# CUCUMBER SELENIDE ALLURE

*Together form a test framework*

*By Jari "Jarru" Heikkilä 05/2018*

*Jari.henrik.heikkila@gmail.com*

# AGENDA

- Introduction part
- Demo part
- Explanation part

# INTRODUCTION PART

# ABOUT THIS PROJECT

## IT IS MY FREETIME PROJECT

- https://github.com/jaheikki/cucumber-selenide-allure

README.md

### Cucumber & Selenide & Allure demo

This is fully working example how run Web UI tests easily with Java with Selenide library (http://selenide.org/). In this example the demo application is set up by docker and docker-compose. The tests will be run by CucumberJVM framework to be able to use Gherkin (BDD) style as testcase descriptions (https://cucumber.io/docs/reference/jvm). And as cherry on top of a cake we use here Allure2 framework for producing world class test report (https://github.com/allure-framework/allure2).

# WHY THIS PROJECT

- I use these tools in my projects, so I would like to keep my knowledge "stored in disk", to get help when new work project starts.
- As a bonus I can share my knowledge to my colleagues, having similar challenges in their projects.

# GOAL FOR THIS DEMO

- Goal is to give initial understanding of tools Cucumber JVM, Selenide and Allure2 and how they can be used together as a framework.
- Goal is NOT to train you the details of tools used in this demo.
- I hope you will learn by doing, in this case by cloning this demo project from github and get the demo running in your PC.

# TOOLS USED IN DEMO

- IntelliJIdea IDE
- 'Cucumber for Java plugin' in IntelliJIdea
- Maven 3
- Java 8
- Cucumber JVM
- Selenide
- Several helper libraries, see pom.xml

# WHY THIS TECHNOLOGY

## JAVA

- The point REALLY is using:
  - JAVA(8)
  - MAVEN(3)
  - IntelliJ Idea IDE (or Eclipse)
  - jUnit or TestNG

  Basicly everything beyond this is nonsense…

# WHY THIS TECHNOLOGY

## CUCUMBER (FOR JAVA)

- Cucumber is a collaboration tool and not just test automation tool, so if it isn't use like that, then it is less relevant. However in case of acceptance testing then Cucumber will still be relevant, because:
- …Cucumber is a way to achieve the Gherkin language (BDD style) and that enables making the test and documentation in same place (file). This will help the stakeholders, customers and managers understand what will be tested.

# WHY THIS TECHNOLOGY

## SELENIDE

- Selenide is a way to make the the WEB UI test on top of Selenium by using Java as programming language.
- Very easy syntax & lot of additional automation -> speeds the WEB UI test development.

# WHY THIS TECHNOLOGY

## ALLURE

- Graphical presentation how tests were executed.
- Allure is just way to make the ‘stakeholder reports’ and support the idea of making code and documentation in same place together with Cucumber feature file.

# CUCUMBER

- The world's most misunderstood collaboration tool
- Aslak HellesøyMar 3, 2014
- https://cucumber.io/blog/2014/03/03/the-worlds-most-misunderstood-collaboration-tool
- In 2003 I became part of a small clique of people from the XP community who were exploring better ways to do TDD. Dan North named this BDD. The idea was to combine automated acceptance tests, functional requirements and software documentation into one format that would be understandable by non-technical people as well as testing tools.

# CUCUMBER EXAMPLE

- Gherkin language (=BDD style) example:

```
@all
@simple_test
Feature: SimpleTests

  Scenario: Test wikipedia
    Given browser should be open in Wikipedia page
    When i write browserstack to search field
      And i press ENTER
    Then i am able to see BrowserStack wikipedia page
```

# CUCUMBER JVM

## THROUGH ALLURE DEPENDENCY

- <**dependency**>
  <**groupId**>io.qameta.allure</**groupId**>
  <**artifactId**>allure-cucumber-jvm</**artifactId**>
  <**version**>2.0-BETA22</**version**>
</**dependency**>
- Yes, Allure 2.0 is still BETA but it works!
- Note that allure-cucumber-jvm dependency brings both Allure 2 and Cucumber JVM dependencies
- Note that Allure needs properly configured “maven-surefire-plugin” and “allure-maven” plugin, see pom.xml

