**Gurkensalat ver. 0.99**

**December 2021**

**© accenture**

Gurkensalat

TEST AUTOMATION FRAMEWORK - DOCUMENTATION

**Credentials**

|  |  |
| --- | --- |
| **Description** | Gurkensalat Test Automation Framework – Documentation |
| **Release date** | December 1st, 2021 |
| **Version** | 0.99 |
| **Author** | david.b.schevela@accenture.com |
| **Author** | matus.simko@accenture.com |
| **Author** | peter.psenka@accenture.com |
| **Reviewer** | branislav.baca@accenture.com |

[1 Introduction 5](#_Toc89339185)

[2 Software and costs 6](#_Toc89339186)

[2.1 CI/CD environment 6](#_Toc89339187)

[2.2 Developer’s environment 6](#_Toc89339188)

[3 Supported Operating system/Browser/WebDriver 7](#_Toc89339189)

[4 Design overview 8](#_Toc89339190)

[5 Prerequisites 9](#_Toc89339193)

[6 Gurkensalat install guide 10](#_Toc89339194)

[6.1 Initial Gurkensalat project setup 10](#_Toc89339195)

[6.1.1 Download Gurkensalat framework 10](#_Toc89339196)

[6.1.2 Copy WebDrivers 10](#_Toc89339197)

[6.1.3 Edit config.properties 11](#_Toc89339198)

[6.2 Import project to Eclipse 14](#_Toc89339199)

[6.2.1 Run Eclipse 14](#_Toc89339200)

[6.2.2 Import project 15](#_Toc89339201)

[6.2.3 Update Maven dependencies 17](#_Toc89339202)

[6.2.4 Run test 18](#_Toc89339203)

[6.3 Import project to IntelliJ IDEA 21](#_Toc89339204)

[6.3.1 Run IntelliJ IDEA 21](#_Toc89339205)

[6.3.2 Import project 21](#_Toc89339206)

[6.3.3 Update Maven dependencies 22](#_Toc89339207)

[6.3.4 Run test 24](#_Toc89339208)

[6.4 Import project to Visual Studio Code 27](#_Toc89339209)

[6.4.1 Run Visual Studio Code 27](#_Toc89339210)

[6.4.2 Import project 27](#_Toc89339211)

[6.4.3 Adding Extensions to Visual Studio 29](#_Toc89339212)

[6.4.4 Run test 31](#_Toc89339213)

[7 Test execution 33](#_Toc89339214)

[7.1 Windows 33](#_Toc89339215)

[7.1.1 Execution From IDE 33](#_Toc89339216)

[7.1.2 Execution From Command Prompt 34](#_Toc89339217)

[7.2 Linux 36](#_Toc89339218)

[7.2.1 Execution From Command Prompt 36](#_Toc89339219)

[7.3 Mac 38](#_Toc89339220)

[7.3.1 Execution From IDE 38](#_Toc89339221)

[7.3.2 Execution From Command Prompt 38](#_Toc89339222)

[8 Gurkensalat project structure 41](#_Toc89339223)

[9 Gurkensalat content 42](#_Toc89339224)

[9.1 Cucumber 42](#_Toc89339225)

[9.2 Gurkensalat Features 48](#_Toc89339226)

[10 HTML Reports 50](#_Toc89339227)

[10.1 Summary section 50](#_Toc89339228)

[10.2 Test result section 51](#_Toc89339229)

[11 Cross-browser remote testing 52](#_Toc89339230)

[11.1 BrowserStack integration 52](#_Toc89339231)

[11.2 Sauce Labs integration 55](#_Toc89339232)

[12 Windows driver integration 59](#_Toc89339233)

[12.1 Winium 59](#_Toc89339234)

[12.2 WinAppDriver - Windows Application Driver 62](#_Toc89339235)

[13 First Test 64](#_Toc89339236)

[13.1 Add URL to config.properties 64](#_Toc89339237)

[13.2 Create unique feature file 64](#_Toc89339238)

[13.3 Run Cucumber script 67](#_Toc89339239)

[13.4 Create unique Page class 69](#_Toc89339240)

[13.5 Create Step definition class 71](#_Toc89339241)

[14 Test Step Actions 74](#_Toc89339242)

# Introduction

***Gurkensalat*** (from German language – Cucumber salad) is a generic open source BDD test automation framework based on *Cucumber*, *Selenium*, *Extent Reports* and *Java*.

We had this feeling that vanilla *Cucumber* framework is not “automation complete” for our professional needs. Kind of plain. So we decided to mix it with other extensions and spice it up a bit. What could go wrong with one-word punchy German name? Hence, Gurkensalat :D

Gurkensalat is suitable for small to mid-size test automation projects to run automated tests on web or on windows desktop application.

Improvements:

* **Fancy HTML reports**

Top notch easy to read HTML reports with lots of details about every action executed. Of course - statistics, timings, actions, locators, screenshots and error messages included.

* **Re-usable steps**

Over 50 predefined re-usable Step Actions available. No need to re-invent the wheel.

* **Conditional waiting**

Steps are executed as soon as possible, no more wasteful waiting.

* **Separated element locators**

No more hard-coded locators in functional code [sigh]. Implemented proper Page Object Model for better separation of locators from execution code. Welcome easy maintenance.

* **Easy assertions**

Validation class ready to be used with expected value, actual value and validation methods. Nicely concentrated in one place.

* **Cross-browser/cross-device support**

So you need to test your application for compatibility on many operating systems, browsers and devices? Don’t worry. We’ve got you covered. Gurkensalat is supporting cloud testing services like BrowserStack or Sauce Labs.

* **Cost effective**

You might think – It is for free, so it is not so effective – and you would be completely wrong. It is very effective and at the same time basically the cheapest framework you can get. You cannot sell it to the customer, but you can sell test automation code developed on project with this framework to the customer. If you plan to test locally and don’t plan to use cross-browser/cross-device cloud testing services - you are safe – it’s for free. For a long time. Extra cross-browser/cross-device testing services supported by framework (such as BrowserStack or Sauce Labs) might be subject to some relatively small third-party fees.

Good luck! Have fun! 😊

# Software and costs

Gurkensalat was built with cost effectivity in mind. Accenture is granting nonexclusive, nontransferable limited license to use the framework on test automation projects. This is so due to license agreements on used framework components which were created under MIT, GNU or Apache licenses.

Specific test automation code (test cases) developed in framework can be treated separately from framework - as project deliverables - and can be part of agreed costs.

Additional extra third party services, usually involved in cross-browser/cross-device testing (such as BrowserStack or Sauce Labs), might be subject to separate additional (small) costs from third parties.

## Software for CI/CD pipeline

List of typically used tools on Continuous Integration / Continuous Deployment (CI/CD pipeline) environment with their versions, costs and licenses. These versions were tested and proven to work together with Gurkensalat seamlessly (you can use any latest version, but with a little risk that it will have some drawbacks):

|  |  |  |  |
| --- | --- | --- | --- |
| Tool | Version | Cost | License |
| [GIT](https://git-scm.com/download/win) | 2.32.0 | Free | GNU |
| [Java – AdoptOpenJDK + HotSpot](https://adoptopenjdk.net/releases.html) | 11.0.11+9 | Free | GNU |
| [Apache Maven](https://maven.apache.org/download.cgi) | 3.8.1 | Free | Apache License 2.0 |

## Software for computer of test automation developer

List of typically used tools running on computer of test automation developer with versions, costs and licenses. These versions were tested and proven to work together with Gurkensalat seamlessly (you can use any latest version, but with a little risk that it will have some drawbacks):

|  |  |  |  |
| --- | --- | --- | --- |
| Tool | Version | Cost | License |
| [Eclipse](https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2021-03/R/eclipse-java-2021-03-R-win32-x86_64.zip) | 2021-03 (4.19.0) | Free | GNU |
| [IntelliJ](https://www.jetbrains.com/idea/download/other.html) | 2021.2 | Free | Apache License 2.0 |
| [Visual Studio Code](https://code.visualstudio.com/Download) | 1.59.1 | Free | MIT |
| [GIT](https://git-scm.com/download/win) | 2.32.0 | Free | GNU |
| [Java – AdoptOpenJDK + HotSpot](https://adoptopenjdk.net/releases.html) | 11.0.11+9 (11 – LTS) | Free | GNU |
| [Selenium](https://www.selenium.dev/downloads/) | 3.141.59 | Free | Apache License 2.0 |
| [Apache Maven](https://maven.apache.org/download.cgi) | 3.8.1 | Free | Apache License 2.0 |
| [Cucumber Eclipse Plugin](https://marketplace.eclipse.org/content/cucumber-eclipse-plugin) | 1.0.0.202106240526 | Free | MIT |

# Supported Operating systems/Browsers/WebDrivers

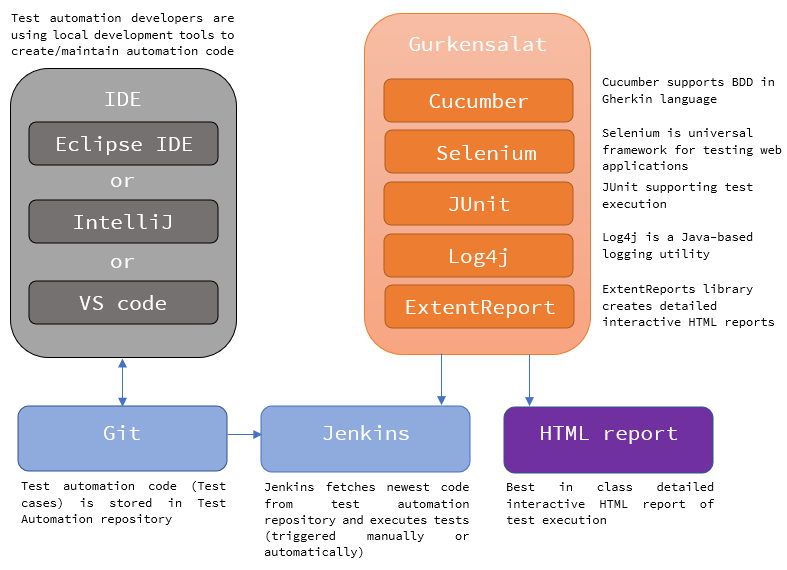
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| OS | OS Version | Browser | Browser Version | WebDriver | WebDriver Version |
| macOS | macOS Mojave 10.14.4 | Firefox | 92.0 | [mozilla/geckodriver](https://github.com/mozilla/geckodriver/releases/tag/v0.29.1) | 0.29.1 |
| macOS | macOS Mojave 10.14.4 | Chrome | 96.0.4664.45 | [chromedriver](https://chromedriver.storage.googleapis.com/index.html?path=96.0.4664.45/) | 96.0.4664.45 |
| macOS | macOS Mojave 10.14.4 | Safari | 12.1 | -- | -- |
| macOS | macOS Mojave 10.14.4 | Edge | 93.0.961.47 | [Microsoft Edge Driver](https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/) | 93.0.964.0 |
| macOS | macOS Mojave 10.14.4 | Opera | 79 | [operadriver](https://github.com/operasoftware/operachromiumdriver/releases/tag/v.93.0.4577.63) | 93.0.4577.63 |
| Linux | Debian 11 | Firefox | 78.14 | [mozilla/geckodriver](https://github.com/mozilla/geckodriver/releases/tag/v0.29.1) | 0.29.1 |
| Linux | Debian 11 | Chrome | 96.0.4664.45 | [chromedriver](https://chromedriver.storage.googleapis.com/index.html?path=96.0.4664.45/) | 96.0.4664.45 |
| Linux | Debian 11 | Edge | 95.0.1 | [Microsoft Edge Driver](https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/) | 95.0.1000 |
| Linux | Debian 11 | Opera | 79 | [operadriver](https://github.com/operasoftware/operachromiumdriver/releases/tag/v.92.0.4515.107) | 93.0.4577.63 |
| Windows | Windows 10 Enterprise 20H2 19042.116 | Firefox | 91.0 | [mozilla/geckodriver](https://github.com/mozilla/geckodriver/releases/tag/v0.29.1) | 0.29.1 |
| Windows | Windows 10 Enterprise 20H2 19042.116 | Chrome | 96.0.4664.45 | [chromedriver](https://chromedriver.storage.googleapis.com/index.html?path=96.0.4664.45/) | 96.0.4664.45 |
| Windows | Windows 10 Enterprise 20H2 19042.116 | Edge | 93.0.961.47 | [Microsoft Edge Driver](https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/) | 93.0.961.47 |
| Windows | Windows 10 Enterprise 20H2 19042.116 | Opera | 78 | [operadriver](https://github.com/operasoftware/operachromiumdriver/releases/tag/v.92.0.4515.107) | 92.0.4515.107 |
| Windows | Windows 10 Enterprise 20H2 19042.116 | IE | 11.789.19041.0 | [InternetExplorerDriver](https://www.selenium.dev/downloads/) | 3.150.2 |

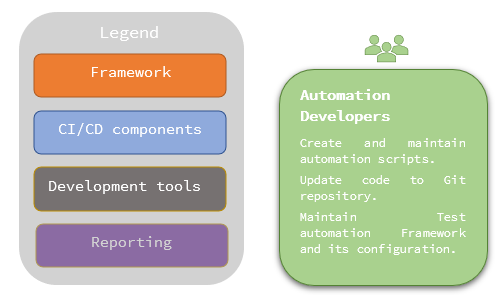
List of tested/supported Operating systems with Browsers and its WebDrivers. It is, for sure, possible to deploy framework also to other operating systems but other combinations were not tested as part of initial release of the framework.

# Design overview

Typical setup of Gurkensalat framework on projects:

Application under Test





# Prerequisites

It is recommended to use the following software suite when starting to work with Gurkensalat for a first time. Gurkensalat was tested to work properly with this setup. It can save you hours of head scratching and debugging. Recommended software suite:

* ***AdoptOpenJDK***

AdoptOpenJDK version 11 x64 installed on your computer

Follow installation guide in *Java/AdoptOpenJDK\_11\_x64*

* ***Chrome browser***

Chrome browser, version 96.0.4664.45

Follow installation guide in *Chrome*

We recommend to verify your framework installation with tested versions of browser/webdriver.

You can upgrade to latest versions immediately after verifying framework works flawlessly on your computer.

* ***Chromedriver***

Chromedriver, version 96.0.4664.45

Follow installation guide in *Chromedriver*

Copy webdriver to corresponding location as explained in the next chapter

* ***Maven***

Maven 3.8.1 version installed on your computer

Follow installation guide in *Maven*

Please select one:

* **Installation for Eclipse IDE:**
  + ***Eclipse IDE*** - Eclipse OXYGEN or the latest version installed on your computer

Follow installation guide in *Eclipse* *IDE*

* + Working ***Cucumber*** plugin installed in your Eclipse

Follow installation guide in *Cucumber - Eclipse plugin*

* **Installation for IntelliJ IDEA:**
  + ***IntelliJ IDEA*** - IntelliJ IDEA 2021.1.2 or the latest installed on your computer

Follow installation guide in *IntelliJ IDEA*

* **Installation for Visual Studio Code:**
  + ***Visual Studio Code*** - Visual Studio Code 1.62.0 system installed on your computer

Follow installation guide in *Visual Studio Code*

# Gurkensalat install guide

## Initial Gurkensalat project setup

This guide is oriented towards Windows OS users. Linux and macOS users may follow similar steps, hints are provided.

### Download Gurkensalat framework

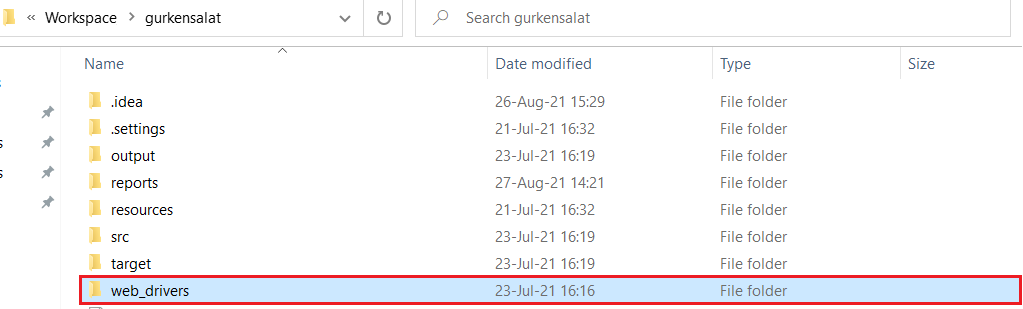
* + Download the latest version of Gurkensalat framework from ***Gurkensalat Framework***
  + Unzip *Gurkensalat framework* into your workspace folder
  + It is recommended to have your Workspace folder directly in *C:\Workspace*

*( /home/workspace for Linux and macOS users)*

* + It is **not recommended** to have your Workspace as cloud folder (OneDrive)

### Copy WebDrivers

* + Copy WebDrivers (e.g. Chromedriver, Geckodriver,…) into your web driver folder in Gurkensalat framework (e.g. C:\Workspace\gurkensalat\web\_drivers for Windows users, /home/workspace/gurkensalat/web\_drivers for Linux and macOS users)

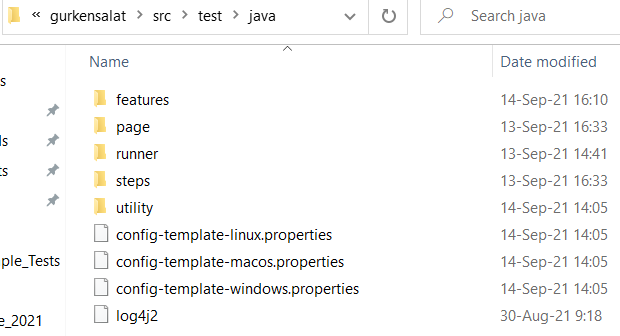


If you decide to use more than one IDE (e.g. for some reason you want to use Eclipse and IntelliJ IDEA), do not use the same project and folder. It is recommended to clone the project from your code base (GIT) to different folders for different IDEs.

### Edit config.properties

* + Navigate to gurkensalat/src/test/java

(e.g. *C:/Workspace/gurkensalat/src/test/java* drivers for Windows users*, /home/workspace/gurkensalat/src/test/java* for Linux and macOS users*)*



* + Make a copy of the config-template file according to your platform

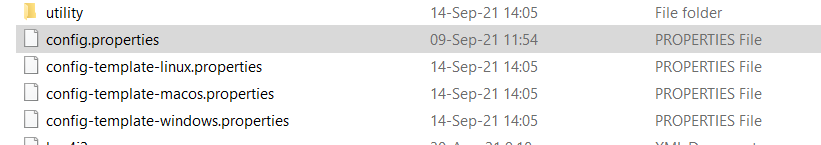
*(e.g. copy config-template-windows.properties* for Windows users,

*copy config-template-linux.properties* for Linux users,

*copy config-template-macos.properties* for macOS users)



* + Paste it into the same folder and rename it config.properties



* + Open config.properties in your favorite text editor.
  + Update config.properties as required. You can delete examples.

It should look similar to the following example (Windows OS example):

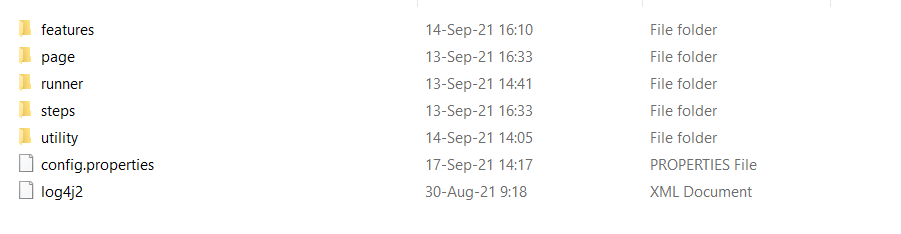




config.properties legend

|  |  |
| --- | --- |
| **FirefoxPath** | Location of firefox.exe file, as geckodriver won’t find .exe (CHANGE this path according to your Mozilla Firefox installation folder) |
| **driverPath** | Location of WebDrivers folder (DO NOT CHANGE this path, paste all WebDrivers in this folder) |
| **reportConfigPath** | System path for Extent Report Configuration .xml file (DO NOT CHANGE) |
| **reportFolderPath** | System path for Reports folder location (DO NOT CHANGE) |
| **featureFilesPath** | System path for Feature files (DO NOT CHANGE) |
| **chromedriverExecFile** | Chrome WebDriver executable file name, as it can be different depending on OS (DO NOT CHANGE) |
| **geckodriverExecFile** | Mozilla Firefox WebDriver executable file name, as it can be different depending on OS (DO NOT CHANGE) |
| **IEdriverExecFile** | Internet Explorer WebDriver executable file name, as it can be different depending on OS (DO NOT CHANGE) |
| **edgedriverExecFile** | Microsoft Edge WebDriver executable file name, as it can be different depending on OS (DO NOT CHANGE) |
| **operadriverExecFile** | Opera WebDriver executable file name, as it can be different depending on OS (DO NOT CHANGE) |
| **bs\_username** | Your BrowserStack username (CHANGE username according to your BrowserStack credentials) |
| **bs\_automate\_key** | Your BrowserStack Automate Key (CHANGE Automate Key according to your BrowserStack credentials) |
| **sl\_username** | Your Sauce Labs username (CHANGE username according to your Sauce Labs credentials) |
| **sl\_automate\_key** | Your Sauce Labs Automate Key (CHANGE Automate Key according to your Sauce Labs credentials) |
| **Google\_URL** | URL to be used in tests (example) |
| **Saucedemo** | URL to be used in tests (example) |

* + - * Save updated config.properties
      * You can delete the other template property files



Due to security reasons, do not commit config.properties to your repository. It can contain some sensitive data (e.g. login credentials). It is highly recommend to add config.properties to git ignore list.

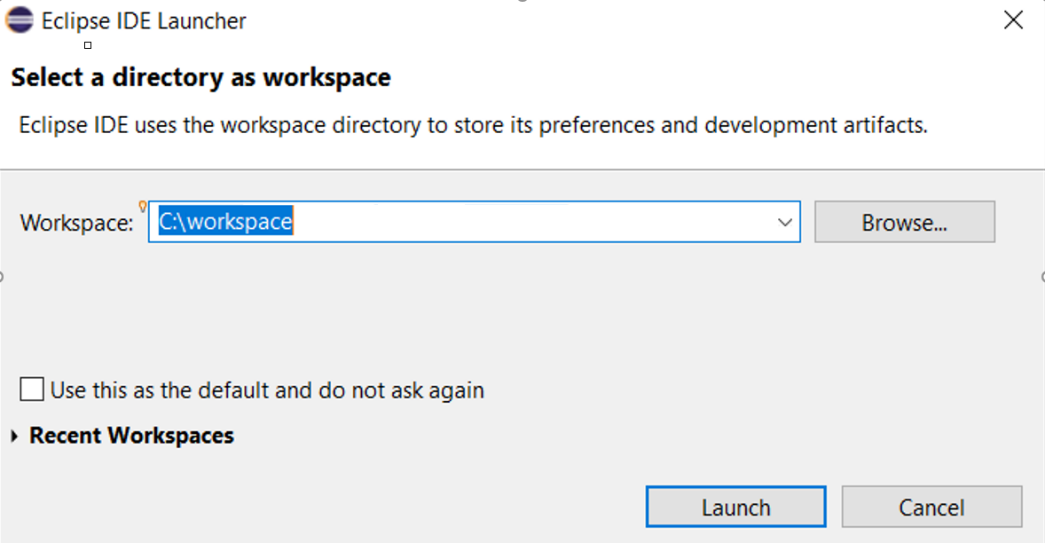
## Import project to Eclipse

### Run Eclipse

* + Double-click the Eclipse icon

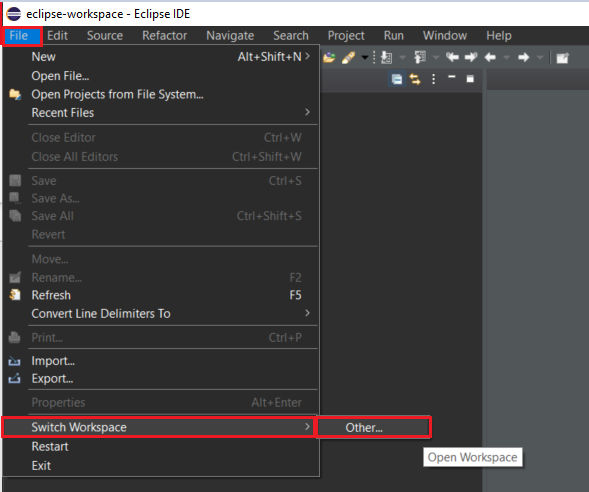


* + Select workspace where your project is located.



