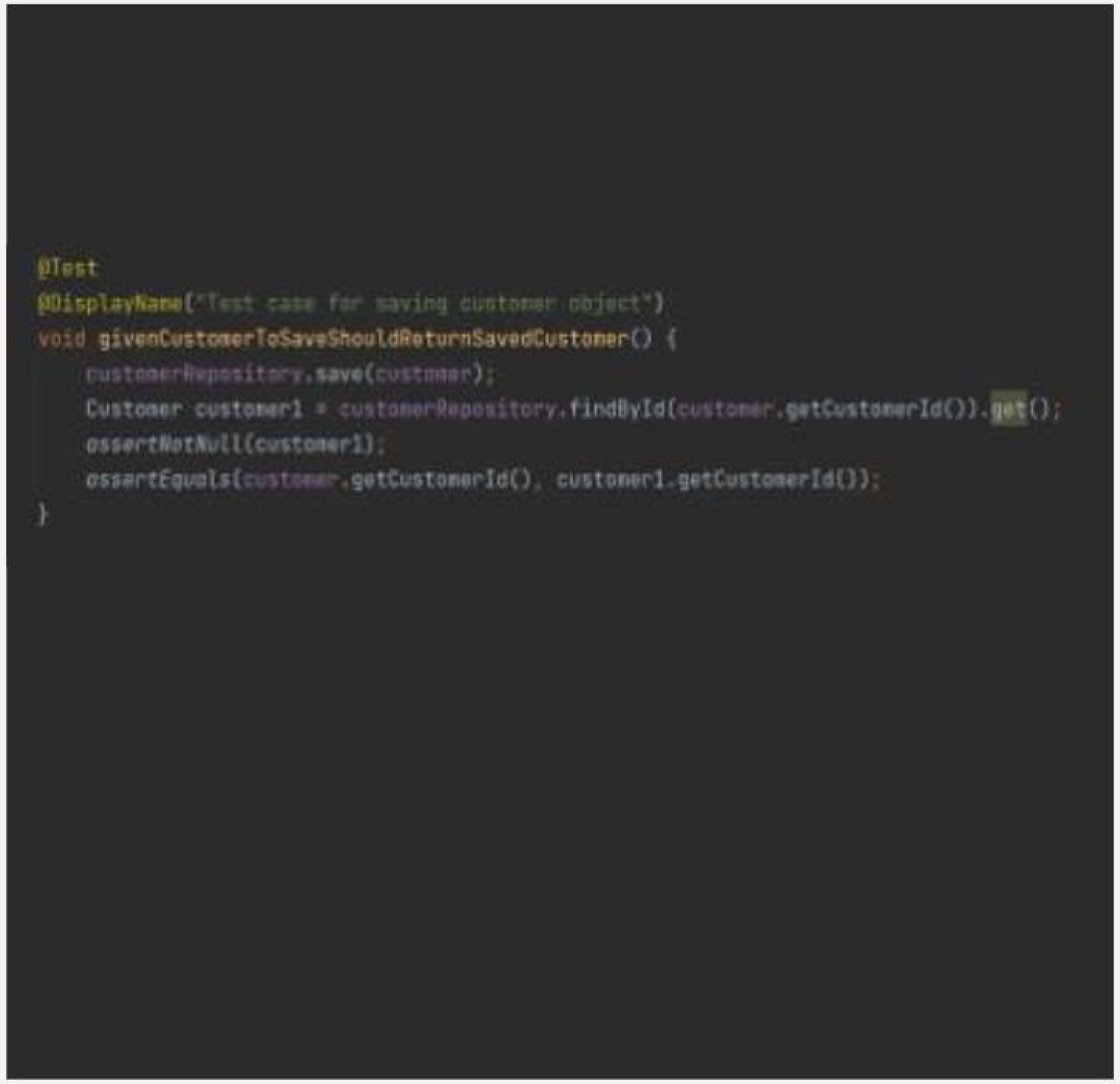


```
@ExtendWith(SpringExtension.class)
@DataMongoTest
class CustomerRepositoryTest {
```

