**Intro**

Selenium webdriver

Cucumber

Jenkins

Java

**Setup**

Environment setup

**Java**

Java JDK

Een JDK biedt de basis die je nodig hebt om te kunnen ontwikkelen met Java: compiler, interpreter, archiver.

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Install Java SE Development Kit 8u181

**System variables**

Noodzakelijk, want wanneer je een java programma runt en dit niet doet, moet je elke keer het pad naar de installatiefolder opgeven. Door dit te doen weet het systeem waar java geïnstalleerd is zodat hij java apps kan runnen.

1. New system variable: JAVA\_HOME, C:\Program Files\Java\jdk1.8.0\_181 (pad naar installatiefolder JDK)
2. Path -> new: plak het pad naar de installatiefolder van de JDK, type “/bin” erachter.
3. Open CMD -> java -version (open als admin)

**Maven**

Build management tool -> POM File -> dependency – haalt relevante jar files voor je project op van het web. Dit creëert een centraal punt waar jar files opgehaald kunnen worden ipv gefragmeteerde libraries.

<https://maven.apache.org/download.cgi>

Kies voor binary zip archive.

Maak directory op C:/Program Files/Dev\_programs/

Unzip maven in deze directory

Set system variables:

1. New system variable: MVN\_HOME, C:\Program Files\Dev\_Programs\apache-maven-3.5.4
2. New system variable: M2, C:\Program Files\Dev\_Programs\apache-maven-3.5.4\bin
3. Path bewerken -> new: %M2%

Open cmd: mvn -v laat zien welke versie je hebt.

**Eclipse**

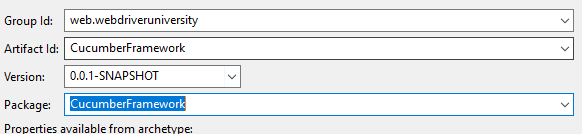
Eclipse is je IDE. Met eclipse kun je applicaties ontwikkelen en testen integreren.

Download en installeer eclipse photon. Kies voor Java. (eclipse for java developers)

Maak een folder op je desktop: CucumberFramework. Dit wordt de workspace van je project.

Begin een nieuw project > Maven Project

Set group ID, artefact ID en package naar de onderstaande waarden



Group Id: naamgeving voor meerdere modules binnen project

Artefact Id: naam van de gecompileerde jar

Setup eclipse met maven:

Ga naar window > preferences > maven > add new installation > {installatiefolder mvn}

Setup java met eclipse:

Ga naar window > preferences > java > add new installation > java vm > {installatiefolder javajdk}

Plaats deze link in default VM arguments: -Dmaven.multiModuleProjectDirectory=M2\_HOME

Het gedrag van de virtual machine wordt beinvloed door dit argument.

Selecteer juiste jdk en sluit af.

Set het correcte build path:

Rechtermuisklik op project

Properties > java build path > library > edit > alternate build path > finish

\*De errors onderin zijn weg.

Als er een compliance error is, ga naar Properties > Java Compiler > Compiler compliance level. Set level naar 1.8.

Setup m2 repo (nodig zodat wanneer je je project update, de files in deze folder terecht komen)

Properties > java build path > library > add variable > select m2 repo

(deze folder is automatisch door maven aangemaakt: Windows – C:\Documents and Settings\{your-username}\.m2

Om te testen: rechtermuisklik op de pom file > run as > maven install. Check je console. Als de depency van junit problemen veroorzaakt, verwijder deze.

Testng

Packages die mogelijkheden bevatten voor testen. Aanvulling op junit.

Help > marketplace > zoek testng > download > confirm

Restart eclipse

Rechtermuisknop op een package > new > check of testng ertussen staat

Cucumber

Ga naar Help > marketplace > search for cucumber > install natural

Restart Eclipse

Ga naar <https://cucumber.github.io/cucumber-eclipse/> en kopieer de link

Ga naar help > new install > plak de link > add (geef een naam)

Zet een vinkje bij cucumber

\*bij java error: set de correcte JDK in window > preferences > installed JRE’s / execution environment

\*bij een junit error: ga op een junit verwijzing staan (bijv test) en klik in de suggesties op add junit to build path.

Tweak Eclipse?

Help > Market place > color theme en darkest color theme

Restart

Chromedriver

<https://chromedriver.storage.googleapis.com/index.html?path=2.41/>

Kies voor chromedriver\_win32.zip

Download chromebrowser (32 bit browser)

Pak chromedriver uit op je bureaublad

**Maven project**

Maak een nieuwe folder aan op je desktop: CucumberFramework.

