45426: Teste e Qualidade de Software

Intregration testing in Spring Boot

Ilídio Oliveira

v2020-03-10

universidade de aveiro
departamento de eletrónica,
telecomunicações e informática
deti

Learning outcomes

Relate the test of API with the right level of testing in the “pyramid of tests”

Explain the structure of a Rest-Assure test

Discuss diferente strategies to test layered applications, such as Spring Boot

Read SpringBoot tests with mocking of services

Popular testing tools for the Java developer

Basics (unit)

JUnit, TestNG

Spock

Hamcrest, AssertJ

Enterprise apps/backend

Arquillian

SpringBoot testing

Mocking objects behavior

Mockito

EasyMock

Web/functional testing

Selenium IDE

API Testing

REST-Assured

Story-driven (BDD)

Cucumber

See also: https://dzone.com/articles/10-essential-testing-tools-for-java-developers

Hamcrest

“Matchers” that can be combined to create flexible expressions of intent (in unit testing)

instead of using JUnit‘s numerous assert methods, we only use the assertThat statement with appropriate matchers

guide/reference

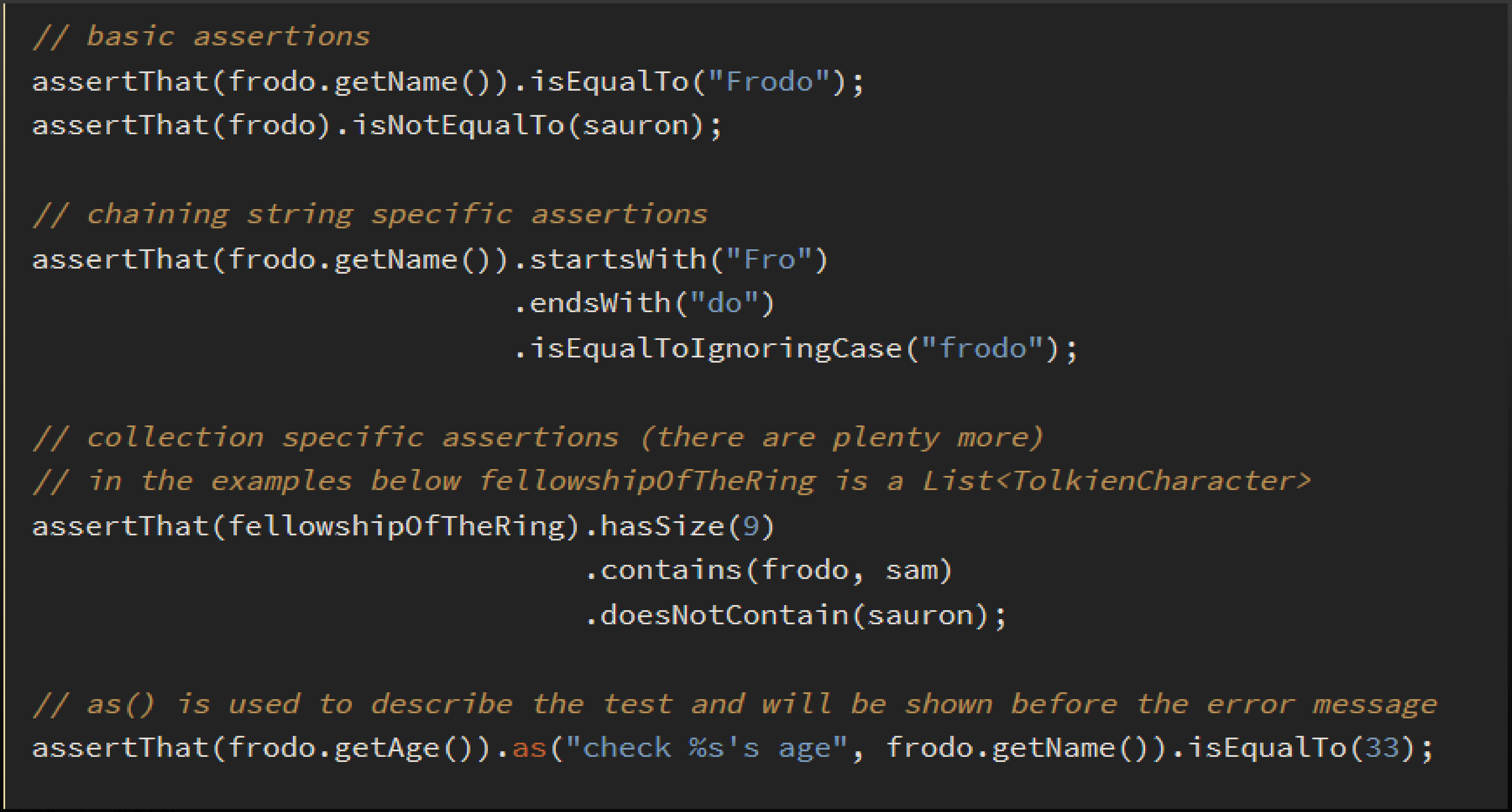
```
assertThat(5, Matchers.equalTo(5));

assertThat(5, Matchers.greaterThanOrEqualTo(5));

assertThat(str1, equalToIgnoringWhiteSpace(str2));

// collections
assertThat(emptyList, empty());

String[] hamcrestMatchers = { "collections", "beans",
"text", "number" };
assertThat("text", isOneOf(hamcrestMatchers));

// object level inspection
assertThat(person, hasProperty("address", equalTo("New
York")));
assertThat(person1, samePropertyValuesAs(person2));
```

