Robot Framework

TEST AUTOMATION FRAMEWORK - DOCUMENTATION

Robot Framework ver. 1.00
October 2021

Credentials

Description	Robot Framework – Documentation
Release date	October 15th, 2021
Version	1.00
Author	marek.lesnovsky@accenture.com
Reviewer	branisalv.baca@accenture.com

1 Introduction

Robot Framework is a generic open source automation framework. It is free to use without licensing costs. has an easy syntax, utilizing human-readable keywords. Its capabilities can be extended by libraries implemented with Python.

Robot Framework is suitable for small to mid-size test automation projects to run automated tests on web.

Improvements:

- ✓ Highly improved folder structure
  Clear and nice project folder structure with reports, web drivers and project files all in separate folders.
- ✓ Auto-generated report folders
  After each run a new report with all screenshots is saved in a new folder with timestamp for better orientation.
- ✓ Automated screenshot naming
  Screenshots are automatically saved with name with proper timestamp in nicely named report folder.
- ✓ 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.
- ✓ Easier reusability of test flows
  Nice and clear parametrization for test flows allows easy and quick runs of multiple same flows with different parameters.
- ✓ More information in terminal
  More information visible in terminal like timestamps, current running keyword , total time elapsed.
- ✓ Webdrivers always in system path.
  No more manual editing of Path variable. Framework will automatically add all web drivers folder into system variables.

2 Software and costs

Using Robot Framework is completely free, no strings attached. Due to its expandable open source nature Robot Framework is extremely versatile for various test and RPA cases. Using it does not require expert skills in coding.
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.

2.1 CI/CD environment

List of tools used on CI/CD environment with its version and cost (can be used any latest version, but it's not tested):

Tool	Version	Cost	License
GIT	2.32.0	Free	GNU
Java – AdoptOpenJDK + HotSpot	11.0.11+9	Free	GNU
Apache Maven	3.8.1	Free	Apache License 2.0

2.2 Developer's environment

List of tools used by automation developers. Each with its version and cost(can be used any latest version, but it's not tested):

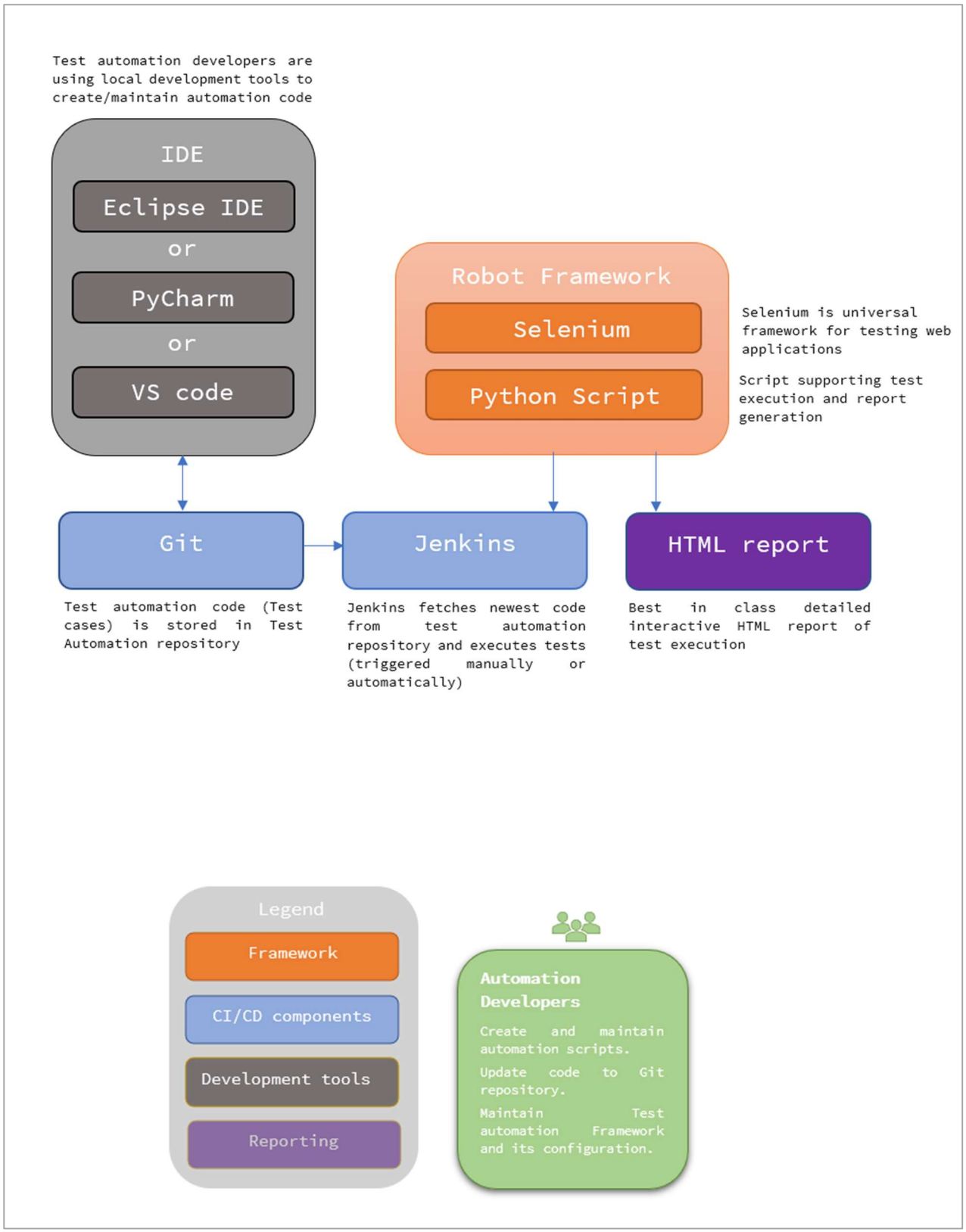
Tool	Version	Cost	License
Eclipse	2021-03 (4.19.0)*	Free	GNU
PyCharm Community	2021.2.3	Free	Open Source
Visual Studio Code	1.80.1	Free	MIT
GIT	2.32.0	Free	GNU
Python	3.11.4	Free	GNU
Robot Framework	6.1	Free	Apache License 2.0
Robotframework-seleniumlibrary	6.1.0	Free	Apache License 2.0

3 Supported Operating system/Browser/WebDriver

List of tested/supported Operating system with Browsers and its WebDriver.

OS	OS Version	Browser	Browser Version	WebDriver	WebDriver Version
Windows	Windows 10 Enterprise 20H2 19042.1348	Firefox	115.0.2	mozilla/geckodriver	0.33.0
Windows	Windows 10 Enterprise 21H2 19042.1348	Chrome	114.0.5735.199	chromedriver	114.0.5735.199
Windows	Windows 11 Enterprise 21H2 22000.2057	Edge	114.0.1823.82	Microsoft Edge Driver	114.0.1823.82

4 Design overview



5 Prerequisites

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

- ***Chrome browser ver. 114.0.5735.199***
- Please download and install latest version of Chrome browser with default settings

- ***Python 3 ver.3.11.4***
  Please install python 3 if you don't have it already.
  Follow installation guide in *Guides/Python*

- ***Chromedriver ver. 114.0.5735.199***
  Please download the actual version of Chromedriver (matching your browser version) from the following URL: https://chromedriver.chromium.org/downloads)
  Follow installation guide in *Chromedriver*
  Copy webdriver to corresponding location as explained later in next chapter

Install one of the following:

- **Installation for PyCharm IDE:**
  - ***PyCharm IDE –*** *PyCharm Community 2021.2.3* or the latest installed on your computer
    Follow installation guide in *PyCharm*

- **Installation for Eclipse IDE:**
  - ***Eclipse IDE –*** *Eclipse IDE* 2021-09 (4.21.0) or the latest installed on your computer
    Follow installation guide in *Eclipse IDE*

- **Installation for Visual Studio Code:**
  - ***Visual Studio Code*** - Visual Studio Code 1.61.1 or the latest installed on your computer
    Follow installation guide in *Visual Studio Code*

6 Robot Framework install guide

6.1 Robot Framework Installation

A precondition for running tests is having both Robot Framework and SeleniumLibrary installed. The easiest way to install Robot Framework along with its dependencies is using pip package manager. There are multiple ways it can be installed.

In this step all necessary dependencies and packages for Robot Framework to work will be installed. You will not be creating framework folder in this step.

Framework can be installed in one of two ways

- Using requirements file where all environment parts are installed at once (subsection 3.1). This also includes some additional utilities. It's highly recommended to install robot framework this way.

- Install every part/feature separately (subsection 3.3).

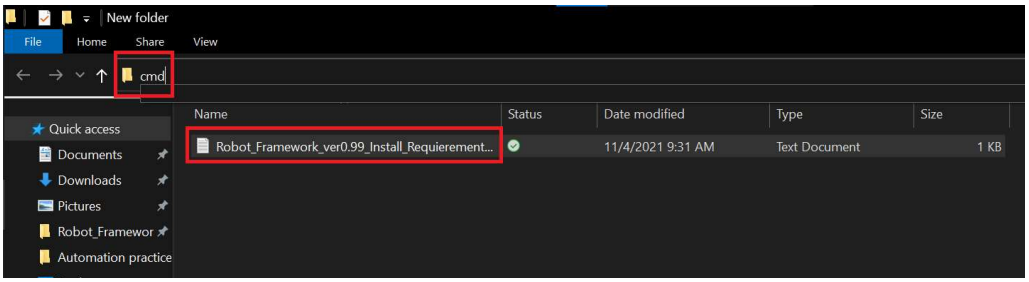
After that continue to subsection 3.4 to verify installation was successful.

6.1.1 Robot Framework installation using requirements file

- Installation using requirements file is easy way to specify a whole environment to be installed with one command

- *Download file* ***Robot_Framework_ver1.00_Install_Requierements.txt*** *from* Teams folder:
  Robot Framework\Robot Framework ver. 1.00

- In Windows, navigate to folder containing ***Robot_Framework_ver1.00_Install_Requierements.txt*** file, then type "*cmd*" and press enter (see picture below).
  Command prompt must be opened in same folder as your ***Robot_Framework_ver1.00_Install_Requierements.txt*** is located in.

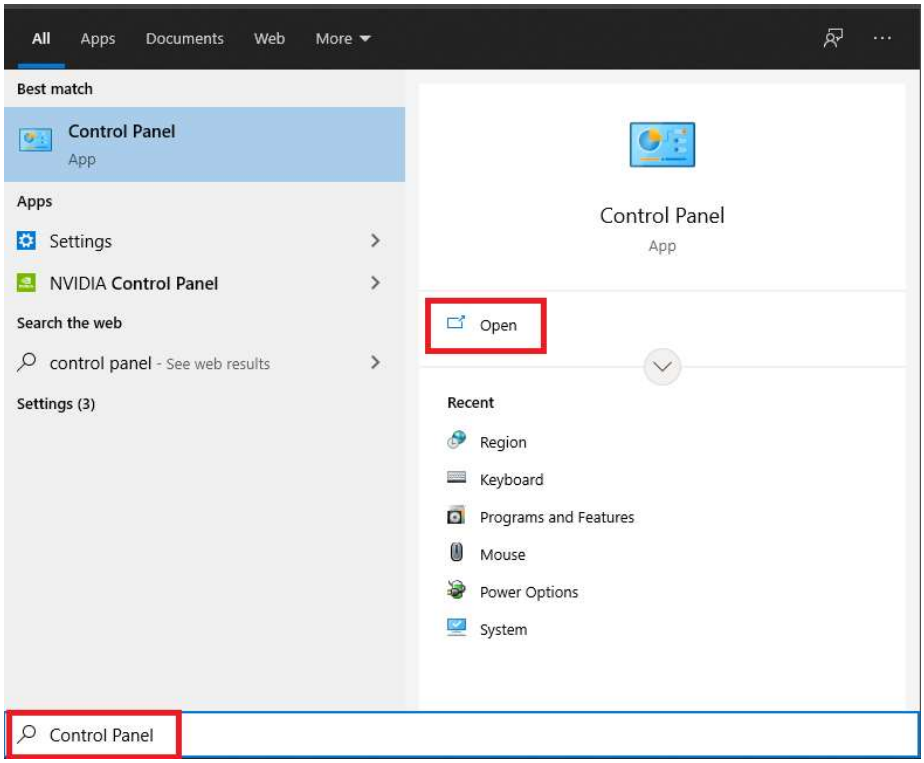


- Command line window will appear with current location set by default
- Now type:
  pip install -r Robot_Framework_ver1.00_Install_Requierements.txt
  and press enter

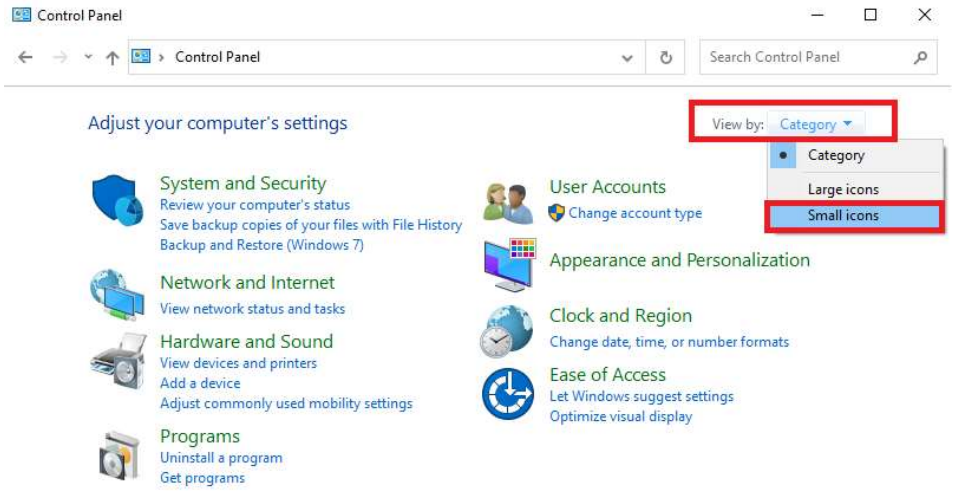
- Even if installation is successful, you will still get a warning message about missing installation location in PATH environment variable. So, **add location from warning into PATH variable**. See subsection 3.2