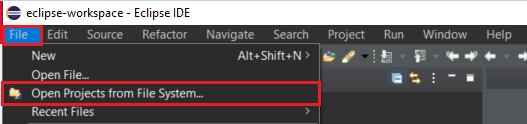
* + If you have already preselected and remembered workspace you can switch it.

In Eclipse click File - > Switch Workspace -> Other …

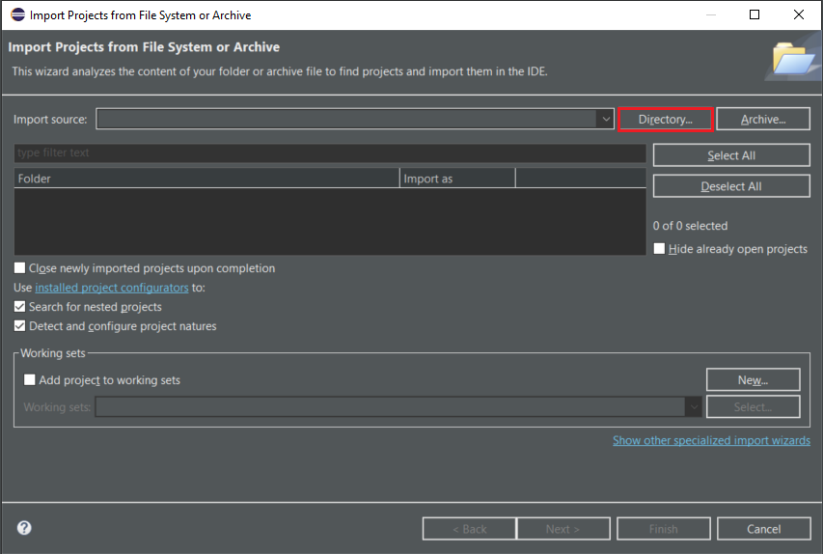


### Import project

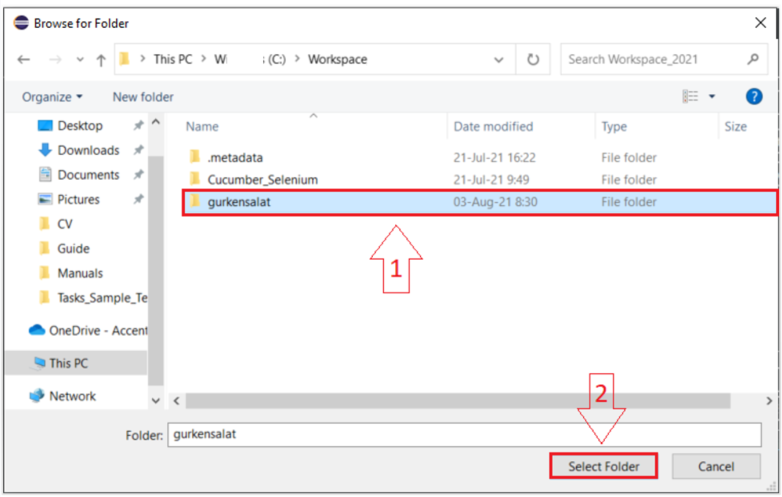
* + Click *File* -> *Open Project from File System.*



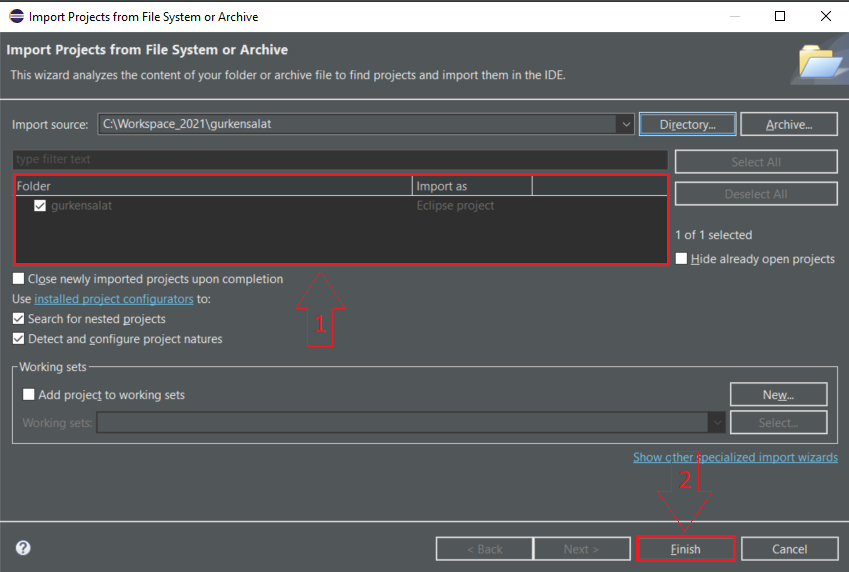
* + Click *Directory…*



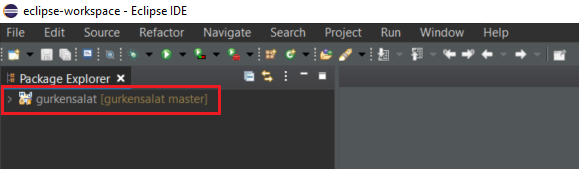
* + Search for *gurkensalat* project location (1) and click *Select Folder* (2).



* + Make sure that gurkensalat is selected (1) and click Finish (2).

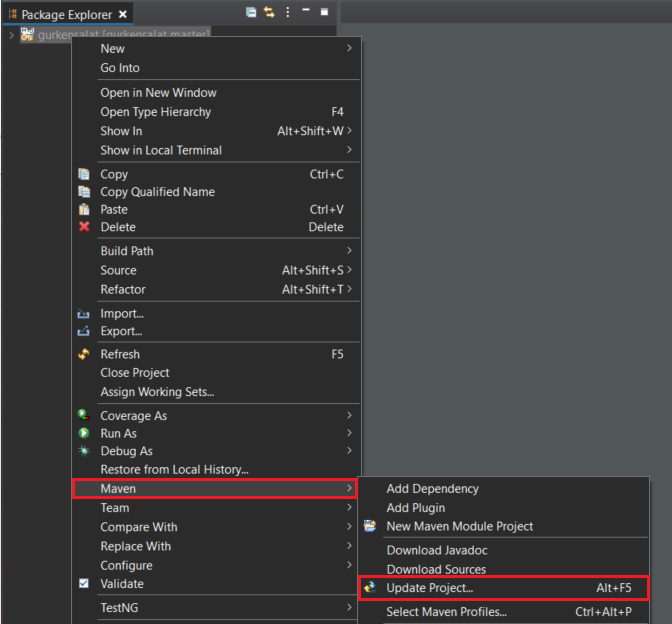


* + *gurkensalat* is now visible in Eclipse.

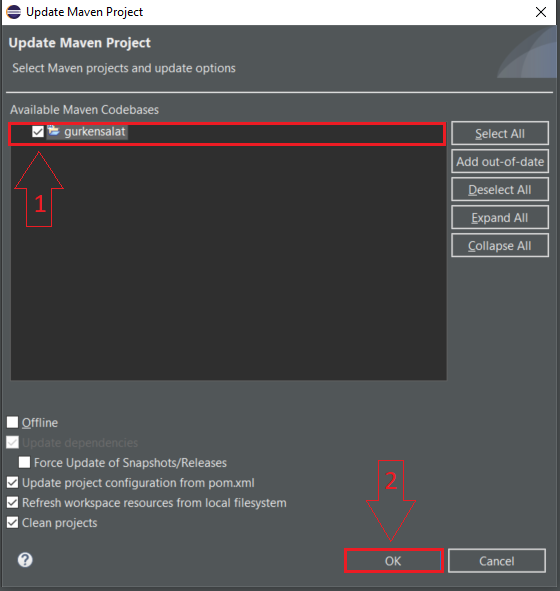
****

### Update Maven dependencies

* + Right-click *gurkensalat* project.
  + Select *Maven* -> *Update Project.*



* + Make sure that gurkensalat is selected (1) and click OK (2).

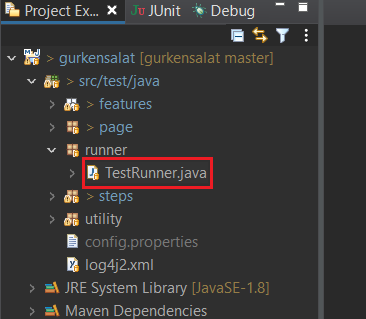


* + Wait for Maven dependencies update to finish (bottom right corner).

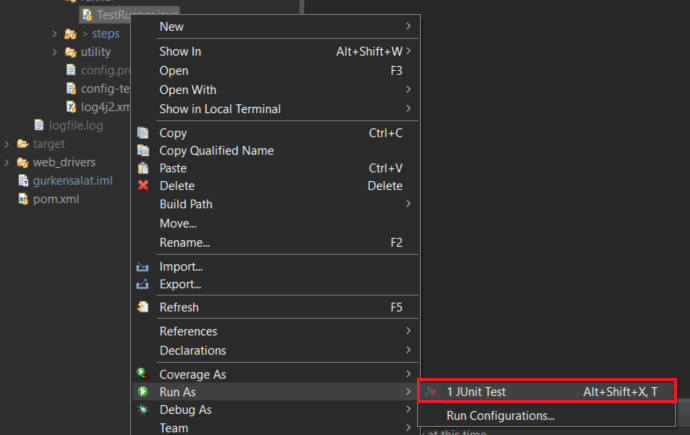


### Run test

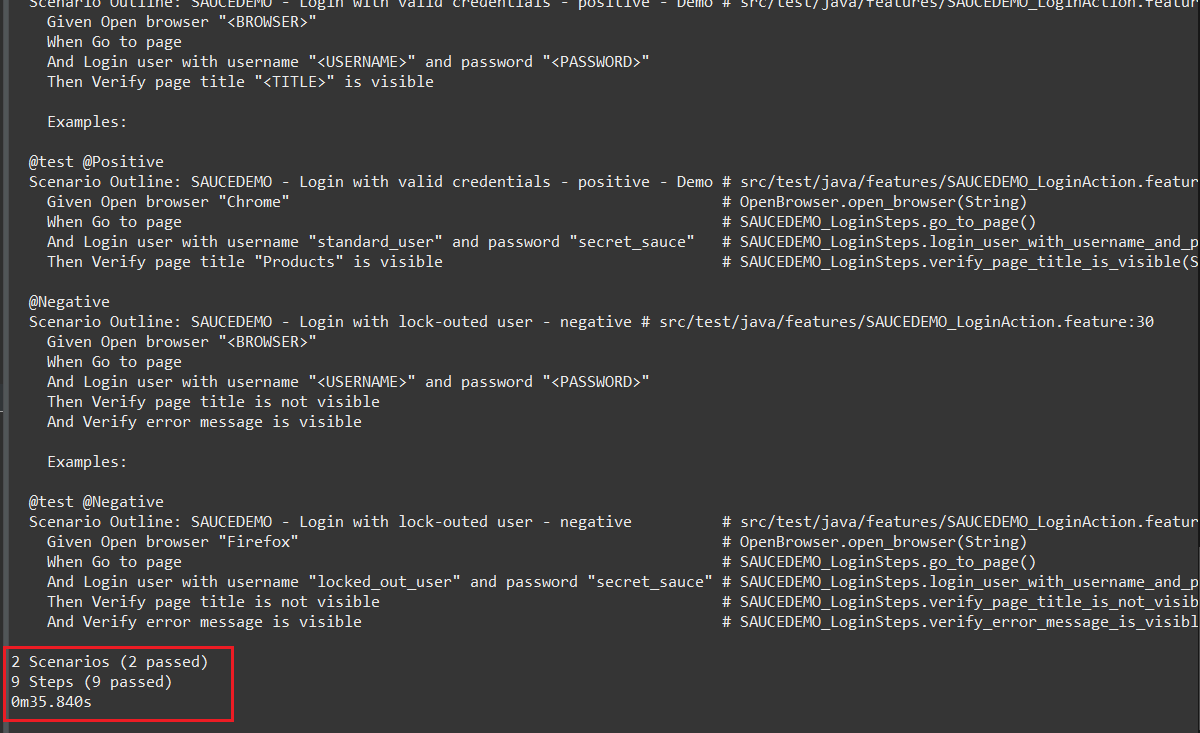
* + Once done, navigate to *TestRunner.java* in Project Explorer *(gurkensalat/src/test/java/runner/TestRunner.java).*



* + Right-click *TestRunner.java* and select *Run As* -> *Junit Test.*

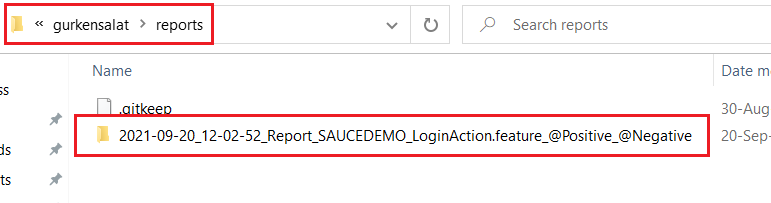


* + Sample test (loginAction.feature) will run. You will see following scenario output in console.

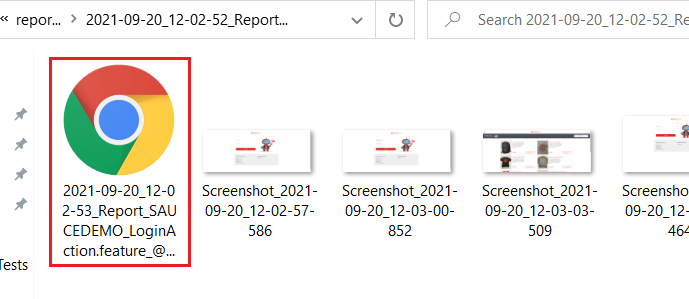


* + In Windows explorer, navigate to *report* folder in *gurkensalat* project (e.g. *C:/Workspace/gurkensalat/reports* for Windows users*, /home/workspace/gurkensalat/reports* for Linux and macOS users).

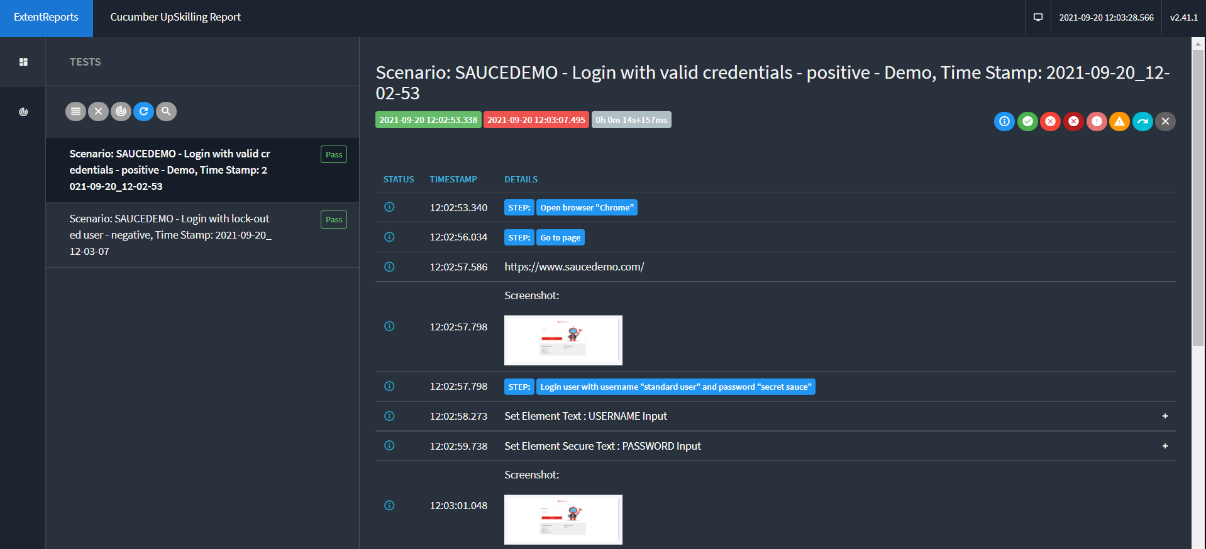
You will see HTML report folder for test you ran.



* + Open the folder and run HTML Document.



* + The HTML report opens in browser. Screenshots and report features are visible.



## Import project to IntelliJ IDEA

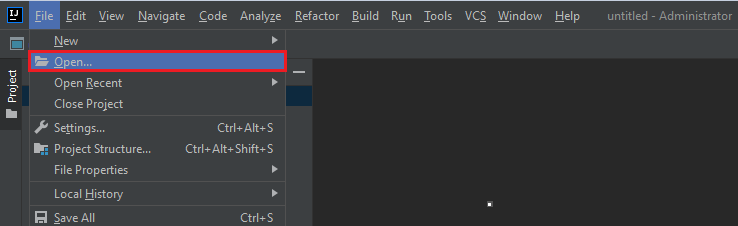
### Run IntelliJ IDEA

* + Double-click the IntelliJ IDEA icon.

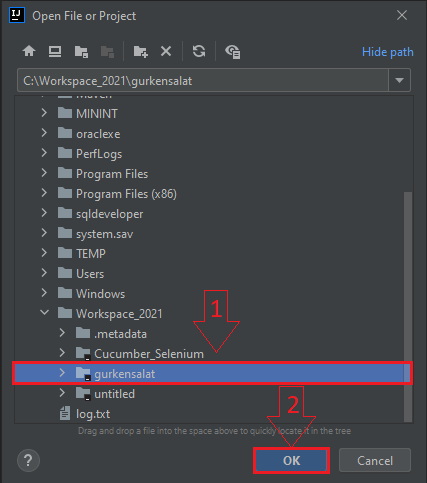


### Import project

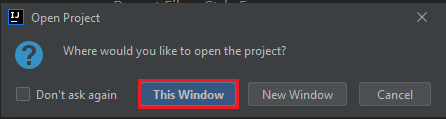
* + If the Welcome screen opens, click *Open.*
  + Otherwise, from the main menu, select *File -> Open*.



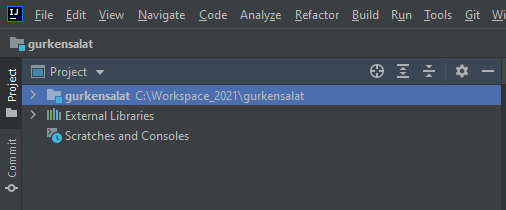
* + In the dialog select the directory in which your *gurkensalat* is located (1) and click *Open* (2).



* + If you have another project open in IntelliJ IDEA, a popup will appear. Click *This Window* button.

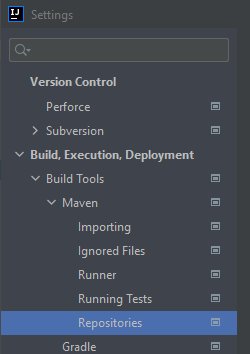


* + *gurkensalat* is now visible in IntelliJ *Project Explorer.*

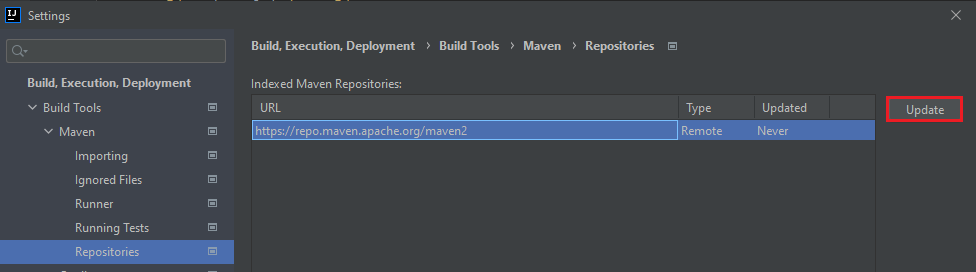


### Update Maven dependencies

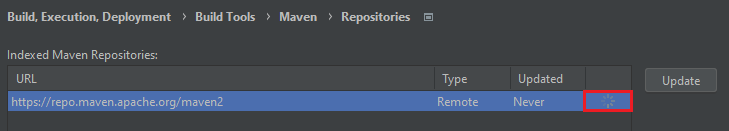
* + In the Settings/Preferences dialog (Ctrl + Alt + S) , go to *Build, Execution, Deployment -> Build Tools -> Maven -> Repositories.*
  + Select Repositories from options on the left.

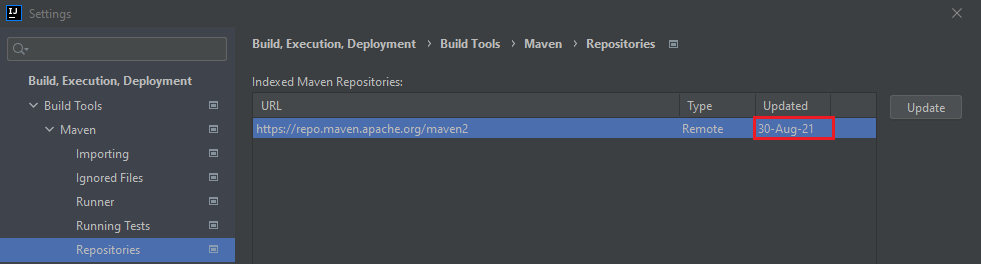


* + On the Repositories page click *Update* to update Maven repositories.

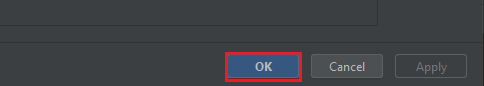


* + Wait for update to finish (it can take 5-10 minutes).



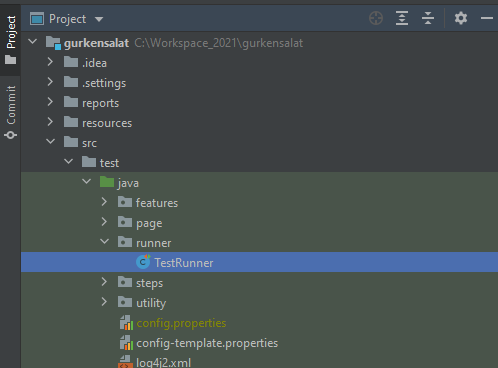


* + After the update is finished click *OK*.