Menu

Slide Note

Testing Repository Layer (contd.)



```
@Test
@DisplayName("Test case for saving customer object")
void givenCustomerToSaveShouldReturnSavedCustomer() {
    customerRepository.save(customer);
    Customer customer1 = customerRepository.findById(customer.getCustomerId()).get();
    assertNotNull(customer1);
    assertEquals(customer.getCustomerId(), customer1.getCustomerId());
}
```

- **In the application, the repository layer is dependent on the external database MongoDB.**
- `CustomerRepository` **is calling the** `save()` **method and saving the customer object in the database.**
- **Here, the Spring Object gets saved in an external database.**
- **For this reason, we call the repository layer testing an "integration testing."**
- **Integration testing verifies that the code works with external dependencies correctly.**
- **Integration testing may detect errors when modules are integrated to build the overall system.**

Menu

00:17/00:00:29 17/ 29

Slide Note

Testing Repository Layer

A supermarket chain needs to perform an analysis of customer data to determine how it can expand its business to new locations.
They have outsourced the work to Zedex Company to create the application. Zedex Company has completed the application and tested the REST endpoints using Postman.
Now they want to automate the testing purpose so they can start with the repository layer.

Check the solution here.

DEMO

Menu

00:18:00:00:29 18/ 29

Slide Note

Quick Check

Which dependency is required to test Spring layers?

1. spring-boot-starter-data-mongodb
2. spring-boot-starter-web
3. spring-boot-starter-test
4. spring-boot-starter-data-jpa

Menu

Slide Note

Quick Check: Solution

Which dependency is required to test Spring layers?

1. spring-boot-starter-data-mongodb
2. spring-boot-starter-web
3. spring-boot-starter-test
4. spring-boot-starter-data-jpa

Menu

00:20 00:00:29 20/ 29

Slide Note

Exploring Mockito Framework

Menu

00:21:00:00:29 21/ 29

Slide Note

Testing Service Layer

- RESTful layers are dependent on each other. The controller is dependent on the service layer. The service layer is dependent on the repository layer and the repository layer is dependent on the external database MongoDB.
- While working with larger enterprise applications, a different team might be developing the controller, service, or repository layers of the application.
- Consider the questions given below:
  - If the repository has not yet been created by the other team, how can one test the service layer?
  - Should the other team wait for the repository code to be created before one starts coding the service?

Menu

Slide Note

Testing Service Layer (contd.)

- In testing, these scenarios are handled using mocking. Since the repository is an external dependency, we can mock it.
- We mock a dependency by creating a mock object of the dependency, which is like a stunt double of an actor in movies.
- Therefore, in our test, you will create a mock object of the repository layer.
- Hence, for testing the service layer, we will mock the repository object instead of using the real object.
- These mocks will be created using Mockito Framework.
- Mocking is done for behavior verification. It means that the mock object verifies that it (the mock object) is being used correctly by the object under test.
- If the verification or tests succeeds, one can assume that the object being tested will correctly use the real collaborator.

Menu

Slide Note

Mockito Framework

- Mockito is a popular mock framework that can be used in conjunction with the JUnit.
- Mockito allows us to create and configure mock objects. Using Mockito significantly simplifies the development for classes with external dependencies.
- When we use Mockito in tests, we can:
  - Mock away external dependencies and insert the mocks into the code under test.
  - Execute the code under test.
  - Validate that the code is executed correctly.