6.1.2 Set up environmental variables

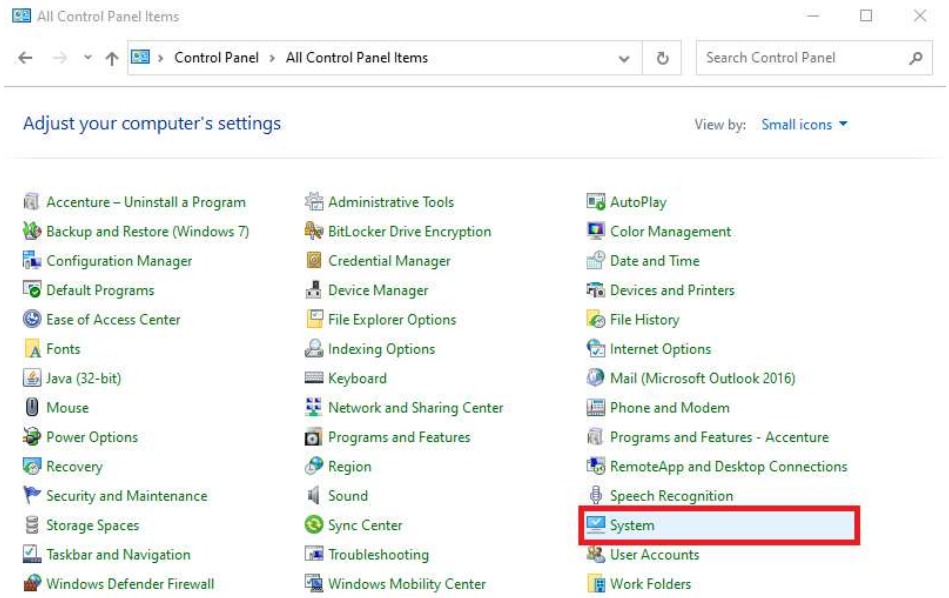
- Open *Control Panel*



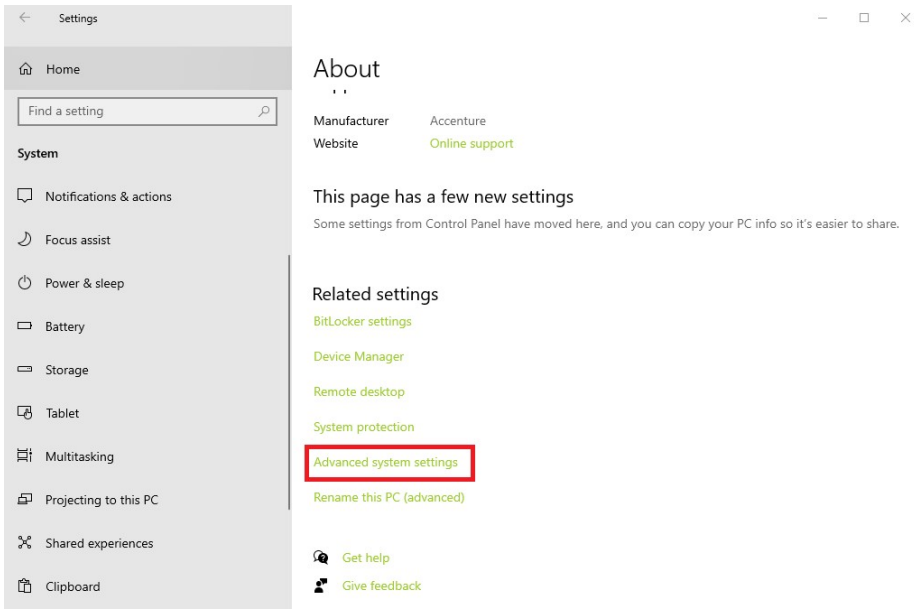
- Set View by: Small Icons



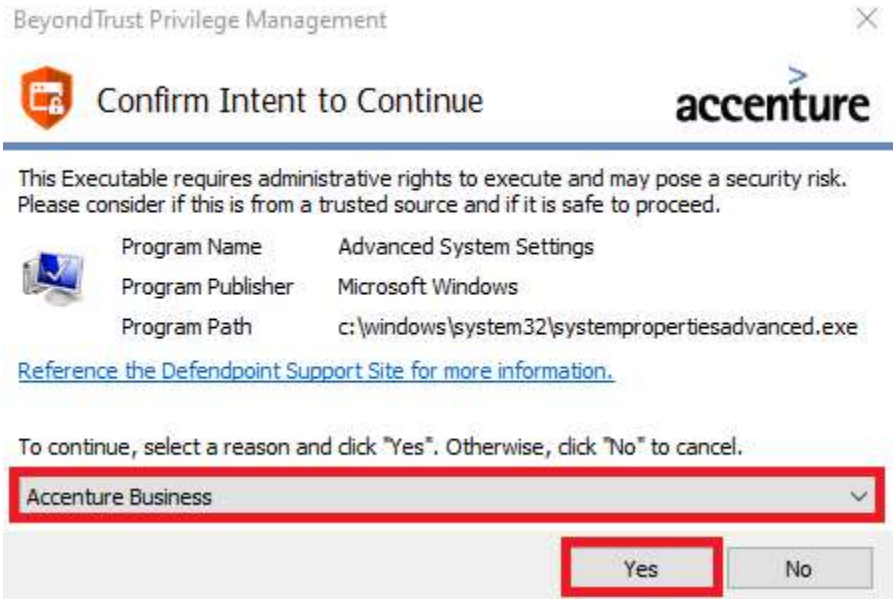
- Open *System* settings



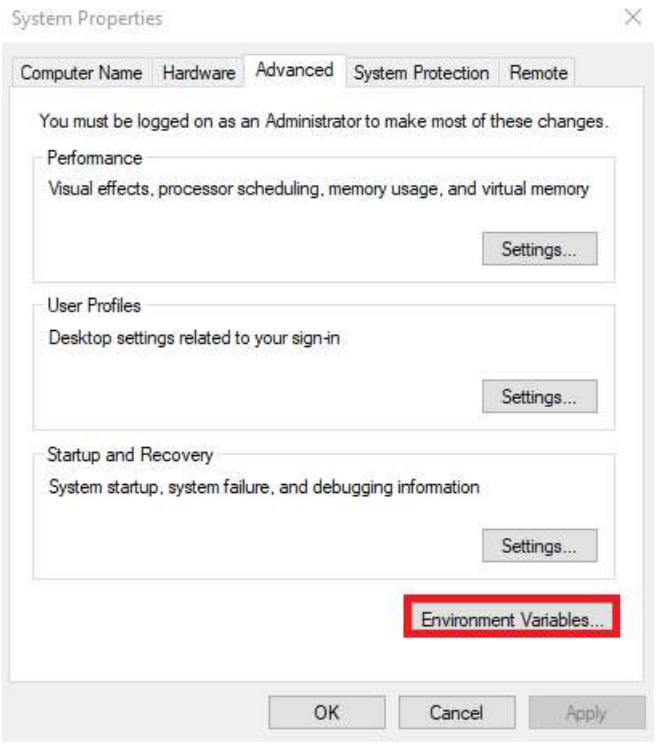
- Scroll the page and click *Advanced system settings*



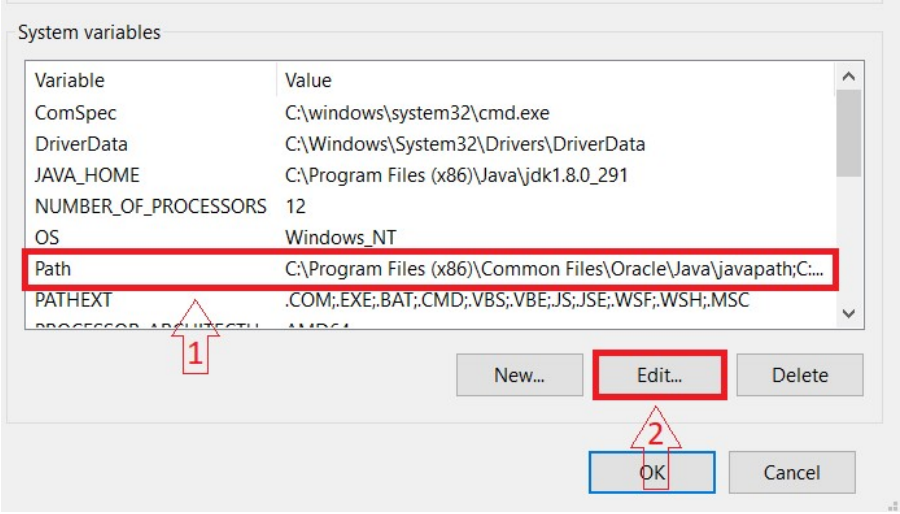
- Confirm Elevation Dialog



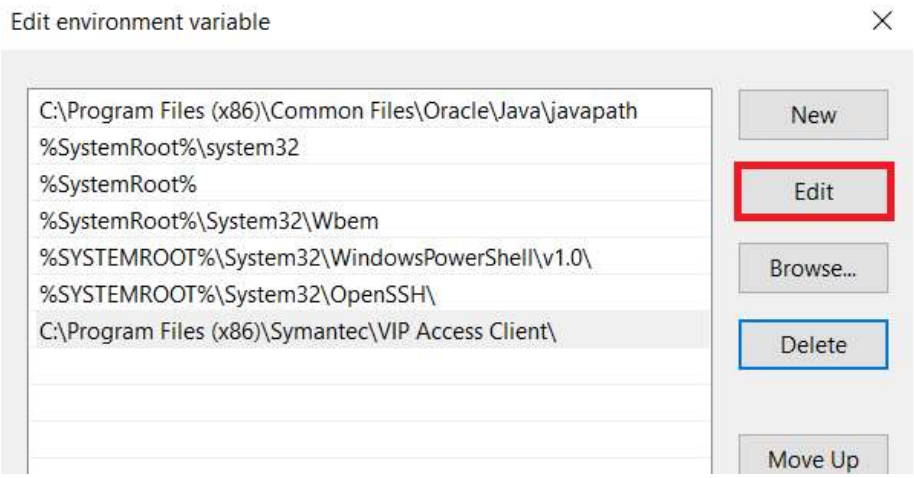
- Select *Environment Variables…* button



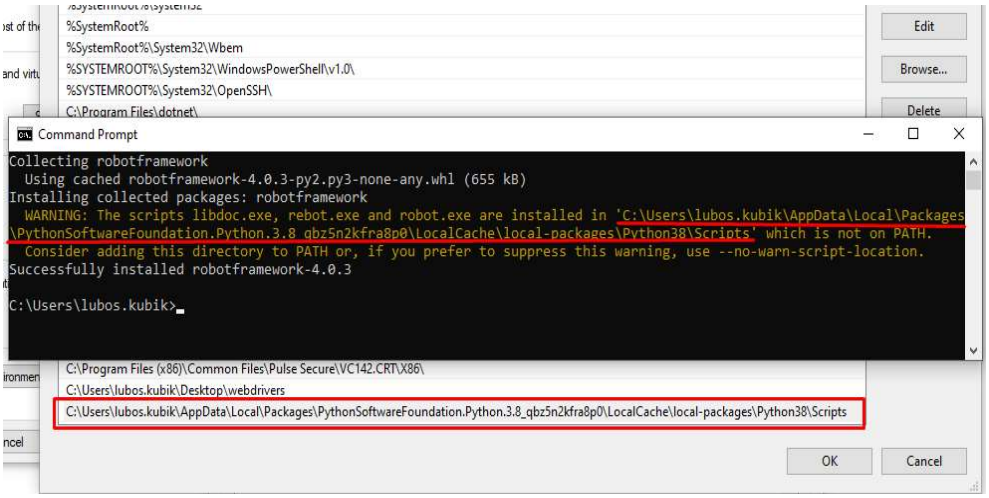
  - In *System variables* list select *Path* variable (1) and click *Edit… (*2).



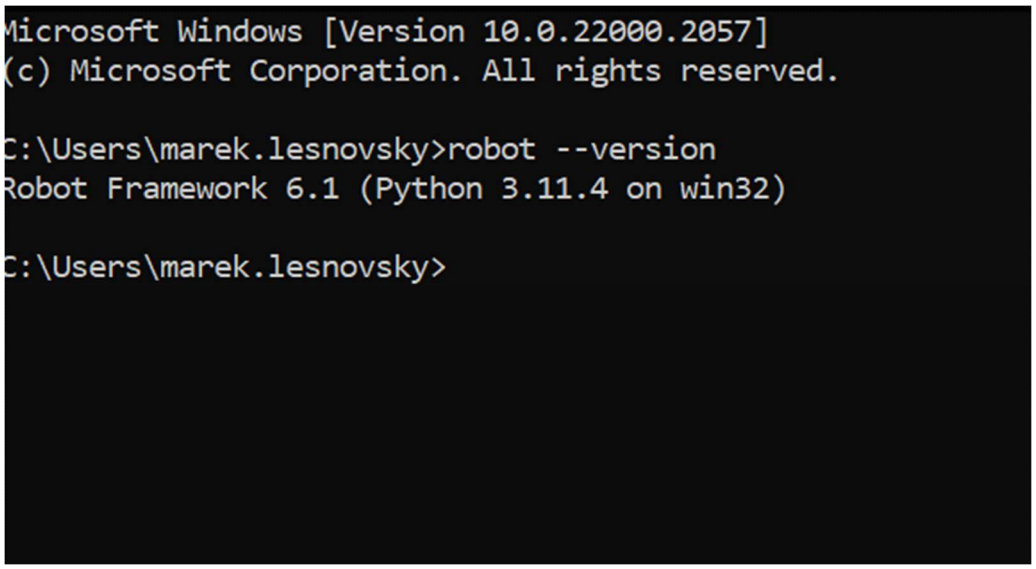
- In the *Edit environment variable* window click *Edit*.



- Add copied path from warning to new line. Restart your device.



- After restarting open Command Prompt, type following command and press Enter
robot –version



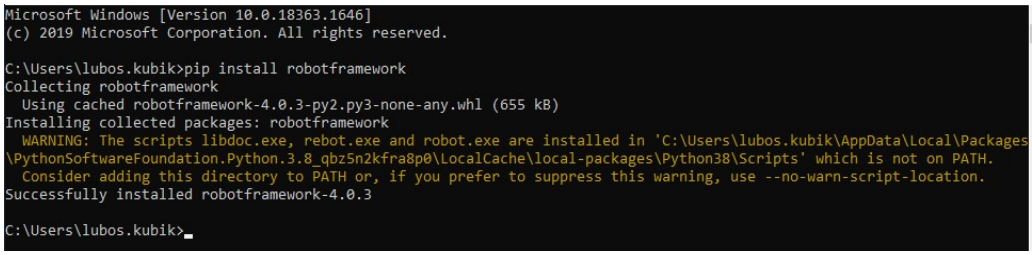
- If command will return version of robot framework your environment is installed along with dependencies.

6.1.3 Separate Installation

Only do when you skipped 3.1 Installation with requirements file.