Oliveira (2018)

AssertJ

Integrated in Spring Boot

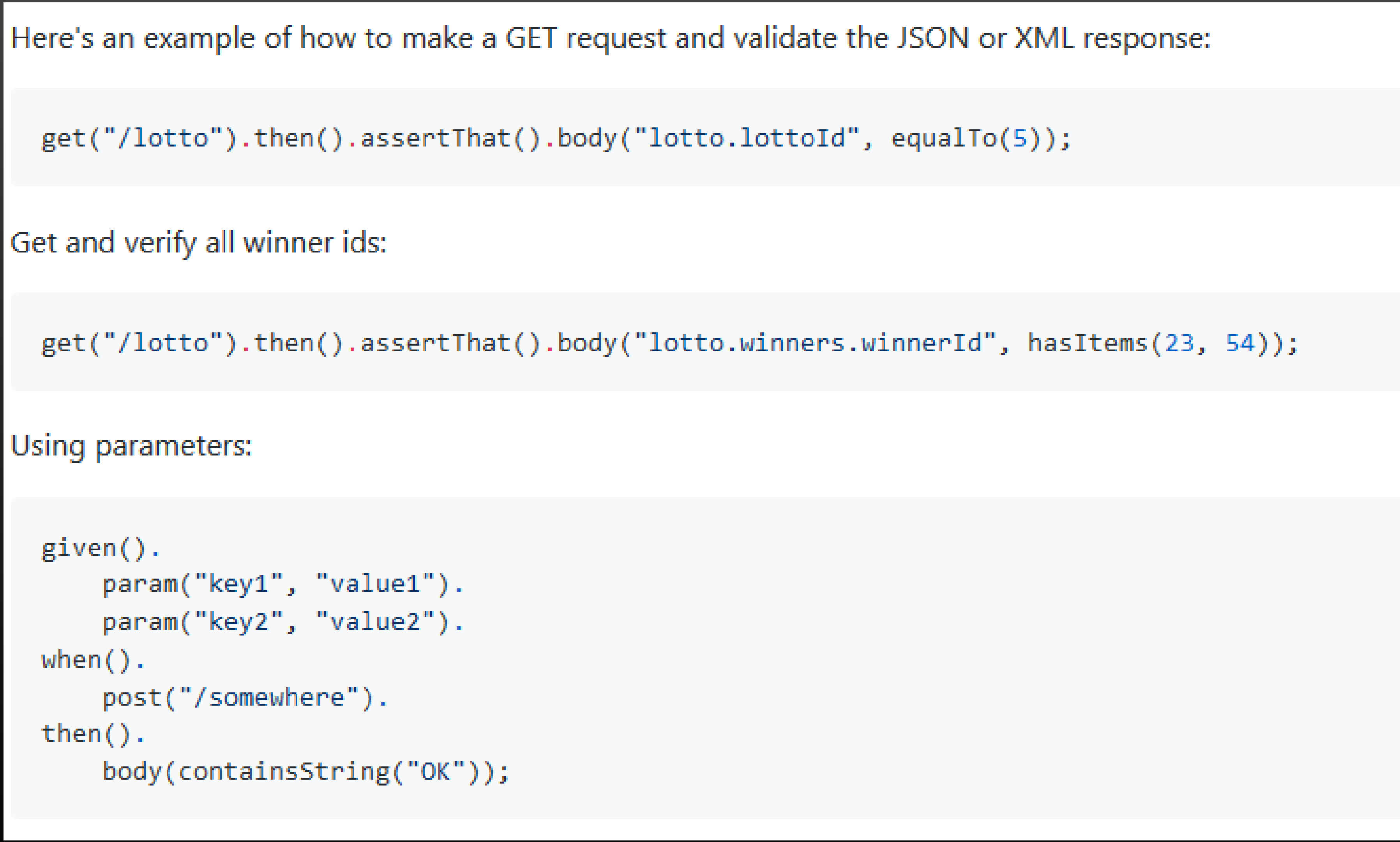
Fluent assertions, with chaining syntax:



```java
// basic assertions
assertThat(frodo.getName()).isEqualTo("Frodo");
assertThat(frodo).isNotEqualTo(sauron);

// chaining string specific assertions
assertThat(frodo.getName()).startsWith("Fro")
                           .endsWith("do")
                           .isEqualToIgnoringCase("frodo");

// collection specific assertions (there are plenty more)
// in the examples below fellowshipOfTheRing is a List<TolkienCharacter>
assertThat(fellowshipOfTheRing).hasSize(9)
                               .contains(frodo, sam)
                               .doesNotContain(sauron);

// as() is used to describe the test and will be shown before the error message
assertThat(frodo.getAge()).as("check %s's age", frodo.getName()).isEqualTo(33);
```

See also: https://www.baeldung.com/introduction-to-assertj

REST-Assured



Here's an example of how to make a GET request and validate the JSON or XML response:

```
get("/lotto").then().assertThat().body("lotto.lottoId", equalTo(5));
```

Get and verify all winner ids:

```
get("/lotto").then().assertThat().body("lotto.winners.winnerId", hasItems(23, 54));
```

