Selenium / Playwright / REST-assured Java Framework

Last updated: 30 October 2021

Contents

Source code download

The source code for the desktop version of this framework is available from GitHub:
https://github.com/brunobosshard/java-framework-desktop.git

The source code for the mobile (Appium) version of this framework is also available from GitHub:
https://github.com/brunobosshard/java-framework-mobile.git

Prerequisites for the desktop version

Absolutely (mandatory) required installs

- Java JDK version 8 (Standard Edition) or higher (Java 17 is recommended) installed and running, either from Oracle https://www.oracle.com/java/technologies/javase-downloads.html , or alternatively a version of the Java Open JDK, for example from: https://openjdk.java.net/.
- The latest stable version of Apache Maven version 3 or higher: https://maven.apache.org/download.cgi .
- For Microsoft Windows (version 10 or 11) Docker Desktop installed and running: https://www.docker.com/products/docker-desktop .
  For Linux, both Docker Engine and Docker Compose installed and running: https://docs.docker.com/engine/install/ and https://docs.docker.com/compose/install/ .

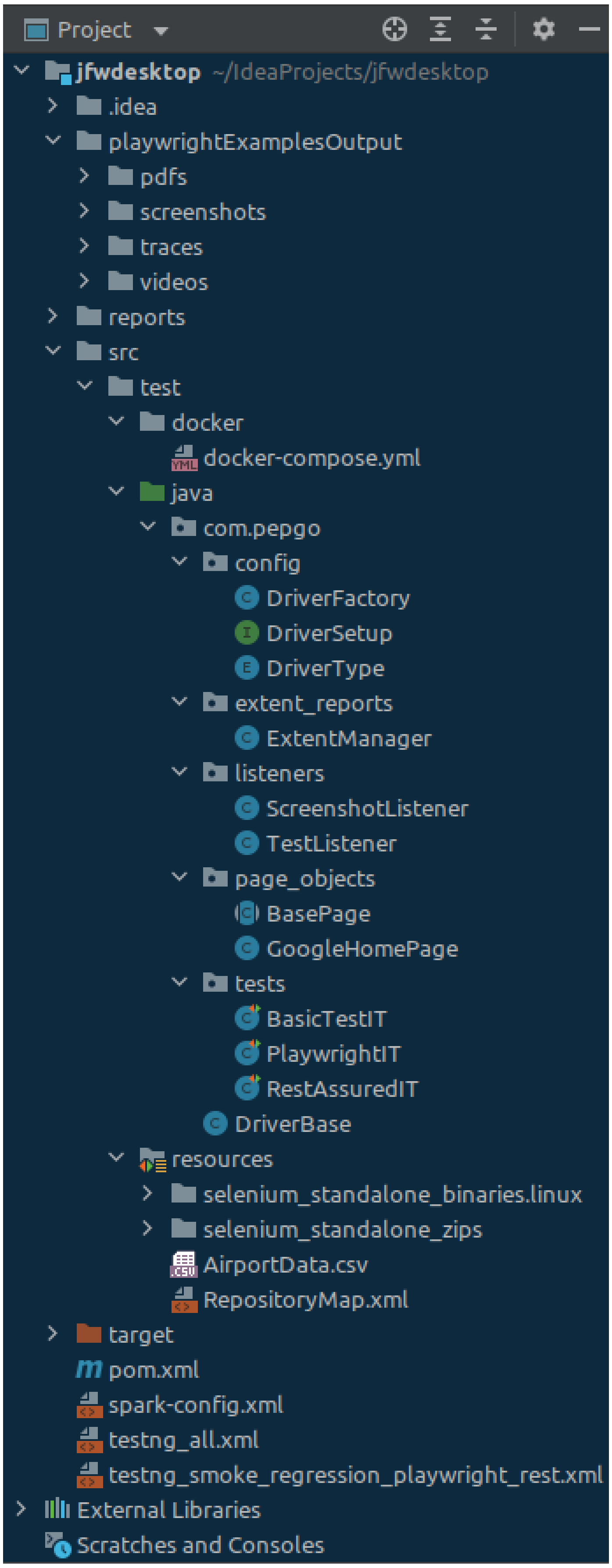
Please see the separate "Docker" section of this document for details.