- Open CMD and type:

pip install robotframework

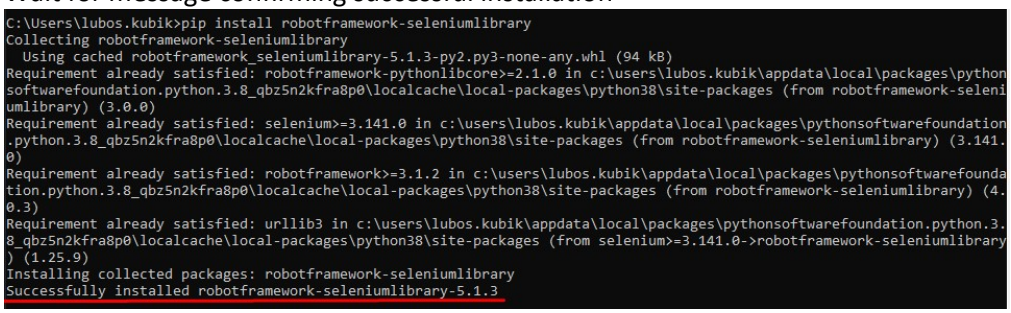


- Even if installation is successful, you will still get a warning message about missing installation location in PATH environment variable. So, add location from warning into PATH variable. See subsection 3.2

- Open CMD and type:

pip install robotframework-seleniumlibrary

- Wait for message confirming successful installation



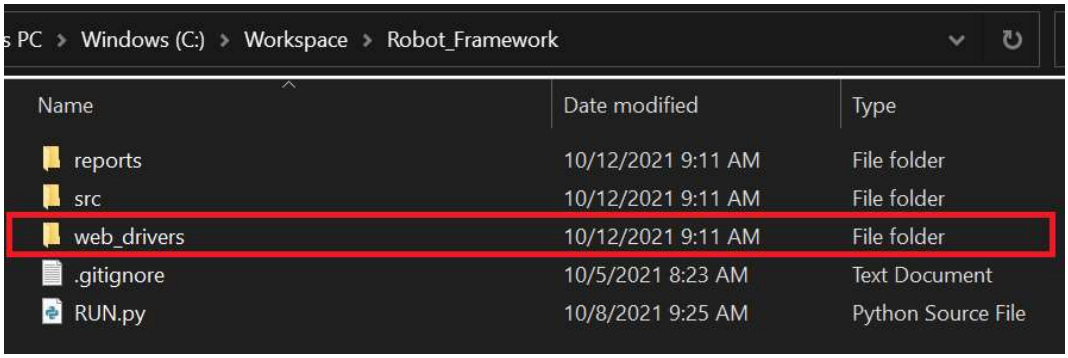
6.2 Initial Robot Framework project setup

6.2.1 Download Robot Framework

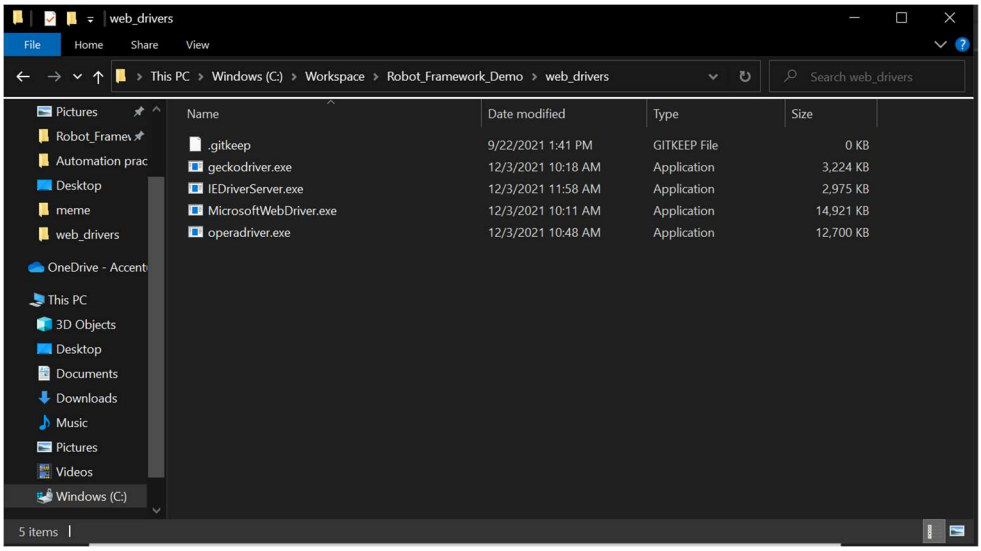
- Download latest version of Robot framework from ***Robot Framework***
- Unzip *Robot framework* into your workspace folder
- It is recommended to have Workspace folder directly in *C:/Workspace (/home/workspace for Linux and macOS users)*
- *It is **not recommended** to have Workspace as cloud folder (OneDrive)*

6.2.2 Copy WebDrivers

- Copy WebDrivers executable (e.g. Chromedriver, Geckodriver,…) into web_driver folder in Robot framework *(e.g. C:/Workspace/robot_framework/web_drivers for Windows users, /home/workspace/robot_framework/web_drivers for for Linux and macOS users)*



- Folder with multiple webdrivers for windows will look like this

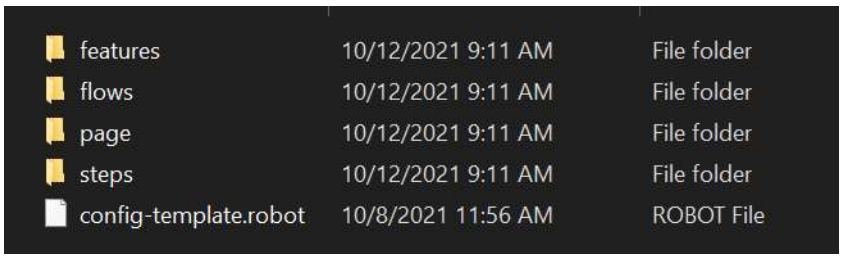


If you decide to use more than one IDE (e.g. for some reason you want to use Eclipse and IntelliJ IDEA), do not use the same project and folder. It is recommended to clone the project from your code base (GIT) to different folders for different IDEs.

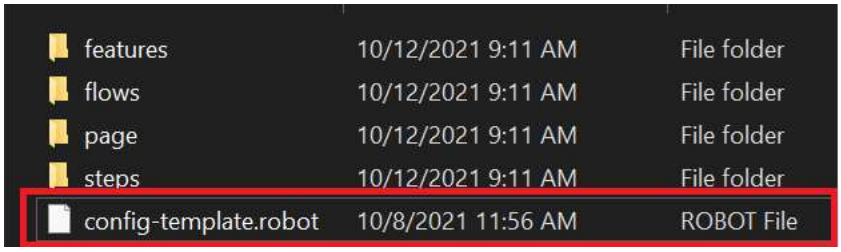
6.2.3 Edit config.robot

- Navigate to Robot_Framework/src

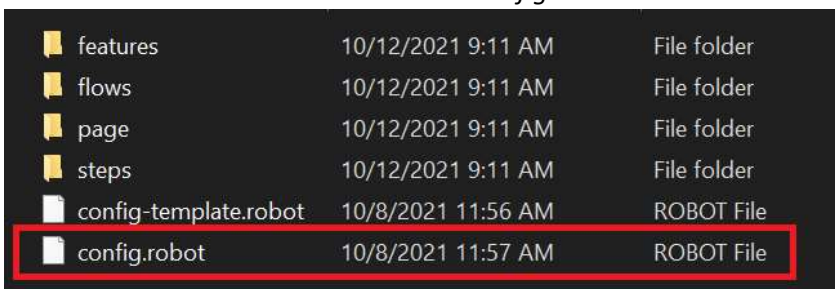
(e.g. *C:/Workspace/Robot_Framework/src* drivers for Windows users*, /home/workspace/gurkensalat/src* for Linux and macOS users*)*



- Make copy of config-template.robot file



- Paste it into same folder and rename it *config.robot*



- Open config.robot in your favorite text editor (e.g. Notepad++)

- Update config.robot as required. You can delete examples.
  It should look similar to following example (Windows OS example):

```
*** Settings ***
Documentation     A resource file with reusable xpaths and variables.
...
...               The system specific keywords created here form our own
...               domain specific language. They utilize keywords provided
...               by the imported SeleniumLibrary and Screenshot.

Library           OperatingSystem

# COPY THIS FILE AND RENAME TO CONFIG.ROBOT

*** Variables ***
#   Global Variable with site you want to test. (URL site)
#   Feel free to add more URL sites
${SERVER}               https://www.saucedemo.com/
${GOOGLE_EXAMPLE}       https://www.google.com/

#   Customize speed of selenium (integer) .
#   Details here https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Set%20Selenium%20Speed
${DELAY}                0

#In case you want to use FIREFOX please fill out path to firefox.exe
${FIREFOX_PATH}    C:\\Program Files\\Mozilla Firefox



#   Path to keyword file.
${KEYWORDS_FILE_PATH}           ../steps/all_keywords.robot

#   Path to POM file.
${POM_FILES_PATH}               ../page/all_pages.robot

#   Path to webdrivers directory
${WEB_DRIVERS_PATH}             ${EXECDIR}/web_drivers/

#   Path to config file (this file)
${CONFIG_FILE_PATH}             ../config.robot

#   Path to file with Flows
${FLOWS_FILE_PATH}              ../flows/flows.robot


#    Browserstack settings

${BROWSERSTACK_USERNAME}        username          # replace with your username
${BROWSERSTACK_ACCESS_KEY}      **********        # replace with your acces key

#    Saucelab Credentials

${SAUCELAB_USERNAME}             username         # replace with your username
${SAUCELAB_ACCES_KEY}            **********       # replace with your acces key

*** Keywords ***
#   Append path to drivers to Enviromental variable PATH
Include Browser Drivers
    Append To Environment Variable    PATH    ${WEB_DRIVERS_PATH}
```

config.robot legend

${SERVER}	URL to be used in tests (example) (CHANGE if needed)
${GOOGLE_EXAMPLE}	URL to be used in tests (example)
${DELAY}	Speed of selenium, 0 is highest speed. (CHANGE if needed)
${FIREFOX_PATH}	System path for firefox.exe (NEEDED ONLY IF USING FIREFOX TO TEST)
${KEYWORDS_FILE_PATH}	System path for Keywords file (DO NOT CHANGE)
${POM_FILES_PATH}	System path for Xpaths files (DO NOT CHANGE)
${WEB_DRIVERS_PATH}	System path for Webdrivers files (DO NOT CHANGE)
${CONFIG_FILE_PATH}	System path for CONFIG file (DO NOT CHANGE)
${FLOWS_FILE_PATH}	System path for Flows files (DO NOT CHANGE)
${BROWSERSTACK_USERNAME}	Change to your Browserstack username (NEEDED ONLY IF USING BROWSERSTACK)
${BROWSERSTACK_ACESS_KEY}	Change to your Browserstack access key (NEEDED ONLY IF USING BROWSERSTACK)
${SAUCELAB_USERNAME}	Change to your Saucelab username (NEEDED ONLY IF USING SAUCELAB)
${SAUCELAB_ACESS_KEY}	Change to your Saucelab access key (NEEDED ONLY IF USING SAUCELAB)

- Save updated config.robot

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

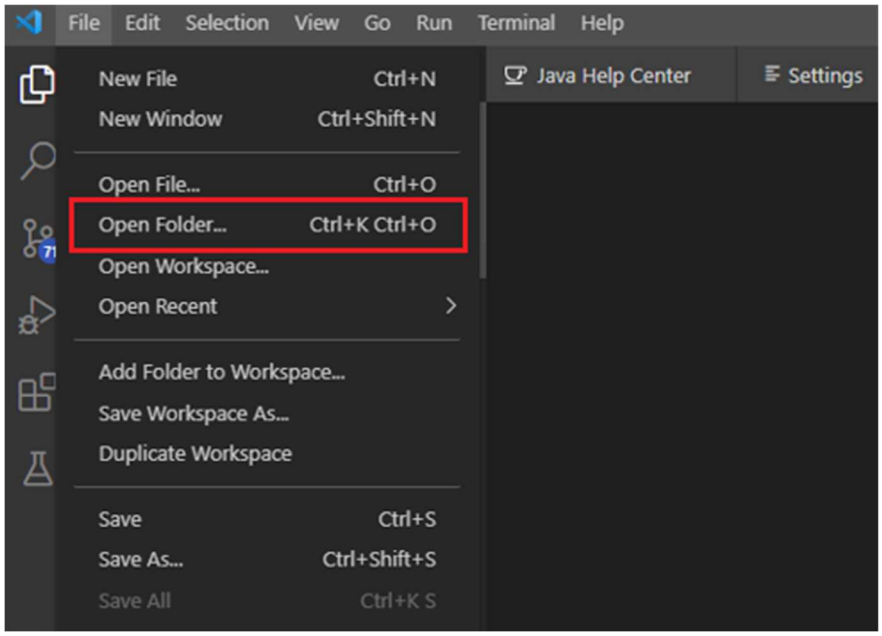
6.3 Import project to Visual Studio Code

6.3.1 Run Visual Studio Code

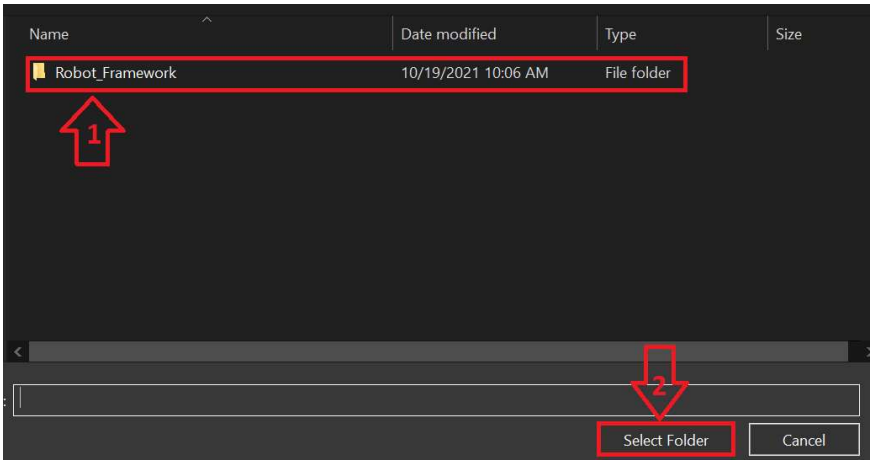
- Double-click the Visual Studio Code icon

6.3.2 Import project

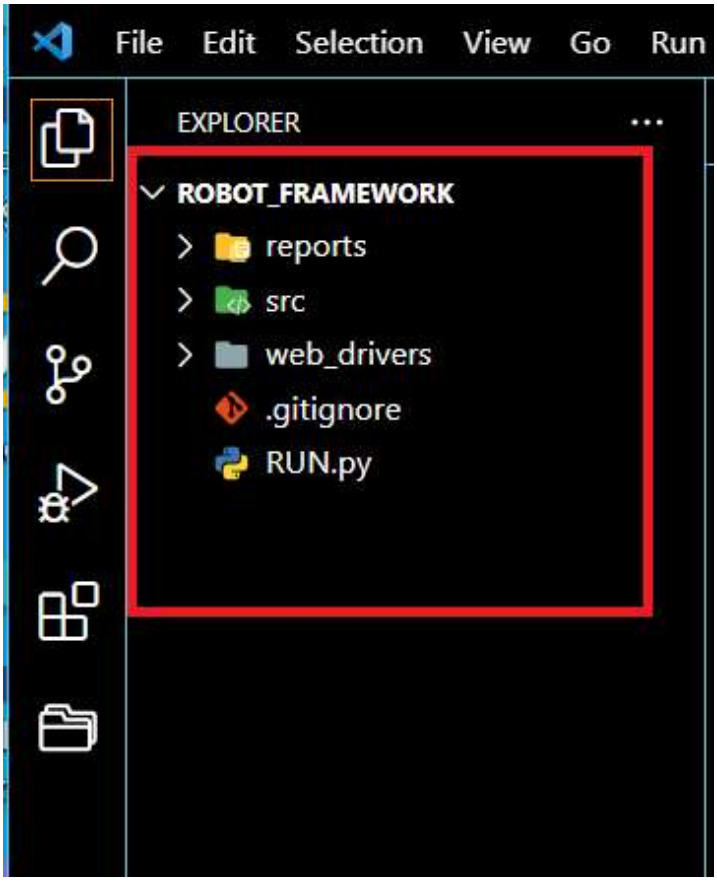
- From the main menu, select *File -> Open Folder*.