Maak een new project aan. Kies voor een maven project.

Open eclipse en selecteer de bovenstaande map als workspace.

Voeg Selenium Java toe aan je POM file: ga naar mvnrepository.com en zoek op Selenium.Kies voor 3.1.1.. Plak de xml code in je POM file tussen <dependencies>

Update het project: rechtermuisknop project > maven > update project > ok

\*alles wat via maven wordt opgehaald wordt opgeslagen in je m2 repo

Plak het XML in de bijlage in de pom file onder properties.

Wijzig deze regel naar de juiste java versie

<!--<executable>C:\Program Files\Java\jdk1.8.0\_131\bin\javac.exe</executable>-->

Rechterklik op project > run as > maven clean

Rechterklik op project > maven > update project > vink de box bij “force” > klik ok

Er zijn geen errors.

**Maven commando’s**

Clean project – verwijderd gecompilede files (gebruik commando bij slecht te verklaren errors)

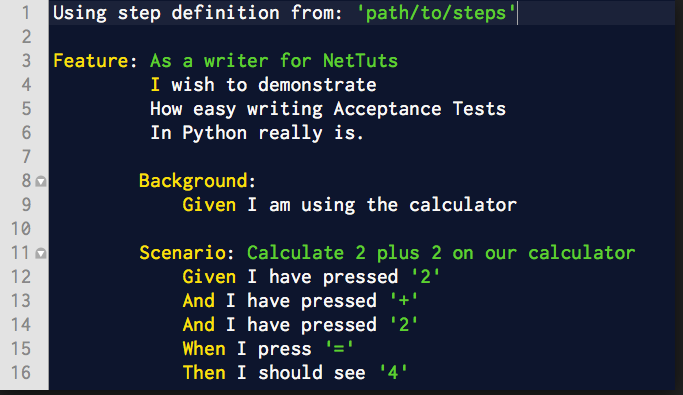
Update project – download relevant jar files

**Gherkin**

Een taal leesbaar voor de business die beschrijft wat de software doet

Bestaat uit:

* Features: vat business requirements van een bepaald deel van een applicatie, bijvoorbeeld “inloggen op de applicatie.” Een feature bestaat uit:



* Step Definitions: voert de stappen uit (in deze cursus zijn de stappen in Java)

**Feature file maken:**

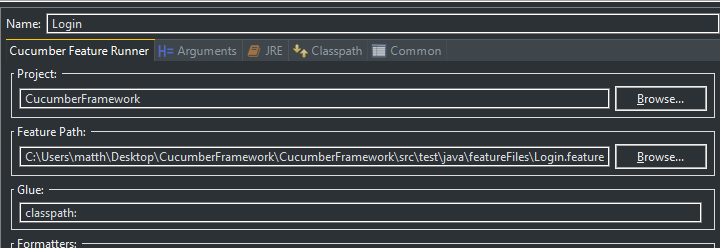
Maak een nieuwe package in src/test/java in cucumberFramework genaamd “cucumberFramework .featureFiles”

Maak in dit package een file aan: Login.feature (Rechter muisknop op package featureFiles: new file.

Plak bijlage twee in het Login feature file

Maak steps aan die de stappen gaan uitvoeren -> rechtermuisknop op Login.feature file > run as > run configurations > Run (naar onderstaand voorbeeld)

De stappen behorend bij de features worden weergegeven in de console.



\*bij melding “eclipse blocked the user operation is waiting” eclipse opnieuw opstarten en het project schoonmaken.

**Step Definitions**

Step definitions voert de stappen van de feature file uit.

Maak een steps package aan in de cucumberFramework package. Noem de package “cucumberFramework .steps”

Maak in de package een java class file aan. Noem deze LoginSteps.

Plaats de code die gegenereerd is vanuit de feature file in class Login Steps (vanaf “given”)

Deze code maakt gebruik van cucumber packages. Deze moeten geimporteerd worden om de code te doen laten werken. Met ctrl + shift + o haal je de benodigde packages op.

Run de configurations van de feature file nogmaals. Je ziet dat er vijf scenario’s geskipped zijn, en 1 ‘pending’ is. Dat komt doordat in de ‘steps’ default een pending exception is aangemaakt. Verwijder die voor elke stap.

Zet in iedere stap een println statement (System.out.println(“Boodschap”). Gebruik bijvoorbeeld de naamgeving van de verschillende stappen.

Kijk in de console of de output klopt.

**Runner class**

Er zijn vaak meerdere feature files en meerdere step classes.