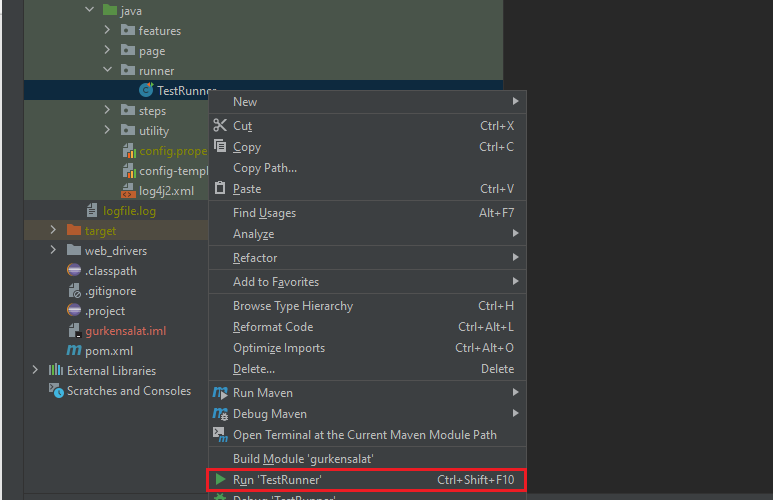


### Run test

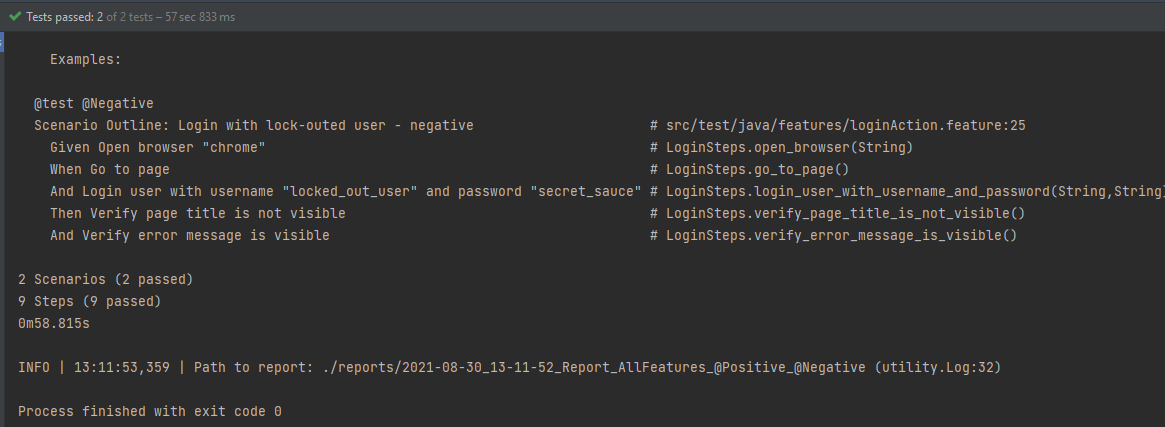
* + Once done, in Project explorer navigate to *TestRunner.java (gurkensalat/src/test/java/runner/TestRunner.java).*



* + Right-click *TestRunner.java* and select *Run ‘TestRunner’.*

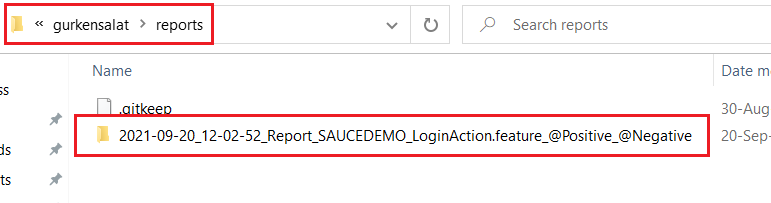


* + Sample test (loginAction.feature) will run. You will see following scenario output in console.

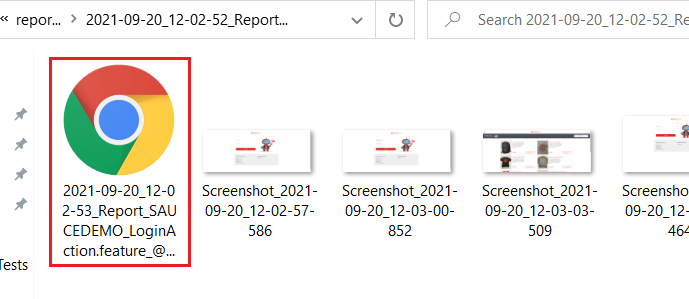


* + In Windows explorer, navigate to *reports* folder in *gurkensalat* project (e.g. *C:/Workspace/gurkensalat/reports* for Windows users*, /home/workspace/gurkensalat/reports* for Linux and macOS users).

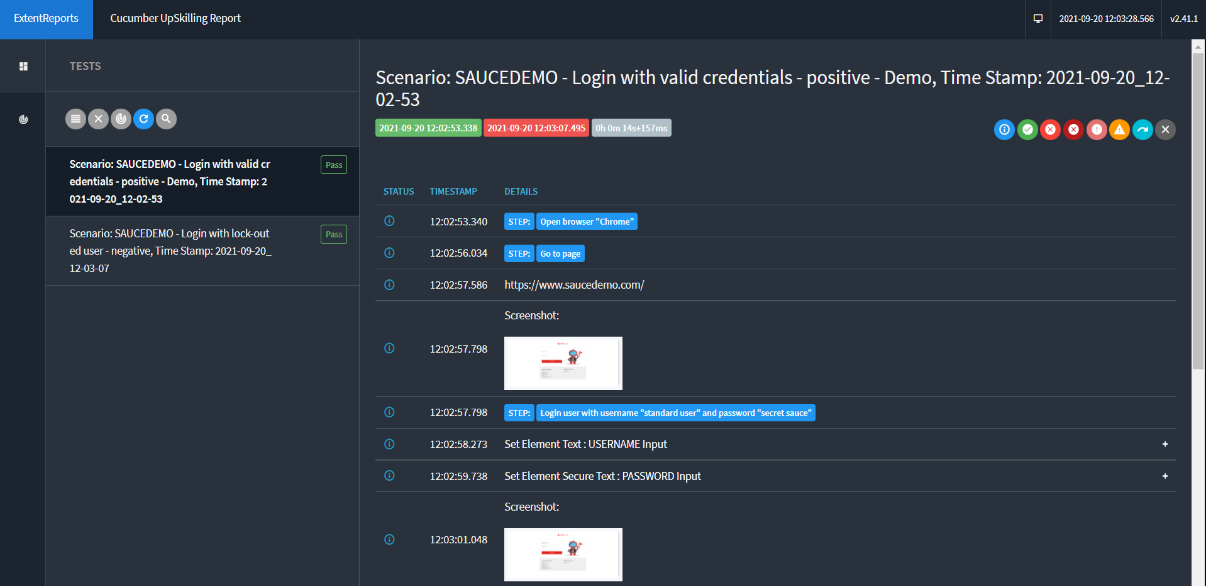
You will see HTML report folder for test you ran.



* + Open the folder and run HTML Document.



* + The HTML report opens in browser. Screenshots and report features are visible.



## Import project to Visual Studio Code

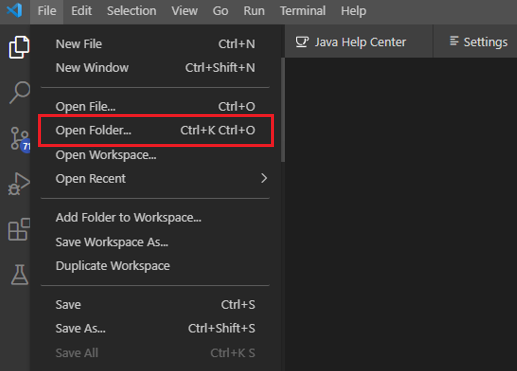
### Run Visual Studio Code

* + Double-click the Visual Studio Code icon.

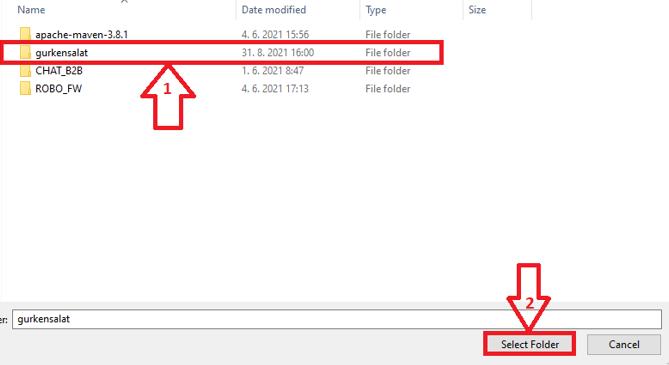


### Import project

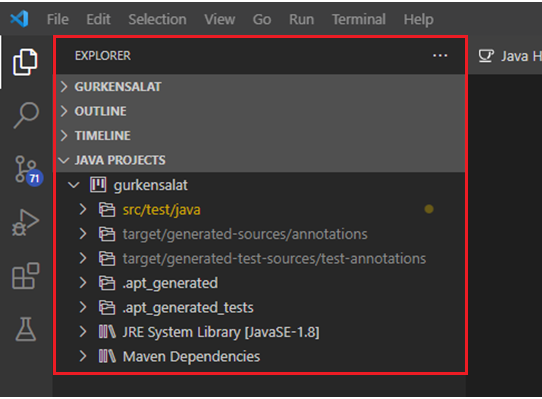
* + From the main menu select *File -> Open Folder*.



* + In the dialog select the directory in which your *gurkensalat* is located (1) and click *Select Folder* (2).

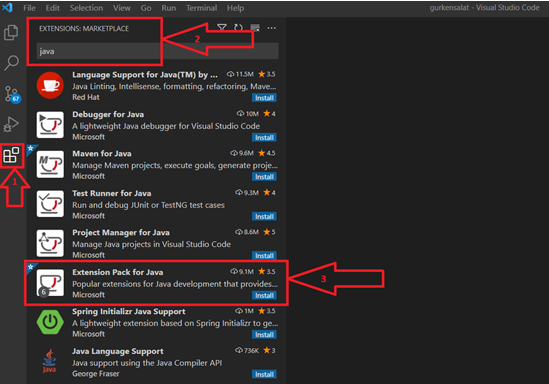


* + *gurkensalat* is now visible in Visual Studio *Explorer*.

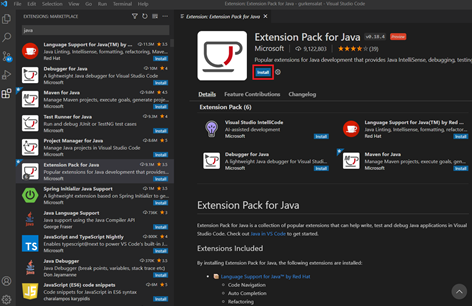


### Adding Extensions to Visual Studio

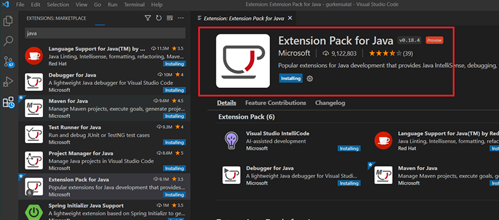
* + In the left panel – click on *Extensions icon* (1).
  + Enter “*Java*” to the search input field (2).
  + Click on the “*Extension Pack for Java*” (3).



* + On the Extension Pack for Java page click *Install* to install Java Extension.



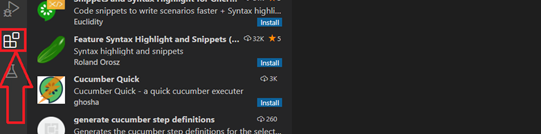
* + Wait for installation process.



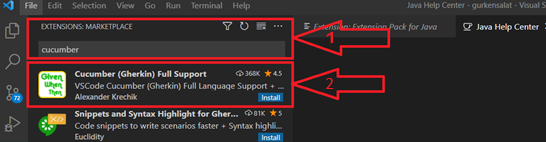
* + After the installation – VS Code will open *Java Help Center* tab.
  + Click on the *skip* button.



* + In the left panel – click on *Extensions icon.*

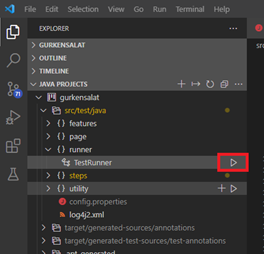
**

* + Enter “*cucumber*” to the search input field (1).
  + Click on the “Cucumber (Gherkin) Full Support” *Install* button (2).

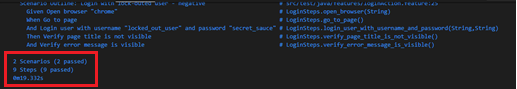


### Run test

* + Once done, in Project explorer – Java Projects, navigate to *TestRunner (gurkensalat/src/test/java/runner/TestRunner.java).*
  + Click Play on the *TestRunner.*

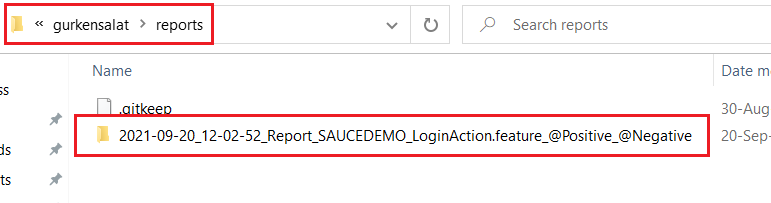


* + Sample test (loginAction.feature) will run. You will see following scenario output in console.

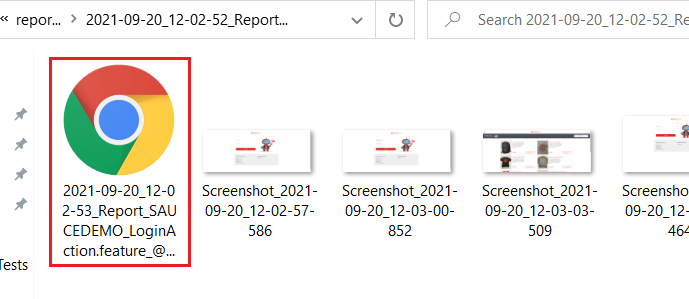


* + In Windows explorer navigate to *report* folder in *gurkensalat* project (e.g. *C:/Workspace/gurkensalat/reports* for Windows users*, /home/workspace/gurkensalat/reports* for Linux and macOS users).

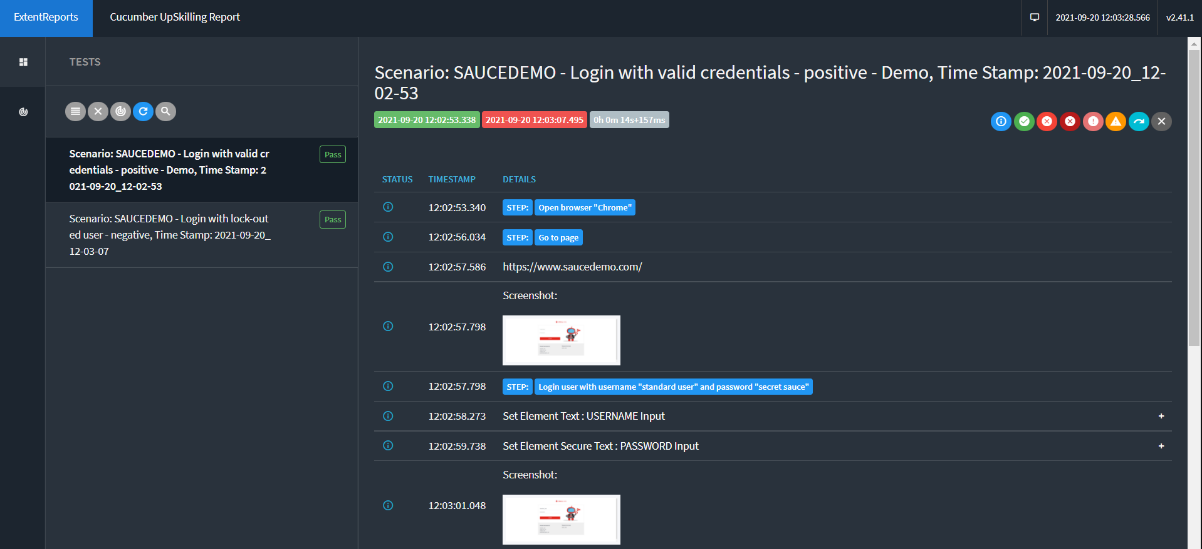
You will see HTML report folder for test you ran.



* + Open the folder and run HTML Document.



* + The HTML report opens in browser. Screenshots and report features are visible.

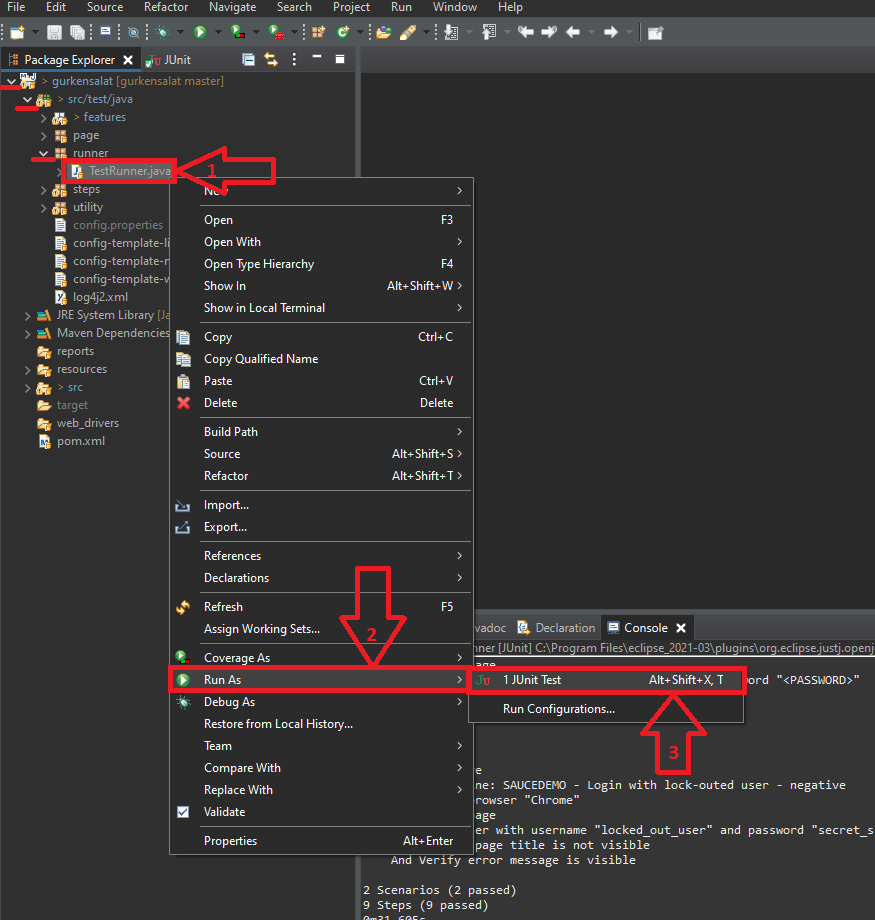


# Test execution

## Windows

### Execution from IDE

* Open IDE (e.g. Eclipse) where you have your *gurkensalat* project imported.
* Right-click on *TestRunner.java* file in IDE Package Explorer (1).
* Click on *Run As* (2).
* Click on *Junit Test* (3).

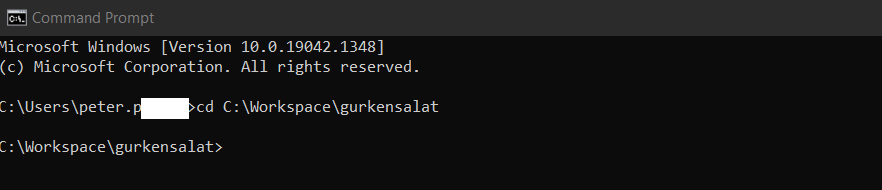


* Test will run again. You will see results in console.

****

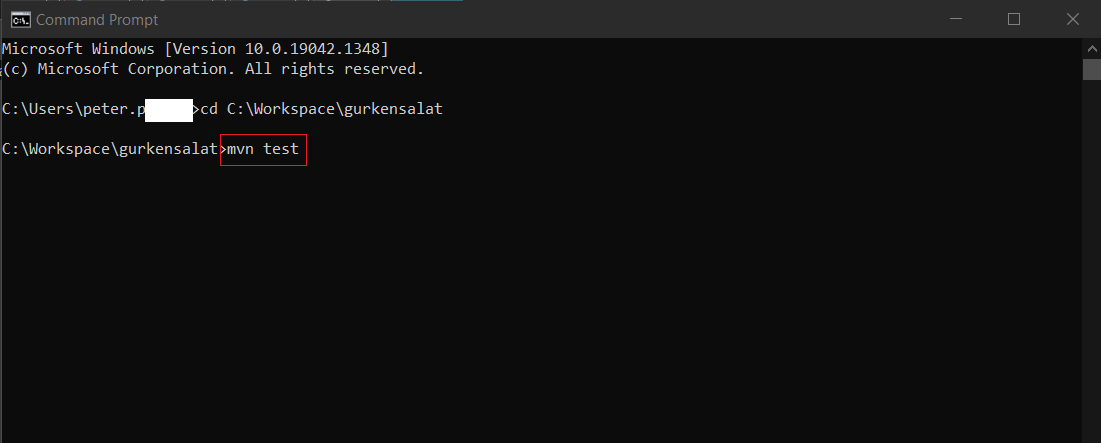
### Execution from Command Prompt

* Open *Command Prompt*.
* Change directory to the root of Gurkensalat project (e.g. C:/Workspace/gurkensalat).

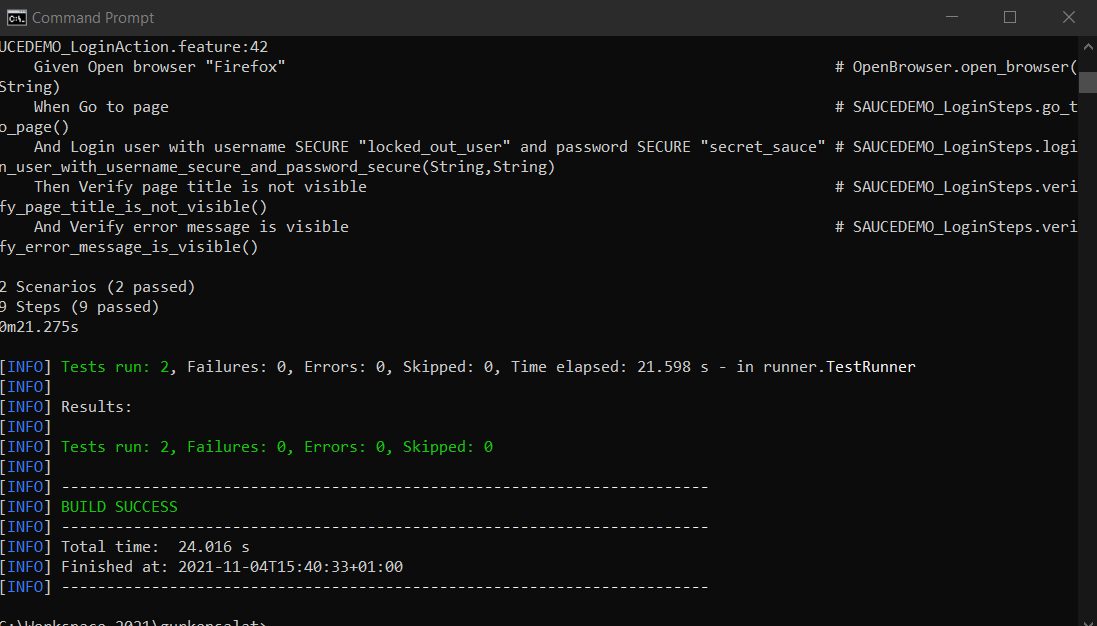


* To run the test run the basic command:

mvn test

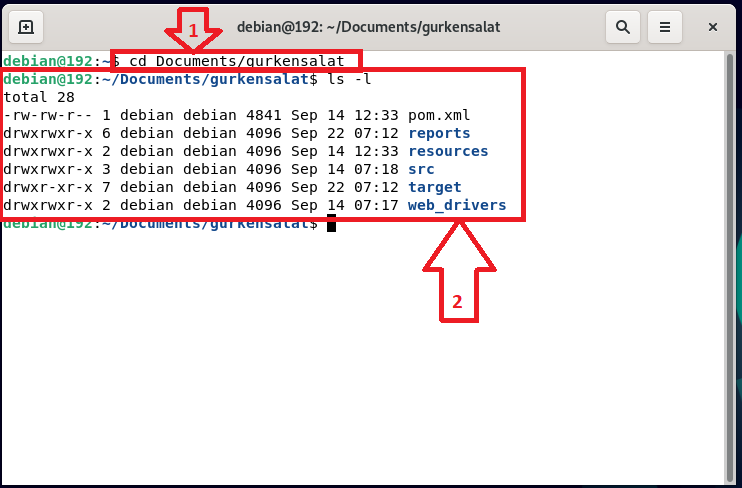


* The test will be executed as it is set in *TestRunner.java* in *@CucumberOption.* You will see the result.