- In the dialog that opens, select the directory in which your *Robot Framework* is located (1) and click *Select Folder* (2).

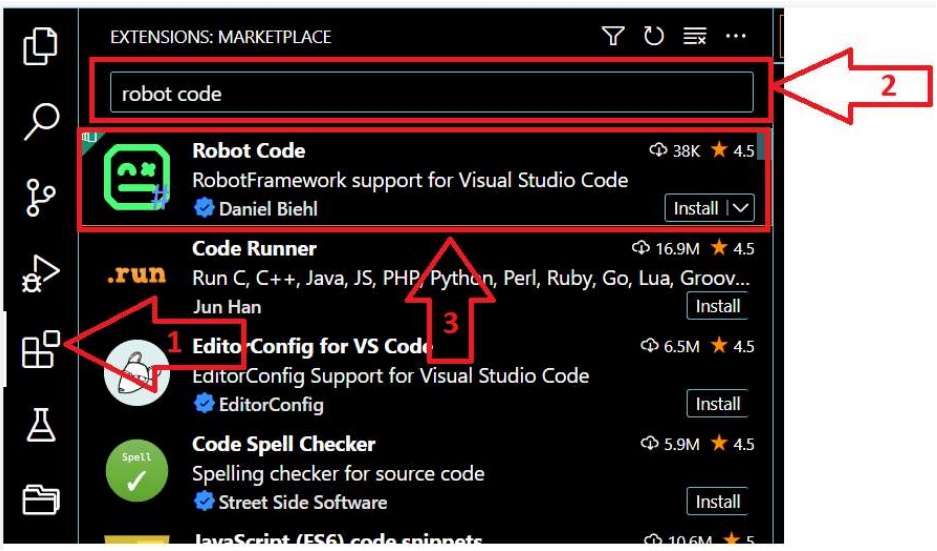


- *Robot Framework* is now visible in Visual Studio *Explorer*.

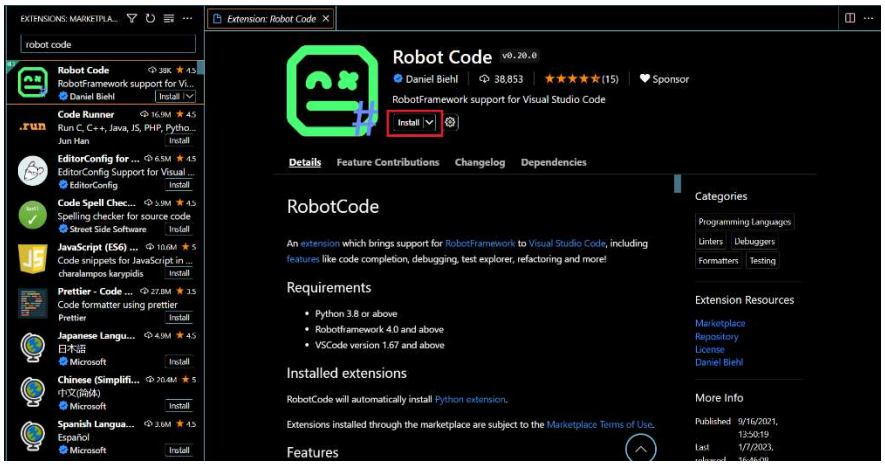


6.3.3 Adding Extensions to Visual Studio – Robot Code

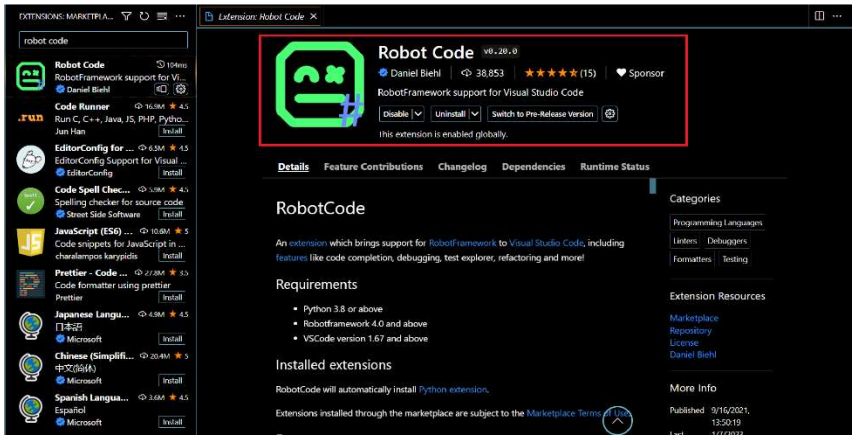
- In the left panel – click on Extensions icon (1)
- Enter "*robot code*" to the search input field (2)
- Click on the "Robot Code".



- On the Robot Code page, click *Install*.

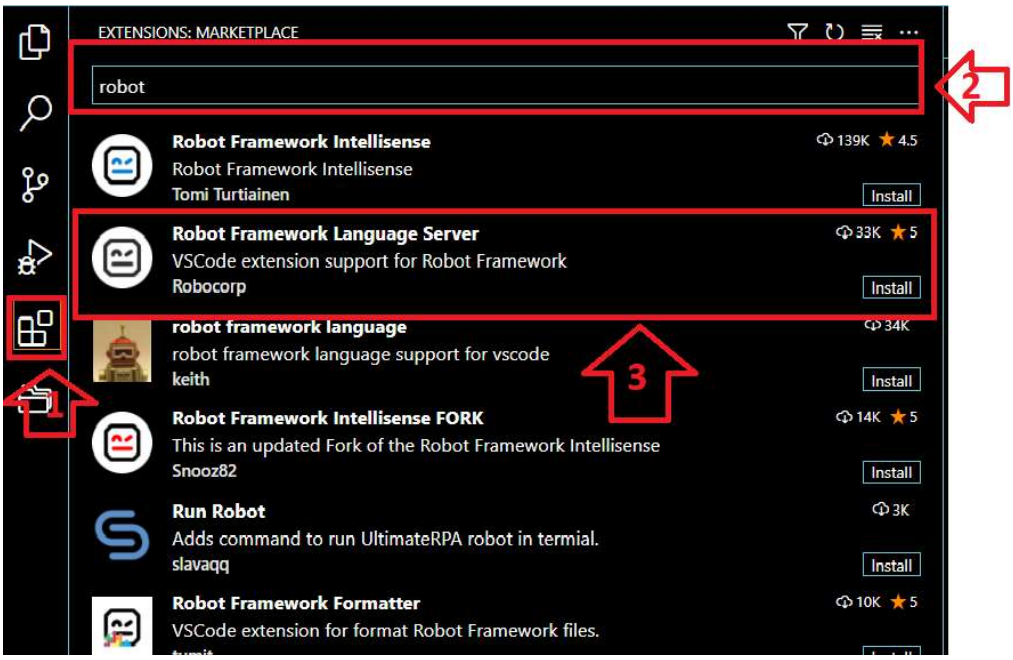


- Wait for installation process.

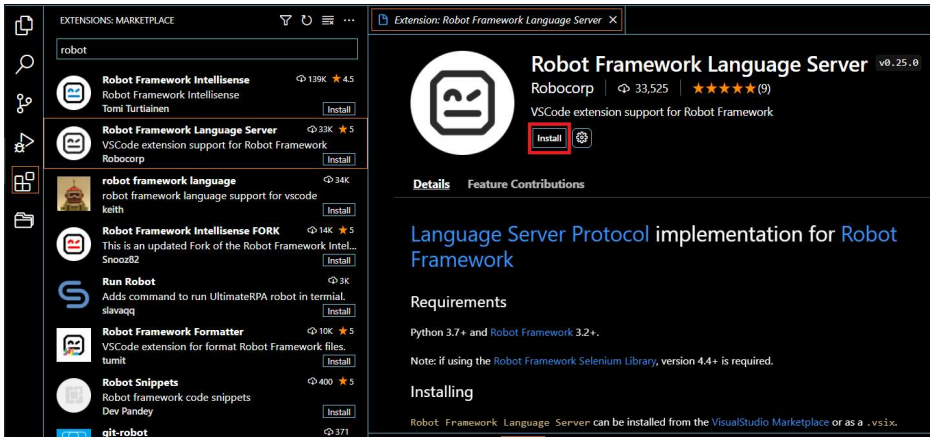


6.3.4 Adding Extensions to Visual Studio – Language Server

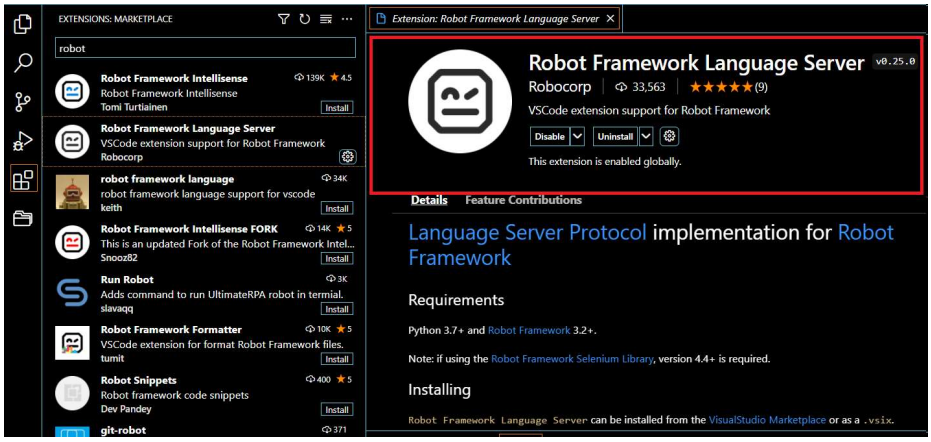
- In the left panel – click on Extensions icon (1)
- Enter "*robot*" to the search input field (2)
- Click on the "Robot Framework Language Server".



- On the Extension Pack for Java page, click *Install* to install Robot Framework.



- Wait for installation process.



6.4 Import Project to Eclipse IDE

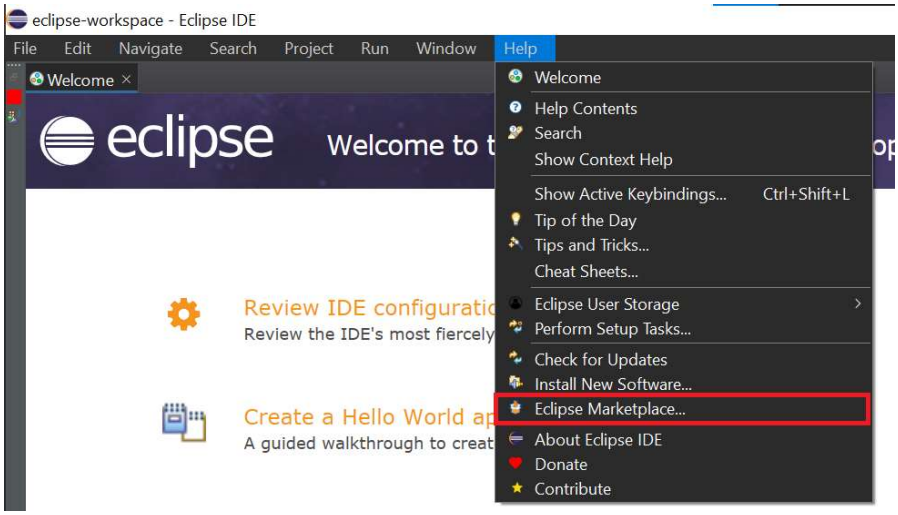
6.4.1 Run Eclipse

- Double-click the Eclipse icon

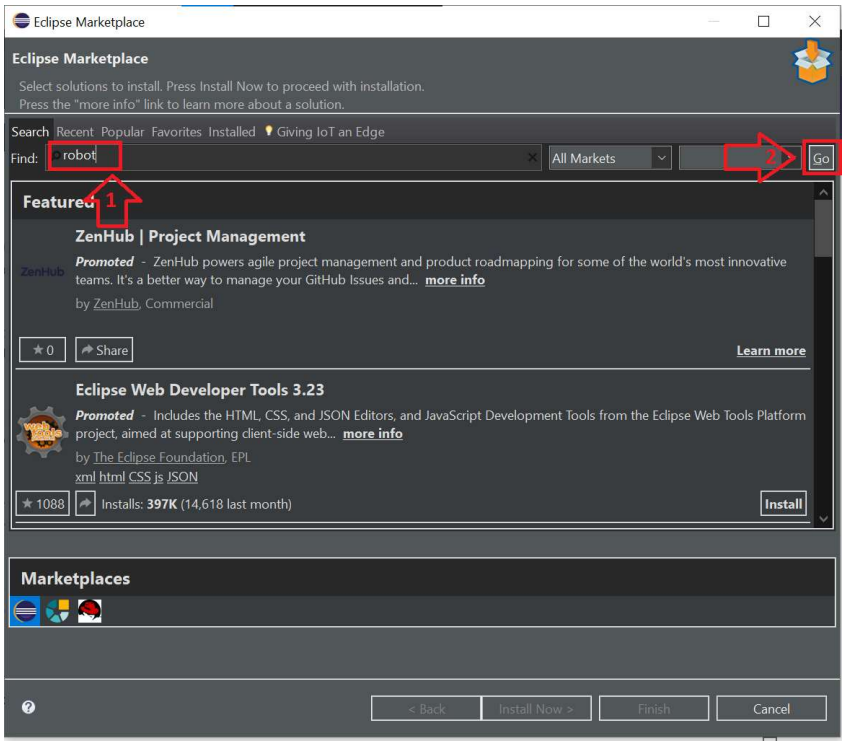
6.4.2 Install required plugins

Standalone Eclipse IDE does not support syntax of Robot Framework. We must install an additional plugin with Eclipse marketplace.