Recommended (optional) installs

- An Integrated Development Environment (IDE), such as IntelliJ IDEA Community Edition is recommended for development and for customisation of the framework: https://www.jetbrains.com/idea/download/ .
- Installation of Google Chrome, Mozilla Firefox, Microsoft Edge, and Opera browsers and their respective driver binaries is recommended on the local file system.
- In Linux and Apple MacOS, all Selenium driver files must be made executable by granting execute permissions with:

```
sudo chmod +x driverfilename
```

Project structure (of the desktop version in IntelliJ IDEA)



```
Project
jfwdesktop ~/IdeaProjects/jfwdesktop
  .idea
  playwrightExamplesOutput
    pdfs
    screenshots
    traces
    videos
  reports
  src
    test
      docker
        docker-compose.yml
      java
        com.pepgo
          config
            DriverFactory
            DriverSetup
            DriverType
          extent_reports
            ExtentManager
          listeners
            ScreenshotListener
            TestListener
          page_objects
            BasePage
            GoogleHomePage
          tests
            BasicTestIT
            PlaywrightIT
            RestAssuredIT
          DriverBase
      resources
        selenium_standalone_binaries.linux
        selenium_standalone_zips
        AirportData.csv
        RepositoryMap.xml
  target
  pom.xml
  spark-config.xml
  testng_all.xml
  testng_smoke_regression_playwright_rest.xml
External Libraries
Scratches and Consoles
```

Framework Highlights

The framework allows flexible browser selection from the command line.

By supplying the "`threads`" argument from the command line, it allows running multiple browser instances in parallel by using the parallel feature of the TestNG unit test framework.

TestNG provides the basis for the framework. The execution mode is defined by the settings of the Maven Failsafe Plugin (in the "`POM.xml`" file). The number of tests that can run concurrently depends on the number of available threads. As with any unit test framework, there are no guarantees in which order tests will be executed. This means that all test cases must be designed to run independently from each other.

The framework downloads and installs driver binaries (such as drivers for Google Chrome, Mozilla Firefox, Microsoft Edge, and Opera) automatically. In Linux and Apple MacOS, the framework also automatically makes the driver files executable (by automatically granting execute permissions "`sudo chmod +x` *driverfilename*" on all driver files).

The framework uses browsers in headless mode by default. Headless mode can be switched off using the "`Dheadless`" argument, for example:

```
mvn clean verify -Dheadless=false
```