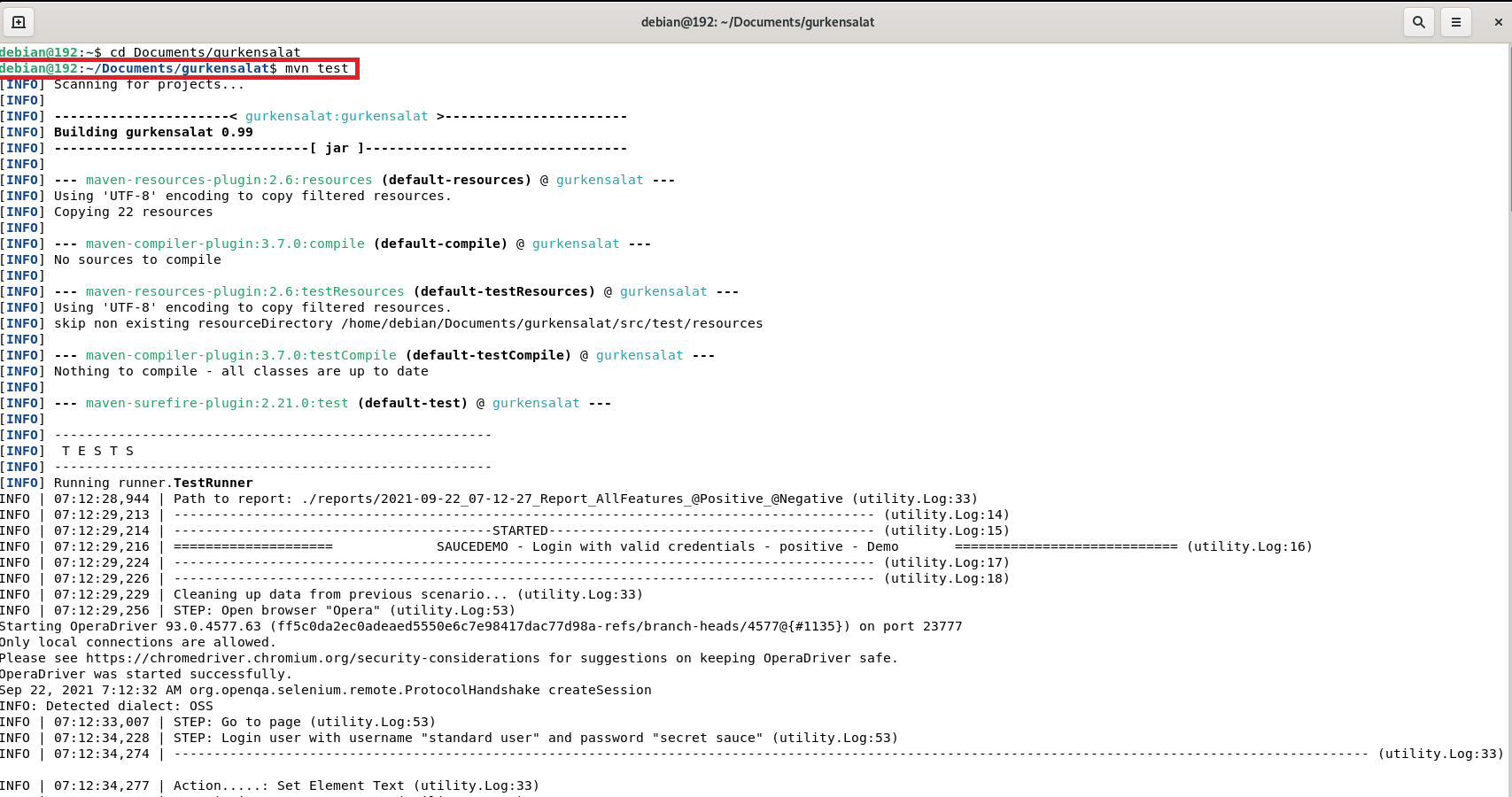
## Linux

### Execution from Terminal

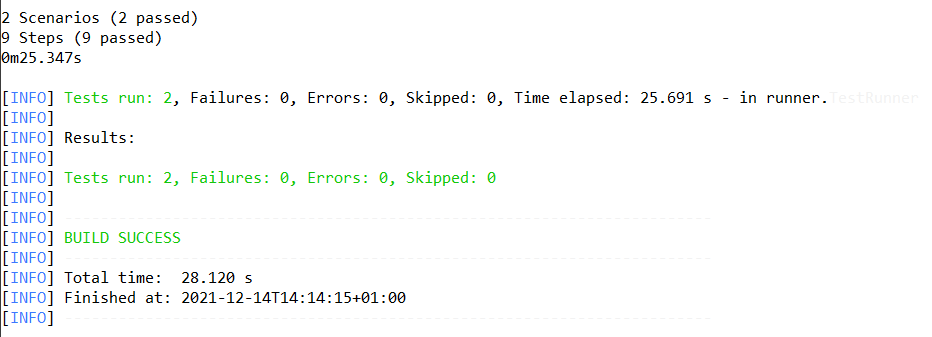
* Open *Terminal* on your local Linux installation.
* Change directory to your location of *gurkensalat* project (1).
* You should be able to see *gurkensalat* directory (2).
* When you are in the gurkensalat directory run the command:

mvn test

* Test will start to execute.



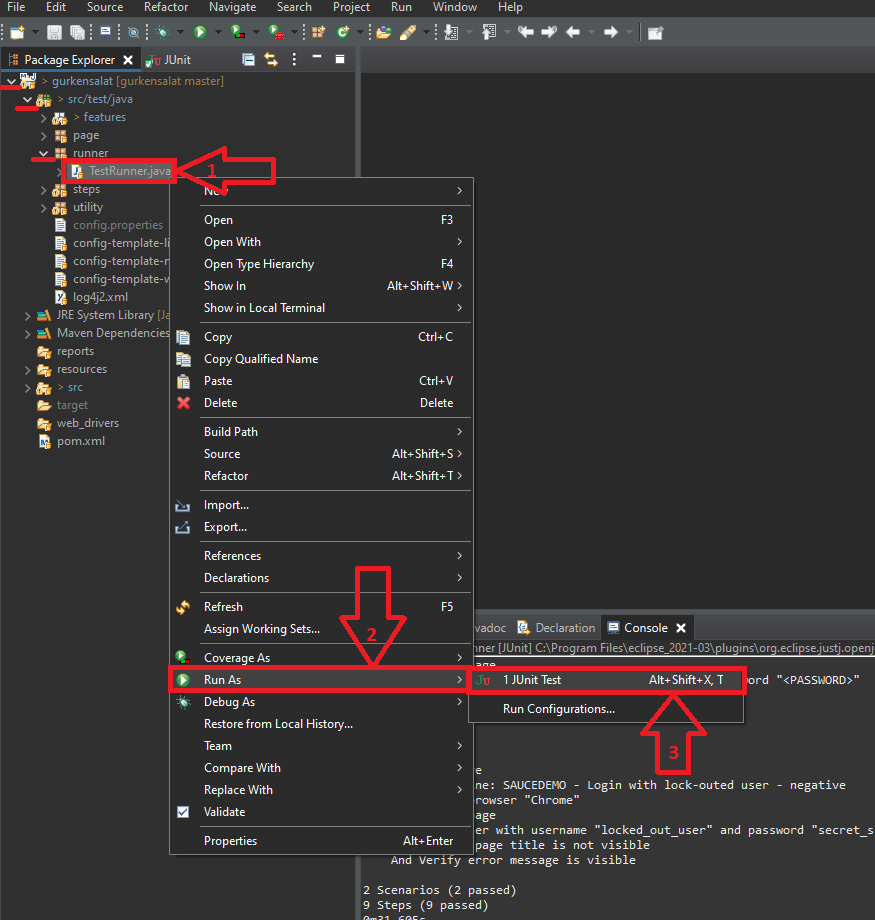
* Test will run. You will see the results.



## Mac

### Execution from IDE

* Open IDE (e.g. Eclipse) where you have your *gurkensalat* project imported.
* Right-click on *TestRunner.java* file in IDE Package Explorer (1).
* Click on *Run As* (2).
* Click on *Junit Test* (3).

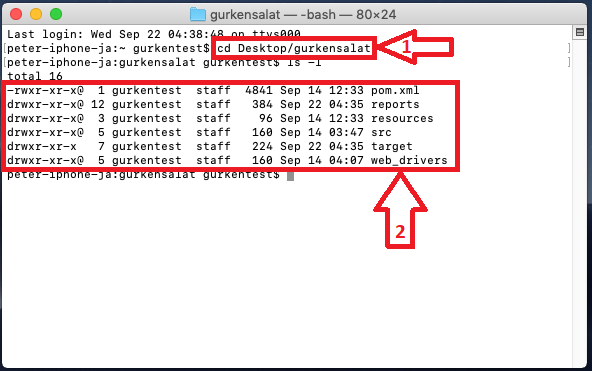


* Test will run. You will see results in console.

****

### Execution from Terminal

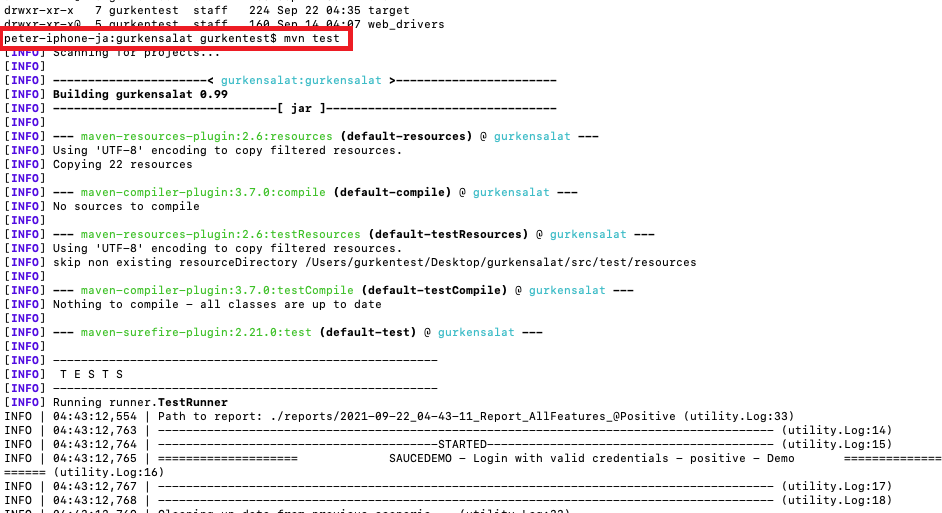
* Open *Terminal* on your local macOS installation.
* Change directory to your location of *gurkensalat* project (1).
* You should be able to see *gurkensalat* directory (2).



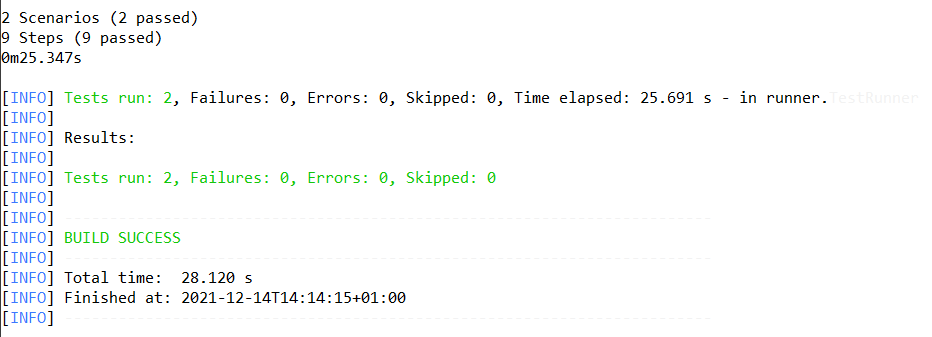
* When you are in the *gurkensalat* directory run the command:

mvn test

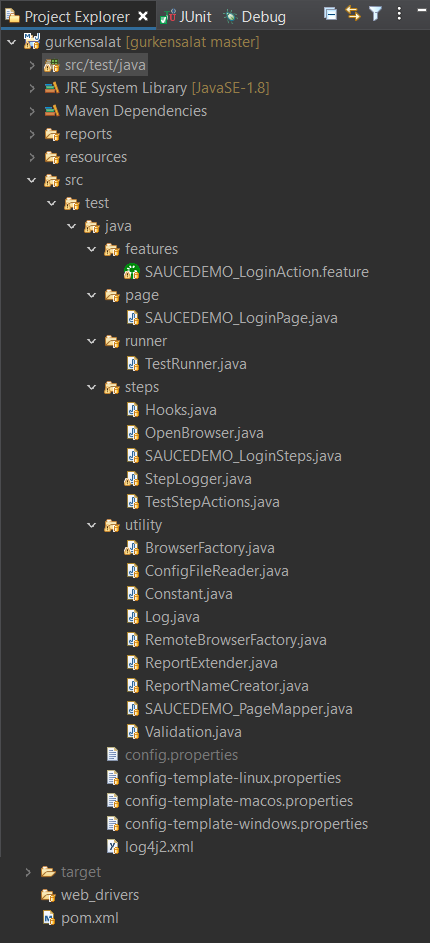
* Test will start to execute.



* Test will run. You will see the results.



# Gurkensalat project structure



*pom.xml* - contains information about the project and configuration details used by Maven to build the project

*config.properties -* used to store project configuration data or settings

*TestStepActions.java* - contains predefined actions/operations on the web application

*steps* – contains step definition. It maps the test case steps in the feature files

*runner – TestRunner.java class* is starting point for Junit to start executing your tests and contains settings for your test

*page* - contains classes with definition of page elements - page object model (POM)

*features* - package where all of your cucumber .feature files (scenarios) are located

*web\_drivers –* folder used to locate your web drivers such is Chromedriver

# Gurkensalat content

*Gurkensalat* is Cucumber-based framework with some useful enhancements.

For detailed Cucumber documentation, please visit official Cucumber website: [*https://cucumber.io/docs/*](https://cucumber.io/docs/)

## Cucumber

Cucumber is a software tool that supports Behavior-Driven Development (BDD). Central to the Cucumber BDD approach is its ordinary language parser called Gherkin.

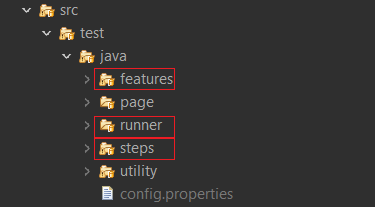
Gherkin is a set of grammar rules that makes plain structured text enough for Cucumber to understand.

For detailed Gherkin documentation, please visit official Gherkin website:

*<https://cucumber.io/docs/>gherkin/*

For Cucumber projects there are three directories with appropriate classes that we want to focus on:

* ***features***
* ***runner***
* ***steps***

******

### Features

***Features package*** is a place where all of your cucumber .feature files are located.

***Feature File*** is an entry point to the Cucumber tests. Here you will describe your tests in descriptive language - Gherkin. A feature file can contain one or multiple scenarios in a single file.

In order for Cucumber to automatically detect the feature file it needs to end with *.feature* file extension. Every *.feature* file conventionally consists of a single software feature we are going to test.

Feature file consists of following components:

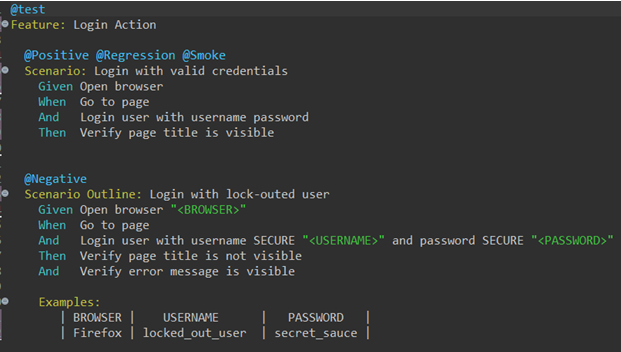
* ***Feature***
* ***Scenario / Scenario Outline***
* ***Examples***
* ***Tags***

***Feature*** describes the current test list which will be executed. Usually, it has the same name as .feature file itself (e.g. loginAction.feature will have Feature: Login Action ).



***Scenario*** describes a particular test case, the steps and expected outcomes.

Example below shows the simplest way to write a feature file scenario (note for masters: yep, it is not parametrized…).



***Scenario Outline***, the same *scenario* can be executed for multiple data sets - *Examples*.

Typically there are two types of steps:

* Actions (like Open browser)
* Verifications (like Verify error message is visible)

Every action step should be followed by a verification step(s).

Parametrization

**Action step:** *Open browser “<BROWSER>”*

Format: Action + Object + Parameter

**Verification step*:*** *Verify error message “ERROR\_MESSAGE” is visible*

*Format: Verify* + Object + Parameter + Condition

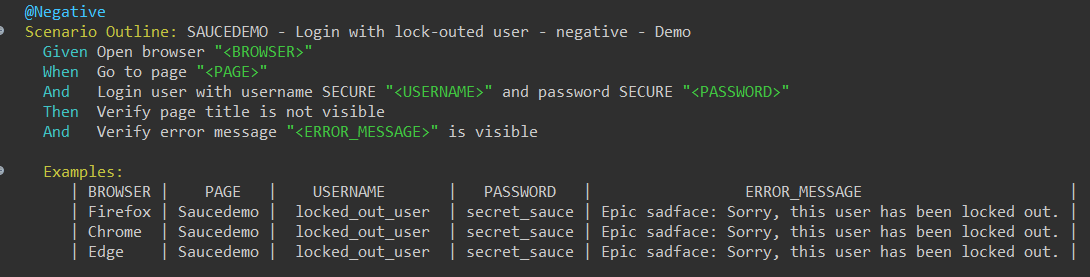
**Keep it clear and simple!**

Object

(browser)

Action (Open)

Parameter (“<BROWSER>”)



Object

(error message)

Condition

(is visible)

Verify

Parameter (“<ERROR\_MESSAGE>”)

***Examples*** are data sets (see screenshot below). They are provided in a tabular structure. One set of parameters corresponds to one line. Individual parameters are in columns and are separated by “|” (e.g. |USERNAME|PASSWORD|). Each column has a Header with the name of the parameter. During a test run one test will be executed for each row in examples.

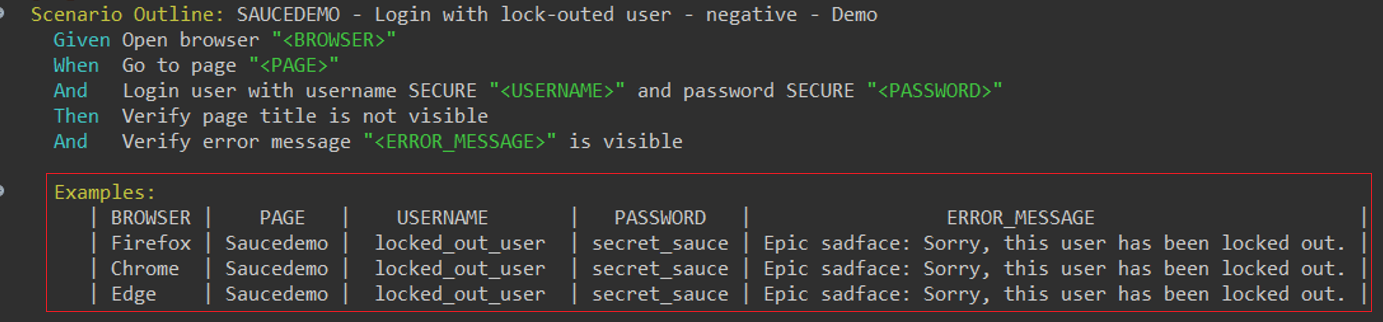
In the screenshot below the test will be executed three times in one test - each execution with one row of parameters.

Follow column BROWSER

First test execution will be with *Firefox* browser

Second test execution will be with *Chrome* browser

Third test execution will be with *Edge* browser

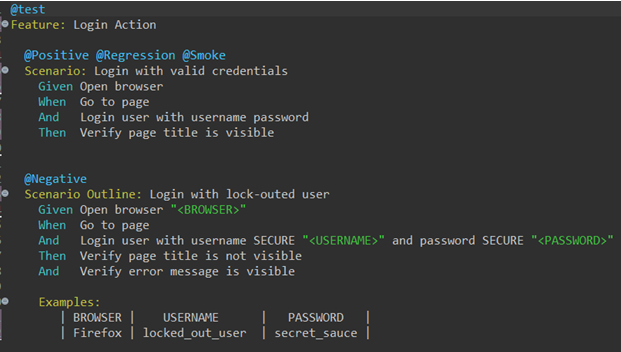


***@Tags*** are the special part of a feature file***.*** In Cucumber they are used to associate a test with particular attribute of test (e.g. smoke, regression, etc.). Tags are optional, not mandatory. You can also use more tags to mark one scenario. Tags are used to group Scenarios/Scenario outlines for execution (e.g. run only *smoke* tests, see details in later chapter.)

Example of a tag:

Every tag begins with at “@”

You can create a tag for just one scenario



You can create a tag for the whole feature file, for all scenarios

Tags are separated by space

### Steps

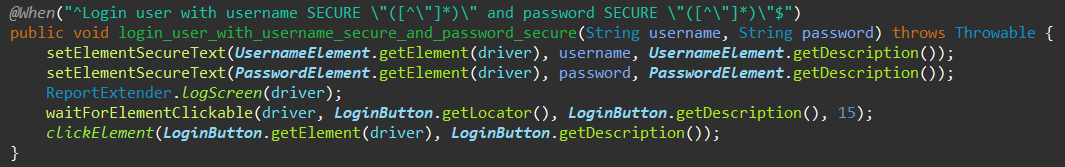
Step definitions map the test case steps from feature files (introduced by Given/When/Then) to code. For a step definition to be executed it must match the step in the feature file. The name of the step in the feature file has to equal the brackets content in the step definition. Step definition itself contains the actual code to be executed in the test scenario in the feature file. The code is composed of step actions from TestStepActions.java.