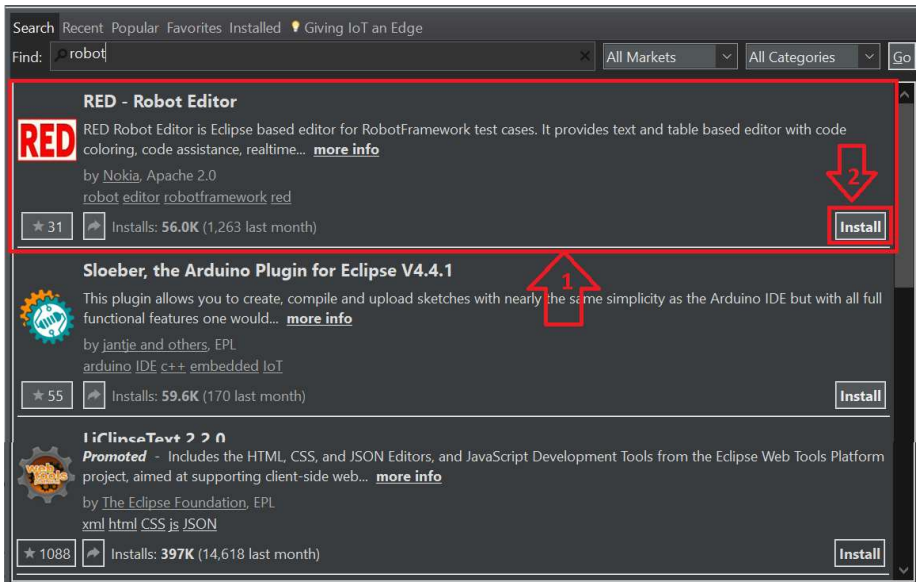
- From menu select *Help -> Eclipse Marketplace*



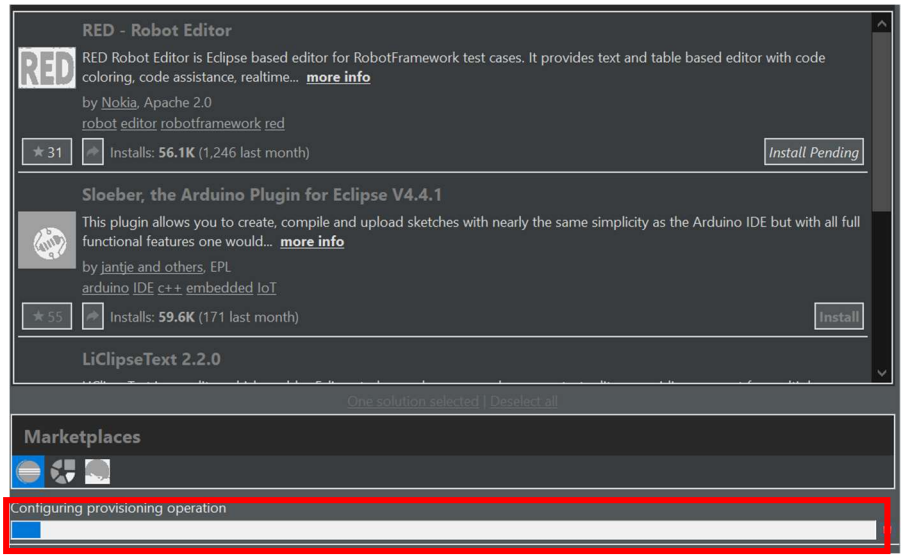
- Windows like this will pop up. Enter *robot* into search field (1) and then press *Go* button (2).



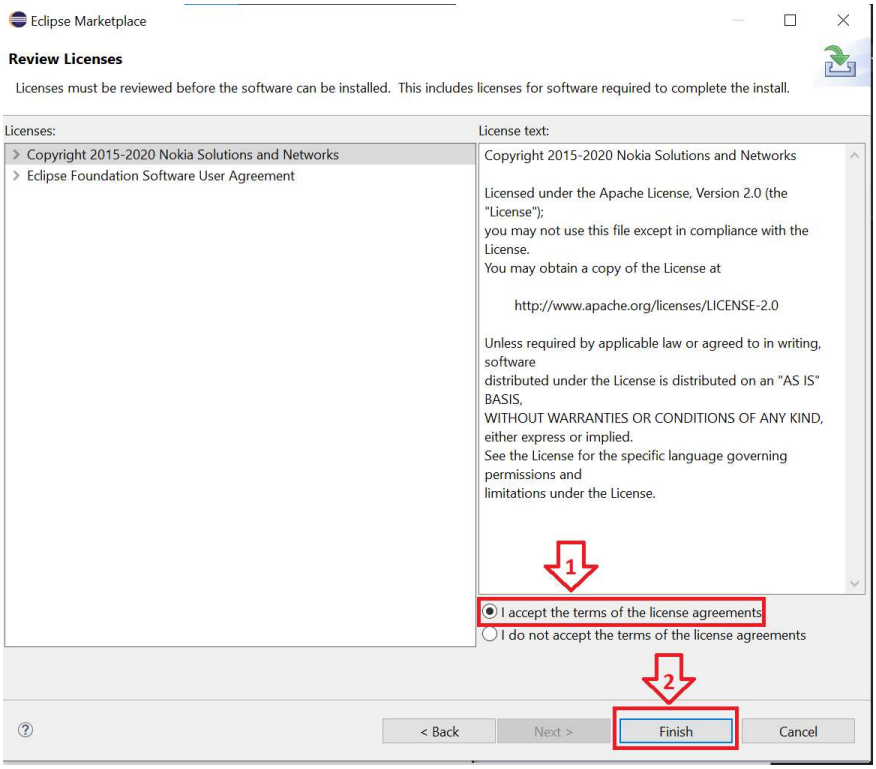
- Find *RED – Robot Editor* and press install.



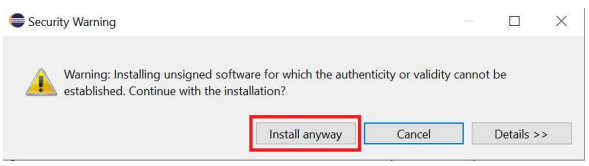
- Wait for Installation to complete.



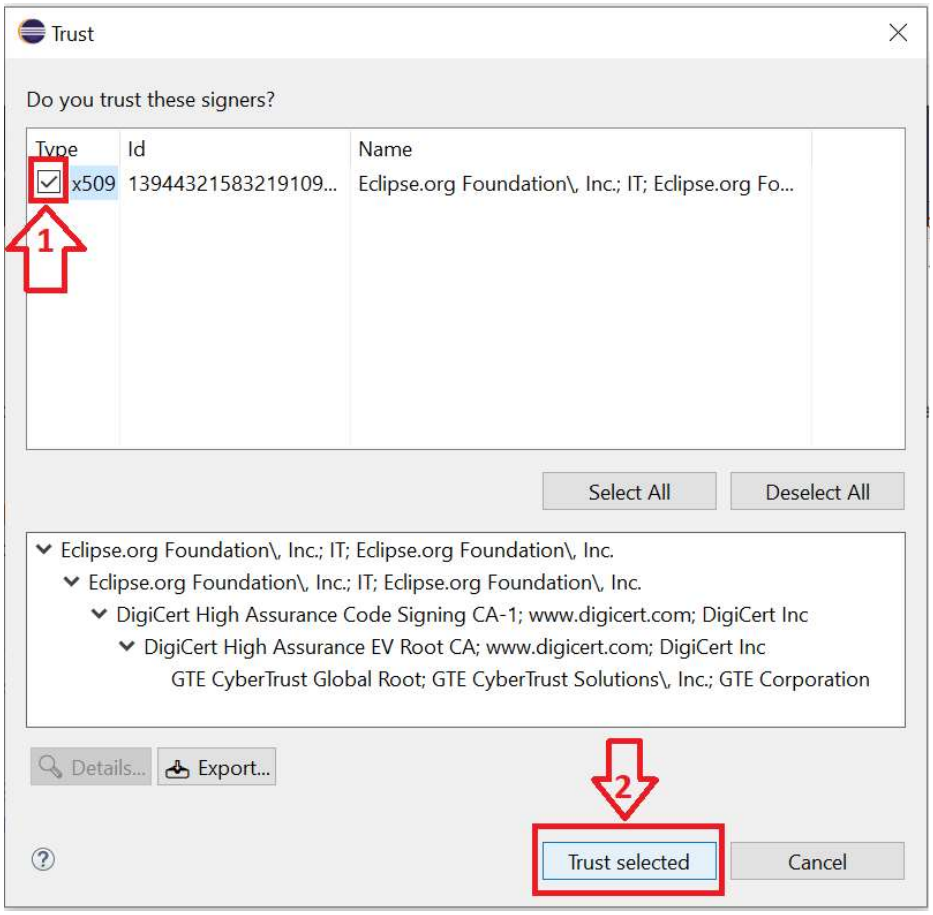
- Window with licenses will pop up. Click *Accept (1)* and then *Finish (2)* button



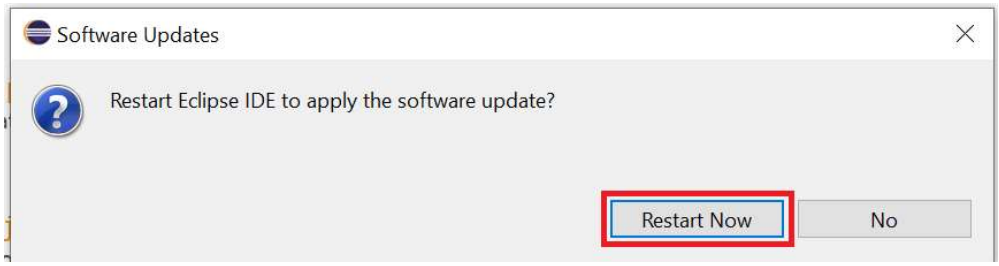
- Security warning will open. Click *Install anyway*



- Click on *checkbox (1)* or Select All button and then on *Trust Selected (2)* button.



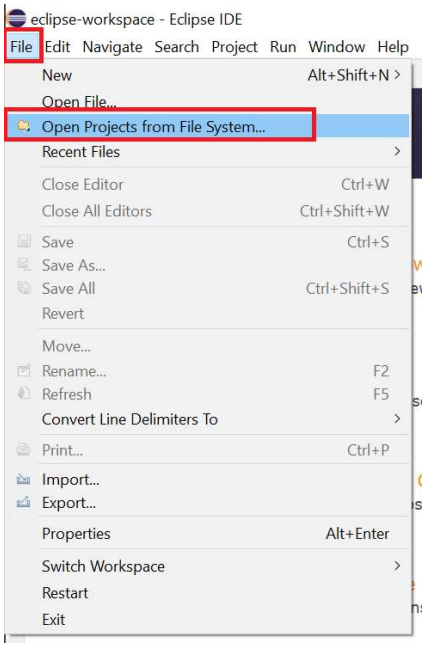
- Click *Restart Now* to restart Eclipse IDE.



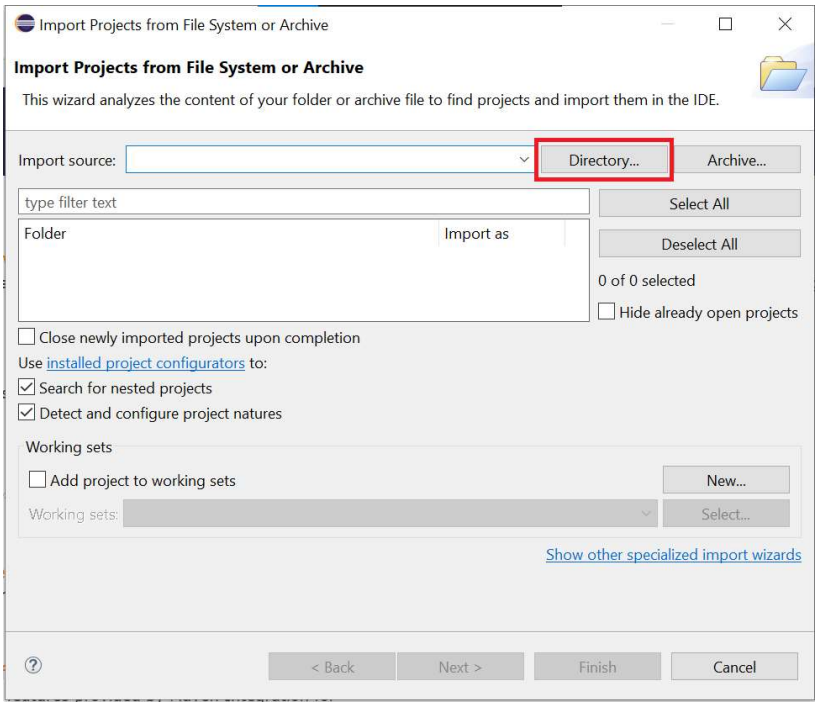
- After restart installation of required plugin will be complete.

6.4.3 Import project

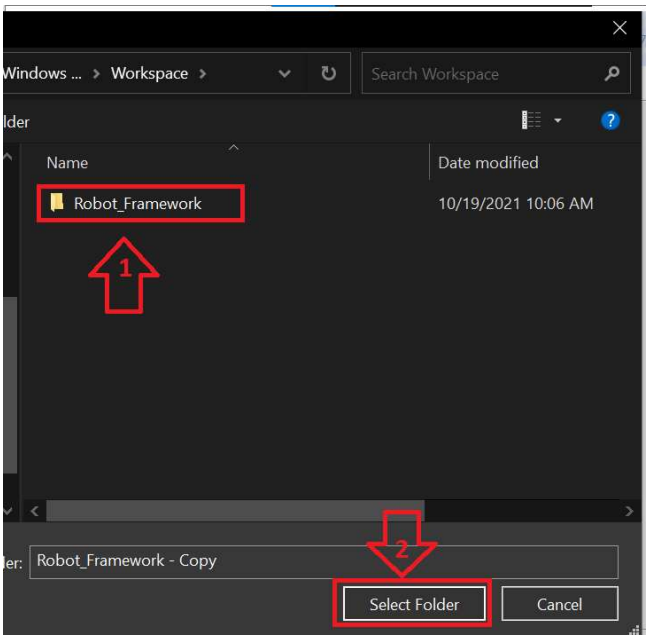
- From the main menu, select *File -> Open Project from File System…*.



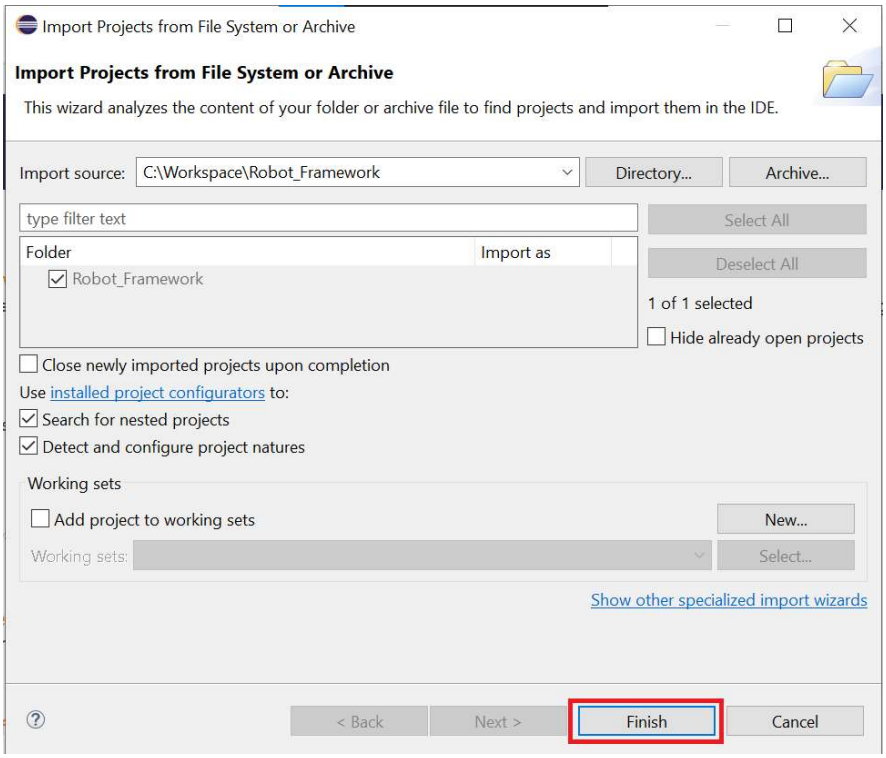
- Choose *Directory…* button



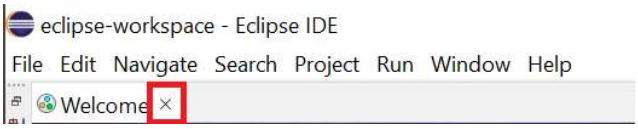
- In the dialog that opens, select the directory in which your *Robot Framework* is located (1) and click *Select Folder* (2).