Met een runner class ben je in staat meerdere feature files te draaien.

Maak een nieuwe package “runner” aan in CucumberFramework.

Maak een class aan en noem deze “MainRunner.”

Om te illustreren dat een runnerclass meerdere feature files kan runnen maken we een tweede feature file aan. Kopieer login.feature in dezelfde package.

Verander in het nieuwe feature file de namen van het scenario en de stappen, zodat de namen uniek zijn in het project.

Run configurations van de nieuwe feature file (login2.feature).

Zet de stappen uit de console in de steps file (onder de eerder gegenereerde stappen).

Verwijder de pendingexceptions en plaats een willekeurig print statement tussen de brackets van de verschillende steps.

Open de runner class

Zet de code in bijlage 3 boven de brackets van de class file.

Features specificeert in welke package de feature files staan.

Glue wijst naar de locatie can de step files.

Monochrome zorgt ervoor dat de output van de stappen beter leesbaar is in de console.

Met tags kan je scenario’s tests geven die gedraaid moeten worden via de runner class.

In plugin specificeren we hoe het rapport eruit moet komen te zien. Pretty zorgt voor verhoogbare leesbaarheid, met “html” krijgen we een html report, met json krijgen we een json report (sommige systemen lezen enkel json bestanden),

Rechterklik op de runnerclass en voer hem uit als junit test. Check in de console of de juiste feature files zijn gerund.

Rechterklik op het project en refresh. Ga dan naar de target folder. Open het report met een browser. Zo kun je zien dat er rapporten gegenereerd worden wanneer de feature files worden uitgevoerd via de runner.

\*Controleer of de packages daadwerkelijk in de CucumberFramework package zitten. Zo niet, geef de feature, steps en runner class een nieuwe naam. Selecteer rename uit het context menu en zet de package name CucumberFramework voor de naam van de file, dus bijv CucumberFramework.featureFile. Dit is nodig om het pad naar de feature- en stepsfile correct te krijgen.

**Selenium Webdriver integreren**

Selenium zorgt ervoor dat we kunnen interacteren met een website. We gaan de testen uitvoeren met FF. We hebben een oudere versie nodig, omdat deze firebug en firepath ondersteunen. Met deze tools kunnen we elementen identificeren op websites.

Installeer de nieuwste versie van FF.

Installeer FF. Kies voor custom installation, en uncheck de boxes voor het creëren van snelkoppelingen op je desktop en het startmenu, omdat we een nieuwere versie van firefox ook blijven gebruiken.

Open de oude versie van FF. Open de settings en zet updates uit, onder advanced.

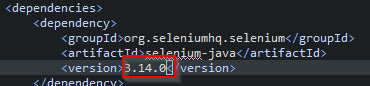
Ga naar addons. Zoek naar firebug en firepath en installeer deze.

Probeer met firebug een aantal elementen te selecteren en kijk in de console naar de html structuur.

\*firebug wordt niet meer ondersteund. We gebruiken daarom de dome explorer.

**Selenium**

Update in de POM selenium naar de laatse versie (3.14.0).



Rechterklik op het project. Ga naar Maven > update project. Vink “force updates” aan en klik op ok.

Download geckodriver: https://github.com/mozilla/geckodriver/releases

geckodriver-v0.21.0-win32.zip (dezelfde win32 versie als FF)

Maak in CucumberFramework een package aan “resources.”

Kopieer chromedriver en geckodriver in deze map.

Maak een object aan: WebDriver Driver; in de class LoginSteps.

Zet een hook voor de code in de steps file: @before(). De code in deze tag wordt altijd uitgevoerd voordat er een step uitgevoerd wordt. In deze stap gaan we geckodriver aanroepen.

Eerst moet er in deze stap een systeem eigenschap geset worden, zodat het systeem weet waar geckodriver te vinden is.

Maak in de before hook een methode aan. Deze method van overal te benaderen zijn en moet setup heten. Plaats in deze methode de onderstaande code: System.setProperty (“webdriver.gecko.driver”, {pad waar geckodriver is opgeslagen}). Check dit met bijlage 4.

Nu kan het object geinstantieerd worden:

**this**.driver = **new** FirefoxDriver();

We willen dat de aangeroepen browser het hele scherm beslaat. Dat doen we door een aantal methodes op het driver object lost te laten.