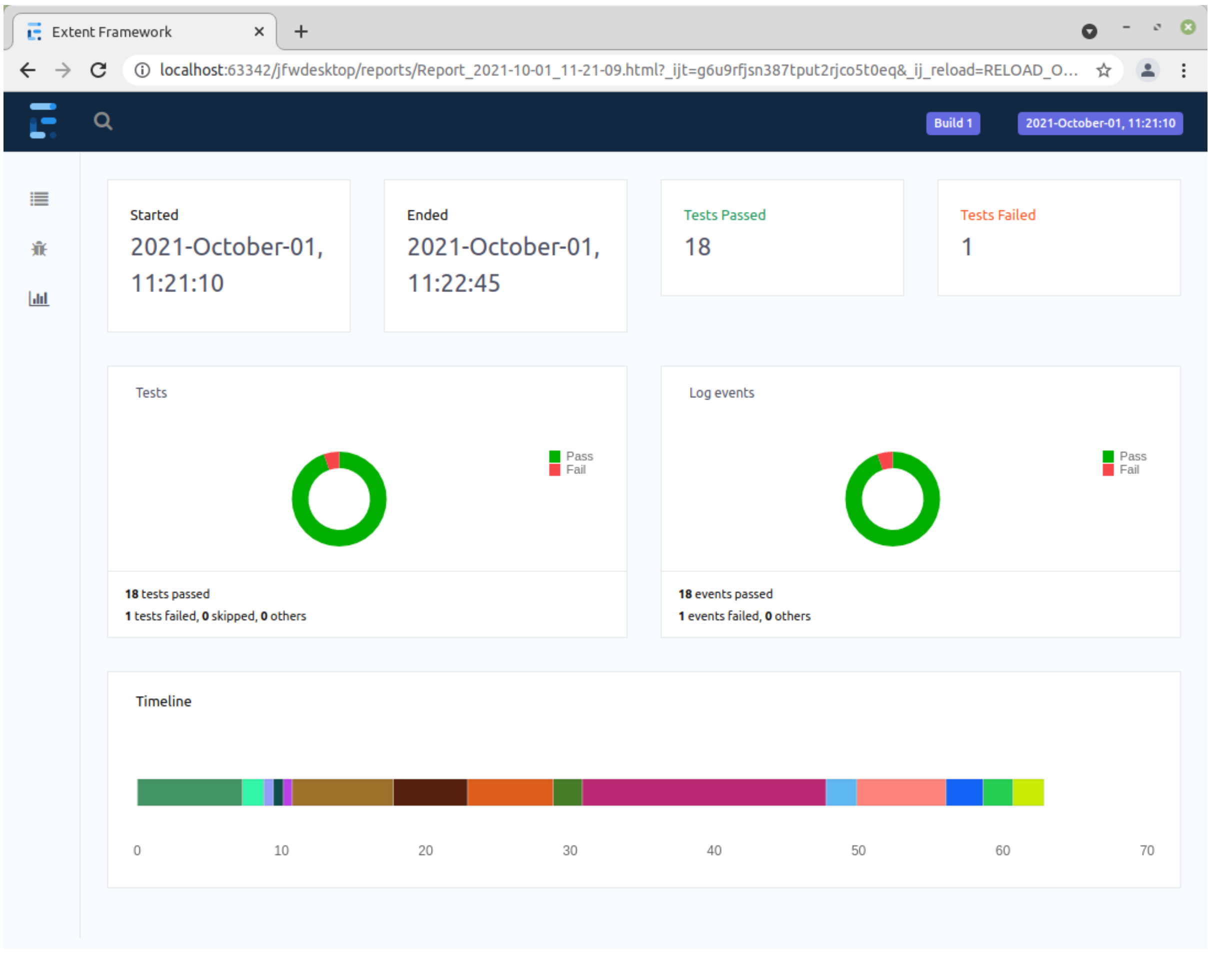
The framework allows using of Remote Grid (cloud) providers. Remote Grids provide the ability to run tests remotely. The following example uses Sauce Labs (https://saucelabs.com/):

```
mvn clean
install -Dremote=true -DseleniumGridURL=http://{username}:{accessKey}@ondemand.sauc
elabs.com:80/wd/hub -Dplatform=win10 -Dbrowser=firefox -DbrowserVersion=87
```

The framework uses Docker for containerisation by default. It uses Docker Compose, which enables the definition and execution of a multiple Docker container system through a YAML file.

The framework implements a ScreenshotListener to automatically take Screenshots in failed tests.

The framework can use JUnit based test reporters for reporting the test results. The framework already includes AventStack ExtentReports, but ExtentReports can easily be exchanged against other reporting libraries by exchanging the test listener class. It is also possible to use multiple reporters by adding more test listener classes. By default, the framework saves the reports to the local "`reports`" folder, but it is recommended to change this folder location to a dedicated (shared) folder on the network, or in the cloud. The location of the reports folder is set in the "`POM.xml`" file. Continuous Integration pipelines might want to store reports as artifacts in a binary artefacts server, such as JFrog Artifactory.


Extent Framework

localhost:63342/jfwdesktop/reports/Report_2021-10-01_11-21-09.html?_ijt=g6u9rfjsn387tput2rjco5t0eq&_ij_reload=RELOAD_O...

Build 1

2021-October-01, 11:21:10

Started
2021-October-01, 11:21:10

Ended
2021-October-01, 11:22:45

Tests Passed
18

Tests Failed
1

Tests

18 tests passed
1 tests failed, **0** skipped, **0** others

Log events

18 events passed
1 events failed, **0** others

Timeline

The framework can be enhanced with Cucumber to accommodate Behaviour-Driven Development (BDD).

Groups

The framework uses the Apache Maven Failsafe plugin for Integration Tests. Failsafe offers great support for the TestNG unit test framework: https://maven.apache.org/surefire/maven-surefire-plugin/examples/testng.html . Among other features, it allows TestNG Suite XML Files for test configuration. It allows using "`groups`" to execute all tests that belong to user-defined Groups, such as "smoke" or "regression".