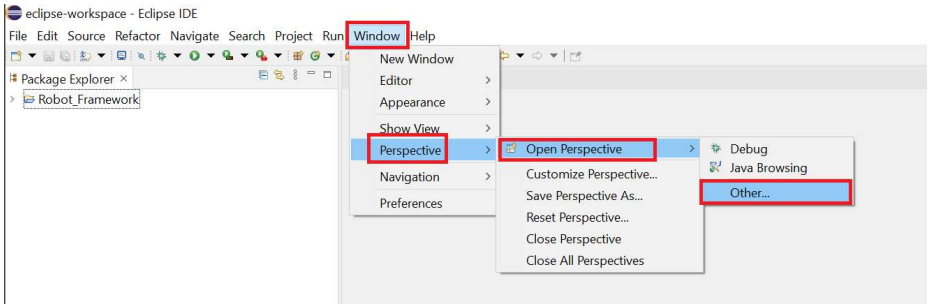
- After selecting the folder click on *the Finish* button.



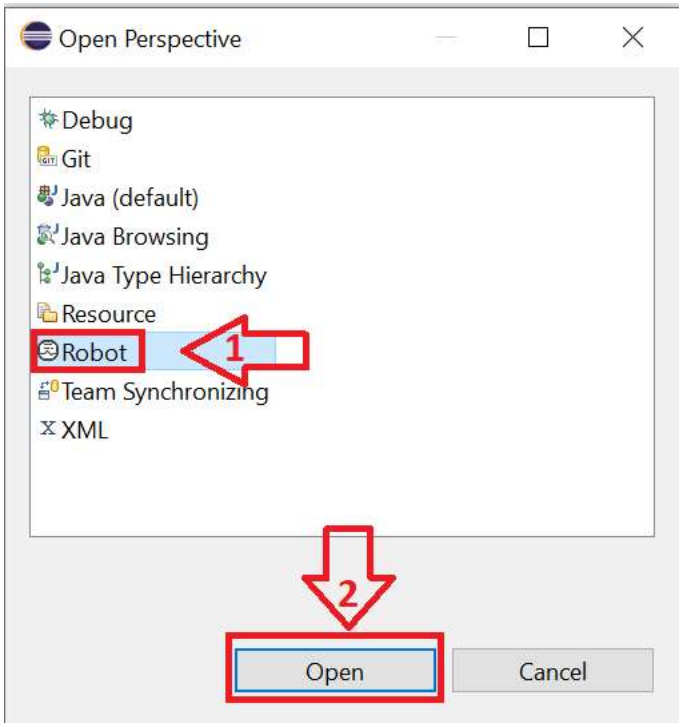
- You can now close the Welcome screen by clicking on x.



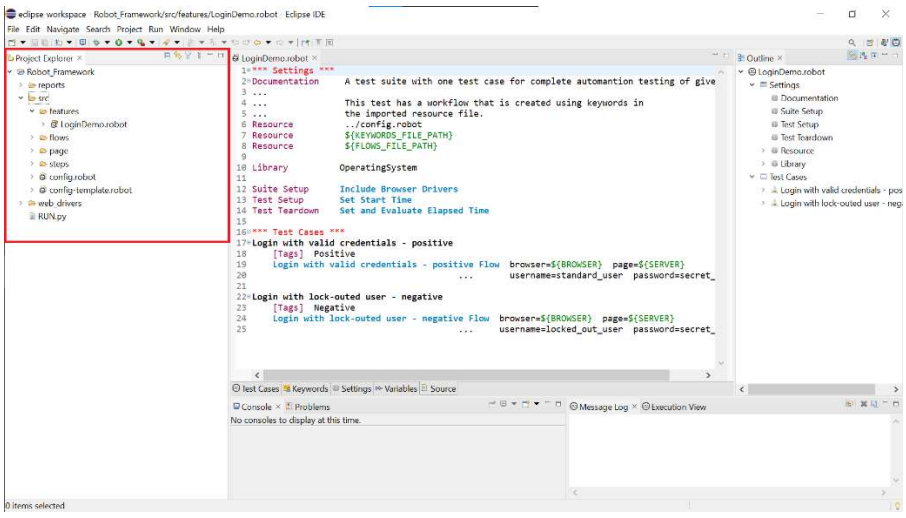
- To open Robot perspective instead of Java from main menu select *Window->Perspective->Open Perspective->Other…*



- In the window select *Robot (*1) and click *Open (2)* button.



- Now you can see your framework in Eclipse Project explorer with Robot perspective on.



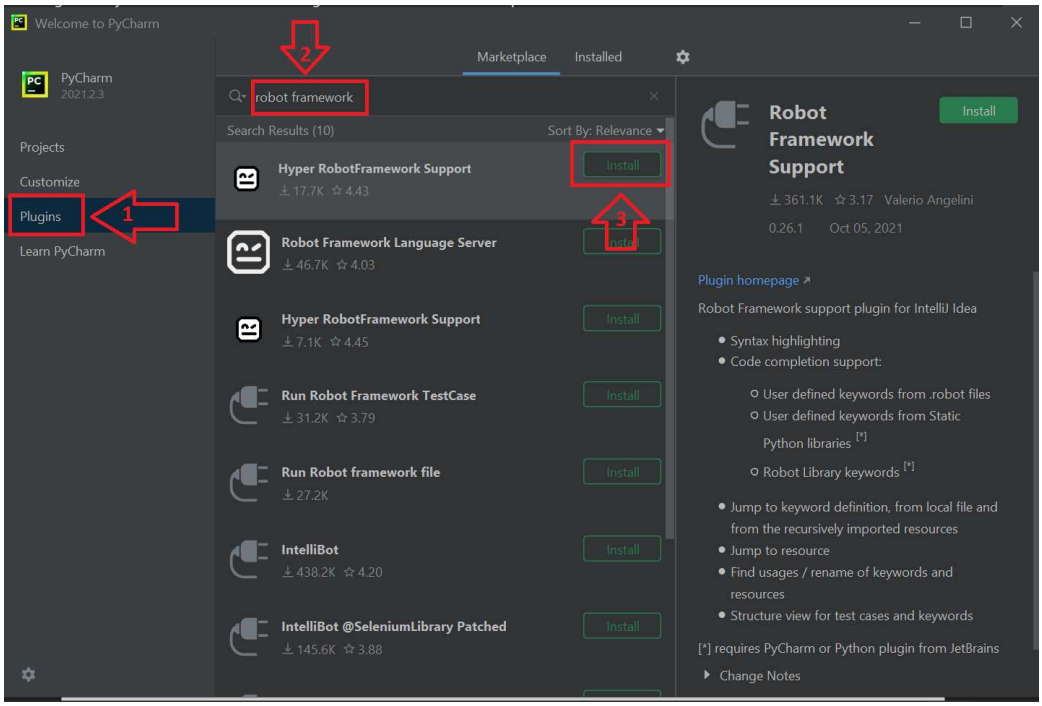
6.5 Import Project to PyCharm IDE

6.5.1 Run Visual Studio Code

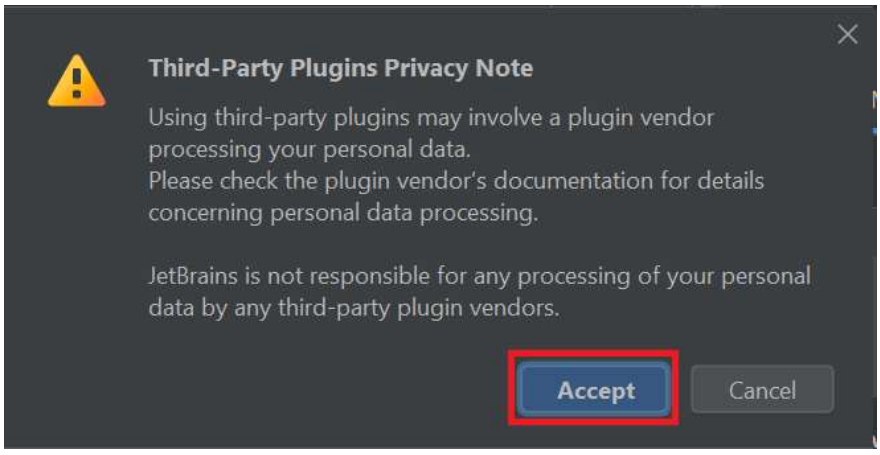
- Double-click the Visual Studio Code icon

6.5.2 Plugin installs

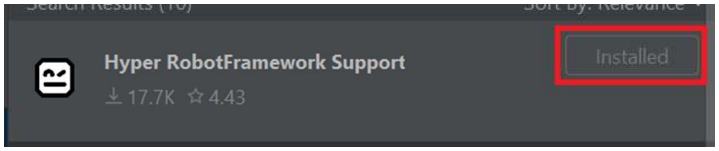
- Click on Plugins. In search bar enter *robot framework, find Hyper RobotFramework Support* and click *Install*



- Window will pop-up, click *Accept*



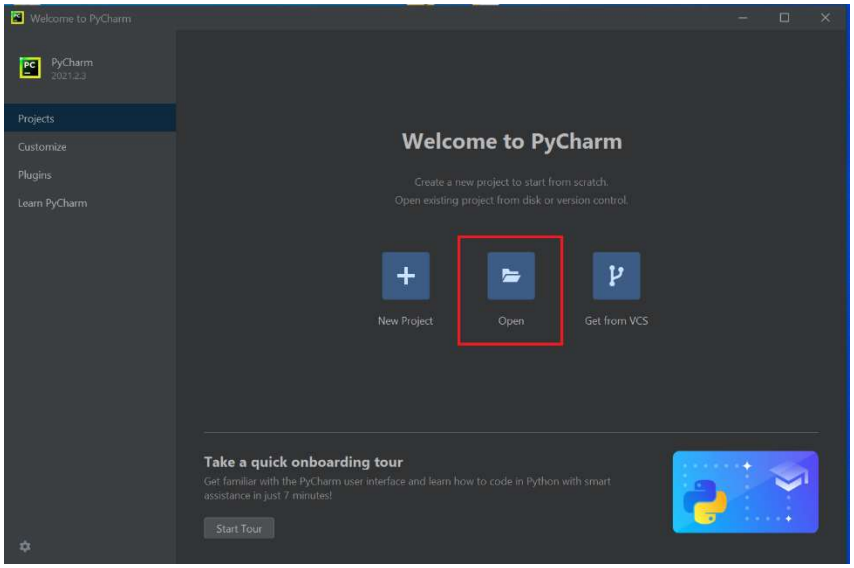
- Plugin will be installed.



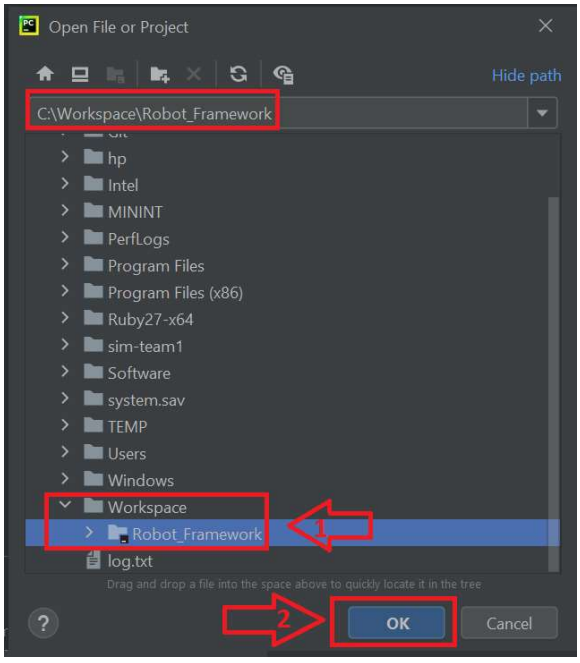
- Restart PyCharm IDE.

6.5.3 Import Project

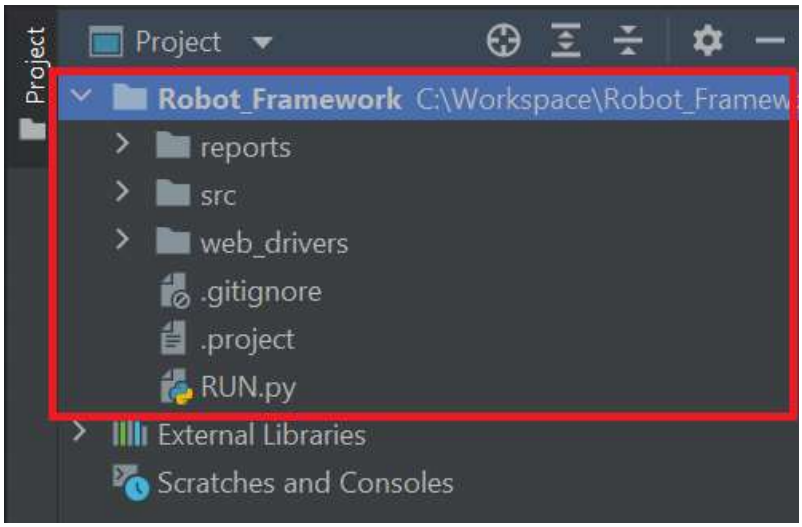
- Click on open



- In the dialog that opens, select the directory in which your *Robot Framework* is located (1) and click *OK* (2).



- Now you can see *Framework* in Project explorer.



7 Run test

There are multiple ways to run tests. We are using custom made Python script or executable file for test runs. It is possible to run the test through basic robot command, but it's not recommended.