Sample step definition example:

Name of the step from feature file (contains regex of parameter)

Given/When/Then Gherkin annotation

Method name should match feature file step, spaces are replaced by underscores



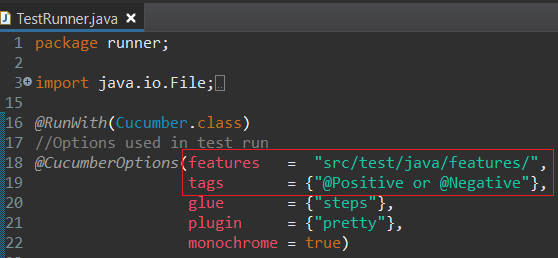
Step definition contains step actions from TestStepActions.java

### Runner

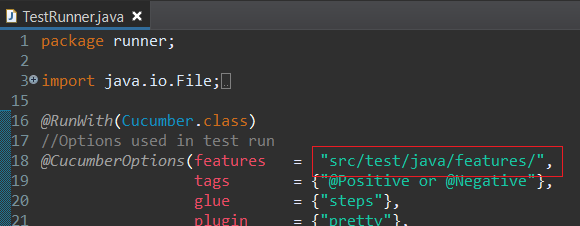
*TestRunner.java* is located in the *runner* folder. It is responsible for test execution.

@CucumberOptions in TestRunner.java are execution settings for your tests. You have a couple of options in there. The tests will be executed according to what is set in here. Let's look at the first two @CucumberOptions:

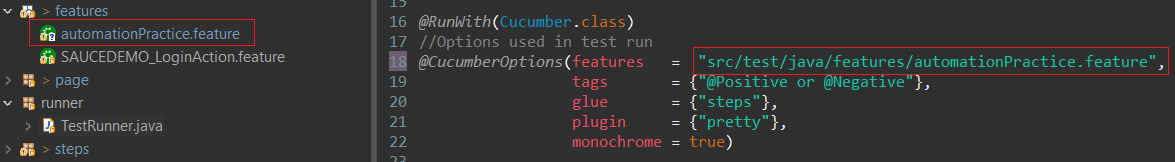
* ***features***
* ***tags***



**features** - represent the path to feature files. It helps Cucumber to locate the feature files in the project folder structure. All we need to do is to specify the path to folder and Cucumber will automatically find all the .*features* files in the folder. By default, the feature file location is set to src/test/java/features/

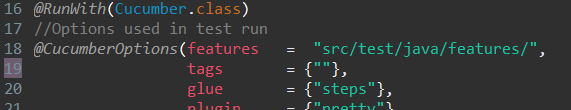


You can also specify which .feature file you want to execute by adding the name of the feature file.

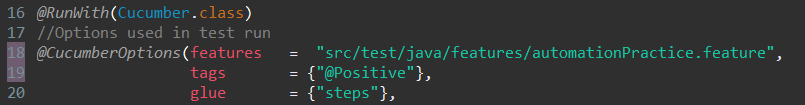


**tags** - option is telling Cucumber to execute selected scenarios with particular tags. You can keep tags empty for running all scenarios.

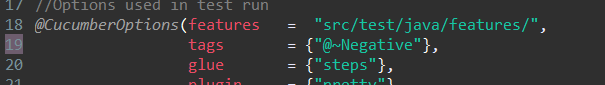
* Following CucumberOptions will execute all feature files and Scenarios, without tag limitations



* Next CucumberOptions will execute only automationPractice.feature and only Scenarios which are tagged as @Positive.

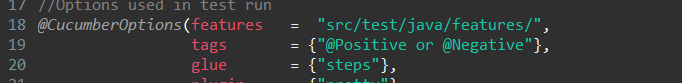


* The following CucumberOptions will run all Scenarios without the @Negative tag (using @~Negative).



* Logical ***OR*** – This CucumberOptions will execute Scenarios that have at least one tag from *tags* option.

Separate a list of tags by commas for a logical OR tag expression (e.g. "@Positive, @Negative"). You can also use *or* word (e.g. "@Positive or @Negative").



* Logical ***AND -*** This CucumberOptions will execute Scenarios that have all tags from *tags* option.

Specifying multiple tag arguments creates a logical AND between each tag expression.

You can also use *and* word.

(e.g. "@Positive and @Regression").



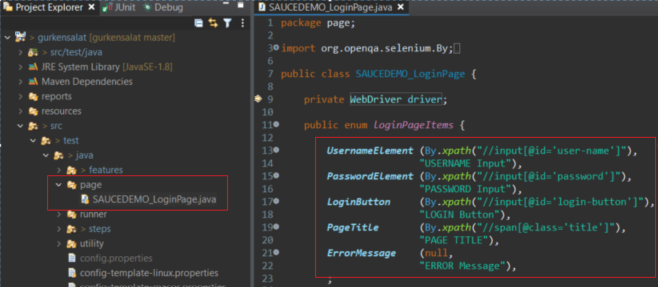
## Extended Components of Gurkensalat

Gurkensalat is a custom framework. It contains some components which help to develop tests, navigate the code and help with code maintenance. The most used for everyday work are two of them:

* ***POM - Page Object Model***
* ***Test Step Actions***

### POM - Page Object Model

POM is a design pattern that creates object repository for web UI elements. Elements are defined only once and stored in a structured way according to pages. Definition of a page element is not mixed with functional code – they are separated. The advantage of the model is that it reduces code duplication and improves test maintenance. Pages of elements are located in *page* folder.

 POM example:

Element locator

Element description

POM class location

Element name

As you can see, there are some patterns that are good to adhere to.

***Page Object Model***

***File/class name:*** SAUCEDEMO\_LoginPage.java

Format:*PORTALNAME in upper case + \_ + PageName in camel case + Page*

Other examples: ALZA\_ProductPage, GOOGLE\_SearchPage,…

***Element name:*** LoginButton

Format: Name of element + Type of element in camel case

Other examples: PageTitle, SearchInput,…

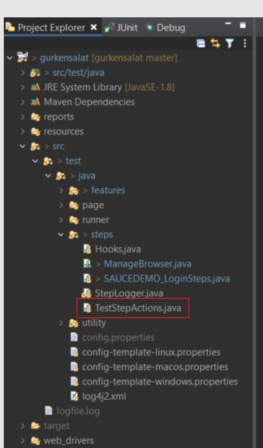
***Element description:*** LOGIN Button

Format: Name of element in UPPER CASE + Type of element in Camel case

Other examples: PAGE Title, SEARCH Input,…

### Test Step Actions

Test Step Actions are located in *steps* folder/package and in *TestStepAction* file/class.



It is a set of user actions to perform any operation on the web application such as click, select drop-down box, wait, etc.

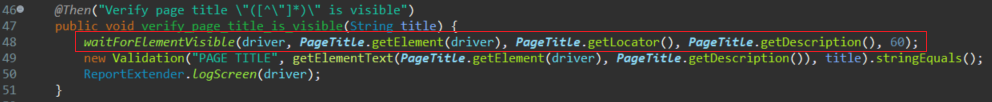
It contains predefined selenium-based actions that are adapted for easier and more elegant use in test steps. There are prepared ready-to-use actions for almost all common actions on web elements. They are ready-made and they serve as an accelerator so you don’t have to create them. They include features like automatic reporting to HTML report and to console.

For detailed information about *Test Step Actions* check last chapter of this document.

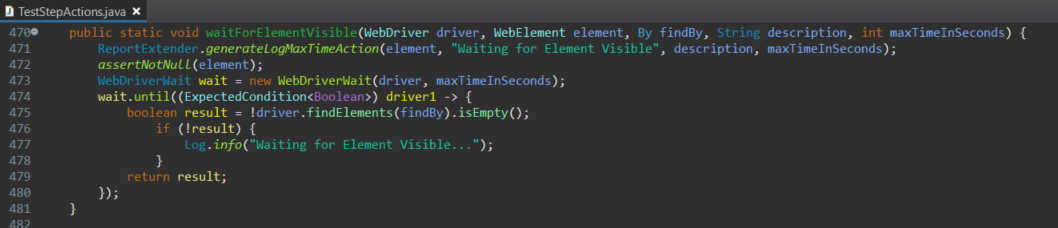
Feature File Step with Test Step Action, example:

Test Step Action

Feature File Step



Definition of one of the Test Step Actions – waitForElementVisible:



Test Step Action definition

contains features like reporting

# HTML Reports

HTML reports are one of the simplest ways to quickly see the test results and share them with others. Report in *Gurkensalat* framework is built on ***Extent Reports*** (one of the best available reporting on market). *Extent Reports* are one of the best built-in ways to generate customizable HTML reports with a pleasing user interface in Selenium web driver. It is an open source library that can be easily configured.

After the test execution, you can find HTML report in reports folder in the root of gurkensalat project.

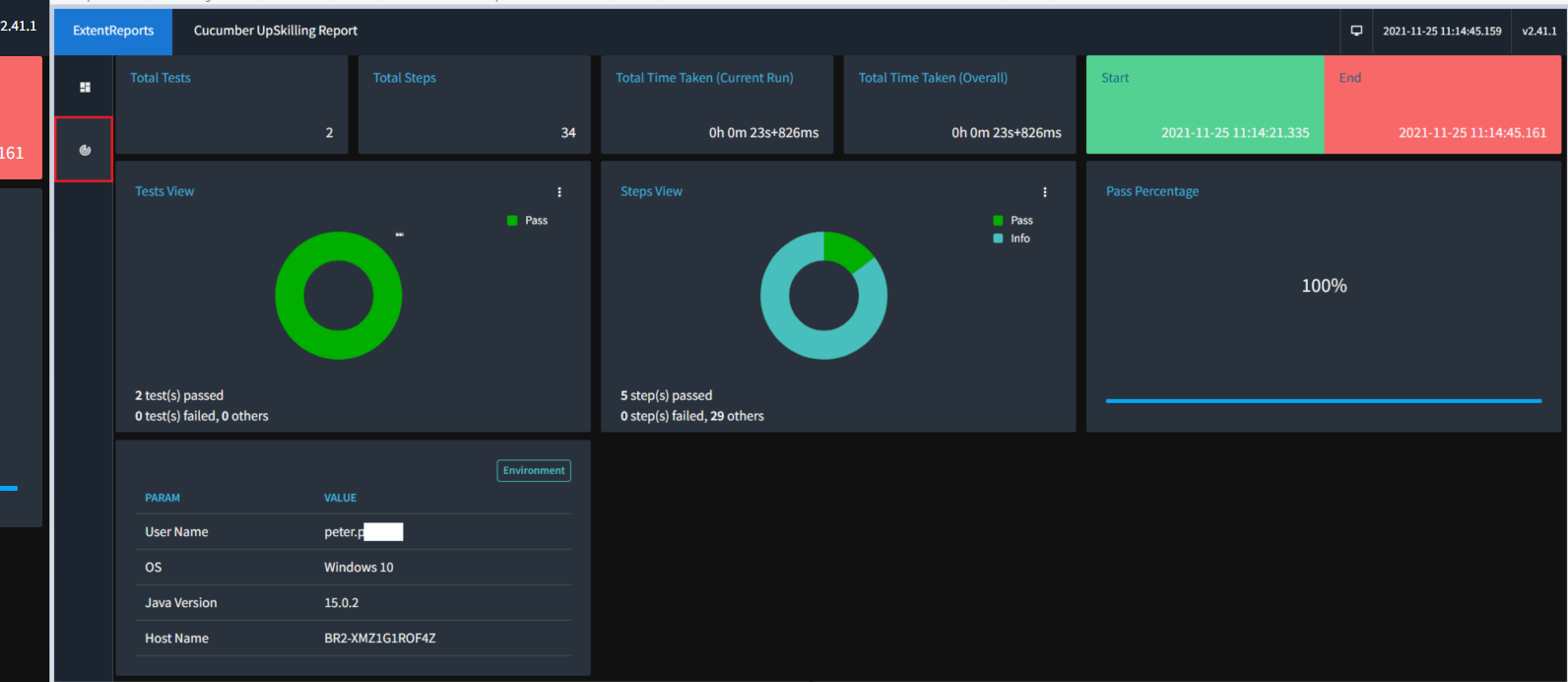
It consists of two sections:

* + - ***Summary section***
    - ***Test result section***

## Summary section

In this section you can see statistics of how many tests and steps were executed in this test run, how many of them passed and how many failed.

You can also see some other information, e.g. when execution started and ended or who executed the test and under which operating system.



Execution Timing

Steps View

Tests View

Environment

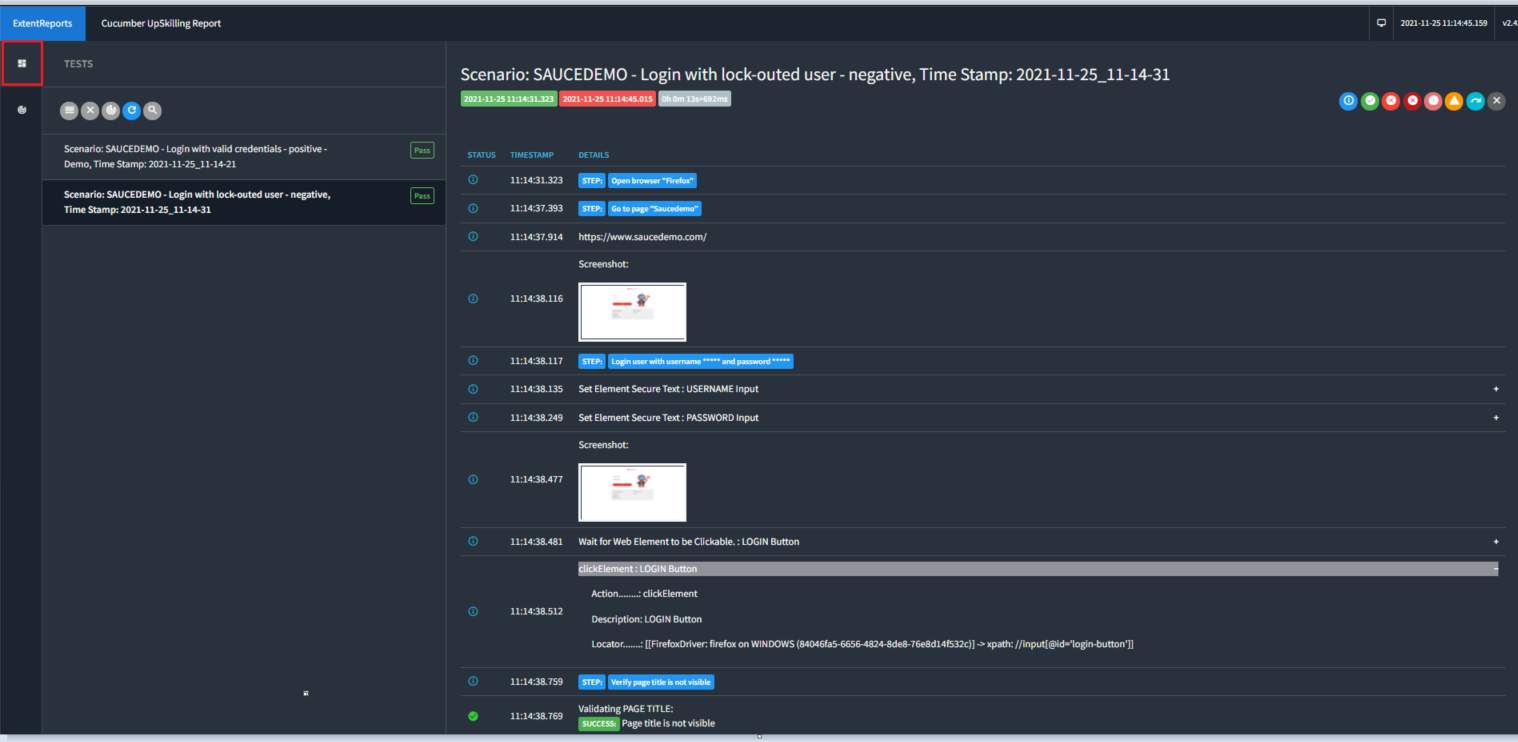
Summary section

## Test result section

In test result section with test details you can see detailed information of the executed test run.

In every report, you can see some very important information:

* + - * ***Actual status*** of the executed test run (Pass/Fail/Warning)
      * ***Test steps*** - action performed over web element with more information in step details (Action, description, locator, timing)
      * ***Screenshots*** - e.g. made after each page action
      * ***URLs*** are recorded
      * ***Validations*** - e.g. displays expected and actual values comparison



Actual status

Screenshot

Test Step

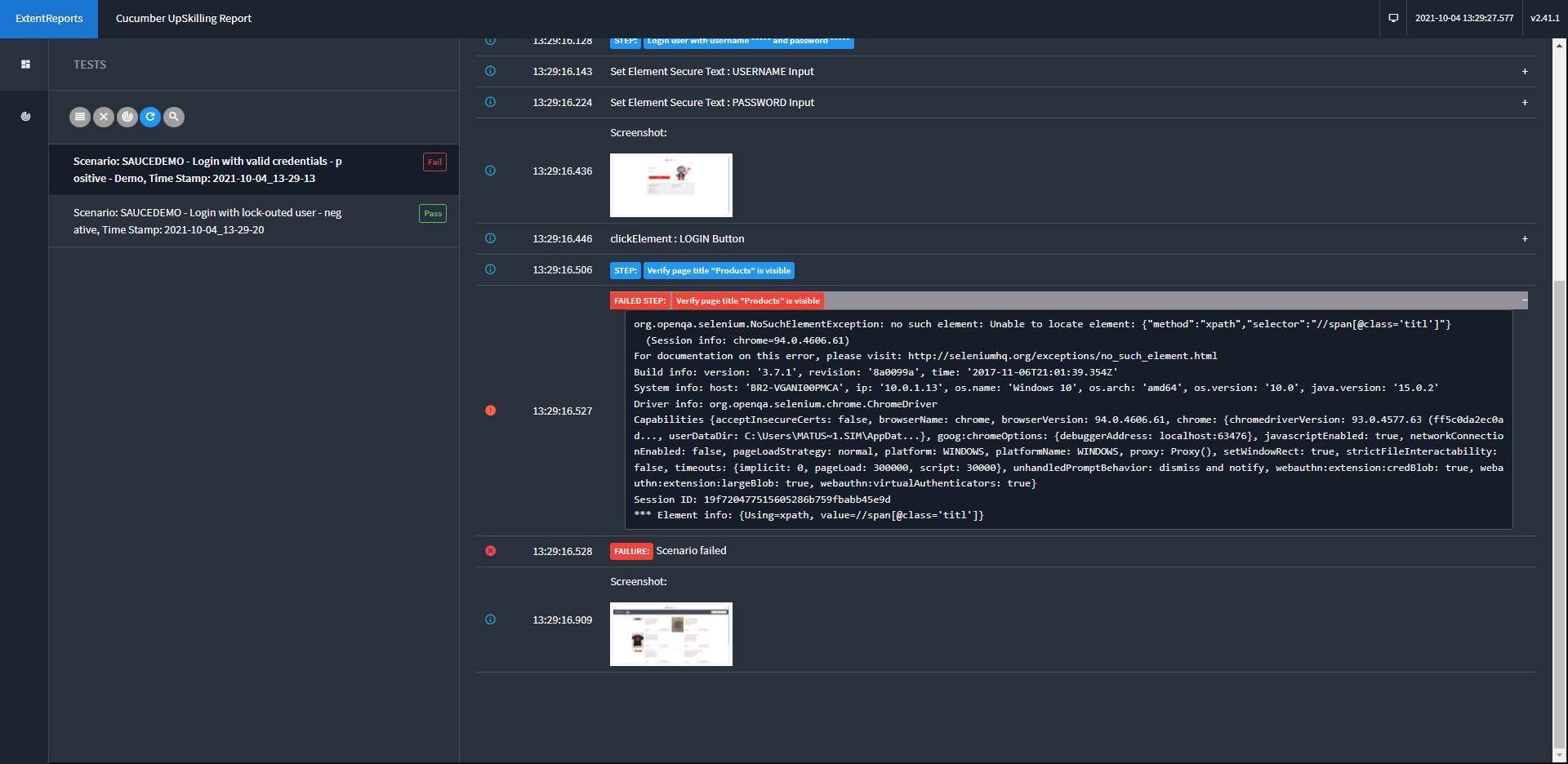
Validation

URL

Test result section

If test is failed, additional information is recorded:

* + - * **Message**
      * **Stack Trace** - in case of failure
      * **Screenshot** - created right after error appears, so we know what exactly happened



Actual status

Stack Trace

Message

Screenshot

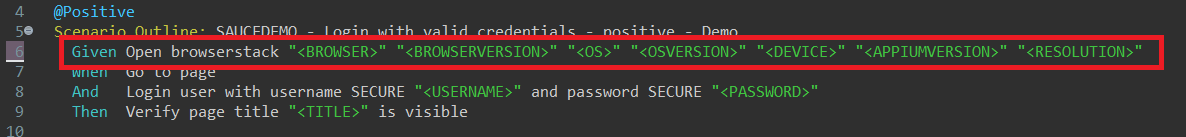
# Cross-browser remote testing

***Gurkensalat*** is offering an integration with cross-browser / cross-device testing services.

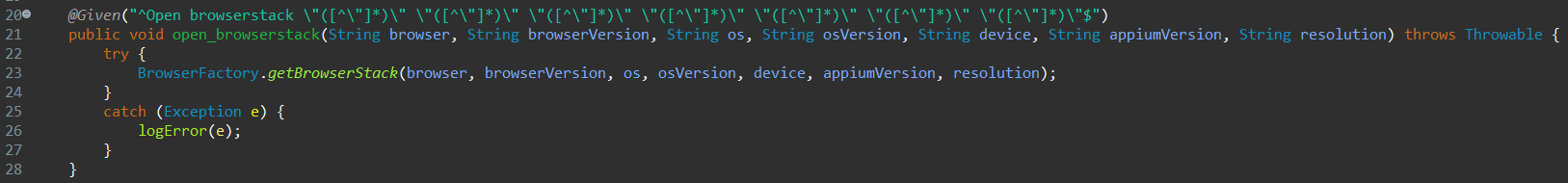
## BrowserStack integration

***BrowserStack*** is a cloud web and mobile testing platform that provides developers with the ability to test their websites and mobile applications across on-demand browsers, operating systems and real mobile devices.

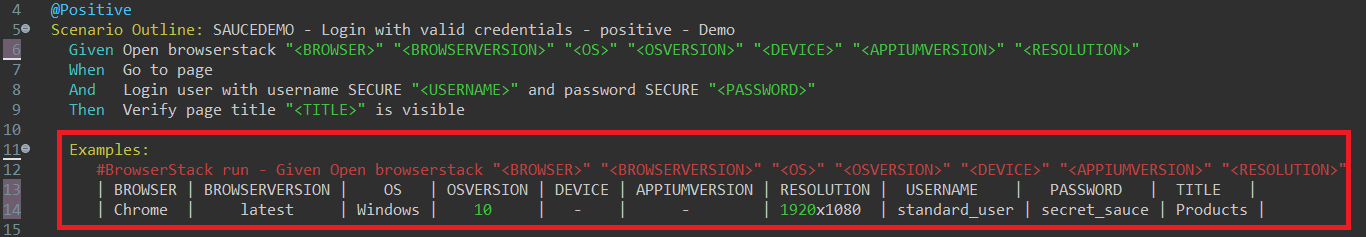
* In feature file you need to use *Given* *Open browserstack "<BROWSER>" "<BROWSERVERSION>" "<OS>" "<OSVERSION>" "<DEVICE>" "<APPIUMVERSION>" "<RESOLUTION>"* instead of *Given Open browser "<BROWSER>"*

**

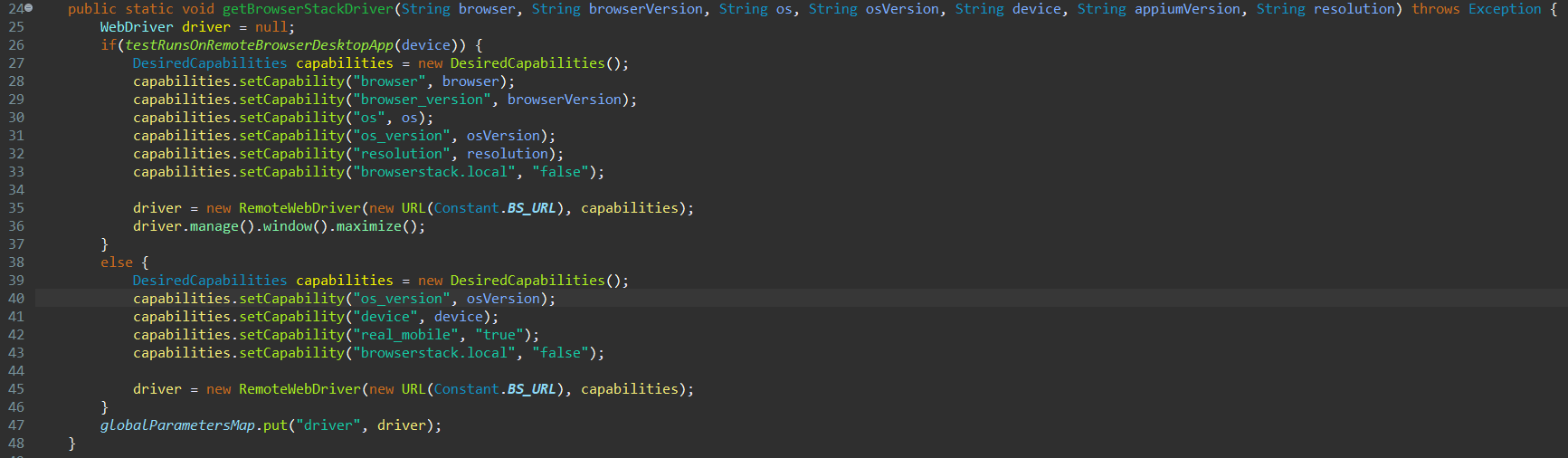
* *open\_browserstack* method is prepared in *steps | ManageBrowser* already.