The following example test annotation makes a test case a member of the two Groups called "regression" and "all":

```
@Test(groups={"regression","all"})
```

The "`DsuiteXmlFile`" argument can be used to specify the TestNG Suite XML file name. Per default (i.e. when the argument is not provided), the example provided in the framework uses the "`testng_smoke_regression_rest.xml`" configuration file, as defined in the "`POM.xml`" file.

The following example runs all tests that belong to the group "all", which deliberately includes one test that fails on the assertion (for demo purposes):
`mvn clean verify -DsuiteXmlFile=testng_all.xml`

Of course, it is also possible to define and use any number of other TestNG Suite XML files to execute just the required test classes, packages and test groups.

Maven Profiles

Maven profiles can be used to run different build configurations for the same project. For example: One profile with Unit Tests can get executed when the code is checked in, and another profile (for example with User Interface end-to-end tests) can get executed when the application has been deployed (for example on a web server).

Profiles can be set with the "`-P`" argument, for example:
`mvn clean verify -P`*myprofilename*

DataProvider

One of the most exiting features of TestNG is the ability to use a different data provider for each test case (for every "`@Test`" annotation). This allows for extremely flexible data-driven testing and for running test cases with almost unlimited sets of data. For example: It is easily possible to supply one test case with data from a text file (CSV format etc.), while supplying another test case with data from a database connection (JDBC).

Data providers can be objects in the (test classes) program code. However, they can also be separate data provider Java classes that read data from any data source, such as text files (CSV format etc.), Microsoft Excel (for example by using Apache POI https://poi.apache.org/), JSON, XML, databases (using JDBC), and more. The example provided in the framework uses the "`opencsv`" parser library to read data from a Comma-Separated Values (CSV) file.

There are plenty of data provider example Java classes and templates for all kind of data sources on the internet that can be easily adapted and used in this framework. Basically, a data provider just has to read the data and return the data as "`Object[][]`" or "`Iterator<Object[]>`".

The following "`@Test`" annotation example from the framework uses a data provider with the name "airportDataProvider":
`@Test(groups={"`*regression*`","`*all*`"}, dataProvider="`*airportDataProvider*`")`

Proxy Server

Corporate environments often enforce using a Proxy Server for internet access. The framework supports this.

The framework allows setting proxy details on the command line, but alternatively, proxy settings can also be pre-configured (for a permanent corporate proxy) in the "`POM.xml`".

The framework features a Boolean ("`useBrowserMobProxy`") in the "`DriverFactory`" class that determines, if BrowserMobProxy should be used or not. This allows using a (corporate) proxy, or a BrowserMobProxy, or both. It ensures that if it is required to use a corporate proxy server to reach the web application under test, it is possible to use the (corporate) proxy as well as BrowserMobProxy. Example use:

```
mvn clean install -DproxyEnabled=true -DproxyHost=localhost -DproxyPort=8080
```

The proxy functionality of the framework can also be used to record Apache JMeter performance test scripts by recording the network traffic generated by Selenium scripts though a Apache JMeter proxy.

Some example commands

Run all tests from the command prompt:

```
mvn clean verify
```

The same, but with ensuring that Apache Maven displays a stack trace:

```
mvn clean verify -e
```

Run all the tests with location of the "Gecko" driver for Mozilla Firefox:

```
mvn clean verify –Dwebdriver.gecko.driver=<PATH_TO_GECKODRIVER_BINARY>
```

Run 2 threads of the tests in the framework with location of the "Gecko" driver for Mozilla Firefox:

```
mvn clean verify -Dthreads=2 –Dwebdriver.gecko.driver=<PATH_TO_GECKODRIVER_BINARY>
```

The same as before, using the framework with the Edge driver (not specifying "`Dbrowser`" would use the default browser of the framework, Google Chrome):

```
mvn clean verify -Dthreads=2 –Dbrowser=edge
```

Playwright

The framework can use Playwright as an alternative or as a supplement to Selenium.

Playwright and Selenium tests can be mixed and matched.

Playwright test classes are written in a similar way to Selenium test classes.