# CUCUMBER JVM

## JUNIT RUNNER

```
@RunWith(Cucumber.class)
@CucumberOptions(
        features = { "classpath:acceptancetests" },
        glue = {"teststepdefinitions"}
)

public class AcceptanceRunnerTest {
```

# CUCUMBER JVM

## FEATURE FILE

```
@order
@all
@link.mylink=jvm
Feature: Microservice acceptance tests

  This test suite verifies that ordering a product from E-commerce Manager UI
  is executed successfully with a new customer.

  @severity=blocker
  @tmsLink=REQ-405
  @link=https://yle.fi/
  Scenario: Order a product from a catalog
    Given E-commerce Manager ui should be open
     And order by Teemu Selanne should not exist
     And product Torspo should not be in the catalog through REST API
     And customer Teemu Selanne should not exist through REST API
     And product Torspo is added to the catalog with price 119.0
     And customer Teemu Selanne is added
    When I order product Torspo as customer Teemu Selanne
    Then I can verify my order of Torspo with price 119.0 by customer Teemu Selanne
```

# CUCUMBER JVM

## TEST STEP DEFINITIONS

- **In feature file:**
  - **And** product **Torspo** is added to the catalog with price **119.0**

- **In test step definitions file:**
  - ```
    @And("product (.*) is added to the catalog with price (.*)")
    public void productIsAddedToTheCatalog(String catalogItem,
    String catalogItemPrice) {
        …;
    ```
    ```

# CUCUMBER JVM

## MULTIPLE TEST STEP DEFS

- For very small project is enough to have just one step definition class file, but most likely you end up situation where steps are needed to be divided into several classes.
- Do NOT do this: "Each feature file has a separate step definition file"
- https://github.com/cucumber/cucumber/wiki/Feature-Coupled-Step-Definitions-(Antipattern)
- Instead use e.g. domain specific division: https://github.com/cucumber/cucumber/wiki/Step-Organization

# CUCUMBER JVM

## SHARING STATE BETWEEN TEST STEP DEF FILES 1/4

- Some dependency injection method needed, scary term?, no need to worry:
- In this demo we use very simple Constructor Dependency Injection, CDI by using PicoContainer (see dependency from pom.xml).
- http://www.thinkcode.se/blog/2017/04/01/sharing-state-between-steps-in-cucumberjvm-using-picocontainer
- https://www.youtube.com/watch?v=upgTg_JiNbY

# CUCUMBER JVM

## SHARING STATE BETWEEN TEST STEP DEF FILES 2/4

- Shared World object class:

```
package cucumber_dependency_injection;

import microservice.pages.ProductsPage;

public class World {

    public ProductsPage msMainPage;

}
```

# CUCUMBER JVM

## SHARING STATE BETWEEN TEST STEP DEF FILES 3/4

- Set value to shared object for keeping the state across classes:

```
public class ProductStepDefs {

    private ProductsPage mainPage;
    private World world;

    public ProductStepDefs(World world) {
        this.world = world;

    }

    static org.apache.log4j.Logger log = org.apache.log4j.Logger.getLogger(
            SFTPService.class.getName());

    @And("E-commerce Manager ui should be open")
    public void eCommerceManagerUiShouldBeOpen() {
        log.info(printMethodName());

        world.msMainPage = Selenide.open(MsVariables.microserviceHost, ProductsPage.class);

    }
```

# CUCUMBER JVM

## SHARING STATE BETWEEN TEST STEP DEF FILES 4/4

- Using preserved state from shared World object:

```
public class CustomerStepDefs {

    private ProductsPage mainPage;
    private World world;

    public CustomerStepDefs(World world) {
        this.world = world;

    }

    @And("customer (.*) (.*) is added")
    public void customerIsAdded(String firstname, String lastname) {
        printMethodName();

        world.msMainPage.navigateToCustomersPage()
                .addCustomer(firstname,lastname)
                .navigateBackToProductsPage();
    }
```

# SELENIDE

- Selenide is a framework for WEB UI test automation powered by Selenium WebDriver that brings the following advantages:
  - Concise fluent API for tests
  - Ajax support for stable tests Powerful selectors
  - Simple configuration
- http://selenide.org/quick-start.html
- http://github.com/codeborne/selenide/wiki/Selenide-vs-Selenium

# SELENIDE EXAMPLE