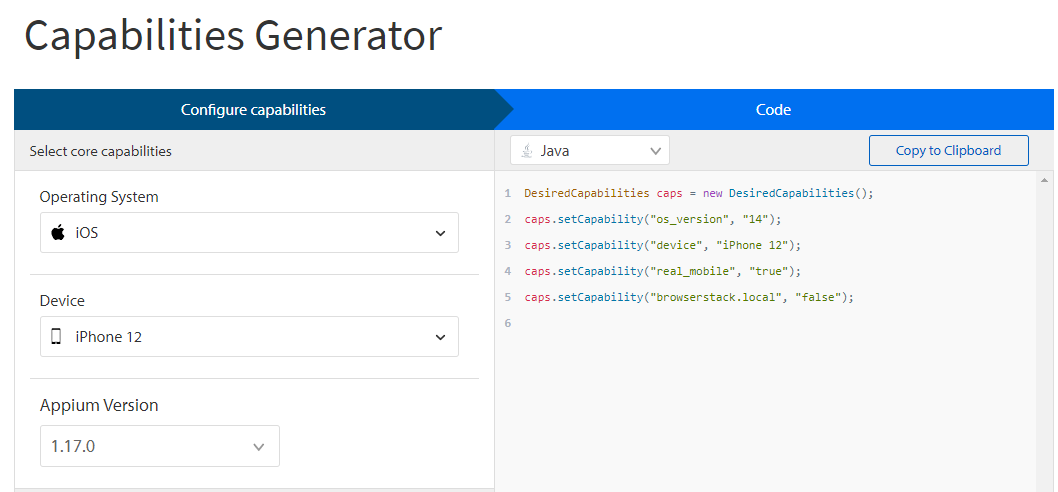
* In *Examples,* fill all parameters (capabilities) required for remote execution.



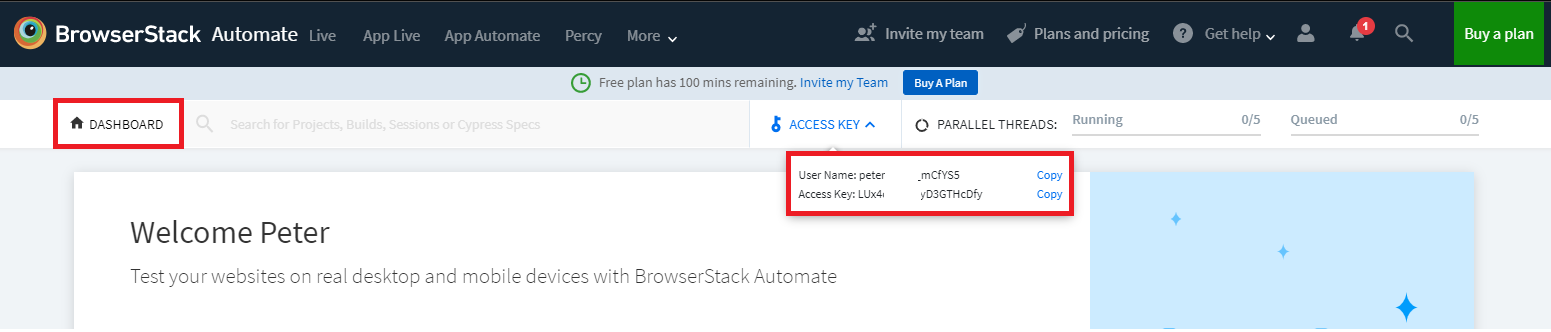
* In *utility | RemoteBrowserFactory* in *getBrowserStackDriver* method you can see all required parameters. There are different parameters for desktop app and mobile app.



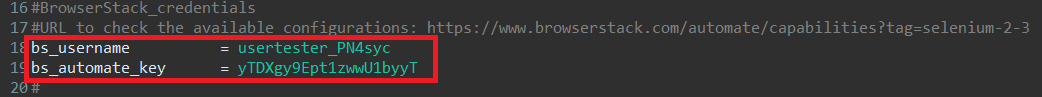
* You can visit [*https://www.browserstack.com/automate/capabilities?tag=selenium-2-3*](https://www.browserstack.com/automate/capabilities?tag=selenium-2-3)for Capabilities Generator. Set your system configuration. You will see all required capabilities.



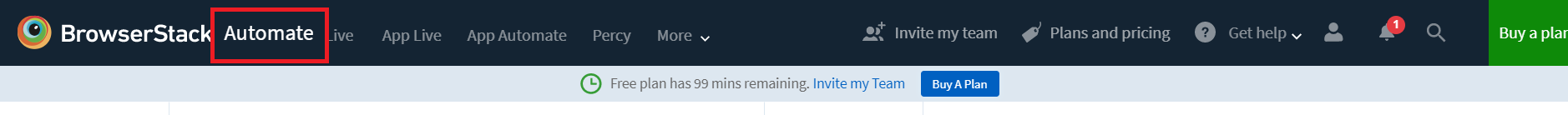
* Create your BrowserStack account. <https://www.browserstack.com/users/sign_up>
* On BrowserStack page, navigate to Dashboard and check for your *User Name* and *Access Key.*



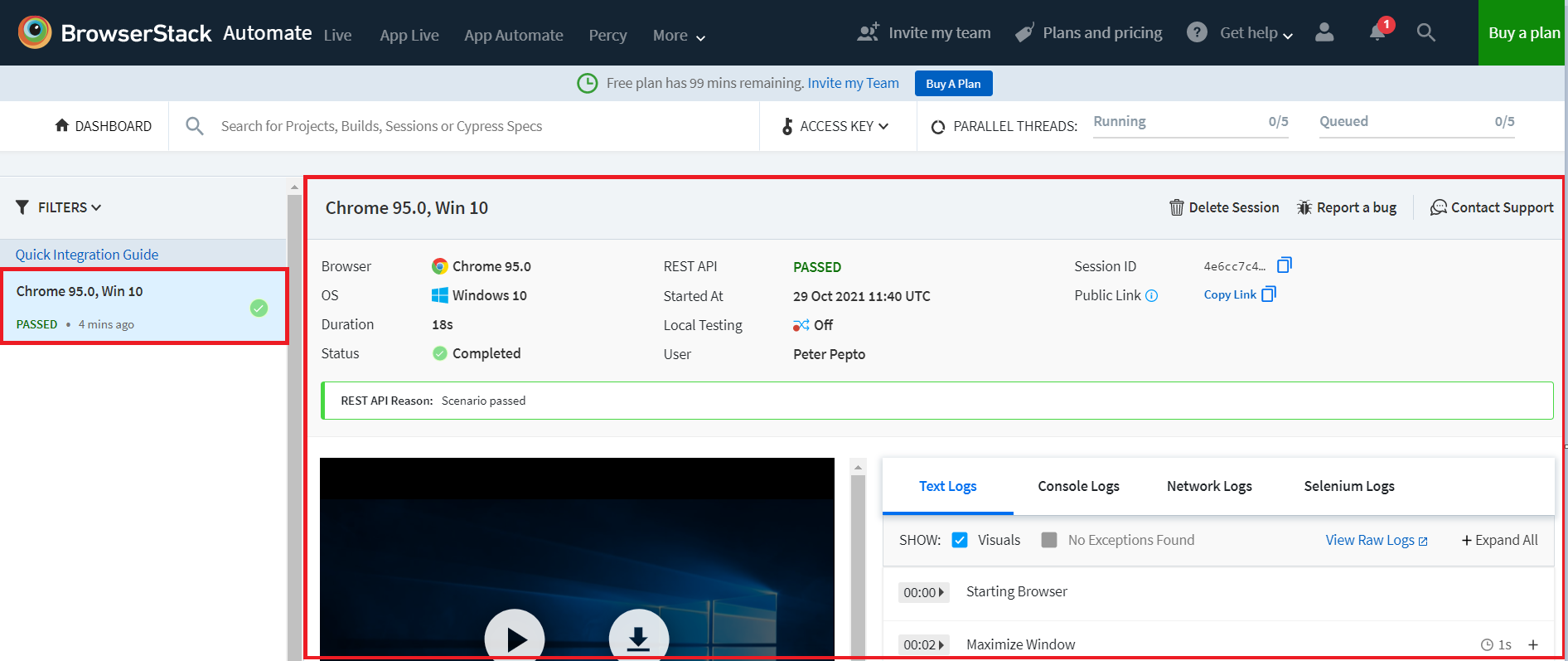
* In config.properties you have to set *bs\_username* (your BrowserStack *User Name*) and *bs\_automate\_key* (your BrowserStack *Access Key*).



* You can run your test in the standard way.
* HTML report will be created as usual.
* On BrowserStack page, navigate to *Automate* tab.



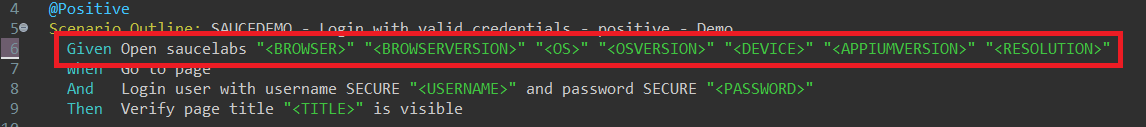
* You can see the test result of the test you ran.



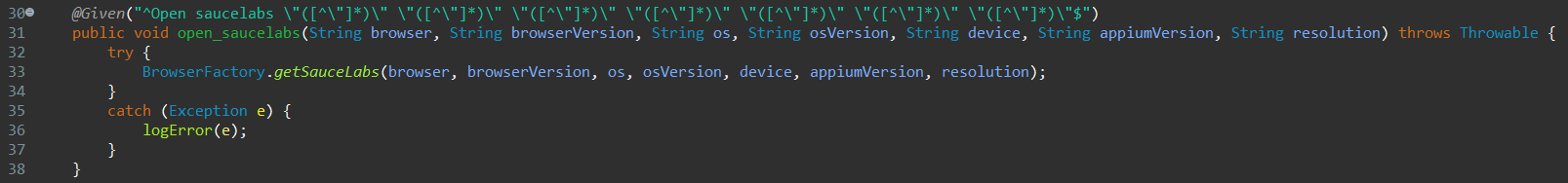
## Sauce Labs integration

**Sauce Labs** is a cloud-hosted, web and mobile application automated testing platform that provides developers with the ability to test their websites and mobile applications across on-demand browsers, operating systems and real mobile devices.

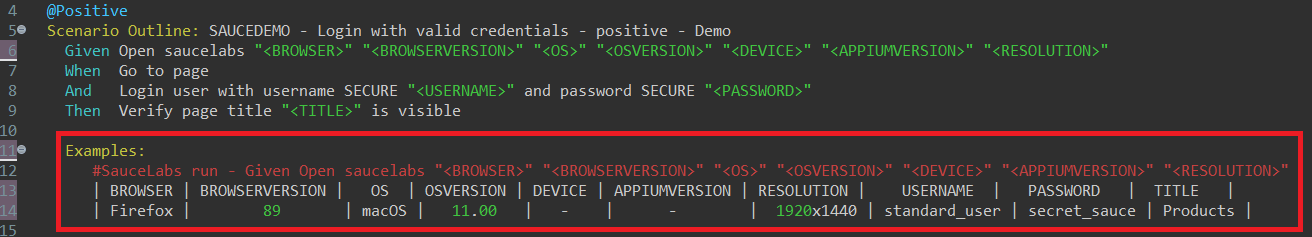
* In feature file you need to use *Given Open saucelabs "<BROWSER>" "<BROWSERVERSION>" "<OS>" "<OSVERSION>" "<DEVICE>" "<APPIUMVERSION>" "<RESOLUTION>"*instead of *Given Open browser "<BROWSER>"*

**

* *open\_saucelabs* method is prepared in *steps | ManageBrowser* already.

**

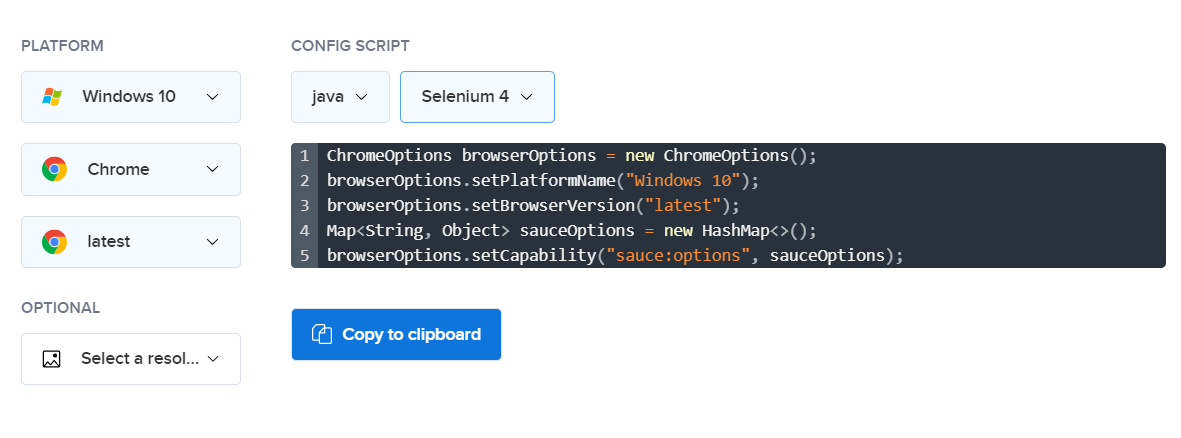
* In *Examples,* fill all parameters (capabilities) required for remote run.



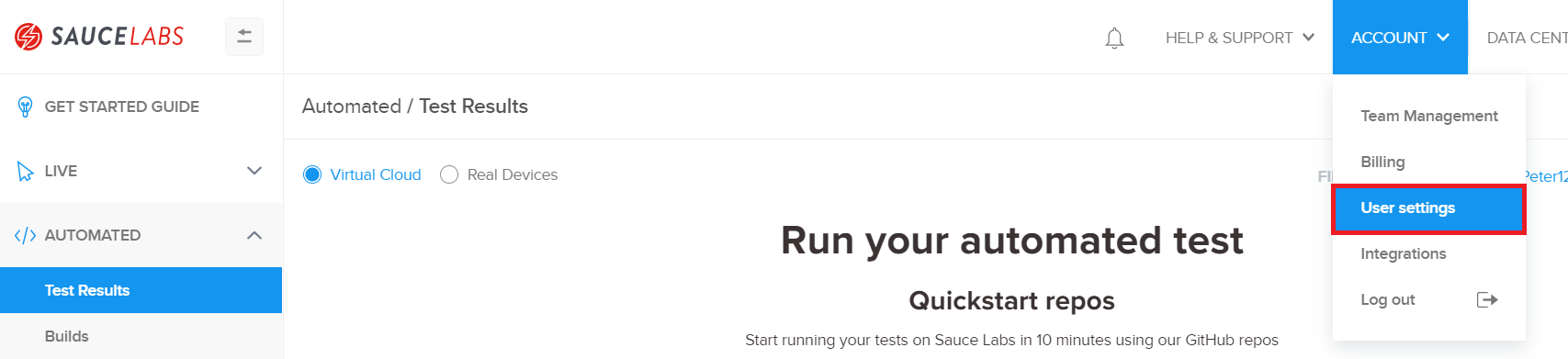
* In *utility | RemoteBrowserFactory* in *getSauceLabsDesktopDriver* method you can see all required parameters. There are different parameters for desktop app and mobile app.



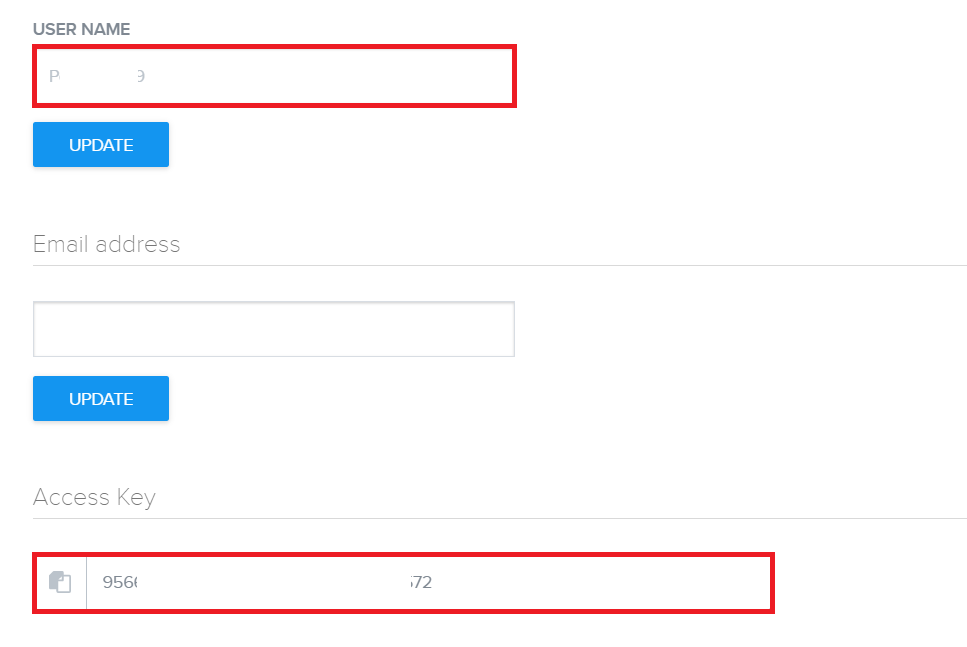
* You can visit <https://saucelabs.com/platform/platform-configurator> for Capabilities Generator. Set your system configuration. You will see all required capabilities.



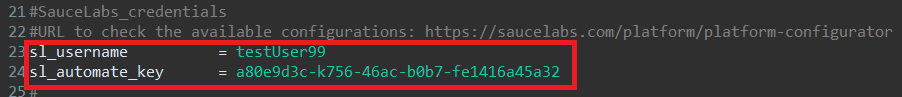
* Create your Sauce Labs account https://saucelabs.com/sign-up.
* On Sauce Labs page, navigate to *ACCOUNT | User settings.*



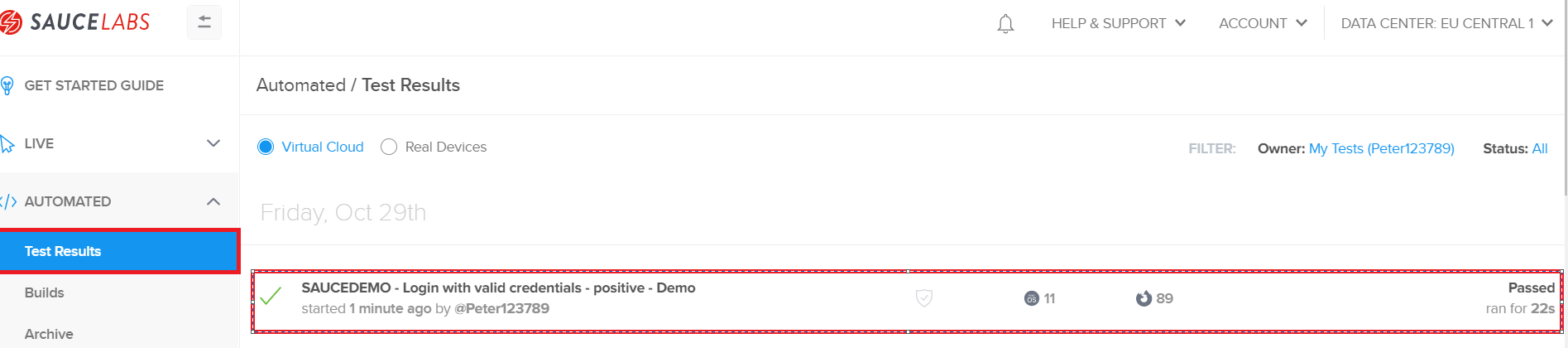
* Check for your *USER NAME* and *Access Key.*



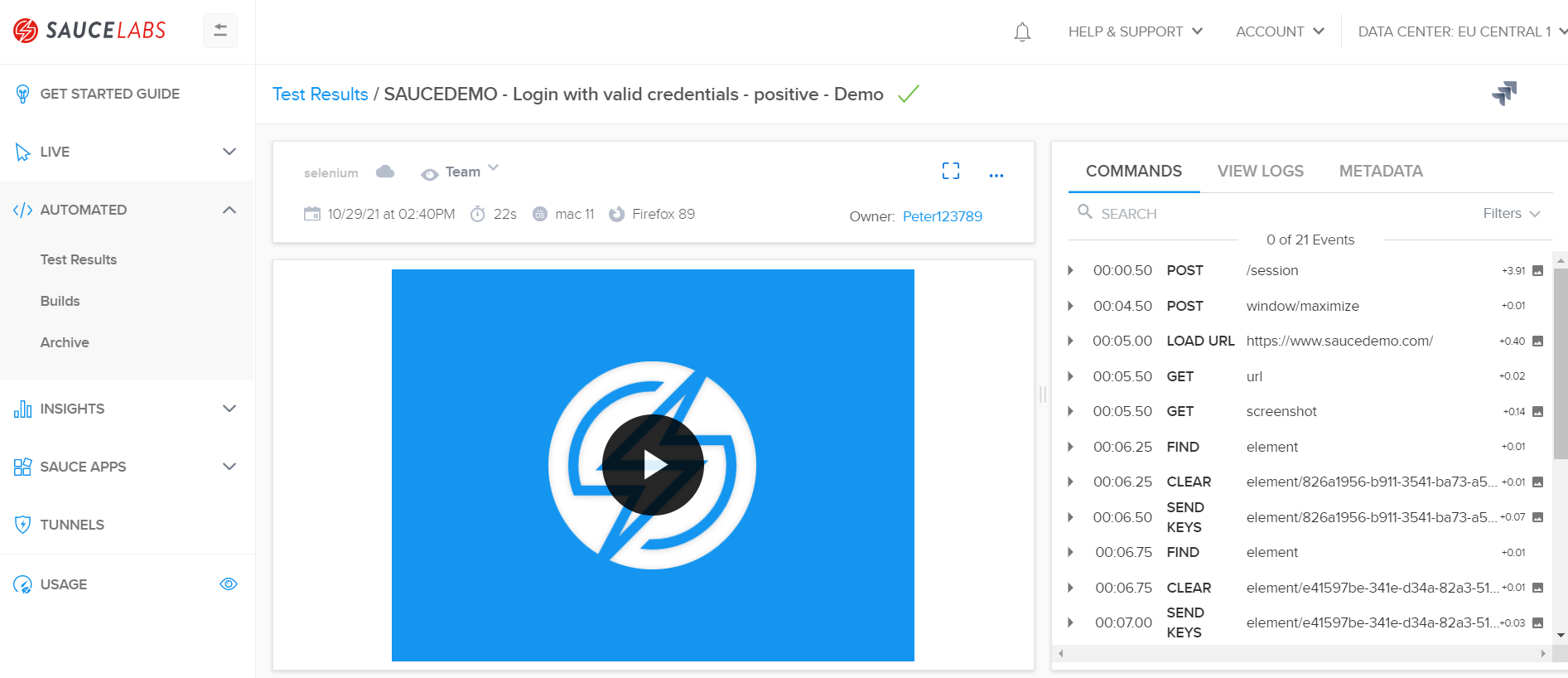
* In config.properties you have to set sl*\_username* (your Sauce Labs *USER NAME*) and sl*\_automate\_key* (your Sauce Labs *Access Key*).