**this**.driver.manage().window().maximize();

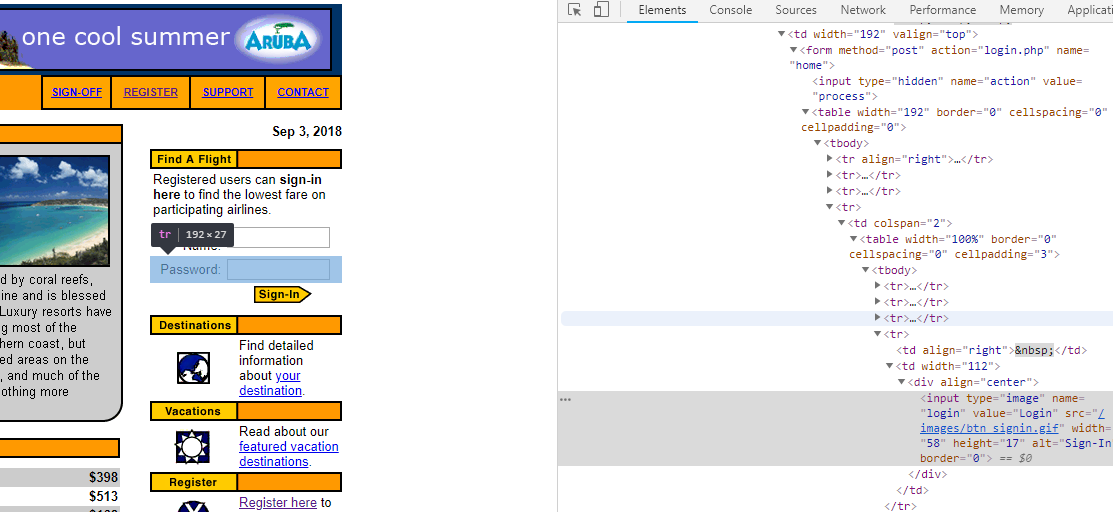
Tot slot willen we dat hij een timeout geeft bij 60 seconden, bijvoorbeeld als hij niet laadt.

**this**.driver.manage().timeouts().pageLoadTimeout(60, TimeUnit.SECONDS);

\*“This” is een keyword. Het is een verwijzing naar de geïnstantieerde variabele van een class.

\*als er errors verschijnen, haal dan de relevante packages binnen door met je muis over het element te hoveren en voor de juiste package

Nu willen we de teststappen uitvoeren. We willen de login functionaliteit van een pagina testen, in dit geval stackoverflow.com. Gebruik de chrome dom explorer om de attributen van de verschillende elementen te inspecteren:



De eerste teststap moet navigeren naar de homepage:

@Given("^user navigates to demoaut website$")

**public** **void** user\_navigates\_to\_stackoverflow\_website() **throws** Throwable {

driver.get("http://www.newtours.demoaut.com");

System.out.println("user navigates to demoaut website");

Dan voeren we onze inloggegevens in (creëer een account voor demoaut)

@Given("^user enters valid username$")

**public** **void** user\_enters\_valid\_username() **throws** Throwable {

driver.findElement(**By**.name("userName")).sendKeys("Pino");

System.out.println("user enters valid username");

@Given("^user enters valid password$")

**public** **void** user\_enters\_valid\_password() **throws** Throwable {

driver.findElement(**By**.name("password")).sendKeys("123");

System.out.println("user enters valid password");

We klikken op login:

@When("^user clicks on the login button$")

**public** **void** user\_clicks\_on\_the\_login\_button() **throws** Throwable {

driver.findElement(**By**.name("userName")).click();

Thread.sleep(1000);

driver.findElement(**By**.name("login")).click();

System.out.println("user clicks login button");

\*let op: er wordt nog een keer geklikt op het username veld, omdat er een waarschuwingsbox verschijnt die voor de login knop verschijnt.

En we verifiëren of we op de juiste pagina zitten:

@Then("^user should be taking to the succesfull login page$")

**public** **void** user\_should\_be\_taking\_to\_the\_succesfull\_login\_page() **throws** Throwable {

WebElement findFlightsButton = driver.findElement(**By**.name("findFlights"));

Thread.sleep(5000);

Assert.assertEquals(**true**, findFlightsButton.isDisplayed());

System.out.println("user should be taking to the succesfull login page");

\*Als er een deprecation error is, zet deze package dan bij je imports: **import** org.junit.Assert;

**Meerdere scenario’s**

In feature files kan je meerdere scenario’s kwijt. Sommige scenario’s hebben gedeelde stappen, het is handig voor het overzicht als deze gedeelde stappen in een algemeen deel worden opgenomen. Dit heet een background.

**Background:**

*Given* user navigates to stackoverflow website

*And* user clicks login button

Nu kunnen de eerste twee stappen worden weggelaten in het scenario.