```java
import com.codeborne.selenide.Condition;
import com.codeborne.selenide.Selenide;
import org.junit.Test;
import org.openqa.selenium.By;

import static com.codeborne.selenide.Selenide.$;

public class SimpleSelenideJunitTests {

    @Test
    public void wikipediaTest() {

        String searchWord = "browserstack";

        Selenide.open( relativeOrAbsoluteUrl: "https://en.wikipedia.org/wiki/Main_Page");

        $(By.xpath("//a[contains(@title,'Visit the main page')]")).shouldBe(Condition.visible);
        $(By.xpath("//input[contains(@placeholder,'Search Wikipedia')]")).shouldBe(Condition.visible).val(searchWord);
        $(By.xpath("//input[contains(@placeholder,'Search Wikipedia')]")).shouldBe(Condition.visible).pressEnter();
        $(By.xpath("//h1[contains(text(),'BrowserStack')]")).shouldBe(Condition.visible);
    }

}
```

# CUCUMBER JVM AND SELENIDE TOGETHER

```
@Given("^browser should be open in Wikipedia page$")
public void browserShouldBeOpenWikipedia() {
    printMethodName();

    Selenide.open( relativeOrAbsoluteUrl: "https://en.wikipedia.org/wiki/Main_Page");
    $(By.xpath("//a[contains(@title,'Visit the main page')]")).shouldBe(Condition.visible);
    Selenide.sleep( milliseconds: 500);

}

@When("^i write (.*) to search field$")
public void iWriteBrowserstackToSearchField(String searchWord) {
    printMethodName();

    $(By.xpath("//input[contains(@placeholder,'Search Wikipedia')]")).shouldBe(Condition.visible).val(searchWord);
    Selenide.sleep( milliseconds: 500);

}
```

# SELENIDE

## WITHOUT MANUAL WEBRIVER BINARY SETTING

- <dependency>
  <groupId>com.codeborne</groupId>
  <artifactId>selenide</artifactId>
  <version>4.9.1</version>
</dependency>

- This is included! (mvn dependency:tree)
  - <dependency>
    <groupId>io.github.bonigarcia</groupId>
    <artifactId>webdrivermanager</artifactId>
    <version>2.0.1</version>
    </dependency>

# SELENIDE

## PAGE OBJECTS WHEN OPEN BROWSER

```
@And("E-commerce Manager ui should be open")
public void eCommerceManagerUiShouldBeOpen() {
    log.info(printMethodName());

    world.msMainPage = Selenide.open(MsVariables.microserviceHost, ProductsPage.class);

}
```

# SELENIDE

## PAGE OBJECTS IN USE

❖ Cucumber step definition with Selenide page object style of UI navigation:

```
@And("product (.*) is added to the catalog with price (.*)")
public void productIsAddedToTheCatalog(String catalogItem, String catalogItemPrice) {
    log.info(printMethodName());