Using parameters:

```
given().
    param("key1", "value1").
    param("key2", "value2").
when().
    post("/somewhere").
then().
    body(containsString("OK"));
```

http://rest-assured.io/

REST-Assured

testing and validation of REST APIs

suggested “lesson”

suggested tutorial/guide

```
@Before
public void setup() {
    RestAssured.baseURI = "https://api.github.com";
    RestAssured.port = 443;
}

...

get("/events?id=390").then().statusCode(200).assertThat()
      .body("data.leagueId", equalTo(35));

when().request("GET",
"/users/eugenp").then().statusCode(200);

with().body(new Odd(5.25f, 1, 13.1f, "X"))
      .when()
      .request("POST", "/odds/new")
      .then()
      .statusCode(201);
```

Oliveira (2018)

Spring Boot testing

A helper framework used to simplify the creation of Spring Framework apps

Provides:

Curated dependencies

“Starter” configurations (data, web, testing,…)

“Opinionated” auto-configuration of many components

Sensible defaults

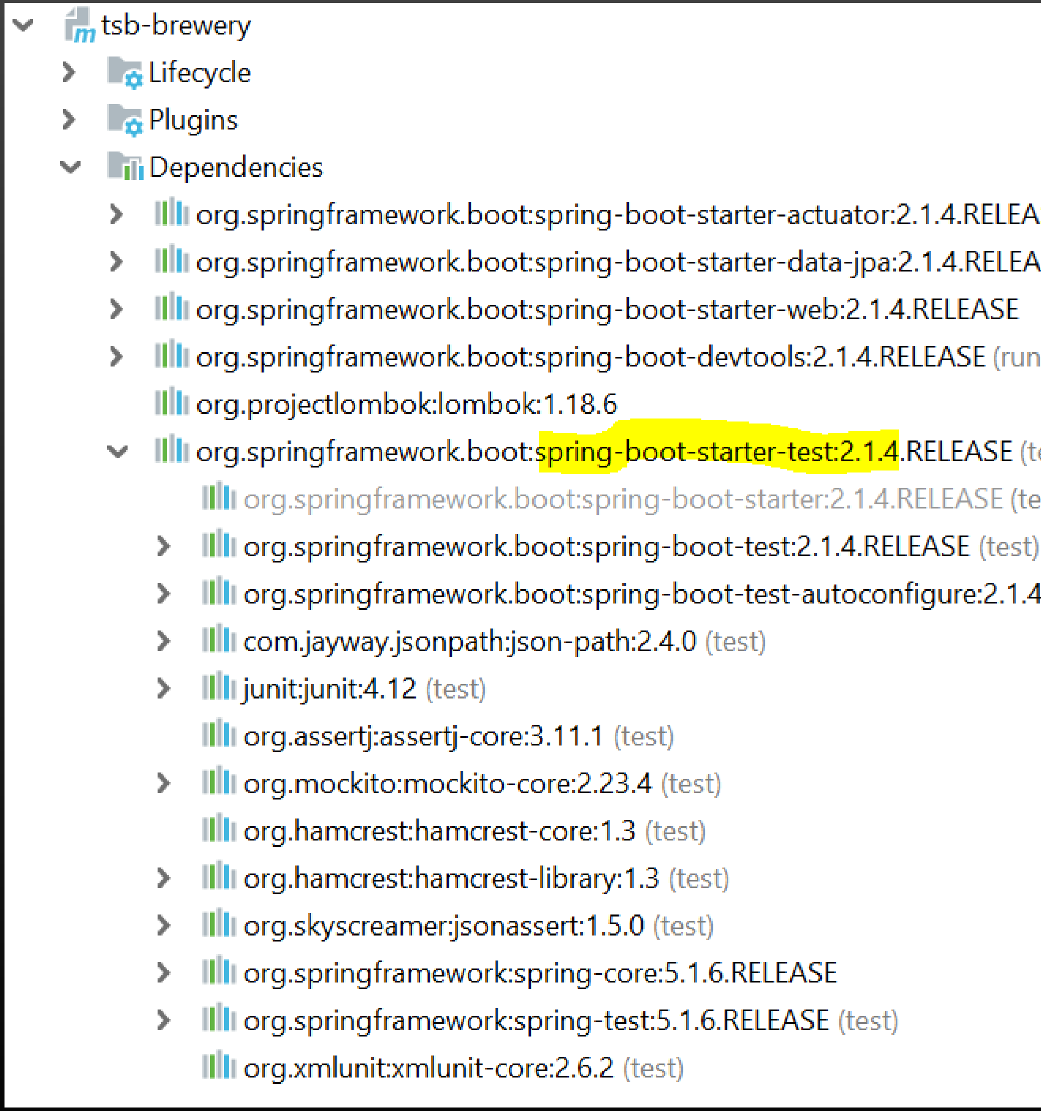
Extending SB philosophy to testing

Test features enabled with

- **spring-boot-starter-test**

Starter provides:

- Testing dependencies
- Testing auto-config



```
tsb-brewery
  Lifecycle
  Plugins
  Dependencies
    org.springframework.boot:spring-boot-starter-actuator:2.1.4.RELEA
    org.springframework.boot:spring-boot-starter-data-jpa:2.1.4.RELEA
    org.springframework.boot:spring-boot-starter-web:2.1.4.RELEASE
    org.springframework.boot:spring-boot-devtools:2.1.4.RELEASE (run
    org.projectlombok:lombok:1.18.6
    org.springframework.boot:spring-boot-starter-test:2.1.4.RELEASE (te
      org.springframework.boot:spring-boot-starter:2.1.4.RELEASE (te
      org.springframework.boot:spring-boot-test:2.1.4.RELEASE (test)
      org.springframework.boot:spring-boot-test-autoconfigure:2.1.4
      com.jayway.jsonpath:json-path:2.4.0 (test)
      junit:junit:4.12 (test)
      org.assertj:assertj-core:3.11.1 (test)
      org.mockito:mockito-core:2.23.4 (test)
      org.hamcrest:hamcrest-core:1.3 (test)
      org.hamcrest:hamcrest-library:1.3 (test)
      org.skyscreamer:jsonassert:1.5.0 (test)
      org.springframework:spring-core:5.1.6.RELEASE
      org.springframework:spring-test:5.1.6.RELEASE (test)
      org.xmlunit:xmlunit-core:2.6.2 (test)
```

Oliveira (2018)

@SpringBootTest

@SpringBootTest annotation

Enable FULL context, using all available auto configurations

Heavy!

better to limit Application Context to a set of spring components that participate in test scenario, by listing them (with annotations)

Slicing the test context

Only load slices of functionality when testing spring boot

@xxxxxTest at class level, e.g.: @DataJpaTest, @DataMongoTest, @JsonTest, @WebMvcTest,...

Mind JUnit version

@RunWith(SpringRunner.class) required for JU4

SpringRunner is an alias for the SpringJUnit4ClassRunner.

Spring Boot components

Components registration

In each layer, we have various components. Simply put, to detect them automatically, Spring uses classpath scanning annotations.

Then, it registers each bean in the ApplicationContext.

A few of these annotations:

@Component: generic stereotype for any Spring-managed component

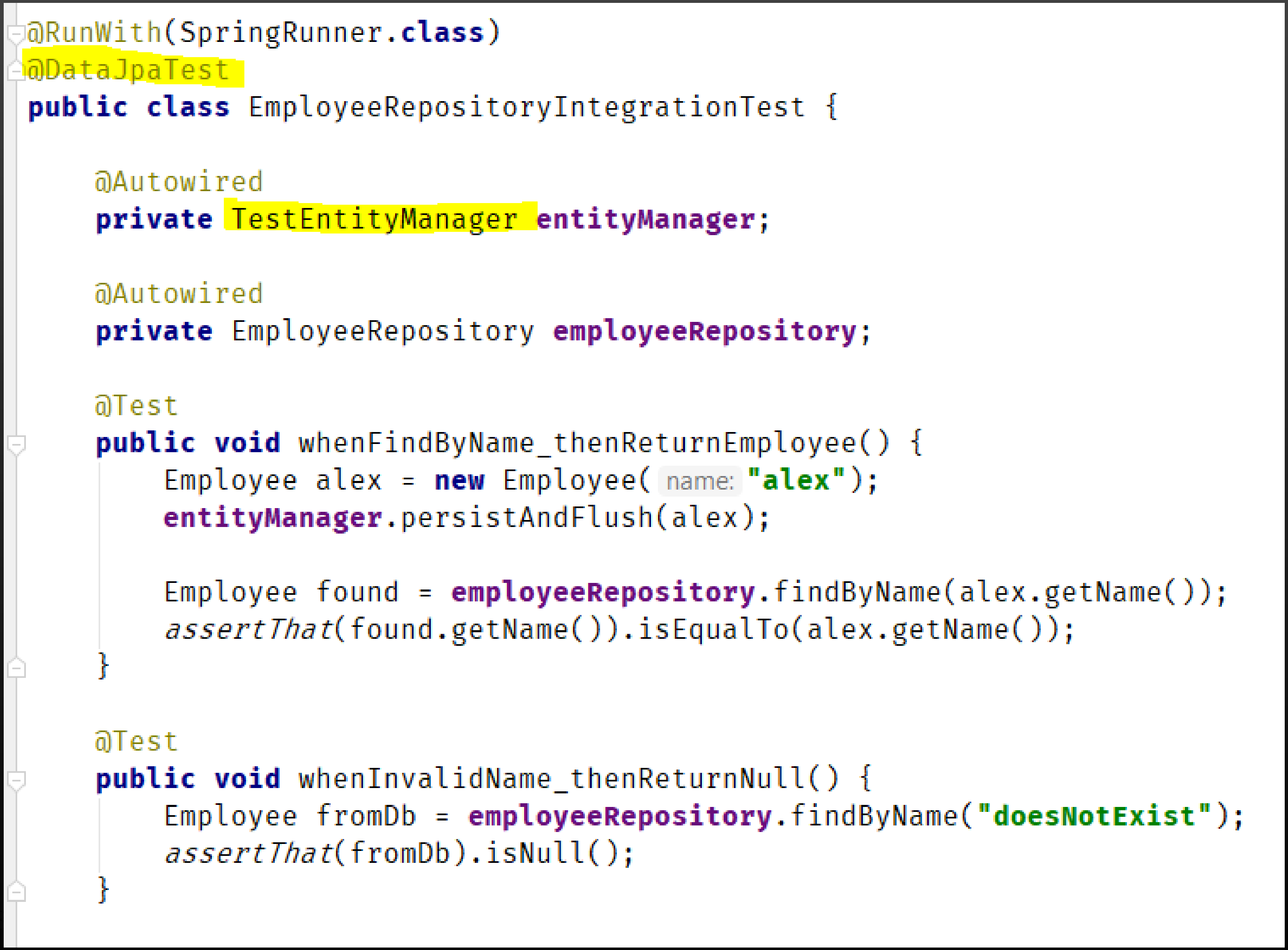
@Service: "components" meant to be used at the service layer

@Repository: classes at the persistence layer, which will act as a database repository

@Service and *@Repository* are special cases of *@Component*.