- In PyCharm
  1. In PyCharm terminal
  2. With configured RUN button
  3. In external console
- In Eclipse IDE
  1. In Eclipse terminal
  2. With configured RUN button
  3. In external console
- Visual Studio Code
  1. In VS Code terminal
  2. In external console

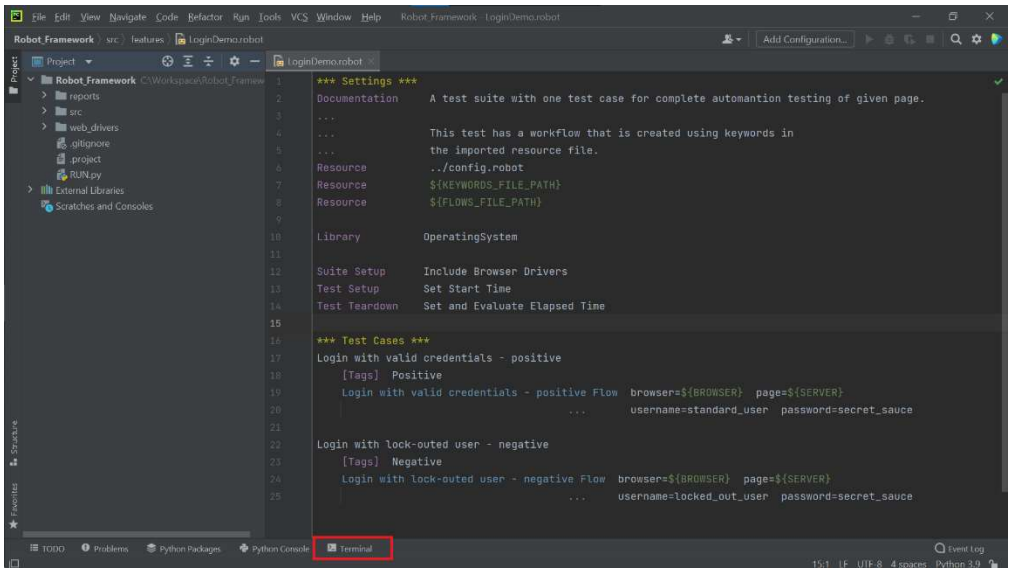
7.1 Run test in PyCharm

Test in PyCharm IDE can be run in these ways

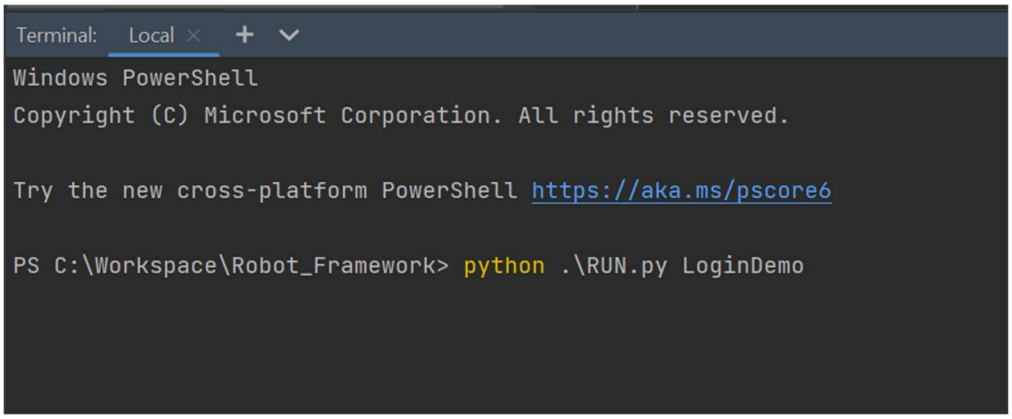
1. In internal Eclipse terminal
2. By configuring Run configuration… after that only by pushing run button
3. In external system terminal, outside of IDE

7.1.1 In PyCharm Terminal

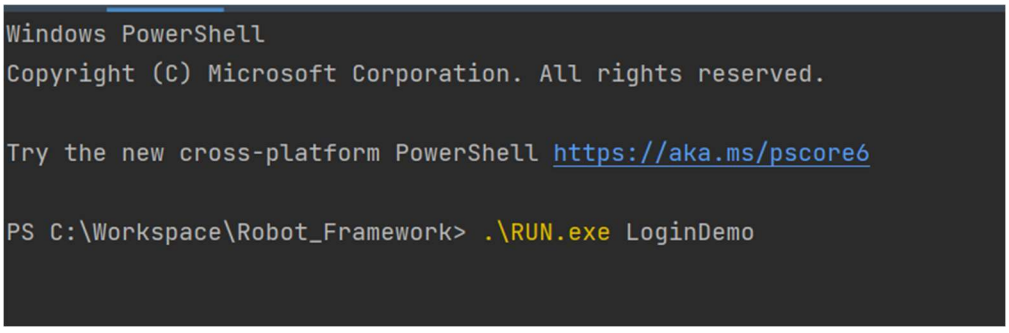
- Open terminal in PyCharm by clicking on terminal bookmark.



- Terminal will open, it works same as external terminal See subsection 9.4

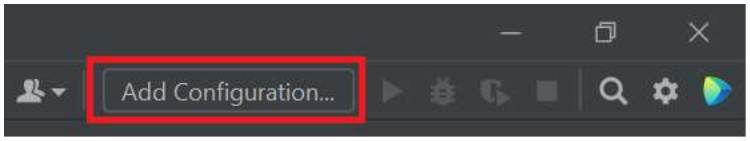


- Or you can use executable.

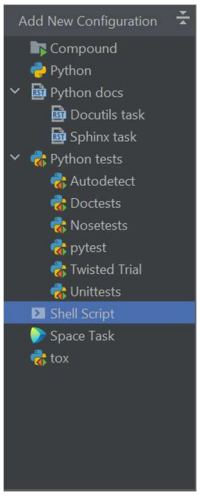


7.1.2 Configure Run button

- Click on *"Add Configuration…"*



- Click on + symbol and choose Shell Script



- Dialog window will pop up

- 1. Name – you can choose whatever name you want, for this example we chose robot_runner.

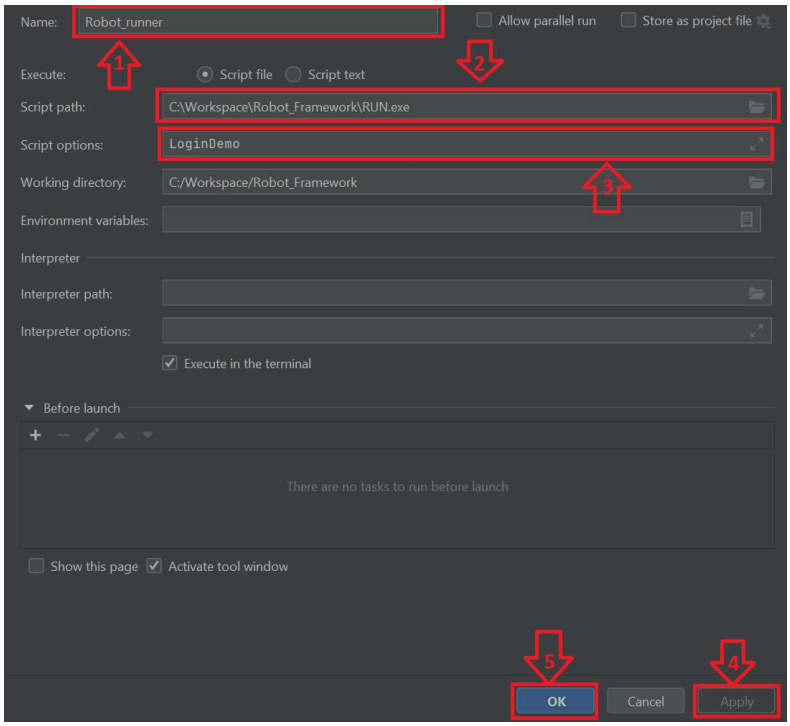
- 2. Path to our RUN.py script, in our case it's

  C:/Workspace/Robot_Framework/RUN.exe

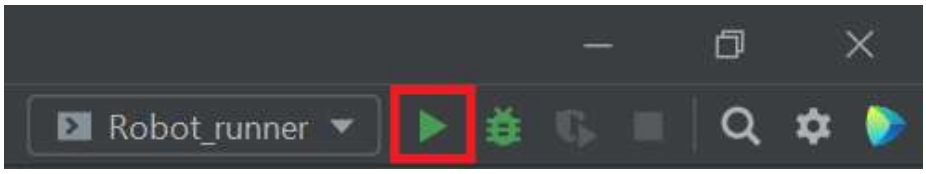
- 3. Parameters to launch script with, in our case its name of our feature file "LoginDemo". Feature file can be found in *C:/Workspace/Robot_Framework/src/features* folder

- 4. Click Apply button

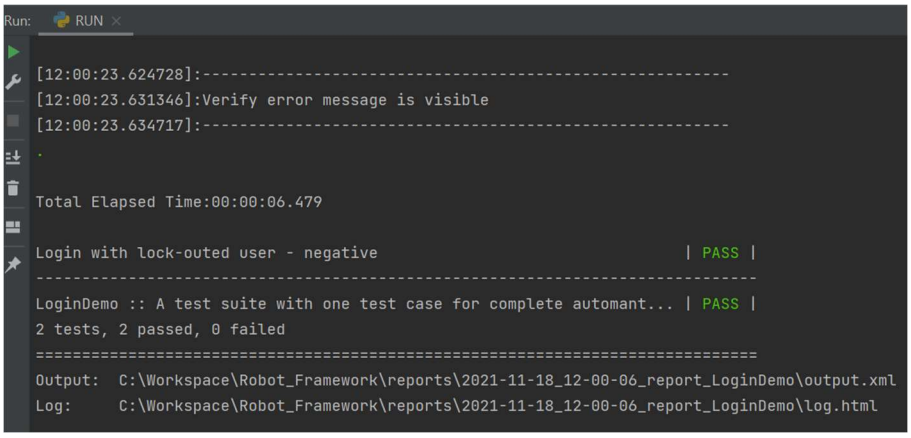
- 5. Click Ok button



- Test can be executed by clicking on the play button.



- You can see output in the console in the bottom part of screen.



- Reports and Logs are saved in the same folder as when executing text in external console. Viz subsection 9.4

7.2 Run test in Eclipse

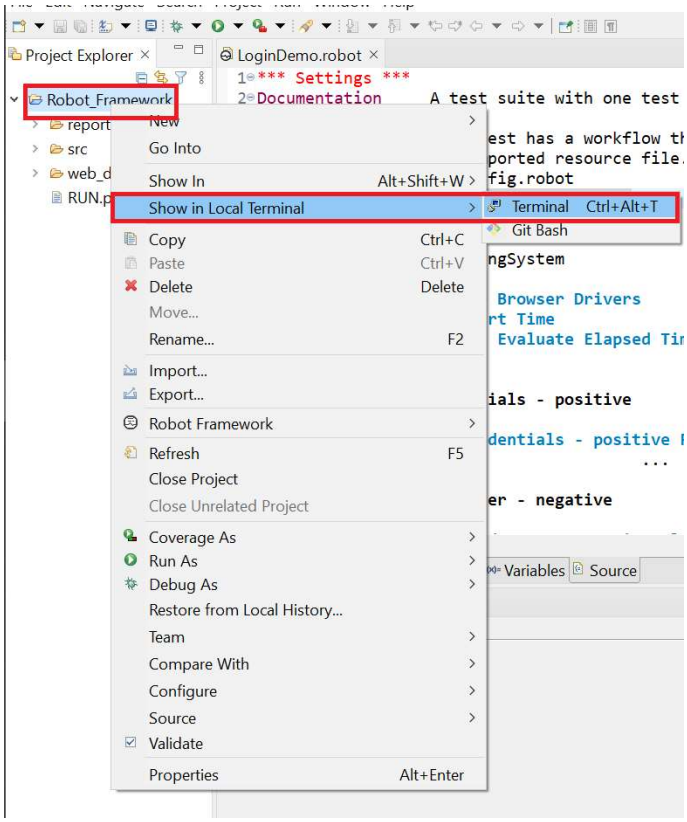
Test in Eclipse IDE can be run in these ways

1. In internal Eclipse terminal
2. By configuring Run configuration… after that only by pushing run button
3. In external system terminal, outside of IDE

//TODO you can choose one or more ways.

7.2.1 Run test in eclipse terminal

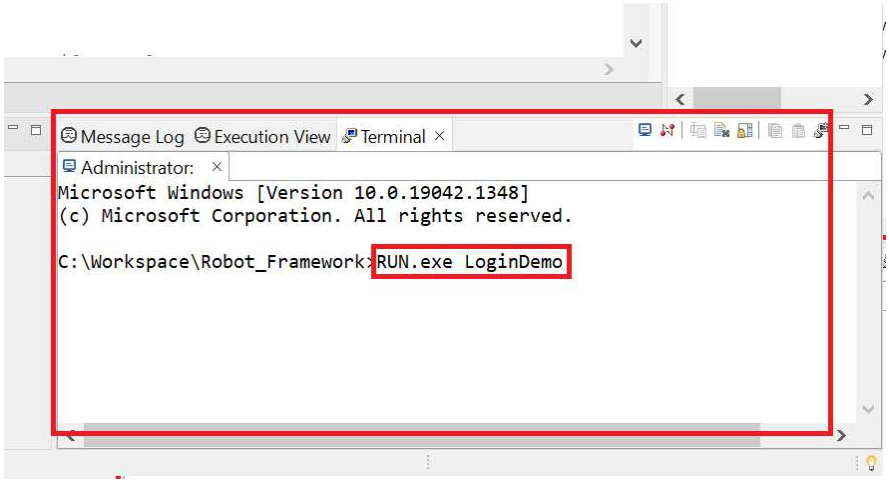
- Right click in your project file. In menu choose “Show in Local Terminal” and “Terminal”



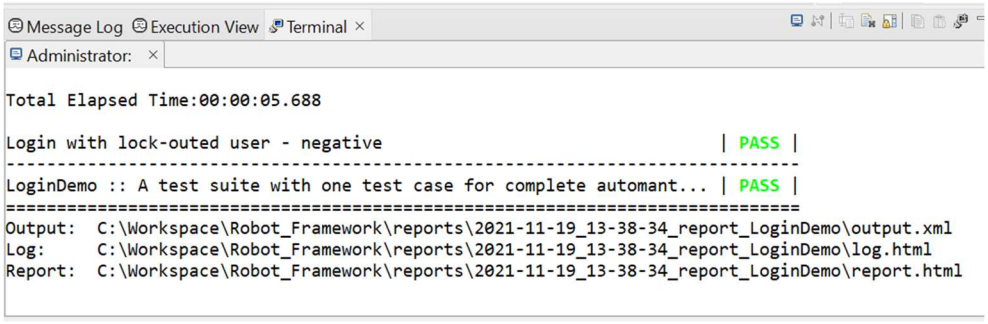
- Terminal will show on the bottom right part of the screen.
- Enter

RUN.exe LoginDemo

And press Enter to run the demo test.

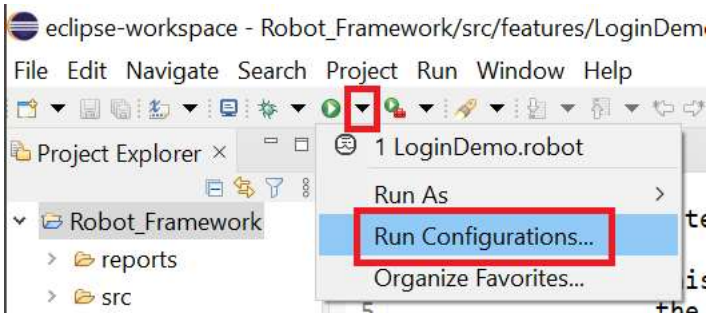


- The test will proceed and you will see the following output in the console.