    world.msMainPage.navigateToAddProductPage()
            .addCatalogItem(catalogItem, catalogItemPrice)
            .navigateBackToProductsPage();
}
```

# SELENIDE

## BROWSER CONFIGURATIONS

- -Dselenide.browser=firefox (or just -Dbrowser=firefox)
  - Or in code Configuration.*browser*=**"chrome"**;
- Custom browser:-

Dbrowser=microservice.browser.Firefox53WindowsRemoteDriverProvider

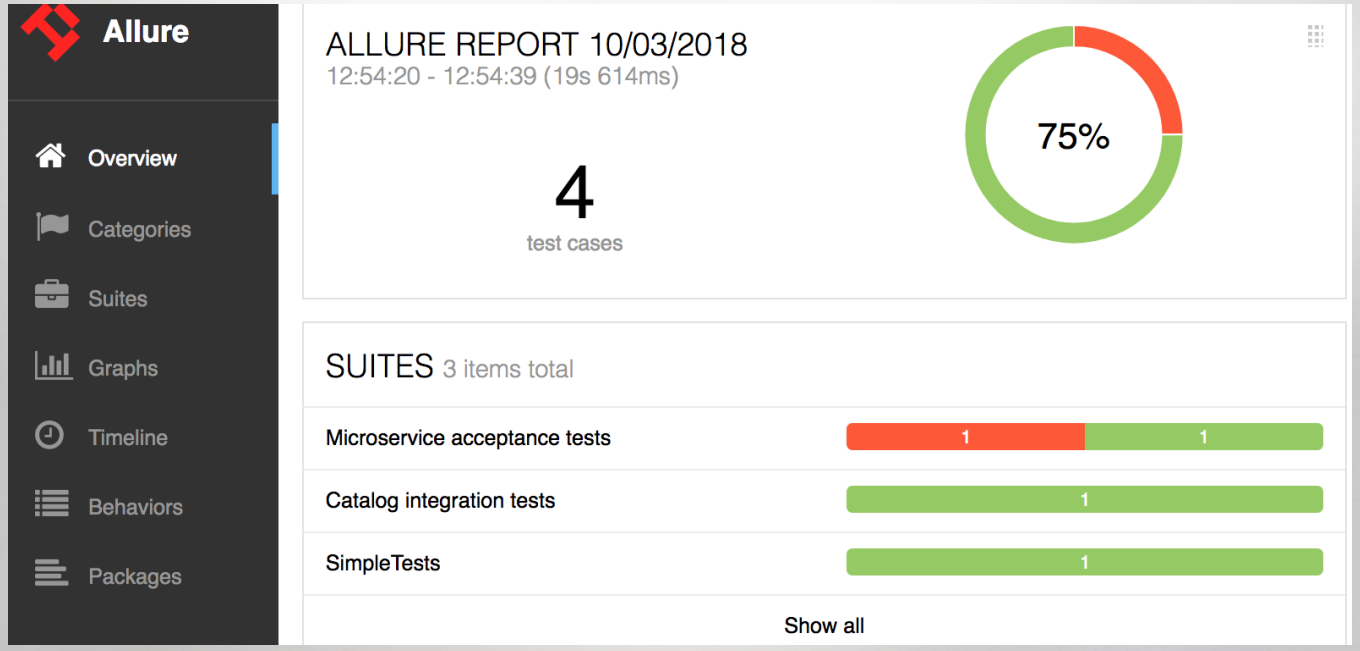
- -Dselenide.browser-size=375x667
- -Dremote=http://<hub host>/wd/hub

# ALLURE

- Allure Framework is a flexible lightweight multi-language test report tool