MOCKITO

Menu

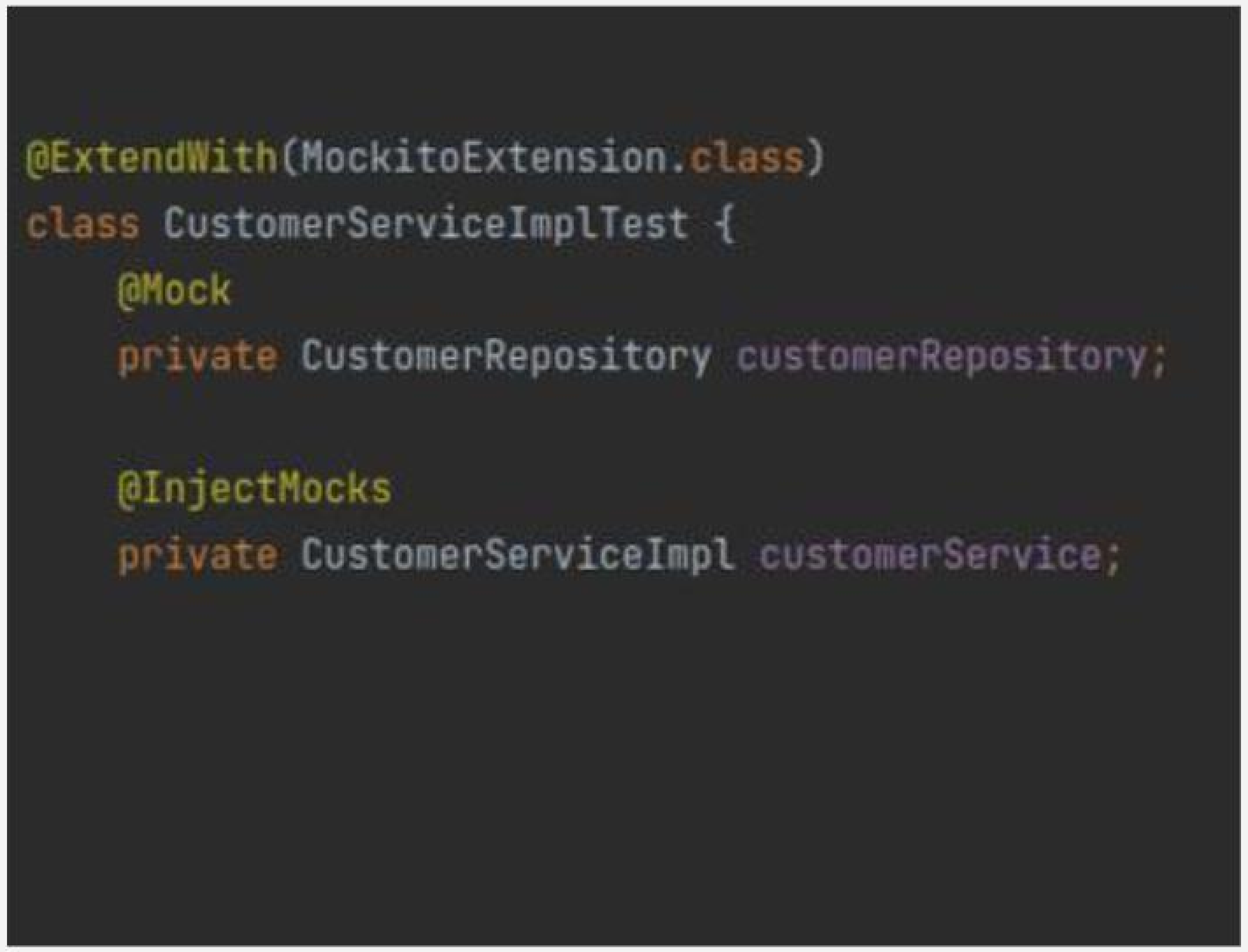
Slide Note

Implementing Service Layer Testing

Slide Note

Testing Service Layer With Mockito

- The test class needs to be created and annotated with `@ExtendWith(MockitoExtension.class).`
- `MockitoExtension` is part of the Mockito library. We use it to perform mocking. This annotation initializes mocks in the test class.
- The `@Mock` annotation is a part of Mockito, which specifies that a repository layer should be mocked. At runtime, Mockito creates a mock object of the repository.
- The `ServiceImpl` class is annotated with `@InjectMocks.`
- `@InjectMocks` creates an instance of the class and injects the mocks that are created with the @Mock annotation into this instance.
- At runtime, Mockito instantiates `ServiceImpl` class and passes the repository mock object to it.



