SPRING IO

#SPRING1018

[Workshop]

Testing every level of your spring microservices application

Jeroen Sterken & Kristof Van Sever

@jeroensterken

@vanseverk

Faros

- IT Consultancy company - Leuven (Belgium)
- Focused on Java and Spring Ecosystem
- Keeping up to date is essential
- Trainings
- Internal workshops
- Reactive Competence Center

www.faros.be
@FarosBelgium

Who are we?

- Jeroen Sterken

  - IT Coordinator @ Faros
  - Spring enthousiast
  - Pivotal Spring teacher

- Kristof Van Sever

  - Technical Consultant @ Faros
  - Reactive Competence center Lead
  - Pivotal Spring teacher

Agenda

9:00am: introduction (5min)
9:05am: Junit 5 (5min)
9:10am: BDD/cucumber (35min)
9:45am: wrap-up part 1 (5min)
9:50am: 10 min break (optional)

10:00am: Spring Cloud Contract (40min)
10:40am: wrap-up (10min)
10:50am: the end

Testing, it’s huge …

- Goal
  - Early detection of issues
- Many ways to test an application
  - Some are “cheaper” than others
  - Automated testing
  - Tests on different levels tend to be a requirement

Testing monoliths

- The entire application is pushed into one domain
- Testing levels
  - Unit Testing
  - Integration Testing
  - User Acceptance Testing

Testing moduliths

- Application separated into modules with clear boundaries (bounded contexts)
- Testing levels
  - Unit Testing
  - Component-Level Testing (through an interface)
  - Integration Testing
  - User Acceptance Testing

Testing microservices

- Application separated into modules with clear boundaries (bounded contexts) as separate applications

- Testing levels
  - Unit Testing
  - Component-Level Testing
  - Integration Testing
  - Communication/Contract-level Testing*
  - User Acceptance Testing

*Can also help with third party integration!

SPRING IO

#SPRING1018

[Presentation]

JUnit 5: What’s new?

Unit tests

- Everbody knows?
  - Testing at the (nearly) smallest level. But what IS the smallest level?
  - Sometimes different components working together
- What's important?
  - To prove to ourselves that our components work
  - Offer a more clear contract about what our components offer
- Come at a cost, so let's make our tests count

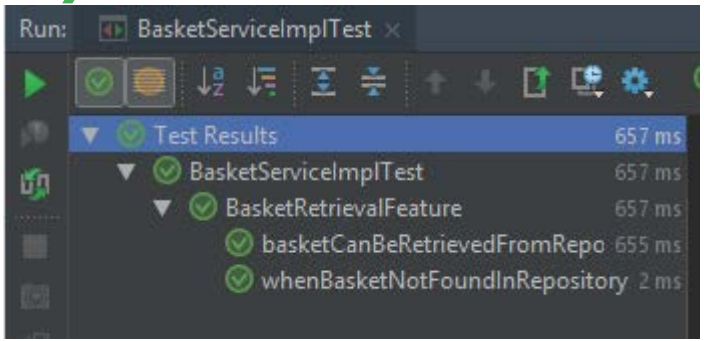
JUnit

- Unit Testing Framework
- Well supported (IDE’s, Maven, CI, ...)
- Integration Testing (JUnit runner)
- Let’s have q uick look at some cool things that JUnit 5 has to offer to developers...

JUnit 5, not a single library anymore

- **JUnit Platform:** Foundation for testing frameworks on the JVM
- **JUnit Jupiter:** New Programming and extension model for JUnit 5
- **JUnit Vintage:** TestEngine for Running JUnit 3 & 4 tests through the JUnit Platform