- https://docs.qameta.io/allure/
- https://github.com/allure-framework/allure-java/blob/master/allure-cucumber-jvm/src/test/resources/features/test.feature

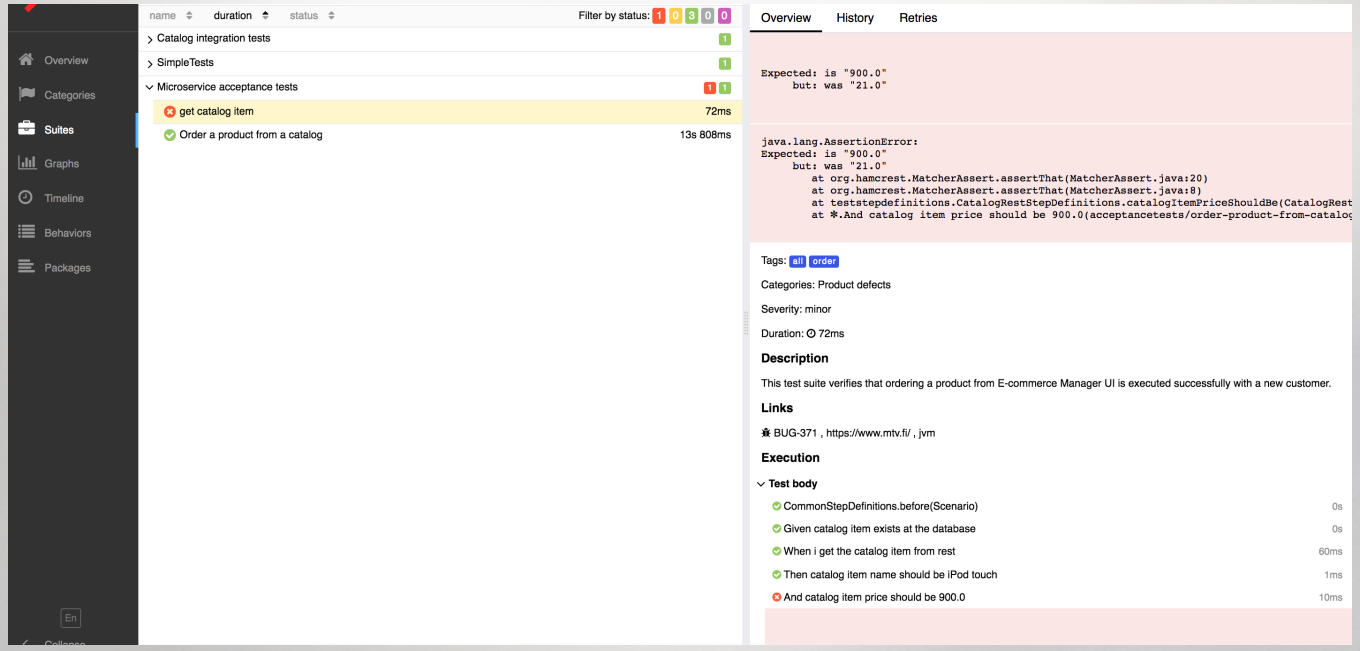
# ALLURE REPORT

## OVERVIEW



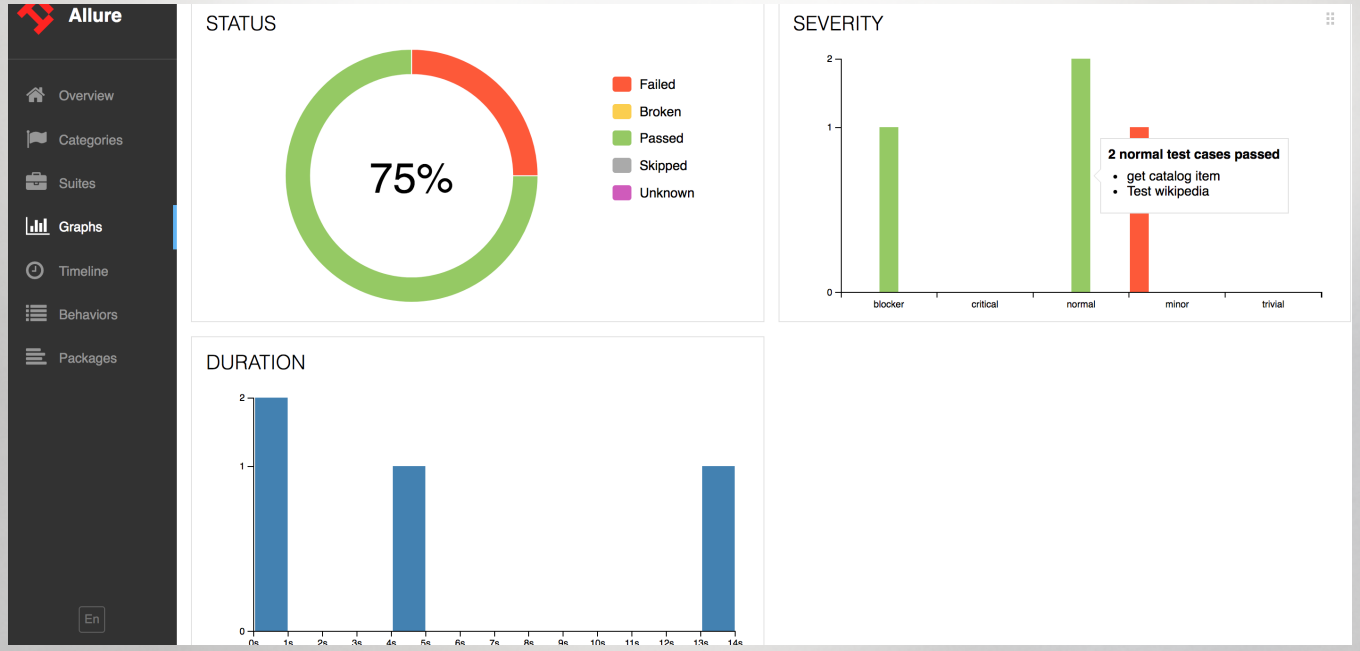
# ALLURE REPORT

## TEST STEP REPORT



# ALLURE REPORT

## GRAPHS



# CUCUMBER AND ALLURE

## TAGS

- With tags you can run whole feature file or just one scenario
- You can run Cucumber test by tag e.g. **-Dcucumber.options="--tags @order"**