We gaan een foutscenario maken, waarbij de gebruiker een incorrecte username opgeeft.

1. Kopieer het login scenario in de feature file. Pas de “given” stappen aan zodat er een ongeldige username wordt opgegeven en een ongeldig wachtwoord. Verander ook de titel van het scenario.
2. Maak in de steps file stappen aan die doen wat de feature file opgeeft. Kopieer dus een bestaande stap en laat deze een ongeldige username opgeven.
3. De gebruiker wordt nu niet naar “succesfull login page” geleid. Daar moet dus ook een nieuwe step voor gemaakt worden met een correcte verificatie. Gebruik bijvoorbeeld de login knop.

**Parametrisering**

Om dubbele code te voorkomen, is het parametriseren van input een handig concept. Neem het invullen van een username. In de vorige les hadden we een ongeldige username en een geldige. Dit kunnen we als volgt parametriseren in de feature:

*And* user enters "valid" username

Dan moeten we de steps file aanpassen zodat de methode een parameter kan ontvangen. Valid moet vervangen worden door deze regular expression:

\"([^\"]\*)\"

Corrigeer de methodenaam.

Dan moet de methode een string als input kunnen gebruiken. Deze voer je als argument op. Zie onderstaand voorbeeld. Totslot moet de sendKeys methode het argument van de user\_enters\_userName in het veldje “username” zetten.

@Given("^user enters \"([^\"]\*)\" username$")

**public** **void** user\_enters\_username(String userName) **throws** Throwable {

driver.findElement(**By**.name("userName")).sendKeys("userName");

System.out.println("user enters valid username");

Nu dit geparametriseerd is kan in de feature file ook de teststap “invalid password” aangepast worden.

*And* user enters "invalid" username

Nu is er slechts 1 step definition nodig voor meerdere teststappen. Handig. Probeer hetzelfde te doen voor wachtwoord en verwijder de onnodige teststappen.

**Data tables**

Stel dat we bij een teststap twee variabelen willen invoeren. Dit kunnen we doen door uit een data tabel te putten. Stel dat we in 1 stap de username en de login willen invoeren. Dit ziet er als volgt uit in de feature file:

**Scenario:** Login to account with correct credentials with data table

*And* user enters valid username and valid password

|Pino|123|

Met run configurations kan je de step die hierbij hoort aanmaken. Het argument van de methode is nu een type data table. Vervolgens word er in de methode een list object gecreëerd die gevuld wordt met de datatable (List is een type raw, .raw maakt een arg1 tot type raw, ctrl linker muisknop voor een uitleg in eclipse). Daarna worden de elementen geïdentificeerd en via de sendkeys methode worden de variabelen uit de data table in de velden gezet.

@Given("^user enters valid username and valid password$")

**public** **void** user\_enters\_valid\_username\_and\_valid\_password(DataTable arg1) **throws** Throwable {

List<List<String>> data = arg1.raw();

driver.findElement(**By**.name("userName")).sendKeys(data.get(0).get(0));

driver.findElement(**By**.name("password")).sendKeys(data.get(0).get(1));

**Examples**

Met examples is het mogelijk om over het scenario te vullen met verschillende variabelen. Zo kun je in 1 scenario bijvoorbeeld een fout en goed situatie opnemen.

**Scenario Outline:** Login to account with correct credentials 2

*Given* user navigates to demoaut website

*And* user enters "*<username>*" username

*And* user enters "*<password>*" password

*When* user clicks on the login button

*Then* user should be taking to the succesfull login page

**Examples:**

|username|password|

|Pino |123|

|Pino |123|

**After hook**

Soms willen we na ieder scenario een generieke stap. Nu is het bijvoorbeeld vrij irritant dat de browser niet afsluit. Dat kunnen we doen door een after hook in te bouwen.

@After()

**public** **void** tearDown(){

driver.manage().deleteAllCookies();

driver.close();

**Tags**

Met Tags in de runner class kan je specifieke features uitvoeren. Plaats de tag van book a flight tussen de accolades van “tags” met accenten. Plaats dezelfde tag, zonder accenten, bovenin de feature class.

tags = {"@BookAFlight"},

*@BookAFlight*

**Feature:** Book a flight

Run de runner class als junit test.

* Tags en hooks kunnen gecombineerd worden, bijvoorbeeld in de situatie wanneer er verschillende features zijn die verschillende setups (before hook) vereist

**TestNg**

Met testng kunnen via een xml 1 of meerdere runner classes gedraaid worden. Als je meerdere runner classes hebt, kan dit handig zijn.