If a test class contains Playwright test cases only, then the test class doesn't need to extend from the "`DriverBase`" class (or the "`AppiumBase`" class in the mobile version of the framework), because none of the Playwright test cases require a web driver. However, it is possible to have Selenium test cases and Playwright test cases in the same test class, and it is even possible to combine Selenium and Playwright testing in the same test case.

Playwright test cases can use the same TestNG assertions as the Selenium test cases for verifying results.

Using Docker with Playwright Java

Playwright provides ready-made Docker images.

Pull the image (specific Playwright version)

```
docker pull mcr.microsoft.com/playwright/java:v1.15.0-focal
```

Run the image (specific Playwright version)

Terminal session as the root user:

```
docker run -it --rm --ipc=host
mcr.microsoft.com/playwright/java:v1.15.0-focal /bin/bash
```

REST API Testing using REST-assured

The framework uses REST-assured for testing REST API's.

REST API test classes are written in a similar way to Selenium test classes.

If a test class contains REST API test cases only, then the test class doesn't need to extend from the "`DriverBase`" class (or the "`AppiumBase`" class in the mobile version of the framework), because none of the REST API test cases require a web driver. However, it is possible to have Selenium test cases and REST API test cases in the same test class, and it is even possible to combine Selenium and REST API testing in the same test case.

REST API test cases can use the same TestNG assertions as the Selenium test cases for verifying results.

REST-assured supports all commonly used REST http methods, including "`GET`", "`POST`", "`PUT`", "`PATCH`" and "`DELETE`".

The demo test cases in the framework also use Google's GSON library for JSON parsing. However, any other Java JSON library (such as Jackson etc.), can also be used instead of Google GSON. Most responses can actually be handled internally by REST-assured without any additional JSON parsing library, because REST-assured already includes the "`jsonPath`" query language (with Groovy syntax) for JSON. The "`jsonPath`" query language does for JSON, what XPath does for XML.

REST-assured test cases always follow the same basic structure:

1. Set a BaseURI for the REST API.
2. Create a Request object for the request that will be sent to the REST API.
3. Create a Response object for the response that will be returned from the REST API.
4. Make the call to the REST API (this is often done together with step 3).
5. Examine the Response object that has been returned from the REST API.

Docker

The framework supports Docker, which needs to be installed separately.

Docker in Microsoft Windows

The (free) Docker Desktop can be downloaded from https://www.docker.com/products/docker-desktop . Please check the Docker documentation for details.

For older version of Microsoft Windows, it might be required to use a legacy (now deprecated) version called "Docker Toolbox".

Docker in Linux

Linux requires the installation of the free Docker Engine and the free Docker Compose.

Linux also requires adding the current user to the "`docker`" group.

The "`docker`" group can be created with the following command. If the "`docker`" group already exists, then a message will complain about this, but it will do no harm:

```
sudo groupadd docker
```

The following command will add the user to the "`docker`" group. After this command, it is recommended to completely log out of the account and log back in (if in doubt, reboot!):

```
sudo usermod -a -G docker $USER
```

Check that Docker works correctly

The correct installation of Docker can be verified by running this command in a terminal in Linux (or command prompt in Windows):

```
docker run hello-world
```

Using Docker with Selenium

The framework starts Docker containers automatically before the Integration Test phase and shuts down Docker containers afterwards.

In effect, working with Docker is exactly the same as working with any other Selenium Grid, it just requires the URL of the (Docker) hub machine.

Docker (i.e. Selenium Grid) can be used like this:

```
mvn clean verify -Dremote=true -Dbrowser=firefox
 -DgridURL=http://{DOCKERMACHINEIPNUMBER}:4444/wd/hub
```

Note: For the local machine, it is possible to use the loopback IP number "`127.0.0.1`", or "`localhost`", for example:

```
mvn clean verify -Dremote=true -Dbrowser=firefox -DgridURL=http://127.0.0.1:4444/wd/hub
```

The framework supports Docker Compose, which is a tool for defining and running multi-container Docker applications through a YAML file.

The framework also alternatively allows defining a particular Docker Selenium Grid execution as standard with the "`<seleniumGridURL>`" setting in the "`POM.xml`", so that the default Docker execution can then be started with just "`mvn clean verify`".