* You can run your test in the standard way.
* HTML report will be created as usual.
* On SauceLabs page, navigate to *AUTOMATED* *| Test Results* option.



* You can see the test result of the test you executed.



# Windows driver integration

Gurkensalat is offering integration with windows drivers.

## Winium

**Winium** is the automation testing tool to automate windows desktop applications. Gurkensalat is ready to use Winium asWinium driver. You need to copy Winium driver into *web\_drivers* folder.

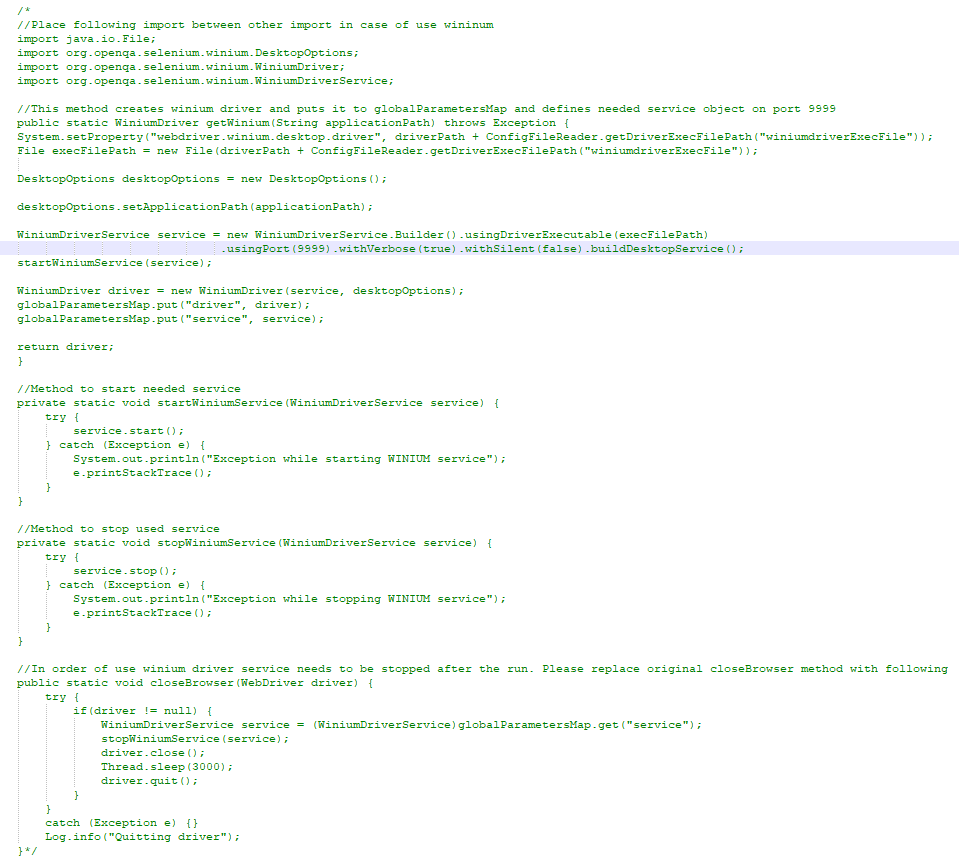
You can download *Winium* driver using following link:

<https://github.com/2gis/Winium.Desktop/releases/>

You can find the prepared functional code in *Gurkensalat.*

It needs to be uncommented in *BrowserFactory*, *ManageBrowser* and *pom file*.

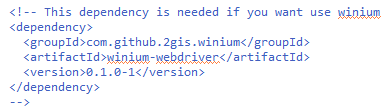
* ***BrowserFactory***- located in *utility – uncomment/delete /\* and \*/*



* ***ManageBrowser*** - located in *steps – uncomment/delete /\* and \*/*

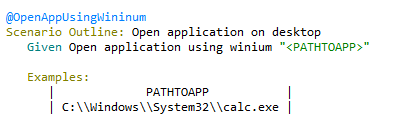


* ***Pom*** - located in *project root – uncomment/delete lines starting with <!-- and -->*



Feature file example:

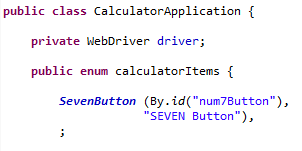
* Open application using the following step:



* Steps from *TestStepActions* can be used in the same way as with Selenium *WebDriver*.
* Locators of application elements can be found using free external application like  and the locators are supported using the built-in page object model.
* Example – How to find the locator of button 7 in Calculator application using UISpy:



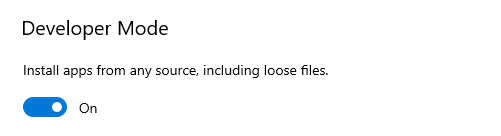
* Choose identifiers you would like to use and find them by their value.



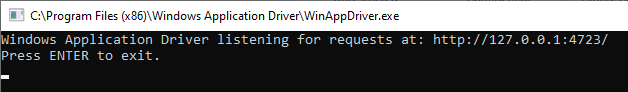
## WinAppDriver - Windows Application Driver

Not directly part of Gurkensalat but it can be accommodated with some minor modifications. *WinAppDriver* needs to be installed on the system from github from official Microsoft repository: [*https://github.com/microsoft/WinAppDriver*](https://github.com/microsoft/WinAppDriver)

* **Prerequisites for running:**
  + Installed Windows application driver
  + Developer mode turned on (*Settings -> Update & Security -> For developers*)



* + Running WinAppDriver.exe

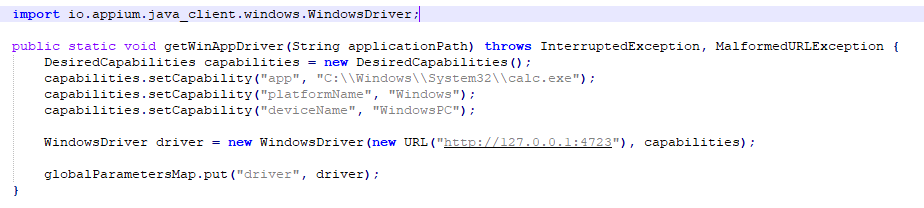


WinAppDriver cannot replace Webdriver. To be able to use WinAppDriver you have to change the driver object from webdriver to WindowsDriver.

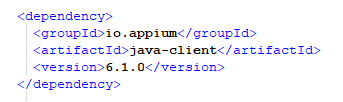
You can find the code in *utility | BrowserFactory*.

At the beginning of BrowserFactory class import the needed library.

Set capability ‘app’ value to path to desired application under test (calc.exe in example).

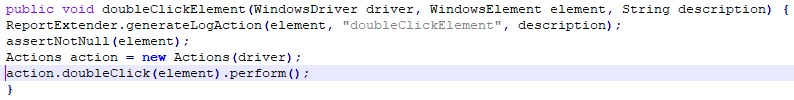


Add dependency to pom:



* In *steps | TestStepActions* you have to replace WebDriver with WindowsDriver

and WebElement with WindowsElement in every method, like in example below.



* To find locators in your application, see chapter on how to find locators for Wininum.

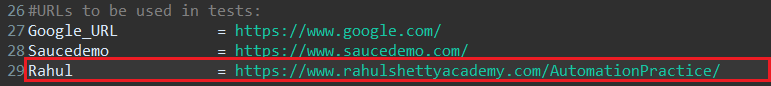
# First Test

To create a new test you should follow a few steps:

1. **Add URL to config.properties**
2. **Create a** **new feature file**
3. **Run Cucumber script**
4. **Create a new Page class**
5. **Create a Step definition class**

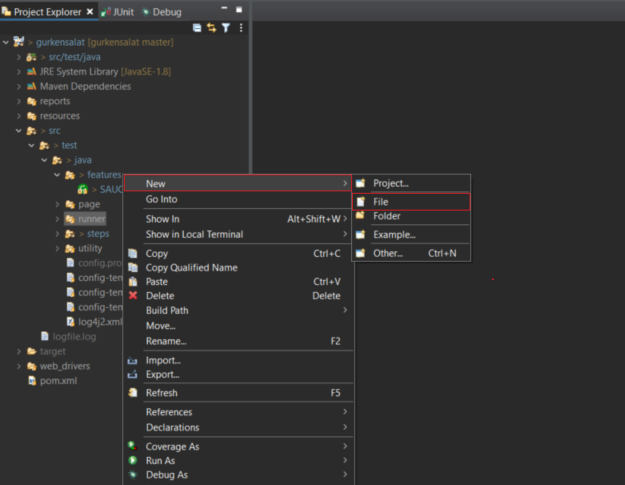
## Add URL to config.properties

* + Create a new record in config.properies and add URL required for the test.

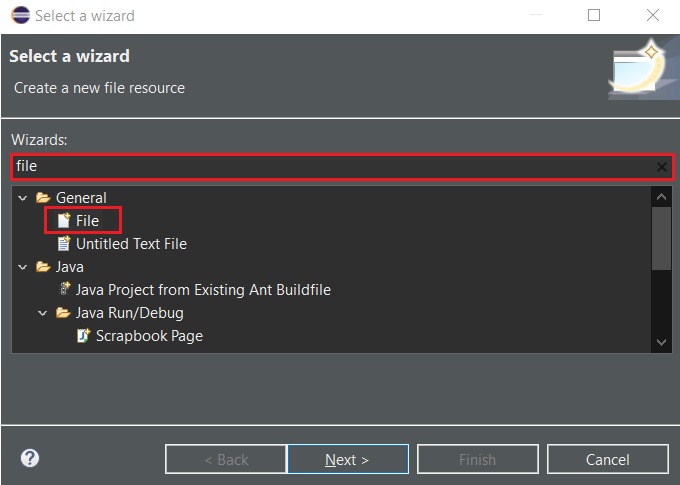


## Create a new feature file

* + Create a new feature file in ***features*** folder. Right-click on the *features* folder and select *New -> File.*



* If *File* is not visible, select *Other…* and search for *File*.



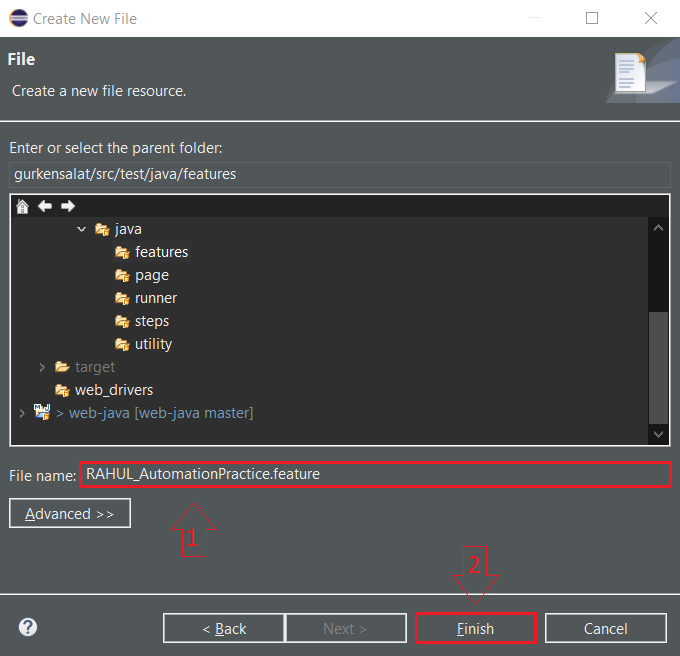
* In order for Cucumber to automatically detect the stories (features), you need to make sure that they carry the *.feature* file extension. Every *.feature* file conventionally consists of a single feature.
  + Name this user story, e.g. *RAHUL\_AutomationPractice.feature*.

Always use naming convention:

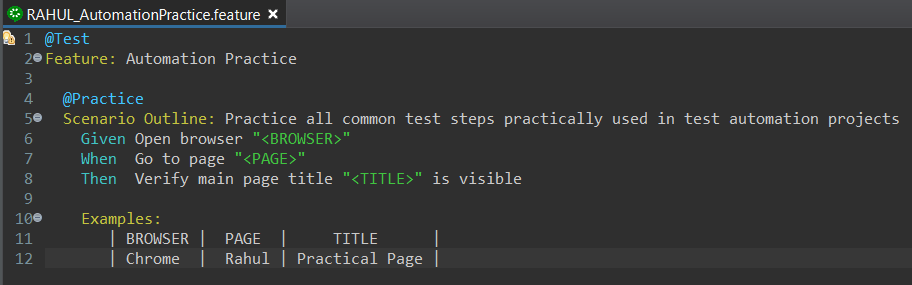
***Feature file name:*** *RAHUL\_AutomationPractice.feature*

*Format: PORTALNAME in upper case + \_ + ScenarioName in camel case + .feature*

*Other examples:* SAUCEDEMO\_LoginAction.feature, ALZA\_OrderProduct.feature,…



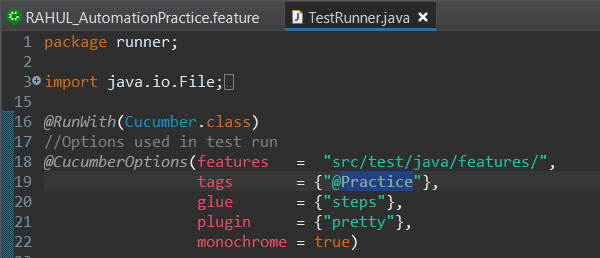
* + You can delete the unnecessary lines from created file and start to write a cucumber scenario. Don’t forget to add *PAGE* parameter into *Examples* with the correct name of URL as you set in config.properties. In BDD terms the scenario would look like the following:



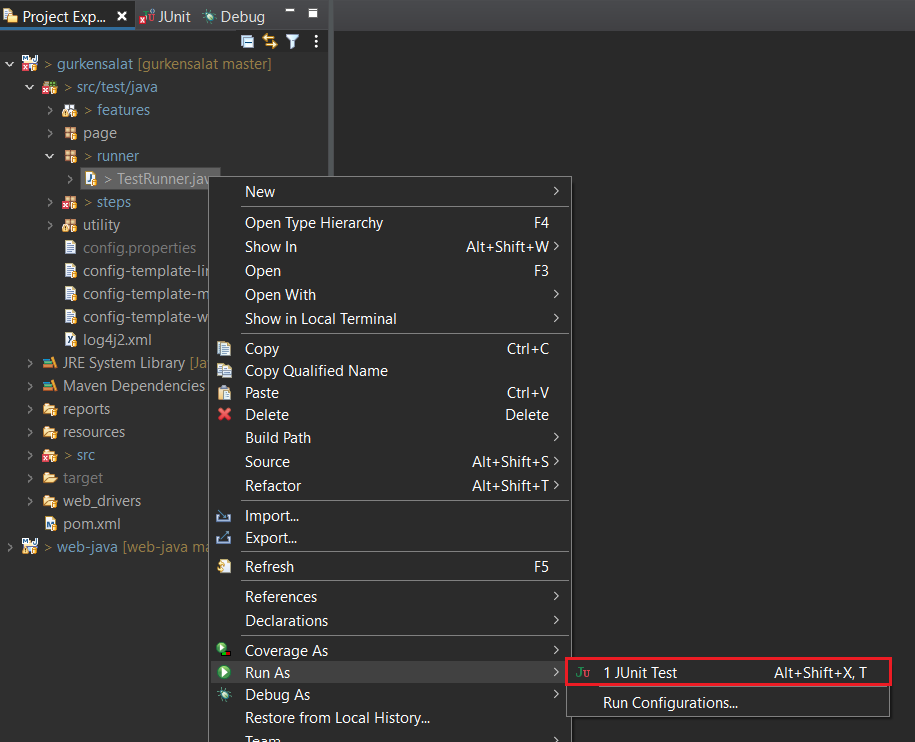
* + Implement the scenario sequentially commands one by one (e.g. implement one step in feature file and continue to the next step from this guide). That is the best way to create functional script in short time and also bring the benefit of your work to the team.

## Run Cucumber script

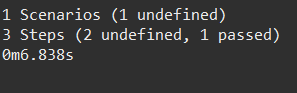
Based on the previous information, set *TestRunner.java* to run your test. You can set the *features* option to run only your .feature file. You can also create your own tag for your scenario and set it in *tags* option (as you can see in example picture).



* + Run cucumber script. Right-click *TestRunner.java* *-> Run As -> JUnit Test.*

****

* + Test will run. At the end you can see:



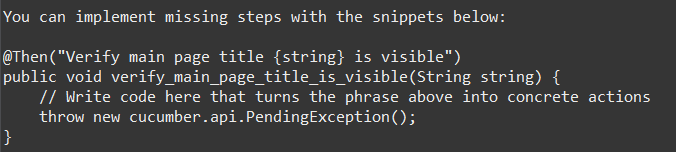
* + As you can see, you have some missing step definitions in *steps* package (*1 undefined*).



*Some of the steps are* already defined.

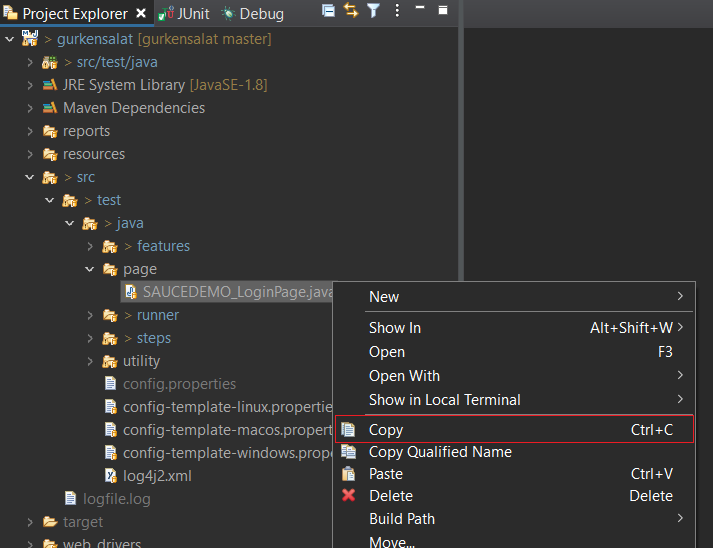
Be careful when implementing steps. Do not duplicate steps. It causes error.

* + Due to a missing step definition in *steps* package there will be a suggestion in the Console to implement the missing steps:

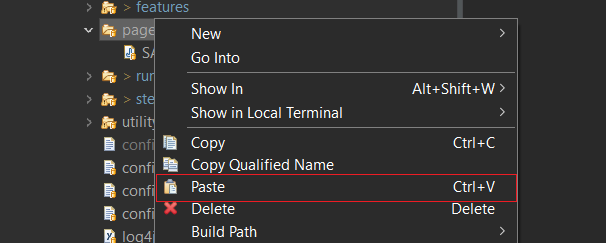


## Create a new Page class

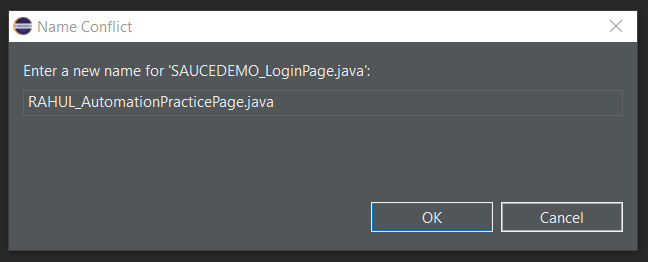
* + Create a page element definition in *page* package. The easiest way to do that is to copy and paste an existing class. Right-click on existing *SAUCEDEMO\_LoginPage.java* class in the *page* package and select *Copy*.



* + Right-click *page* package and select *Paste.*



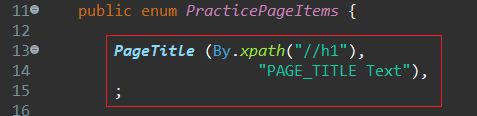
* + Enter a new name, e.g. *RAHUL\_AutomationPracticePage*. Always use naming convention.



* + Change the class name and enum name.

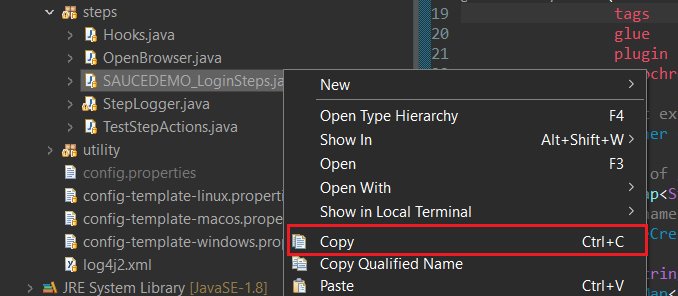


* + You can now add page elements required for the test in the step definition. More about steps in the next section.



## Create a Step definition class

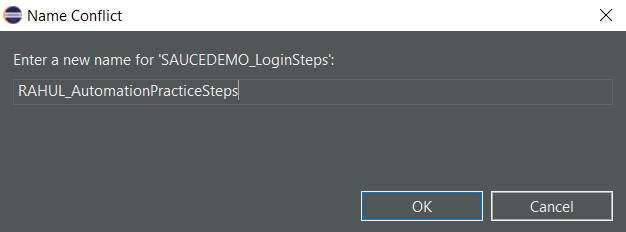
* **Step definition** 
  + Create a step definition. The easiest way to do that is to copy and paste an existing class. Right-click existing *SAUCEDEMO\_LoginSteps.java* class in the *steps* package and select *Copy*.