Oliveira (2018)

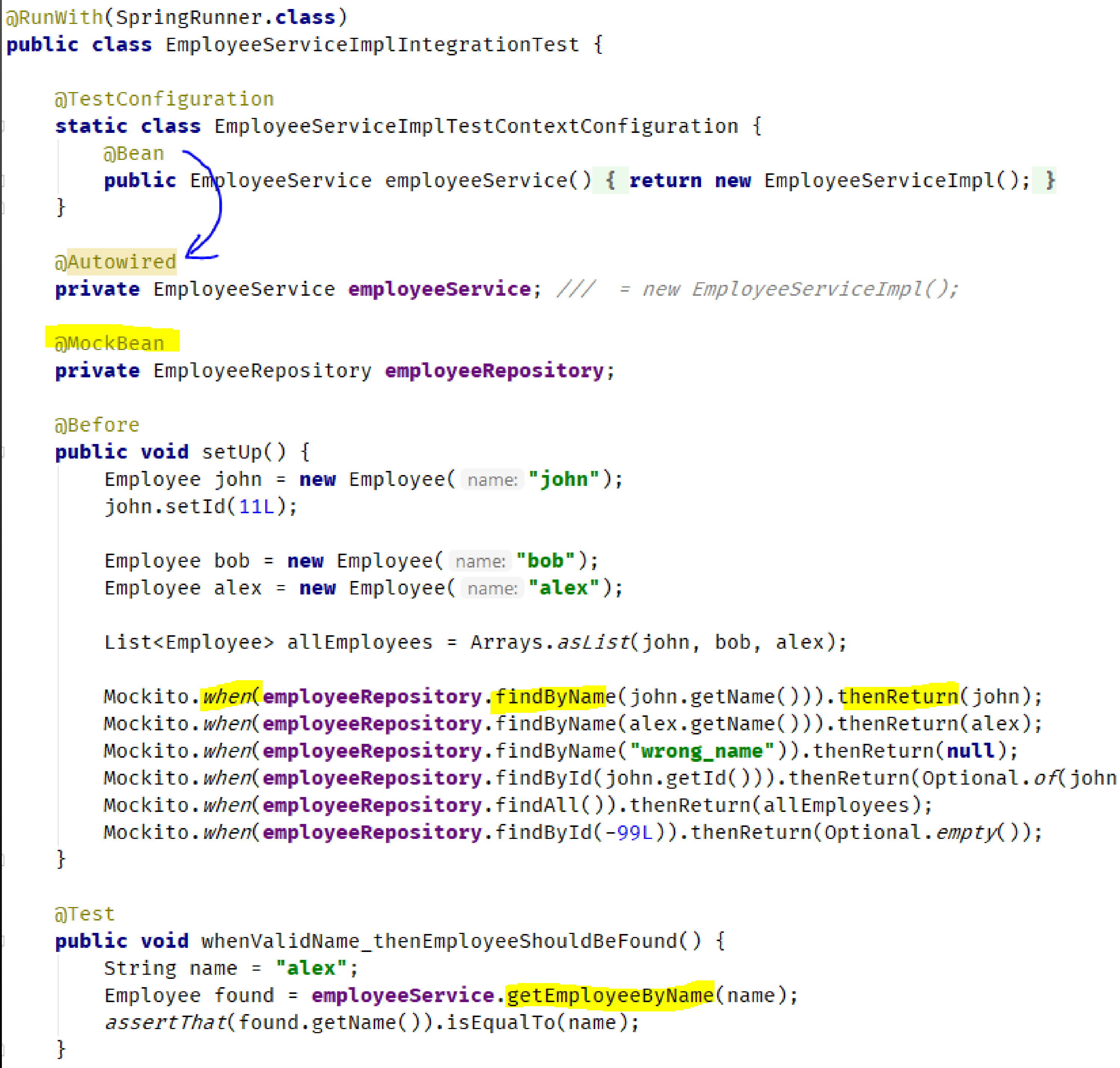
Testing the data persistence/repository services



```java
@RunWith(SpringRunner.class)
@DataJpaTest
public class EmployeeRepositoryIntegrationTest {

    @Autowired
    private TestEntityManager entityManager;

    @Autowired
    private EmployeeRepository employeeRepository;

    @Test
    public void whenFindByName_thenReturnEmployee() {
        Employee alex = new Employee( name: "alex");
        entityManager.persistAndFlush(alex);

        Employee found = employeeRepository.findByName(alex.getName());
        assertThat(found.getName()).isEqualTo(alex.getName());
    }

    @Test
    public void whenInvalidName_thenReturnNull() {
        Employee fromDb = employeeRepository.findByName("doesNotExist");
        assertThat(fromDb).isNull();
    }
```