Starting up Docker manually

The framework downloads, installs, and runs all Docker containers automatically and shuts them down correctly after use.

This means that this section will normally not be required, as all of this will usually be done automatically by the framework.

However, it is also possible to manually start up Docker and to use it like any other Selenium Grid.

The following 4 images need to be installed ("pulled"). It is also possible to automatically pull the latest versions by not including the version number in the pull commands:

```
docker pull selenium/hub:3.141.59
docker pull selenium/node-firefox:3.141.59
docker pull selenium/node-chrome:3.141.59
docker pull selenium/node-opera:3.141.59
```

Selenium hub can then be started with these commands:

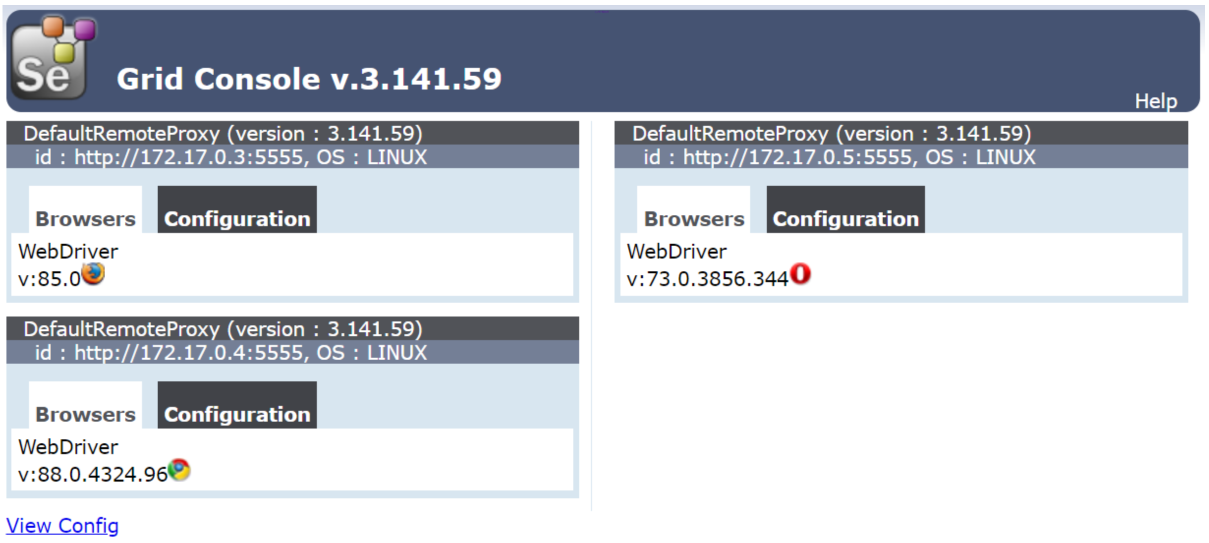
```
docker run -d -p 4444:4444 --name selenium-hub selenium/hub:3.141.59
docker run -d --link selenium-hub:hub selenium/node-firefox:3.141.59
docker run -d --link selenium-hub:hub selenium/node-chrome:3.141.59
docker run -d --link selenium-hub:hub selenium/node-opera:3.141.59
```

It is possible to start multiple instances of the same image with the same commands. This allows running more parallel threads.

The installation can be verified in a browser with:

```
http://{DOCKERMACHINEIPNUMBER}:4444/grid/console
```

This should result in something like this (for just one instance of Mozilla Firefox, Google Chrome, and Opera each):


