Testing in Spring Boot

UA/DETI/TQS
Ilídio Oliveira (ico@ua.pt)
v2019-04-23.

universidade de aveiro
departamento de eletrónica, telecomunicações e informática
deti

Popular testing tools for Java

Basics (unit)

- JUnit, TestNG
- Spock
- Hamcrest, AssertJ

Mocking objects behavior

- Mockito
- EasyMock

API Testing

- REST-Assured

Enterprise apps/backend

- Arquillian
- SpringBoot testing

Web/functional testing

- Selenium IDE

Story-driven (BDD)

- Cucumber

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));
```

AssertJ

Write fluent assertions:

```
// 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);
```

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"));
```

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);
```

Spring Boot testing

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

Provides:

- Curated dependencies
- Quick add dependencies
- “opinionated” auto-configuration of many components
- Sensible defaults

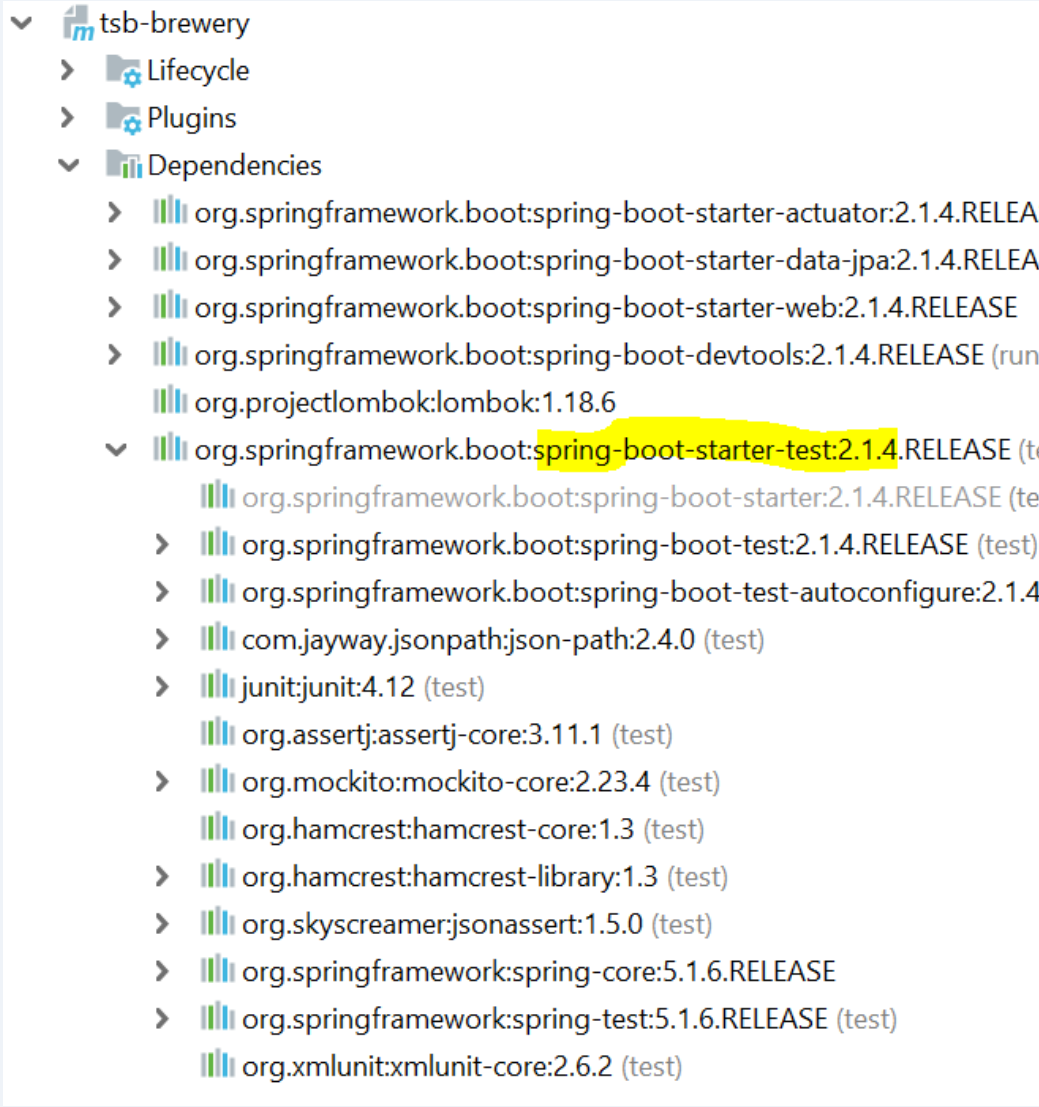
Extending SB filosophy to testing

Test features enabled with

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

Starter provides:

- Testing dependencies
- Testing auto-config



@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 (annotation declaration)

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.

SB concepts

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*.

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();
    }
```

Testing the service layer, isolating the persistence with mocks

```java
@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)

```java
@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)

```java
@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(webEnvironm
public class HttpRequestTes

    @LocalServerPort
    private int port;

    @Autowired
    private TestRestTemplat

    @Test
    public void greetingSho
        assertThat(this.res
                String.clas
    }
}
```

```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")));
    }
}
```

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

    @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