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="Cucumber Framework">

<test name="Cucumber Tests" junit="true">

<classes>

<class name="CucumberFramework.runner.MainRunner" />

</classes>

</test>

</suite>

**Testen draaien vanaf commandline**

Om testen te draaien, moet deze depency in je pom file zitten:

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.19.1</version>

<configuration>

<suiteXmlFiles>

<suiteXmlFile>testng.xml</suiteXmlFile>

</suiteXmlFiles>

<testErrorIgnore>false</testErrorIgnore>

<testFailureIgnore>false</testFailureIgnore>

</configuration>

</plugin>

Ga naar de directory waar de POM in zit:

CD C:\Users\matth\Desktop\CucumberFramework\CucumberFramework

Voer uit mvn clean compile test

**Bijlage 1**

POM file

<build>

<pluginManagement>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

<fork>true</fork>

<!--<executable>C:\Program Files\Java\jdk1.8.0\_121\bin\javac.exe</executable>-->

<executable>${env.JAVA\_HOME}\bin\javac.exe</executable>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.19.1</version>

<configuration>

<suiteXmlFiles>

<suiteXmlFile>testng.xml</suiteXmlFile>

</suiteXmlFiles>

<testErrorIgnore>false</testErrorIgnore>

<testFailureIgnore>false</testFailureIgnore>

</configuration>

</plugin>

</plugins>

</pluginManagement>

</build>

<reporting>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-report-plugin</artifactId>

<version>2.19.1</version>

</plugin>

</plugins>

</reporting>

<dependencies>

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>3.6.0</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

</dependency>

<dependency>

<groupId>info.cukes</groupId>

<artifactId>cucumber-jvm</artifactId>

<version>1.2.5</version>

<type>pom</type>

</dependency>

<dependency>

<groupId>info.cukes</groupId>

<artifactId>cucumber-junit</artifactId>

<version>1.2.5</version>

<scope>test</scope>

</dependency>

<!-- https://mvnrepository.com/artifact/info.cukes/cucumber-core -->

<dependency>

<groupId>info.cukes</groupId>

<artifactId>cucumber-core</artifactId>

<version>1.2.5</version>

</dependency>

<!-- https://mvnrepository.com/artifact/info.cukes/cucumber-html -->

<dependency>

<groupId>info.cukes</groupId>

<artifactId>cucumber-html</artifactId>

<version>0.2.3</version>

</dependency>

<!-- https://mvnrepository.com/artifact/info.cukes/cucumber-java -->

<dependency>

<groupId>info.cukes</groupId>

<artifactId>cucumber-java</artifactId>

<version>1.2.5</version>

</dependency>

<!-- https://mvnrepository.com/artifact/info.cukes/cucumber-jvm-deps -->

<dependency>

<groupId>info.cukes</groupId>

<artifactId>cucumber-jvm-deps</artifactId>

<version>1.0.5</version>

</dependency>

<!-- https://mvnrepository.com/artifact/info.cukes/gherkin -->

<dependency>

<groupId>info.cukes</groupId>

<artifactId>gherkin</artifactId>

<version>2.12.2</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.hamcrest/hamcrest-all -->

<dependency>

<groupId>org.hamcrest</groupId>

<artifactId>hamcrest-all</artifactId>

<version>1.3</version>

</dependency>

<!-- https://mvnrepository.com/artifact/info.cukes/cucumber-picocontainer -->

<dependency>

<groupId>info.cukes</groupId>

<artifactId>cucumber-picocontainer</artifactId>

<version>1.2.5</version>

</dependency>

<!-- https://mvnrepository.com/artifact/info.cukes/cucumber-testng -->

<dependency>

<groupId>info.cukes</groupId>

<artifactId>cucumber-testng</artifactId>

<version>1.2.5</version>

</dependency>

<!-- Extent Reports -->

<dependency>

<groupId>com.aventstack</groupId>

<artifactId>extentreports</artifactId>

<version>3.0.5</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.freemarker/freemarker -->