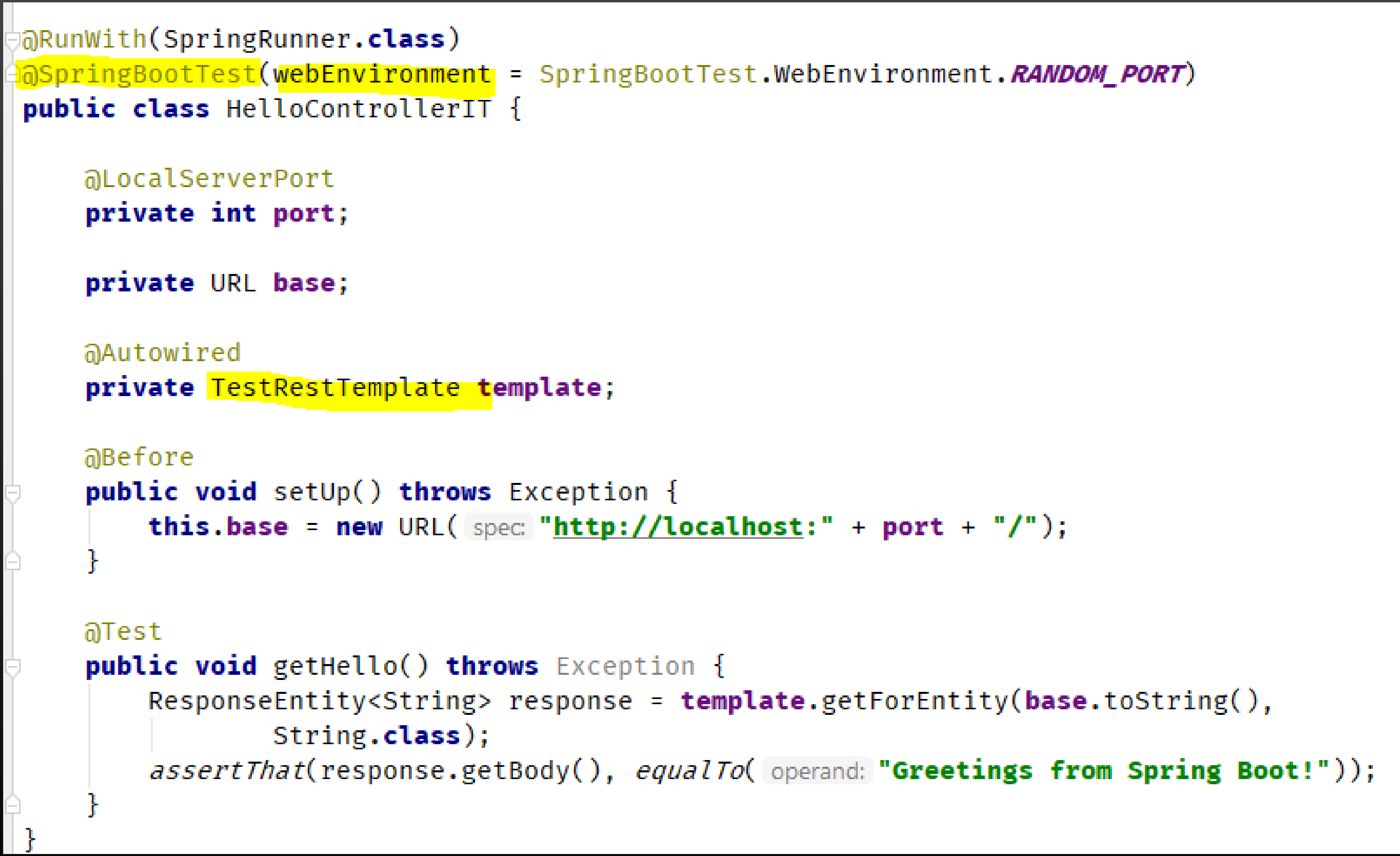
I Oliveira (2018

Testing the service layer, isolating the persistence with mocks



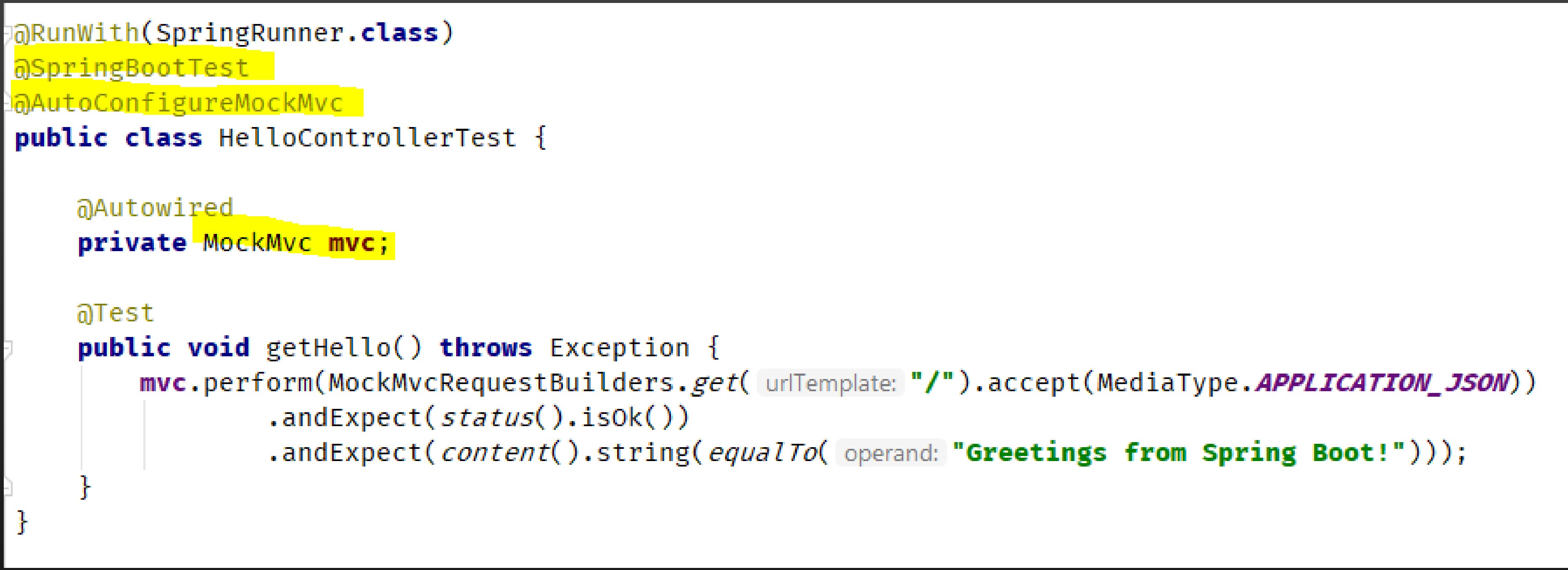
```
@RunWith(SpringRunner.class)
public class EmployeeServiceImplIntegrationTest {

    @TestConfiguration
    static class EmployeeServiceImplTestContextConfiguration {
        @Bean
        public EmployeeService employeeService() { return new EmployeeServiceImpl(); }
    }

    @Autowired
    private EmployeeService employeeService; ///  = new EmployeeServiceImpl();

    @MockBean
    private EmployeeRepository employeeRepository;

    @Before
    public void setUp() {
        Employee john = new Employee( name: "john");
        john.setId(11L);

        Employee bob = new Employee( name: "bob");
        Employee alex = new Employee( name: "alex");

        List<Employee> allEmployees = Arrays.asList(john, bob, alex);

        Mockito.when(employeeRepository.findByName(john.getName())).thenReturn(john);
        Mockito.when(employeeRepository.findByName(alex.getName())).thenReturn(alex);
        Mockito.when(employeeRepository.findByName("wrong_name")).thenReturn(null);