Grid Console v.3.141.59

Help

DefaultRemoteProxy (version : 3.141.59)
id : http://172.17.0.3:5555, OS : LINUX

Browsers Configuration

WebDriver
v:85.0

DefaultRemoteProxy (version : 3.141.59)
id : http://172.17.0.5:5555, OS : LINUX

Browsers Configuration

WebDriver
v:73.0.3856.344

DefaultRemoteProxy (version : 3.141.59)
id : http://172.17.0.4:5555, OS : LINUX

Browsers Configuration

WebDriver
v:88.0.4324.96

View Config

The command "`docker container ls`" lists all running containers as follows (except for different "NAMES", because Docker assigns names randomly):

```
CONTAINER ID   IMAGE                              COMMAND                  CREATED          STATUS         PORTS                      NAMES
309a2b02b10f   selenium/node-opera:3.141.59       "/opt/bin/entry_poin…"   2 minutes ago    Up 2 minutes                              laughing_meninsky
850521a8c7f9   selenium/node-chrome:3.141.59      "/opt/bin/entry_poin…"   2 minutes ago    Up 2 minutes                              unruffled_bose
efa242aee459   selenium/node-firefox:3.141.59     "/opt/bin/entry_poin…"   2 minutes ago    Up 2 minutes
priceless_cartwright
342052a1df3d   selenium/hub:3.141.59              "/opt/bin/entry_poin…"   2 minutes ago    Up 2 minutes   0.0.0.0:4444->4444/tcp     selenium-hub
```

Important Docker commands

To dispose a container, the container has to be first stopped and then removed.

Stop a specific running container:

`docker container stop` ***[container_id]***

Stop all running containers:

```
docker container stop $(docker container ls -aq)
```

Remove a specific stopped container:

`docker container rm` ***[container_id]***

Remove all stopped containers:

```
docker container rm $(docker container ls -aq)
```

Technical Notes for the desktop version of the framework

The "`maven-failsafe-plugin`" runs tests in the Integration Test phase (such as Selenium and REST API tests). It picks up test class files that end in "`IT`" by default.

In contrast, the "`maven-surefire-plugin`" runs tests in the Unit Test phase. It picks up test class files that end in "`Test`" by default.

The framework uses TestNG assertions.

The framework uses a special maven plugin called "`driver-binary-downloader-maven-plugin`" https://github.com/Ardesco/selenium-standalone-server-plugin/ to download and install binary drivers. In Linux and Apple MacOS, it also automatically makes the downloaded driver files executable (by automatically granting execute permissions "`sudo chmod +x` *driverfilename*" on all driver files).

The file "`RepositoryMap.xml`" is where the download locations and versions for all driver binaries are defined. To update the drivers, it is recommended to regularly update the settings in this XML file. To ensure the integrity of the downloaded driver files, a (SHA1) hash code should be calculated for each driver file. There are many free tools available to calculate a (SHA1) hash code for a file.

The Docker implementation of this framework is based on the "`Docker-Maven-Plugin`" from fabric8.io : https://dmp.fabric8.io/

The mobile version of the framework (using Appium)

The example tests in the mobile framework cover both mobile, as well as Playwright and REST API tests.

The example mobile tests in the mobile framework use the standard Android calculator app. If you are using an Android phone with a customised Android version (for example a "Samsung" device), then you will most likely have to change the "`appPackage`" and "`appActivity`" settings in "`CalculatorIT.java`" to match your customised calculator app. In this case, you will also most likely have to change the Page Objects element identifiers in "`CalaculatorPageObject.java`" to match those of your calculator app.

The mobile framework expects a real phone to be connected (via USB cable) to your test execution machine, or a virtual AVD device (using Android Studio). If you use a real phone, you need to make sure that this phone has "Developer options" and "USB debugging" enabled, and that you can connect to it from your test execution machine (you can check this for example with:

```
adb devices
```

(You might have to add the Android SDK to the "path" of your operating system).

The mobile framework requires Google Android Studio https://developer.android.com/studio/ (if you have enough space on your machine), or at least Android Debug Bridge (ADB), which is also useful to query the phone you are using to get hold of the package name and launch activity. ADB is available from: https://developer.android.com/studio/command-line/adb .

Appium Server

The mobile framework also requires the Appium Server.

The Appium Server is available in two versions: As a Command-Line Interface (CLI) version, and as a Graphical User Interface (GUI) version.