<dependency>

<groupId>org.freemarker</groupId>

<artifactId>freemarker</artifactId>

<version>2.3.26-incubating</version>

</dependency>

<!-- https://mvnrepository.com/artifact/net.masterthought/cucumber-reporting -->

<dependency>

<groupId>net.masterthought</groupId>

<artifactId>cucumber-reporting</artifactId>

<version>3.6.0</version>

</dependency>

<dependency>

<groupId>com.vimalselvam</groupId>

<artifactId>cucumber-extentsreport</artifactId>

<version>2.0.4</version>

</dependency>

</dependencies>

</project>

**Bijlage 2**

Feature file

Feature: Login to account

Existing user should be able to login to account using correct account

Scenario: Login to account with correct credentials

Given user navigates to demoaut website

And user clicks login button

And user enters valid username

And user enters valid password

When user clicks on the login button

Then user should be taking to the succesfull login page

**Bijlage 3**

import org.junit.runner.RunWith;

import cucumber.api.CucumberOptions;

import cucumber.api.junit.Cucumber;

@RunWith (Cucumber.class)

@CucumberOptions(

features = {"src/test/java/CucumberFramework/featureFiles/"},

glue = {"CucumberFramework.steps"},

monochrome = true,

tags = {},

plugin = {"pretty","html:target/cucumber","json:target/cucumber.json","com.cucumber.listener.ExtentCucumberFormatter:target/repo"}

)

**Bijlage 4**

WebDriver driver;

@Before()

public void setup(){

System.setProperty("webdriver.gecko.driver", "C:\\Users\\matth\\Desktop\\CucumberFramework\\CucumberFramework\\src\\test\\java\\CucumberFramework\\resources\\geckodriver.exe");

**Bijlage 5**

@Then("^user should not be taking to the succesfull login page$")

**public** **void** user\_should\_\_not\_be\_taking\_to\_the\_succesfull\_login\_page() **throws** Throwable {

WebElement loginButton = driver.findElement(**By**.name("login"));

Thread.sleep(5000);

Assert.assertEquals(**true**, loginButton.isDisplayed());

System.out.println("user should be taking to the succesfull login page");

**Feature:** Login to account

Existing user should be able to login to account using correct account

**Background:**

*Given* user navigates to demoaut website

**Scenario:** Login to account with correct credentials

*And* user enters valid username

*And* user enters valid password

*When* user clicks on the login button

*Then* user should be taking to the succesfull login page

**Scenario:** Login to account with incorrect username

*And* user enters invalid username

*And* user enters invalid password

*When* user clicks on the login button

*Then* user should not be taking to the succesfull login page

Bijlage 6

**Feature:** Login to account

Existing user should be able to login to account using correct account

**Background:**

*Given* user navigates to demoaut website

**Scenario:** Login to account with correct credentials

*And* user enters "valid" username

*And* user enters "valid" password

*When* user clicks on the login button

*Then* user should be taking to the succesfull login page

**Scenario:** Login to account with incorrect username

*And* user enters "invalid" username

*And* user enters "invalid" password

*When* user clicks on the login button

*Then* user should not be taking to the succesfull login page

@Given("^user enters \"([^\"]\*)\" password$")

**public** **void** user\_enters\_password(String password) **throws** Throwable {

driver.findElement(**By**.name("password")).sendKeys("password");

System.out.println("user enters valid password");

Bijlage 7

Steps voorbeeld

package CucumberFramework.steps;

import java.util.List;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.Keys;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

import cucumber.api.DataTable;

import cucumber.api.java.After;

import cucumber.api.java.Before;

import cucumber.api.java.en.Given;

import cucumber.api.java.en.Then;

import cucumber.api.java.en.When;

import org.junit.Assert;

public class LoginSteps {

WebDriver driver;

@Before()

public void setup(){

System.setProperty("webdriver.gecko.driver", "C:\\Users\\matth\\Desktop\\CucumberFramework\\CucumberFramework\\src\\test\\java\\CucumberFramework\\resources\\geckodriver.exe");

this.driver = new FirefoxDriver();

this.driver.manage().window().maximize();

this.driver.manage().timeouts().pageLoadTimeout(60, TimeUnit.SECONDS);

}

@After()

public void tearDown(){

driver.manage().deleteAllCookies();

driver.close();

}

@Given("^user navigates to demoaut website$")

public void user\_navigates\_to\_demoaut\_website() throws Throwable {

driver.get("http://www.newtours.demoaut.com/");

System.out.println("user navigates to demoaut website");

}

@Given("^user enters \"([^\"]\*)\" username$")

public void user\_enters\_username(String userName) throws Throwable {

driver.findElement(By.name("userName")).sendKeys(userName);

System.out.println("user enters valid username");

}

@Given("^user enters \"([^\"]\*)\" password$")

public void user\_enters\_valid\_password(String password) throws Throwable {

driver.findElement(By.name("password")).sendKeys(password);

System.out.println("user enters valid password");

}

@Given("^user enters invalid username$")

public void user\_enters\_username() throws Throwable {

driver.findElement(By.name("userName")).sendKeys("Aart");