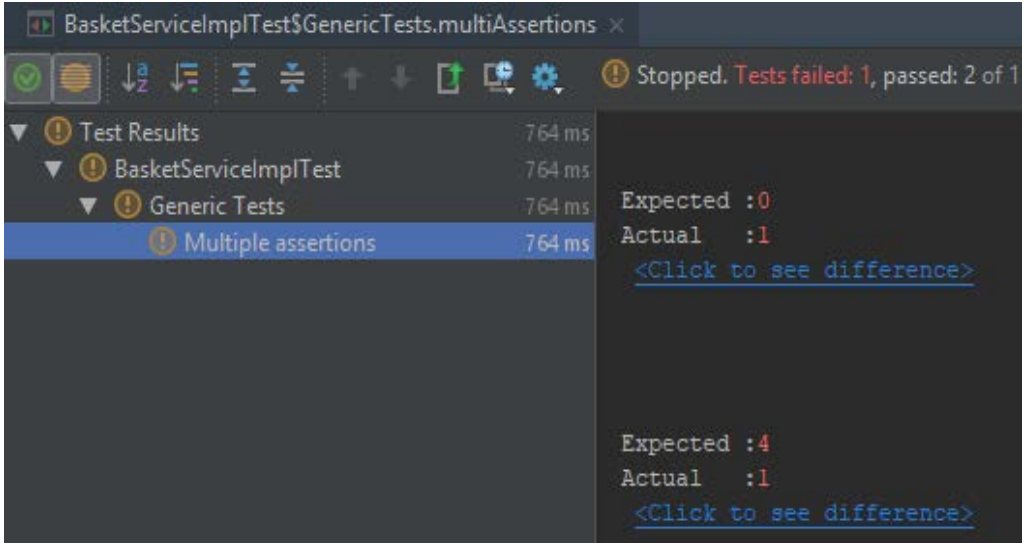
@Nested Tests (+ assertThrows)




```
@Nested
@DisplayName("BasketRetrievalFeature testing")
class BasketRetrievalFeature {

  @Test
  void whenBasketNotFoundInRepositoryAnIllegalArgumenExceptionIsThrown() {
    final int basketId = 123;

    when(basketRepositoryMock.findById(eq(basketId))).thenReturn(Optional.empty());

    assertThrows(IllegalArgumentException.class, () -> t.getExpectedBasket(basketId));
  }

  @Test
  public void basketCanBeRetrievedFromRepository() {
    ...
}
```

Better Lambda Support & Multi-Assertions

```
@DisplayName("Multiple assertions")
@Test
void multiAssertions() {
  int[] numbers = {0, 1, 2, 3, 4};
  Assertions.assertAll("Assumed Numbers",
      () -> assertEquals(numbers[0], 1),
      () -> assertEquals(numbers[2], 2),
      () -> assertEquals(numbers[4], 1)
  );
}
```



Parametrized Tests

- @ValueSource: values
- @MethodSource: from method
- @CSVSource: multiple literals
- @ArgumentSource: Custom

```
@DisplayName("Multiple assertions")
@Test
void multiAssertions() {
  int[] numbers = {0, 1, 2, 3, 4};
  Assertions.assertAll("Assumed Numbers",
      () -> assertEquals(numbers[0], 1),
      () -> assertEquals(numbers[2], 2),
      () -> assertEquals(numbers[4], 1)
  );
}
```

```
@DisplayName("Times two multiplying")
@ParameterizedTest(name = "{0} times 2 should be {1}")
@CsvSource({ "1, 2", "2, 4", "4, 8"})
void numbersShouldBeMultipliedByTwo(int firstNumber, int secondNumber) {
  assertEquals(secondNumber, firstNumber * 2);
}
```

Spring 5 JUnit 5 Integration Tests

- @SpringJUnitConfig = @ExtendWith(SpringExtension.class) + @ContextConfiguration

```
@SpringJUnitConfig(classes = SystemTestConfig.class)
public class UserBasketManagementImplTest {

  @Autowired
  private UserBasketManagementImpl t;

  @Test
  public void createNewBasketTest() {
    assertEquals(1, t.createNewBasket());
  }
}
```

SPRING IO

#SPRING1018

[Presentation + Workshop]

Behaviour Driven Testing through Cucumber

Unit Tests are sometimes too limited

- Prove a single unit works
- Does not prove that a combination of unit works

Integration and Component Tests are the answer

- Testing Multiple parts together
- Often along with databases, messaging systems, etc
- Not a test in isolation, everyone can play along

But who should really be writing these tests?

- **Developers:** Supply technical support
- **Domain experts:** Perhaps a bit surprising, but why not?