        Mockito.when(employeeRepository.findById(john.getId())).thenReturn(Optional.of(john
        Mockito.when(employeeRepository.findAll()).thenReturn(allEmployees);
        Mockito.when(employeeRepository.findById(-99L)).thenReturn(Optional.empty());
    }

    @Test
    public void whenValidName_thenEmployeeShouldBeFound() {
        String name = "alex";
        Employee found = employeeService.getEmployeeByName(name);
        assertThat(found.getName()).isEqualTo(name);
    }
```

Testing the REST controller (full stack, web server started)

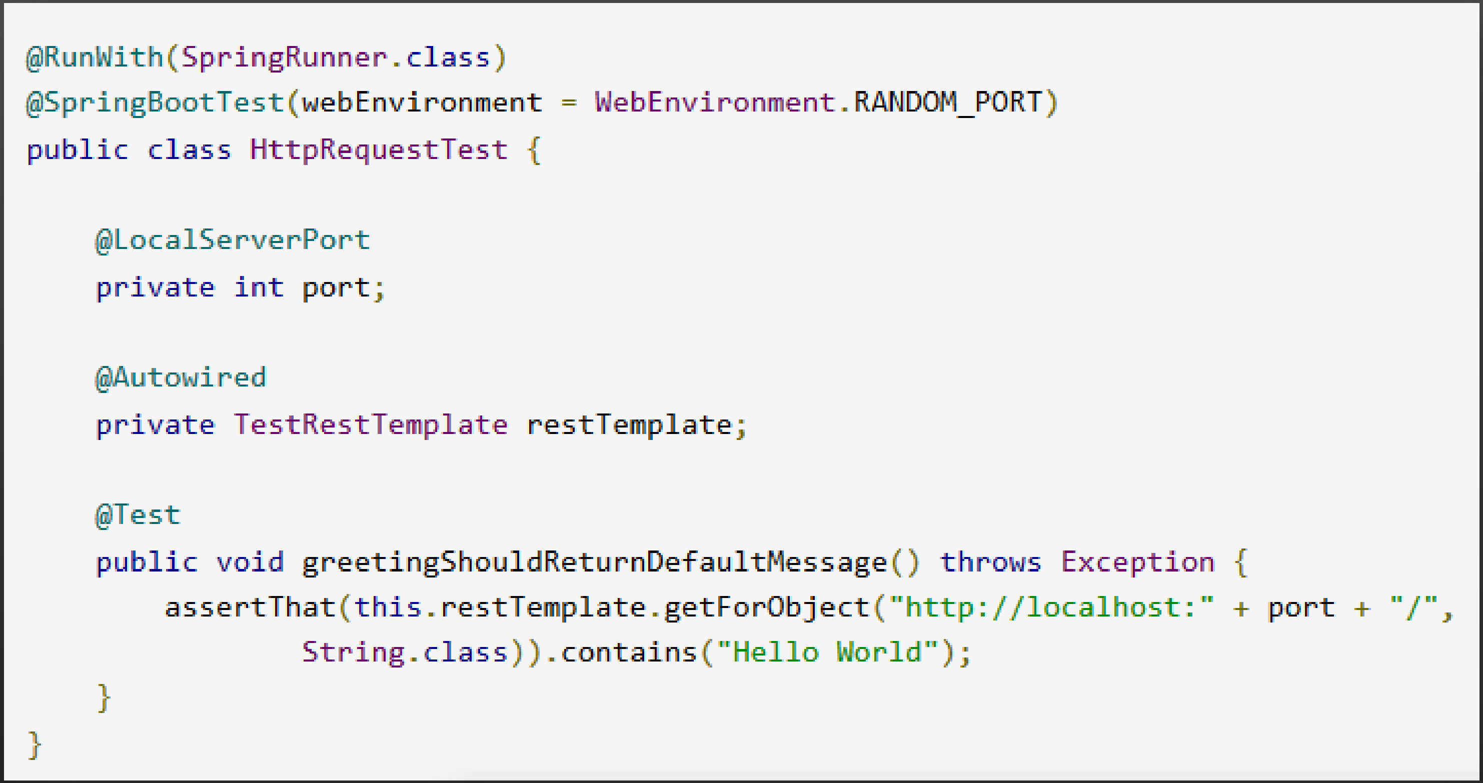


```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
public class HelloControllerIT {

    @LocalServerPort
    private int port;

    private URL base;

    @Autowired
    private TestRestTemplate template;

    @Before
    public void setUp() throws Exception {
        this.base = new URL( spec: "http://localhost:" + port + "/");
    }

    @Test
    public void getHello() throws Exception {
        ResponseEntity<String> response = template.getForEntity(base.toString(),
                String.class);
        assertThat(response.getBody(), equalTo( operand: "Greetings from Spring Boot!"));
    }
}
```