System.out.println("user enters valid username");

}

@Given("^user enters invalid password$")

public void user\_enters\_invalid\_password() throws Throwable {

driver.findElement(By.name("password")).sendKeys("bullshit");

System.out.println("user enters valid password");

}

@When("^user clicks on the login button$")

public void user\_clicks\_on\_the\_login\_button() throws Throwable {

driver.findElement(By.name("userName")).click();

Thread.sleep(1000);

driver.findElement(By.name("login")).click();

System.out.println("user clicks login button");

}

@Then("^user should be taking to the succesfull login page$")

public void user\_should\_be\_taking\_to\_the\_succesfull\_login\_page() throws Throwable {

WebElement findFlightsButton = driver.findElement(By.name("findFlights"));

Thread.sleep(10000);

Assert.assertEquals(true, findFlightsButton.isDisplayed());

System.out.println("user should be taking to the succesfull login page");

}

@Then("^user should not be taking to the succesfull login page$")

public void user\_should\_\_not\_be\_taking\_to\_the\_succesfull\_login\_page() throws Throwable {

WebElement loginButton = driver.findElement(By.name("login"));

Thread.sleep(5000);

Assert.assertEquals(true, loginButton.isDisplayed());

System.out.println("user should be taking to the succesfull login page");

}

@Given("^user enters valid username and valid password$")

public void user\_enters\_valid\_username\_and\_valid\_password(DataTable arg1) throws Throwable {

List<List<String>> data = arg1.raw();

driver.findElement(By.name("userName")).sendKeys(data.get(0).get(0));

driver.findElement(By.name("password")).sendKeys(data.get(0).get(1));

}

@When("^the user finds a flight$")

public void the\_user\_finds\_a\_flight() throws Throwable {

Thread.sleep(5000);

WebElement CheckboxTypeoneWay = driver.findElement(By.cssSelector("input[value='oneway']"));

CheckboxTypeoneWay.click();

driver.findElement(By.name("passCount")).sendKeys(Keys.DOWN, Keys.DOWN, Keys.DOWN, Keys.ENTER);

driver.findElement(By.name("fromPort")).sendKeys(Keys.DOWN, Keys.DOWN, Keys.DOWN, Keys.ENTER);

//select month

driver.findElement(By.name("fromMonth")).sendKeys(Keys.DOWN, Keys.DOWN, Keys.DOWN, Keys.ENTER);

//

driver.findElement(By.name("toPort")).sendKeys( Keys.DOWN, Keys.ENTER);

//select day

driver.findElement(By.name("fromDay")).sendKeys(Keys.DOWN, Keys.DOWN, Keys.DOWN, Keys.ENTER);

driver.findElement(By.name("toMonth")).sendKeys(Keys.DOWN, Keys.DOWN, Keys.DOWN, Keys.ENTER);

driver.findElement(By.name("toDay")).sendKeys(Keys.DOWN, Keys.DOWN, Keys.DOWN, Keys.DOWN, Keys.ENTER);

WebElement CheckBoxBusinessclass1 = driver.findElement(By.cssSelector("input[value='Business']"));

driver.findElement(By.name("airline")).sendKeys(Keys.DOWN, Keys.DOWN, Keys.ENTER);

CheckBoxBusinessclass1.click();

driver.findElement(By.name("findFlights")).click();

}

@When("^the user selects a flight$")

public void the\_user\_selects\_a\_flight() throws Throwable {

Thread.sleep(5000);

driver.findElement(By.name("reserveFlights")).click();

}

@When("^the user secures a purchase$")

public void the\_user\_secures\_a\_purchase() throws Throwable {

Thread.sleep(5000);

driver.findElement(By.name("passFirst0")).sendKeys("Pino");

driver.findElement(By.name("passLast0")).sendKeys("Sesam");

driver.findElement(By.name("passFirst1")).sendKeys("Tommy");

driver.findElement(By.name("passLast1")).sendKeys("Sesam");

driver.findElement(By.name("passFirst2")).sendKeys("Inie Minie");

driver.findElement(By.name("passFirst3")).sendKeys("Sesam");

driver.findElement(By.name("creditnumber")).sendKeys("ikhebmoneyG");

driver.findElement(By.name("buyFlights")).click();

}

@Then("^the user sees the flight confirmation page$")

public void the\_user\_sees\_the\_flight\_confirmation\_page() throws Throwable {

Thread.sleep(5000);

driver.getPageSource().contains("New York to Frankfurt");

}

}