Presenting: Cucumber feature files (1)

- Describe Features and functionalities in “plain English”
- Follows BDD standards to describe scenarios (testing behaviour, rather than implementation)
- Arguments can be passed into the requirements
- Uses a language called “Gherkin”, a Business Readable Domain Specific Language (BRDSL)

```
#Feature file
Feature: A new empty basket can be created and filled with Tapas

  Scenario: Client creates a new Basket, and verifies it's empty
    When the user creates a new Basket
    Then the total number of items in the Basket with id 1 equals 0
```

Presenting: Cucumber feature files (2)

- Support for step keywords: **Given** (put in a known state), **When** (Describe Key Action), **Then** (Verification of result), **And/But** (append a when or then to an existing one)

- Support for translations (Feature -> Functionaliteit (Dutch))

```
Scenario Outline: eating Tapas
  Given there are <start> Tapas
  When I eat <eat> Tapas
  Then I should have <left> Tapas

  Examples:
    | start | eat | left |
    |  12   |  5  |  7   |
    |  20   |  5  |  15  |
```

Presenting: Cucumber Step Definitions

- Implementation of the different steps
- Used to glue steps to technical impl

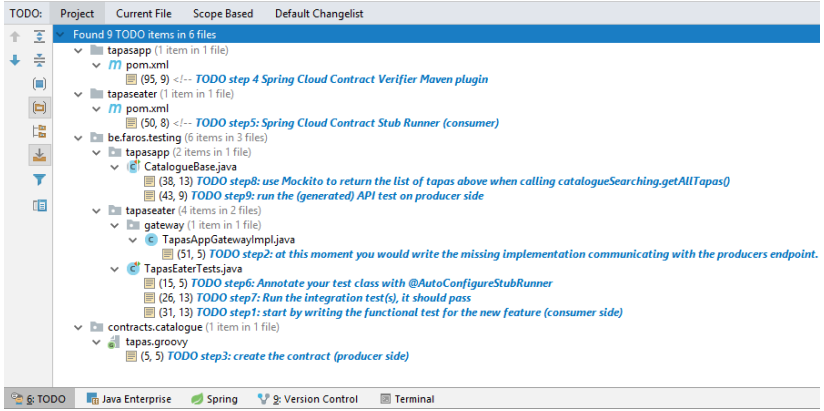
```
@When("^the user creates a new Basket$")
public void theUserCreatesANewBasket() {
  userBasketManagement.createNewBasket();
}

@Then("^the total number of items in the Basket with id (\\d+) equals (\\d+)$")
public void theTotalNumberOfItemsInTheBasketWithIdEquals(int basketId, int totalNumber) {
  assertThat(userBasketManagement.retrieveListOfAllTapasOrdersInBasket(basketId).stream()
      .map(to -> to.getAmount())
      .reduce(0L, Long::sum))
      .isEqualTo(totalNumber);
}
```

EXERCISE

Exercise: tapas Store

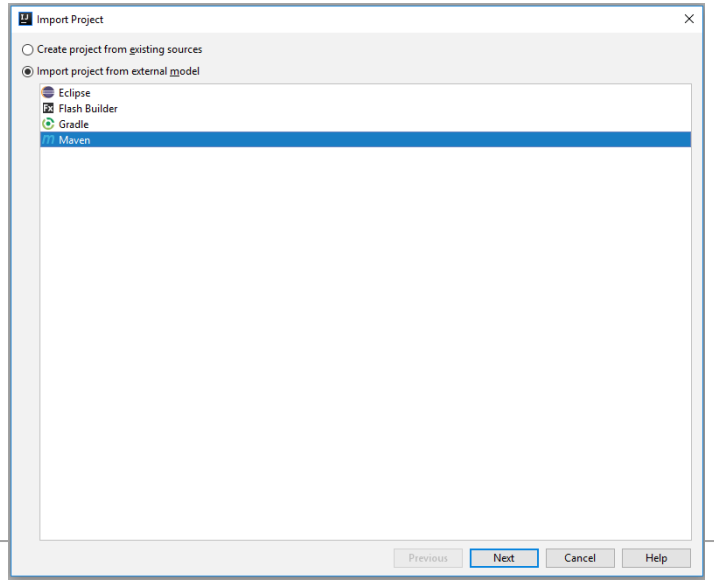
- Features:
  - Create a new basket
  - Add tapas to your basket
  - Calculate the total price