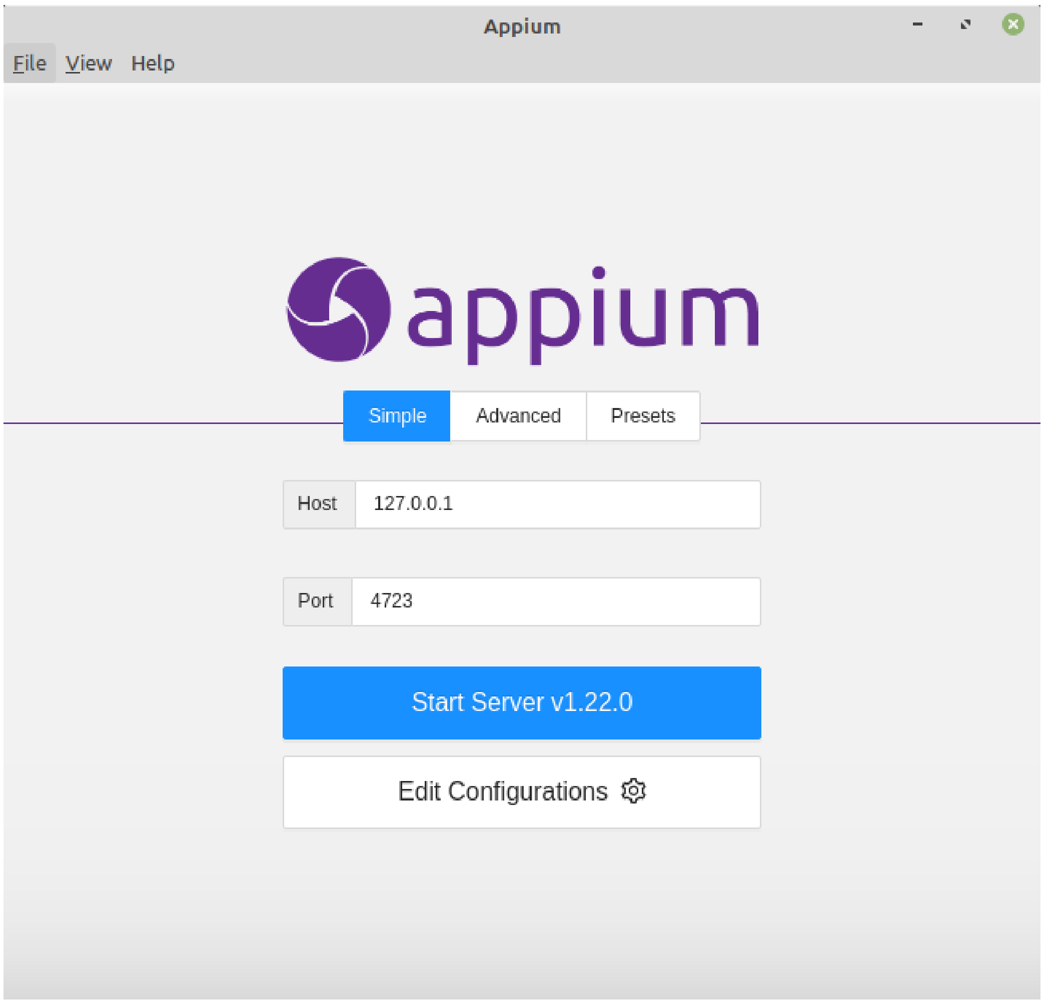
The CLI version might be better suited for scripted environments, such as Continuous Integration (CI) pipelines.

The Graphical User Interface (GUI) version of Appium Server

The GUI version of Appium Server is available from https://github.com/appium/appium-desktop/releases . On Linux, follow these steps:

1. Download the "`Appium-Server-GUI-*.AppImage`".
2. Make it executable by "`sudo chmod a+x Appium-Server-GUI*.AppImage`".
3. Run "`Appium-Server-GUI*.AppImage`".

The mobile framework expects that the Appium Server has already been started when you execute the tests. The Appium Server must be running locally on the test execution machine under the IP address "127.0.0.1"and the default port number "4723".



Host 127.0.0.1

Port 4723

The Command-Line Interface (CLI) version of Appium Server

Appium Server is a Node.js application.

Appium Server therefore requires the installation of Node as a prerequisite.

After installing Node, the Appium Server can then be installed with:

```
npm install -g Appium
```

This example starts the Appium Server on the host IP number 127.0.0.1 and on port 4723:

```
Appium -a 127.0.0.1 -p 4723
```

Execute Appium tests

If you have already started the Appium Server, then you can execute the tests as usual with:

```
mvn clean verify
```