```
@order
@all
Feature: Microservice acceptance tests
```

# CUCUMBER AND ALLURE

## SEVERITY, ISSUE&TMS LINKS, PLAIN LINKS

❖ You can add description, severity, links to issues and test management system (tms) as well as plain links visible in report. Note: for issues and tms use allure.properties for adding link patterns.

```
@link.mylink=jvm
@order
@all
Feature: Microservice acceptance tests

  This test suite verifies that ordering a product from E-commerce Manager UI
  is executed successfully with a new customer.

  @severity=blocker
  @tmsLink=REQ-405
  @link=https://yle.fi/
  @order1
  Scenario: Order a product from a catalog
```

# CUCUMBER AND ALLURE

## ALLURE.PROPERTIES

- For link patterns e.g for @tmsLink=REQ-405

allure.properties

```
1  allure.link.mylink.pattern=https://cucumber.io/docs/reference/{}
2  allure.link.issue.pattern=https://jira.public.mycompany.com/issue/{}
3  allure.link.tms.pattern=https://jira.public.mycompany.com/tms/{}
```

# THE DEMO PART

- Based on running the script:  cucumber-demo.sh
  - Note: Remember to source needed env variables (e.g. in this demo source ~/bs_creds.sh (personal credentials not published in github))
  - Note: Start VNC viewer before demo script

# CONTINUOS INTEGRATION DEMO

## FOR RUNNING CUCUMBER JVM TESTS

- You can run run tests with Maven in any CI system, like Jenkins, GoCD etc and get the test running automatic.
- In this demo GoCD is used when 3 testcases are running WEB UI tests PARALLELLY to BrowserStack.
- See ../cucumber-selenide-allure/cucumber.andon.yml as just example when using GoCD (with some company specific enlargement in yaml format).
- GoCD Yaml plugin spec:

https://github.com/tomzo/gocd-yaml-config-plugin

- NOTE: Setupping CI/GoCD system is not part of this demo!

# EXPLANATION PART

## LOOKING UNDER THE HOOD

# IDE VS MAVEN 1/2

- Develop and run tests in IDE, in this case IntelliJ Idea.
- In IDE you can:
  - Run all tests in all feature files from Cucumber jUnit runner class
  - Run single feature file (all scenarios inside single file)
  - Run single scenario (single test)
  - Add VM options e.g. '-Dbrowser=firefox' in Run/ Debug configuration in IntelliJ Idea.
  - Debug, use of breakpoints etc.

# IDE VS MAVEN 2/2

- Maven is for:
  - Running tests in CI environment like Jenkins or GoCD
  - Getting Allure test report automatically.
  - Run specific amount of tests is based on Cucumber tags (e.g. --tags @simple_test).

# RUN CUCUMBER

## WITH TAG @ORDER AND GENERATE REPORT

- Mvn clean install -Dcucumber.options="--tags @order"

# CUSTOM BROWSER

## AND LOCAL GRID

❖ -

Dselenide.browser=microservice.browser.FirefoxLinuxGridDriverPro

vider -Denv=local-grid

- -Denv=local-grid means using local-grid.properties file as parameter source for tested service
- Note: In this case the UI test is run inside docker container so the UI is not found from localhost
  - microservice_host=http://localhost:8080 vs
  - **microservice_host=http://zuul:8080 #this is used**

# CUSTOM BROWSER

## WHEN RUNNING TO BROWSERSTACK

❖ -

Dbrowser=microservice.browser.SamsungGalaxyS7_V6_0_RemoteDriverProvider −Dlocal

- -Dlocal = true means running tests in BrowserStack towards local UI
- Note: -Denv not needed to define since local.properties is used by default

# USEFUL LIBRARIES

## TYPESAFE LIBRARY FOR PROPERTIES

- <dependency>
  <groupId>com.typesafe</groupId>
  <artifactId>config</artifactId>
  <version>1.2.0</version>
  </dependency>