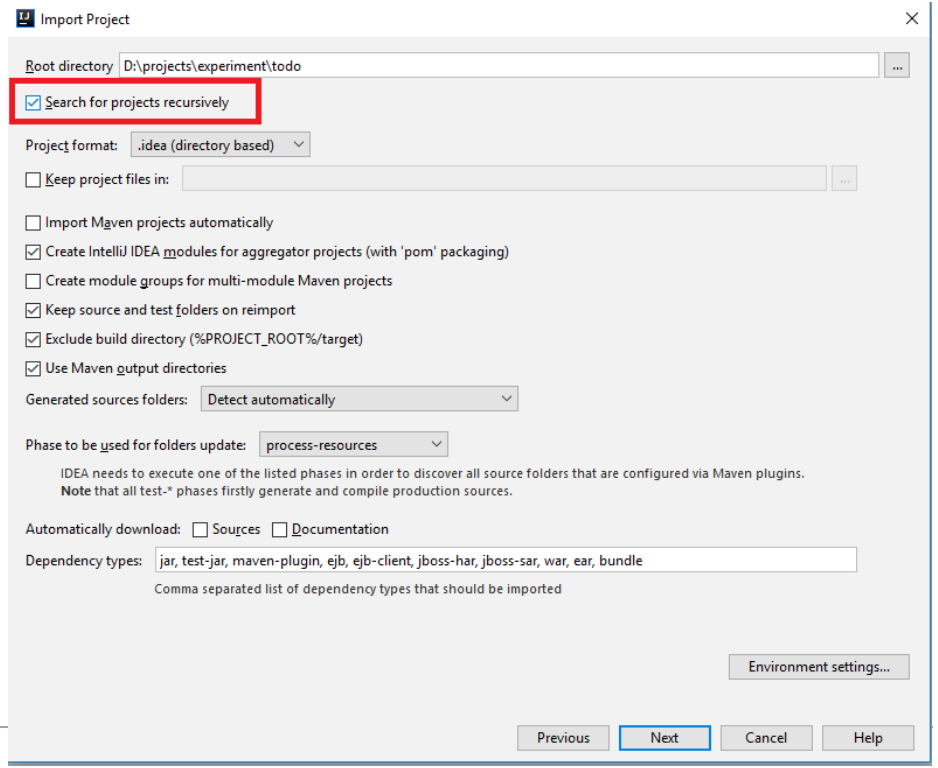
Exercise


TODO:
Project
Current File
Scope Based
Default Changelist
Found 9 TODO items in 6 files
tapasapp (1 item in 1 file)
pom.xml
(95, 9) <!-- TODO step 4 Spring Cloud Contract Verifier Maven plugin
tapaseater (1 item in 1 file)
pom.xml
(50, 8) <!-- TODO step5: Spring Cloud Contract Stub Runner (consumer)
be.faros.testing (6 items in 3 files)
tapasapp (2 items in 1 file)
CatalogueBase.java
(38, 13) TODO step8: use Mockito to return the list of tapas above when calling catalogueSearching.getAllTapas()
(43, 9) TODO step9: run the (generated) API test on producer side
tapaseater (4 items in 2 files)
gateway (1 item in 1 file)
TapasAppGatewayImpl.java
(51, 5) TODO step2: at this moment you would write the missing implementation communicating with the producers endpoint.
TapasEaterTests.java
(15, 5) TODO step6: Annotate your test class with @AutoConfigureStubRunner
(26, 13) TODO step7: Run the integration test(s), it should pass
(31, 13) TODO step1: start by writing the functional test for the new feature (consumer side)
contracts.catalogue (1 item in 1 file)
tapas.groovy
(5, 5) TODO step3: create the contract (producer side)
6: TODO
Java Enterprise
Spring
9: Version Control
Terminal

Exercise

- Import project (IDE)
  - Maven project





Exercise

- https://github.com/faros/bdd-cucumber.git