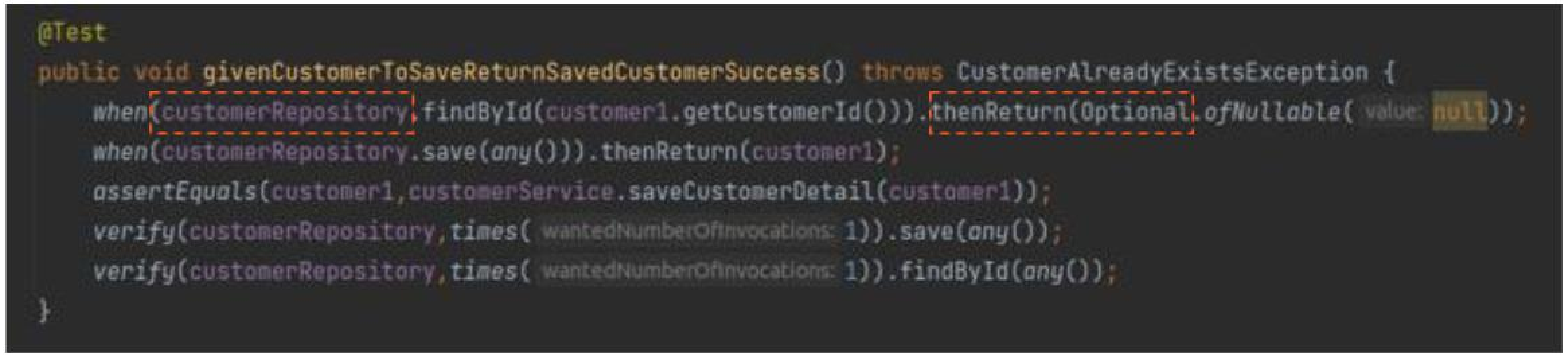
```java
@ExtendWith(MockitoExtension.class)
class CustomerServiceImplTest {
    @Mock
    private CustomerRepository customerRepository;

    @InjectMocks
    private CustomerServiceImpl customerService;
```

Slide Note

Testing Service Layer With Mockito (contd.)

The `when …. Then` is the crux of the Mockito testing.



```
@Test
public void givenCustomerToSaveReturnSavedCustomerSuccess() throws CustomerAlreadyExistsException {
    when(customerRepository.findById(customer1.getCustomerId())).thenReturn(Optional.ofNullable( value: null));
    when(customerRepository.save(any())).thenReturn(customer1);
    assertEquals(customer1,customerService.saveCustomerDetail(customer1));
    verify(customerRepository,times( wantedNumberOfInvocations: 1)).save(any());
    verify(customerRepository,times( wantedNumberOfInvocations: 1)).findById(any());
}
```

When a new object is to be saved in the database, the assumption is that the object is not present, so when the `findById` method is used, the expectation is that null must be returned. This is done by the `when` method of the Mockito. It mocks the repository layer and sets an expectation.

- **When the `findById` is called, the expectation is to return a null object.**
- **Similarly, while saving the object, the expectation is that the object will get saved and the saved object will be returned.**
- **The when method of Mockito is not calling the customeRepository in real, it's just going to `mock` it.**
- **The verify method checks how many times mock method is getting called.**

Menu

Slide Note

Testing Service Layer

A supermarket chain needs to perform an analysis of customer data to determine how it can expand its business to new locations.
They have outsourced the work to Zedex Company to create the application. Zedex Company has completed the application and tested the REST endpoints using Postman.
Now they want to automate the testing purpose, so they will start with the repository, service, and controller layers.

Check the solution here.

DEMO

Menu

00:28:00:00:29 28/ 29