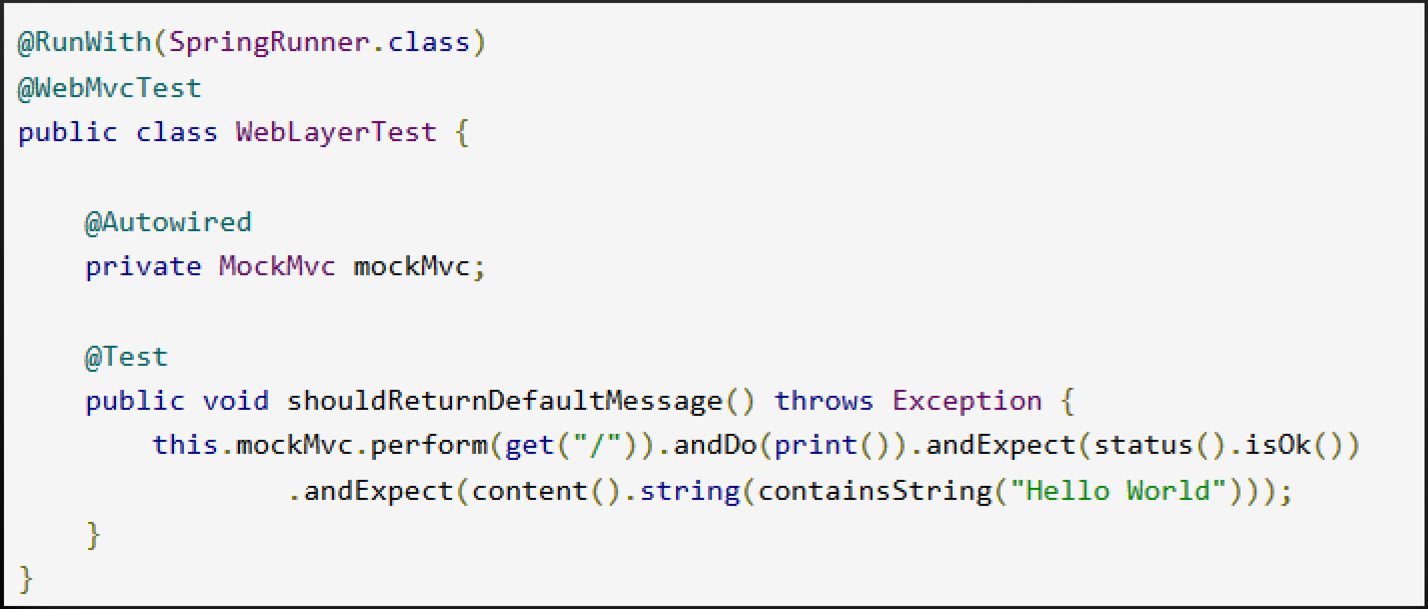
Testing the controller (mocked web context)



```
@RunWith(SpringRunner.class)
@SpringBootTest
@AutoConfigureMockMvc
public class HelloControllerTest {

    @Autowired
    private MockMvc mvc;

    @Test
    public void getHello() throws Exception {
        mvc.perform(MockMvcRequestBuilders.get( urlTemplate: "/").accept(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk())
                .andExpect(content().string(equalTo( operand: "Greetings from Spring Boot!")));
    }
}
```



```java
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
public class HttpRequestTest {

    @LocalServerPort
    private int port;

    @Autowired
    private TestRestTemplate restTemplate;

    @Test
    public void greetingShouldReturnDefaultMessage() throws Exception {
        assertThat(this.restTemplate.getForObject("http://localhost:" + port + "/",
                String.class)).contains("Hello World");
    }
}
```



```java
@RunWith(SpringRunner.class)
@WebMvcTest
public class WebLayerTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void shouldReturnDefaultMessage() throws Exception {
        this.mockMvc.perform(get("/")).andDo(print()).andExpect(status().isOk())
                .andExpect(content().string(containsString("Hello World")));
    }
}
```

References

Spring.io docs

Testing the web layer

Eugen Paraschiv's tutorials

Testing in Spring Boot