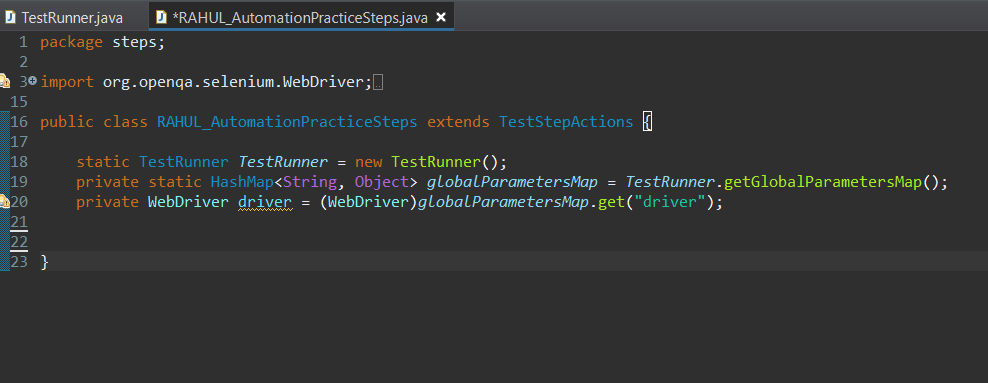
* + Right-click *steps* package and select *Paste.*



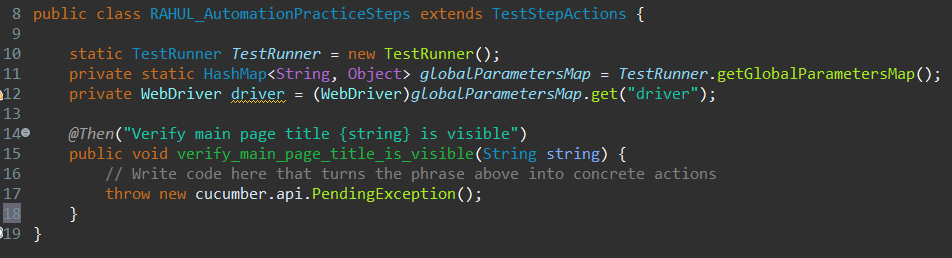
* + Enter a new name, e.g. *RAHUL\_AutomationPracticeSteps*. Always use naming convention.



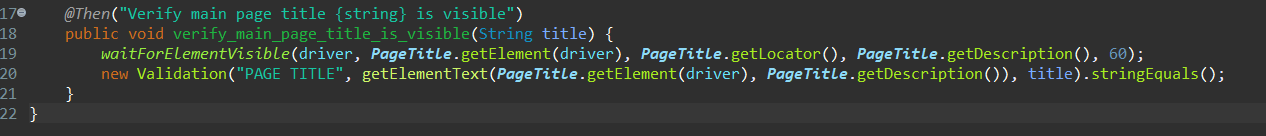
* + Delete all the steps copied from the original class. It should look as follows:



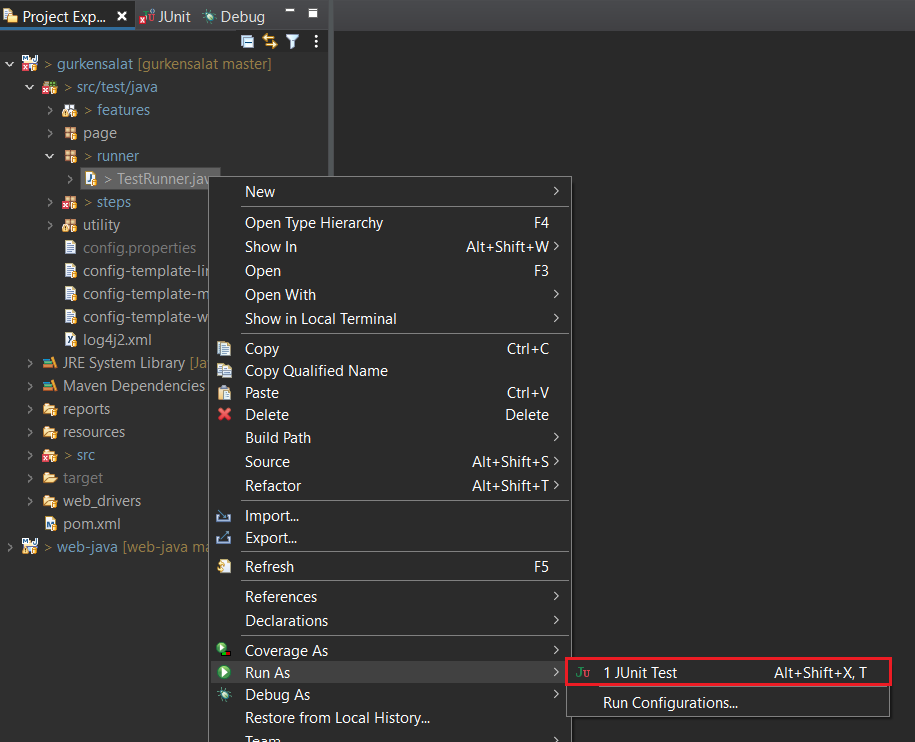
* + You can now implement your step definitions. Copy-paste the offered steps from the console to *RAHUL\_AutomationPracticeSteps.java*.



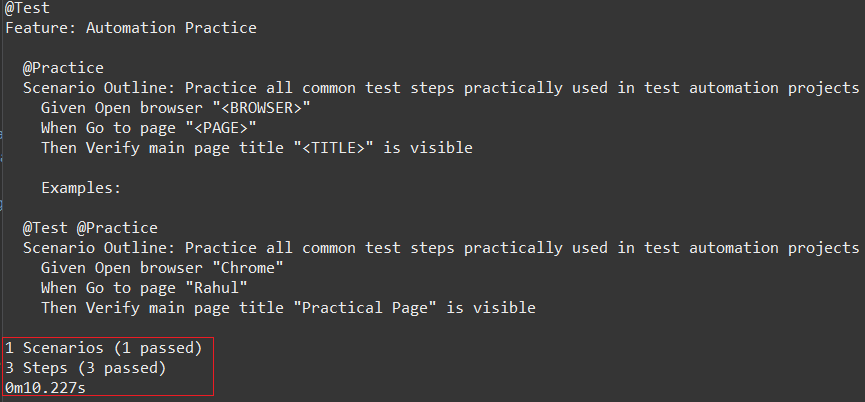
* + Now you can start to implement the missing steps:



* + Once the steps are implemented, run the test. Right-click TestRunner.java -> *Run As -> JUnit Test.*

****

* + Test should run. In Console, you should see the results:



# Test Step Actions

waitForFullPageLoad (WebDriver driver, int maxTimeInSeconds)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

@param maxTimeInSeconds - [int] - max. waiting time in seconds.

Description:

Waits until a page is fully loaded or maxTimeInSeconds is reached.

Creates a record in logger.

Example:

waitForFullPageLoad(driver, 60);

scrollElementIntoView (WebDriver driver, WebElement element)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

@param element - [WebElement] - web element specified by id/xpath path.

Description:

Scrolls specific element to view (top of screen).

Example:

scrollElementIntoView(driver, element);

scrollElementIntoMiddleOfScreen (WebDriver driver, WebElement element)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

@param element - [WebElement] - web element specified by id/xpath path.

Description:

Scrolls specific element to view (middle of screen).

Example:

scrollElementIntoMiddleOfScreen(driver, Back.getElement(driver));

scrollPageIntoBottom (WebDriver driver)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

Description:

Scroll view to the bottom of the page.

Example:

scrollPageIntoBottom(driver);

setFocusToElement (WebDriver driver, WebElement element)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

@param element - [WebElement] - web element specified by id/xpath path.

Description:

Moves focus to a specific element. If html/body is the element xpath, the whole page will be focused instead.

Example:

setFocusToElement(driver, AdviceBar.getElement(driver));

blurFocusOfActiveElement (WebDriver driver)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

Description:

Loses focus/blurs active element.

Example:

blurFocusOfActiveElement (driver);

clickElement (WebElement element, String description)

Parameters**:**

@param element - [WebElement] - web element specified by id/xpath path.

@param description - [String] - description on the element (user friendly).

Description:

Clicks on element, creates a record in logger and report.

Example:

clickElement(LoginButton.getElement(driver), LoginButton.getDescription());

doubleClickElement (WebDriver driver, WebElement element, String description)

Parameters:

@param driver - [WebDriver] - current Webdriver.

@param element - [WebElement] - web element specified by id/xpath path.

@param description - [String] - description on the element (user friendly).

Description:

Double clicks on element, creates a record in logger and report.

Example:

doubleClickElement(driver, page.customerFieldElement(accno), FromTable.getDescription());

clickElementUsingJavascript (WebDriver driver, WebElement element, String description)

Parameters**:**

@param driver - [WebDriver] - current Webdriver.

@param element - [WebElement] - web element specified by id/xpath path.

@param description - [String] - description on the element (user friendly).

Description:

Clicks on element using Javascript, creates a record in logger and report.

Example:

clickElementUsingJavascript(driver,Checkout.getElement(driver),

Checkout.getDescription());

clickElementUsingEnterButton (WebElement element, String description)

Parameters**:**

@param element - [WebElement] - web element specified by id/xpath path.

@param description - [String] - description on the element (user friendly).

Description:

Clicks on element using Enter button, creates a record in logger and report.

Example:

clickElementUsingEnterButton(caseRecord, "Cases list -> case link");

scrollAndClickElementUsingJavascriptAndWait (WebDriver driver, WebElement element, String description,

int timeInMilliseconds)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

@param element - [WebElement] - web element specified by id/xpath path.

@param description - [String] - description on the element (user friendly).

@param timeInMilliseconds - [int] - wait time in milliseconds.

Description:

Scrolls and clicks on element using javascript and waits, creates a record in logger and report.

Example:

scrollAndClickElementUsingJavascriptAndWait(driver,

LoginWithUsernameLink.getElement(driver), LoginWithUsernameLink.getDescription());

clickElementAndWait (WebElement element, String description , int timeInMilliseconds)

Parameters**:**

@param element - [WebElement] - web element specified by id/xpath path.

@param description - [String] - description on the element (user friendly).

@param timeInMilliseconds - [int] - wait time in milliseconds.

Description:

Clicks on element and waits, creates a record in logger and report.

Example:

clickElementAndWait(PortingYes.getElement(driver), PortingYes.getDescription(),

2000);

waitForElementClickable (WebDriver driver, By findBy, int maxTimeInSeconds)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

@param findBy – [By] - specified by xpath path.

@param maxTimeInSeconds - [int] - max. waiting time in seconds.

Description:

Waits for element to be clickable, creates a record in logger and report.

Example:

waitForElementClickable(driver, ShowLess.getLocator(), 50);

scrollAndClickElement (WebDriver driver, WebElement element, String description)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

@param element - [WebElement] - web element specified by id/xpath path.

@param description - [String] - description on the element (user friendly).

Description:

Scrolls and clicks on element, creates a record in logger and report.

Example:

scrollAndClickElement(driver, AddProduct.getElement(driver),

AddProduct.getDescription());

scrollClickElementAndWait (WebDriver driver, WebElement element, String description, int timeInMilliseconds)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

@param element - [WebElement] - web element specified by id/xpath path.

@param description - [String] - description on the element (user friendly).

@param timeInMilliseconds - [int] - wait time in milliseconds.

Description:

Scrolls and clicks on element and waits, creates a record in logger and report.

Example:

scrollClickElementAndWait(driver, Options.getElement(driver),

Options.getDescription(), 500);

checkCheckbox (WebElement element, String description)

Parameters**:**

@param element - [WebElement] - web element specified by id/xpath path.

@param description - [String] - description on the element (user friendly).

Description:

Clicks on checkbox box. Does nothing if the checkbox is already checked by default on the webpage. Creates a record in logger and report.

Example:

checkCheckbox(Phonebook.getElement(driver), Phonebook.getDescription());

uncheckCheckbox (WebElement element, String description)

Parameters**:**

@param element - [WebElement] - web element specified by id/xpath path.

@param description - [String] - description on the element (user friendly).

Description:

Unchecks a checkbox box. Does nothing if the checkbox is already unchecked by default on the webpage. Creates a record in logger and report.

Example:

uncheckCheckbox(Phonebook.getElement(driver), Phonebook.getDescription());

getElementText (WebElement element, String description)

Parameters**:**

@param element - [WebElement] - web element specified by id/xpath path.

@param description - [String] - description on the element (user friendly).

@return value - text of the element.

Description:

Retrieves text of the element. Creates a record in logger and report.

Example:

getElementText(ActiveAccountMessage.getElement(driver), ActiveAccountMessage.getDescription());

getDisplayedText (WebElement element, String description)

Parameters**:**

@param element - [WebElement] - web element specified by id/xpath path.

@param description - [String] - description on the element (user friendly).

@return value – visible text of the element.

Description:

Retrieves text visible on the webpage. Creates a record in logger and report.

Example:

getDisplayedText(CaseId.getElement(driver), CaseId.getDescription());

setElementText (WebElement element, String value, String description)

Parameters**:**

@param element - [WebElement] - web element specified by id/xpath path.

@param value - [String] – text of the element that is going to be filled.

@param description - [String] - description on the element (user friendly).

Description:

Fills in an input field. Creates a record in logger and report.

Example:

setElementText(UsernameElement.getElement(driver), username, UsernameElement.getDescription());

setElementTextAndConfirm (WebElement element, String value, String description)

Parameters**:**

@param element - [WebElement] - web element specified by id/xpath path.

@param value - [String] – text of the element that is going to be filled.

@param description - [String] - description on the element (user friendly).

Description:

Fills in an input field and hits enter. Creates a record in logger and report.

Example:

setElementTextAndConfirm(Product\_Basket.DROPDOWN\_SEARCH\_TEXTBOX.getElement(driver),

firstLanguage, Product\_Basket.DROPDOWN\_SEARCH\_TEXTBOX.getDescription());

setElementTextAndWait (WebElement element, String value, String description, int timeInMilliseconds)

Parameters**:**

@param element - [WebElement] - web element specified by id/xpath path.

@param value - [String] – text of the element that is going to be filled.

@param description - [String] - description on the element (user friendly).

@param timeInMilliseconds - [int] - wait time in milliseconds.

Description:

Fills in an input field and waits. Creates a record in logger and report.

Example:

setElementTextAndWait(Email.getElement(driver), defaultEmail, Email.getDescription());

setElementSecureText (WebElement element, String value, String description)

Parameters**:**

@param element - [WebElement] - web element specified by id/xpath path.

@param value - [String] – text of the element that is going to be filled.

@param description - [String] - description on the element (user friendly).

Description:

Fills in an input field, but masks it with "\*" in logs/report. Used mainly for passwords. Creates a record in logger and report.

Example:

setElementSecureText(PasswordElement.getElement(driver), password,

PasswordElement.getDescription());

selectElementFromListByVisibleText (WebElement element, String value, String description)

Parameters**:**

@param element - [WebElement] - web element specified by id/xpath path.

@param value - [String] – text of the option that is going to be selected.

@param description - [String] - description on the element (user friendly).

Description:

Selects element from a specific visible text from dropdown list. Creates a record in logger and report.

Example:

selectElementFromListByVisibleText(DisplayProducts.getElement(driver), "All",

DisplayProducts.getDescription());

selectElementFromListByValue (WebElement element, String value, String description)

Parameters**:**

@param element - [WebElement] - web element specified by id/xpath path.

@param value - [String] – value from dropdown element that is going to be selected.

@param description - [String] - description on the element (user friendly).

Description:

Selects a value from a specific dropdown list. Creates a record in logger and report.

Example:

selectElementFromListByValue(DisplayProducts.getElement(driver), "-1",

DisplayProducts.getDescription());

selectElementFromButtonList (WebElement element, WebDriver driver, String value, String description)

Parameters**:**

@param element - [WebElement] - web element specified by id/xpath path.

@param driver - [WebDriver] - current WebDriver.

@param value - [String] – text of the option that is going to be selected.

@param description - [String] - description on the element (user friendly).

Description:

Selects a value from an old type of dropdown, that is actually values + links. Creates a record in logger and report.

Example:

selectElementFromButtonList(Activity.getElement(driver), driver, activityTranslated,

Activity.getDescription());

getPreselectedOption (By findBy, WebDriver driver, String description)

Parameters**:**

@param findBy - [By] - specified by xpath path.

@param driver - [WebDriver] - current WebDriver.

@param description - [String] - description on the element (user friendly).

@return defaultItem - return preselected option.

Description:

Retrieves preselected option from dropdown.

Example:

getPreselectedOption(SelectListReason.getLocator(), driver, SelectListReason.getDescription());

sleep (int timeInMilliseconds)

Parameters**:**

@param timeInMilliseconds - [int] - wait time in milliseconds.

Description:

Thread sleep action. Use only when no other wait is effective.

Example:

sleep(1000);

waitForElementVisible (WebDriver driver, WebElement element, By findBy,

String description, int maxTimeInSeconds)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

@param element - [WebElement] - web element specified by id/xpath path.

@param findBy - [By] - specified by xpath path.

@param description - [String] - description on the element (user friendly).

@param maxTimeInSeconds - [int] - max. waiting time in seconds.

Description:

Waits until an element appears on page or maxTimeInSeconds is reached.

Creates a record in logger and report.

Example:

waitForElementVisible(driver, PageTitle.getLocator(), 60);

waitIfElementAppears (WebDriver driver, By findBy, String description, int maxTimeInSeconds)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

@param findBy - [By] - specified by xpath path.

@param maxTimeInSeconds - [int] - max. waiting time in seconds.

@param nameOfElement - [String] - Element name.

@return true/false - return true/false option.

Description:

Waits if element appears on page or maxTimeInSeconds is reached.

If not it creates a log and the script continues.

Creates a record in logger and report.

Example:

waitIfElementAppears(driver, PageTitle.getLocator(), PageTitle.getDescription(), 60);

refreshPage (WebDriver driver)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

Description:

Refresh page.

Example:

refreshPage(driver);

waitForElementDisappear (WebDriver driver, WebElement element, By findBy,

String description, int maxTimeInSeconds)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

@param element - [WebElement] - web element specified by id/xpath path.

@param findBy - [By] - specified by xpath path.

@param description - [String] - description on the element (user friendly).

@param maxTimeInSeconds - [int] - max. waiting time in seconds.

@return result - return only if element successfully disappear.

Description:

Waits until an element disappears from page or maxTimeInSeconds is reached.

Creates a record in logger and report.

Example:

waitForElementDisappear(driver, ActiveSpinner.getLocator(), 100);

waitForAlert (WebDriver driver, int maxTimeInSeconds)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

@param maxTimeInSeconds - [int] - max. waiting time in seconds.

Description:

Waits until an alert pop-up appears on page or maxTimeInSeconds is reached.

Creates a record in logger and report.

Example:

waitForAlert(driver);

acceptAlert (WebDriver driver)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

Description:

Accepts alert pop-up (clicks YES/OK/Accept/...).

Creates a record in logger and report.

Example:

acceptAlert(driver);

switchToAlert(WebDriver driver)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

Description:

Switch to Alert. Creates a record in logger and report.

Example:

switchToAlert(driver);

switchToParentalWindow (WebDriver driver)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

Description:

Switch driver to parental window. Creates a record in logger and report.

Example:

switchToParentalWindow(driver);

dismissAlert (WebDriver driver)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

Description:

Declines alert pop-up (clicks NO/Decline/Dismiss...).

Creates a record in logger and report.

Example:

dismissAlert(driver);

verifyElementIsPresent (WebDriver driver, By locator, String description)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

@param locator- [By] - specified by xpath path.

@param description - [String] - description on the element (user friendly).

@return true - if element is present on the page / false - if element is not present on the page.

Description:

Checks if a specific element is present on the page. Creates a record in logger.

Example:

verifyElementIsPresent(driver, ProspectButtonxpath.getXpathLocator(),

ProspectButton.getDescription());

verifyIsSelected (WebDriver driver, By locator, String description)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

@param locator- [By] - specified by xpath path.

@param description - [String] - description on the element (user friendly).

@return true - if element is selected / false - if element is not selected.

Description:

Checks if a specific element is selected. Creates a record in logger.

Example:

verifyIsSelected(driver, OptionOneCheckbox.getLocator(), OptionOneCheckbox.getDescription())

verifyText (WebElement element, String value, String description)

Parameters**:**

@param element - [WebElement] - web element we are checking.

@param value - [String] - value that is expected to be found.

@param description - [String] - description on the element (user friendly).

@return true - if values match / false - if values do not match.

Description:

Checks if a specific element has a specific text exactly.

Creates a record in logger and report.

Example:

verifyText(PageTitle.getElement(driver), "EXPECTED\_PAGE\_TITLE", PageTitle.getDescription());

switchToFrameByName (WebDriver driver, String frameName)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

@param frameName - [String] - name of the iFrame.

Description:

Switches to child iFrame. Creates a record in logger.

Example:

switchToFrameByName(driver, PageFrame.getElementFrame());

switchToFrameByLocator (WebDriver driver, By by)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

@param by - [By] - specified iFrame by xpath path.

Description:

Switches to child iFrame. Creates a record in logger.

Example:

switchToFrameByLocator(driver, PageFrame.getLocator());

printCurrentFrameName (WebDriver driver)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

Description:

Print current frame name into logger.

Example:

printCurrentFrameName(driver);

switchToParentFrame (WebDriver driver)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

Description:

Switches to parent iFrame. Use multiple times in case of nested iFrames.

Example:

switchToParentFrame(driver);

stepOutOfAllFrames (WebDriver driver)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

Description:

Switches out of the iFrames.

Example:

stepOutOfAllFrames (driver);

switchToBrowserTab (WebDriver driver, int windowIndex)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

@param windowIndex - [int] - browser tab index counted from the left, starting with 1.

Description:

Switches to another browser tab according to its index.

Creates a record in logger and report.

Example:

SwitchToBrowserTab(driver,2);

copyToClipboardManually (WebDriver driver)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

Description:

Manually selects all and copies it to clipboard (CTRL + A, CTRL + C). Creates a record in logger.

Example:

copyToClipboardManually(driver);

pasteFromClipboardManually (WebDriver driver)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

Description:

Manually paste from clipboard (CTRL + V). Creates a record in logger.

Example:

pasteFromClipboardManually(driver);

pressKey (WebDriver driver, WebElement element, String key)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

@param element - [WebElement] - web element specified by id/xpath path.

@param key - [String] - if it is a standard key of alphabet/numeric, just send the key in param in quotation marks (e.g. key="t").

Press SHIFT/TAB/ALT/CTRL/F1-12/ENTER keys

@param key - [String] – modifier, starts with Keys. + name of the key, no quotation marks (e.g. key=Keys.CONTROL). Creates a record in logger.

Description:

Press key action.

Example:

pressKey(driver, ConfirmButton.getElement(driver), Keys.ENTER.toString());

pressTwoKeysAtOnce (WebDriver driver, WebElement element, String key1, String key2)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

@param element - [WebElement] - web element specified by id/xpath path.

@param key1 and key2 - [String] - if it is a standard key of alphabet/numeric, just send the key in param in quotation marks (e.g. key="t").

Press SHIFT/TAB/ALT/CTRL/F1-12/ENTER keys

@param key1 and key2 - [String] - start with Keys. + name of the key, no quotation marks (e.g. key=Keys.CONTROL).

Description:

Press key action. Creates a record in logger.

Example:

pressTwoKeysAtOnce(driver, MainInput.getElement(driver), Keys.CONTROL.toString(),

Keys.DIVIDE.toString());

getLatestWindowFocused (WebDriver driver)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

Description:

In case of multiple tabs/windows being open, this method moves focus to the last open window/tab. Creates a record in logger.

Example:

getLatestWindowFocused(driver);

mouseOverElementAndClick (WebDriver driver , WebElement element, String description)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

@param element - [WebElement] - web element specified by id/xpath path.

@param description - [String] - description on the element (user friendly).

Description:

Hover Mouse Over Element and Click.

Creates a record in logger and report.

Example:

mouseOverElementAndClick(driver, FlashesIcon.getElement(driver),

OpenPartRequestsIcon.getDescription());

getSessionValue (WebDriver driver , String key)

Parameters**:**

@param driver - [WebDriver] - current WebDriver.

@param key - [String] - session key, where the value is stored.

Description:

Retrieves value stored in session. Creates a record in logger.

Example:

getSessionValue(driver, "dcBasketResponse");