```
public class MsVariables {

    private static final String env = System.getProperty( key: "env", def: "local");
    private static final String propertiesFile = env + ".properties";
    private static final Config config = ConfigFactory.load(propertiesFile).withFallback(ConfigFactory.load( resourceBasename: "general.properties"));
```

# USEFUL LIBRARIES

## SCP & SFTP

- See SSHService and SFTPService classes in helper package
- See SSH examples in demo package

```
public void iTransferCSVFilesToSFTPServer() throws SftpException {
    log.info(printMethodName());

    String remoteCSVDir = "/csvdir/csvfiles";
    SFTPService sftp = new SFTPService( user: "sshuser",  host: "192.168.0.10",  port: 22,  sshPrivatekeyFilePath: "privateKeyFile");

    //delete old folder if exists
    sftp.removeFolderRecursively(remoteCSVDir);

    //create new directory for csv files
    sftp.createDirectory(remoteCSVDir);

    //upload new files
    sftp.uploadFiles(remoteCSVDir,  ...localFiles: "src/test/resources/mycsvfile1",
            "src/test/resources/mycsvfile2"
    );

    sftp.exit();
}
```

# USEFUL LIBRARIES

## SPRING-JDBC STUFF FOR DB CONNECTIONS

- See dependencies from pom.xml
- See .. src/test/resources/customer.sql

```
public CustomersPage addCustomer(String firstname, String lastname) {
    printMethodName();

    MyDatasource customerDatasource=new MyDatasource(mariaDBJDBCUrl,mariaDBJDBUser,mariaDBJDBCPassword);

    customerDatasource.runSqlScript("customer.sql");

    this.customerJDBCTemplate = new JdbcTemplate(customerDatasource.getDataSource());
    String email = SQLCommon.sqlQueryForString(customerJDBCTemplate, SQL: "SELECT email FROM customerdb.customer WHERE lastname='"+lastname+"' AND firstname='"+firstname+"';");
    String street = SQLCommon.sqlQueryForString(customerJDBCTemplate, SQL: "SELECT street FROM customerdb.customer WHERE lastname='"+lastname+"' AND firstname='"+firstname+"';");
    String city = SQLCommon.sqlQueryForString(customerJDBCTemplate, SQL: "SELECT city FROM customerdb.customer WHERE lastname='"+lastname+"' AND firstname='"+firstname+"';");
```

# USEFUL LIBRARIES

## JERSEY AND JACKSON FOR REST SERVICES

- See .. src/test/resources/acceptancetests/get-catalog-item.feature
- See .. src/main/java/microservice/helper/RESTService.java

```
public class RESTService {

    public static String getJsonFromUrl(final String url) {
        printMethodName();

        try {

            ClientConfig clientConfig = new DefaultClientConfig();
            clientConfig.getClasses().add(JacksonJaxbJsonProvider.class);

            Client client = Client.create(clientConfig);
            client.addFilter(new LoggingFilter(System.out));

            WebResource webResource = client.resource(url);

            ClientResponse response = webResource.accept(MediaType.APPLICATION_JSON).get(ClientResponse.class);

            System.out.println("Response status:" + response.getStatus());

            if (response.getStatus() != 200) {
                throw new RuntimeException("Failed : HTTP error code : " + response.getStatus());
            }

            return response.getEntity(String.class);
```

# THE END

- Now already, another hour of demo, please…