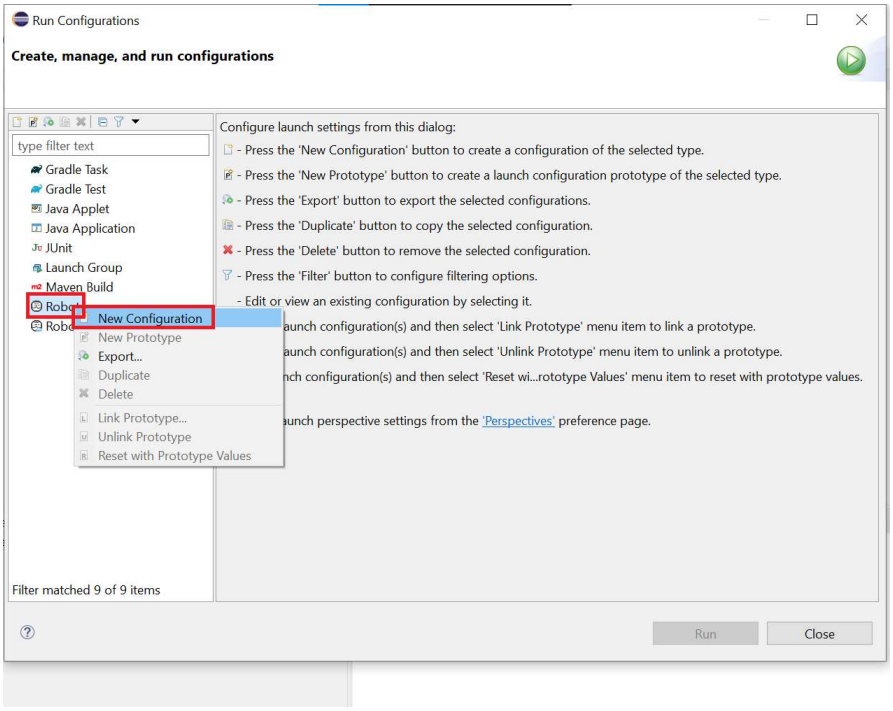


7.2.2 Set up Run Configuration

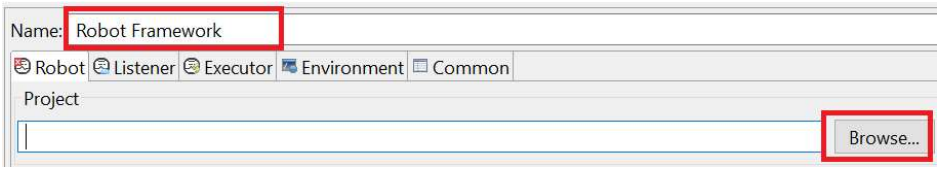
- Click on down arrow near play button and choose *Run Configurations…*



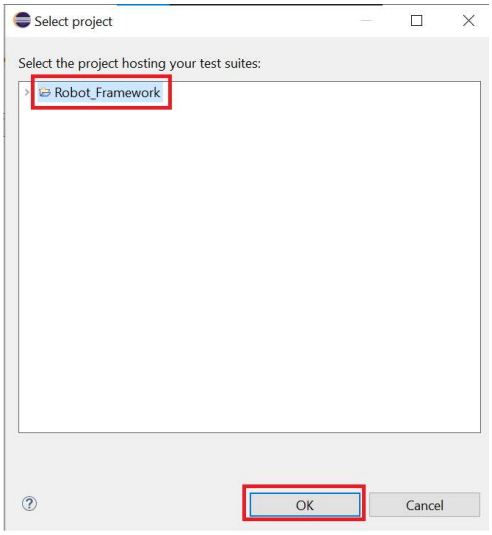
- Right click on robot and choose *New Configuration*



- Choose Whatever name you want, we chose “Robot Framework”
- Click on Browse… button in project section.

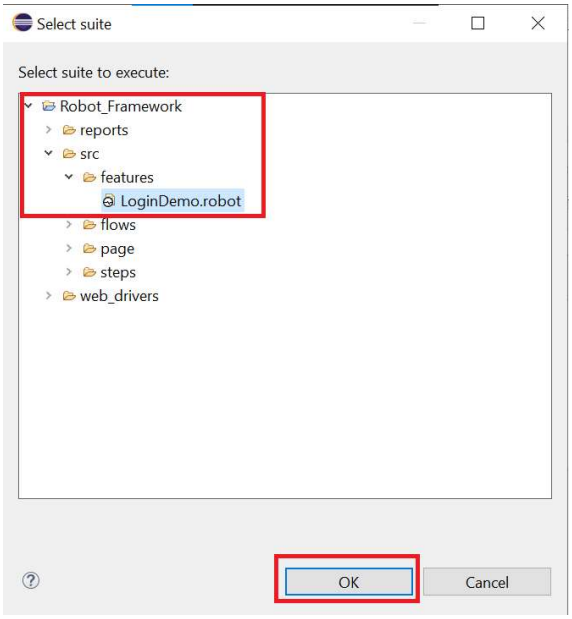


- Click on Robot_framework project and click OK

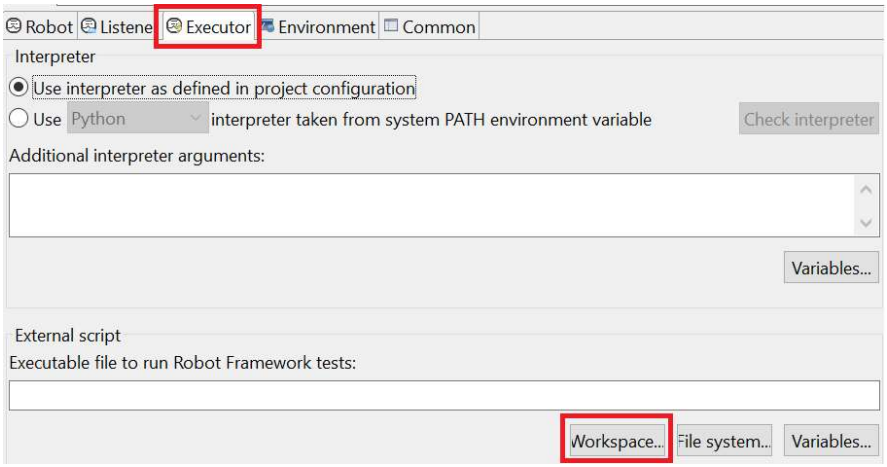


- In Suite Click on *Browse…*

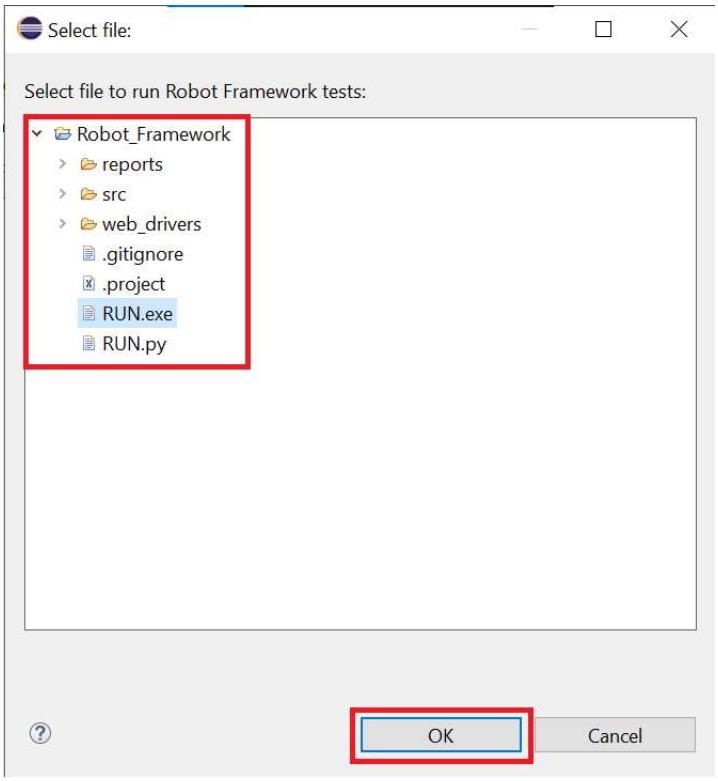
- Select your features file in *Robot_Framework -> src -> features*, in our case *LoginDemo.robot* and then click *OK* button



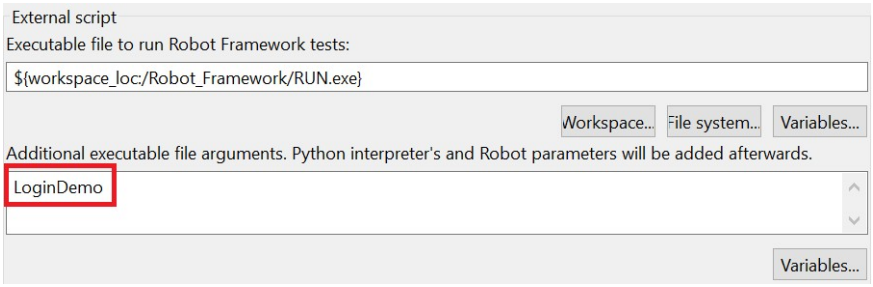
- Click on *Executor* bookmark and under External script section select *Workspace…*



- Select *RUN.exe* in *Robot_Framework* and click *OK*

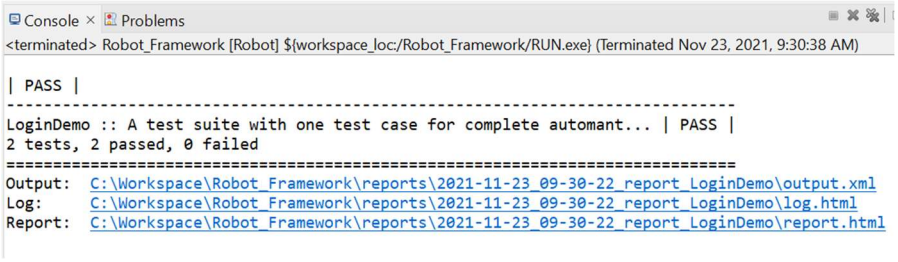


- Into *Additional executable file arguments* insert name of your feature file, in our case *LoginDemo*



- 1. Click *Apply*
- 2. Click *Run*

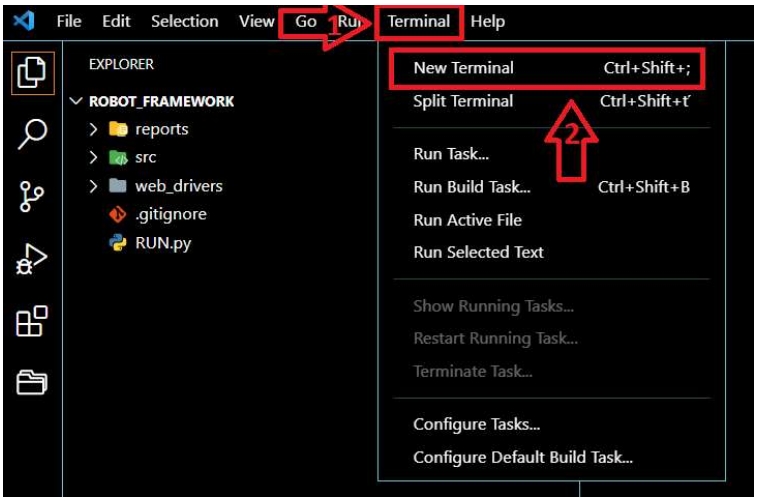
- The test will run and you will see the following output in console.



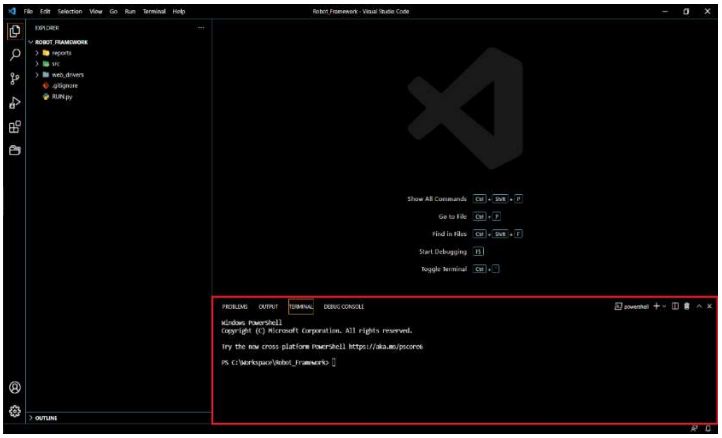
- Next time you can run the test simply by clicking the run button.

7.3 Run test in Visual Code terminal

- On upper menu select “Terminal” (1)
- Other options will roll out. Select new terminal
  //TODO run config



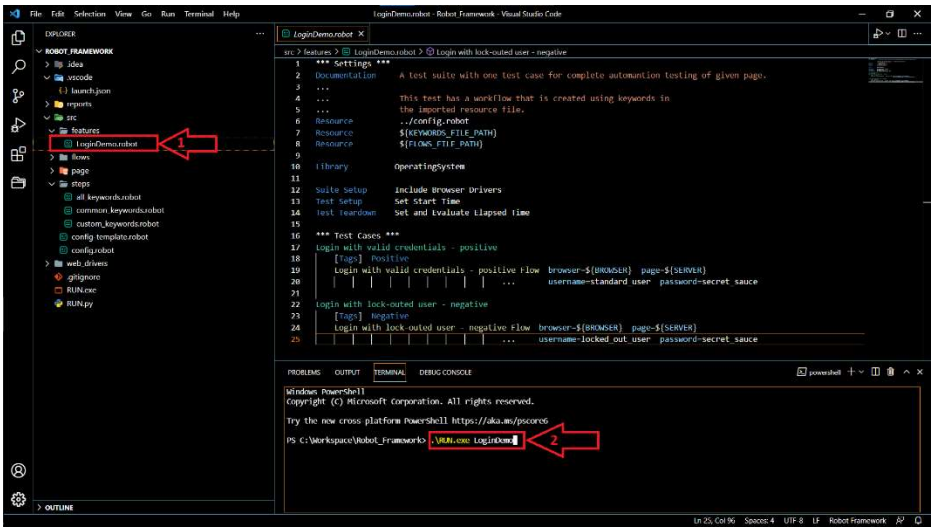
- Terminal will pop-up in bottom part of the screen



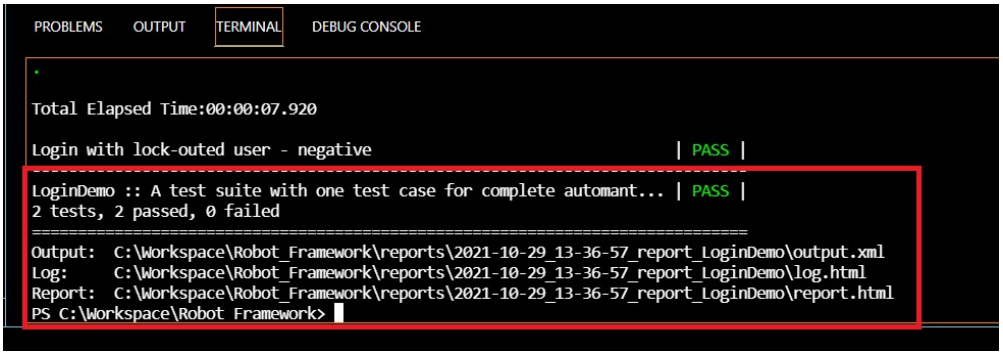
- Run script with command *RUN.exe "name of feature.robot file we want to run"* we can find feature file in *src/features* folder of project (1) in our case command will look like this

RUN.exe LoginDemo

  - and press Enter.

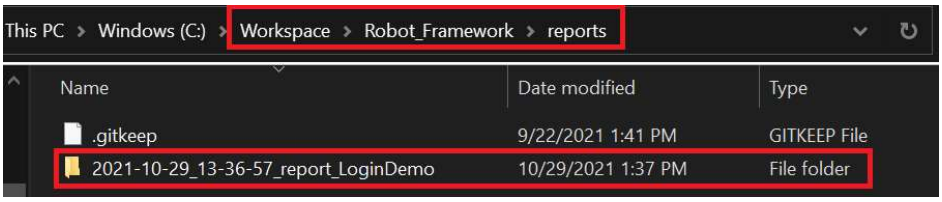


- Sample test (LoginDemo.robot) will run. You will see the following scenario output in the console.



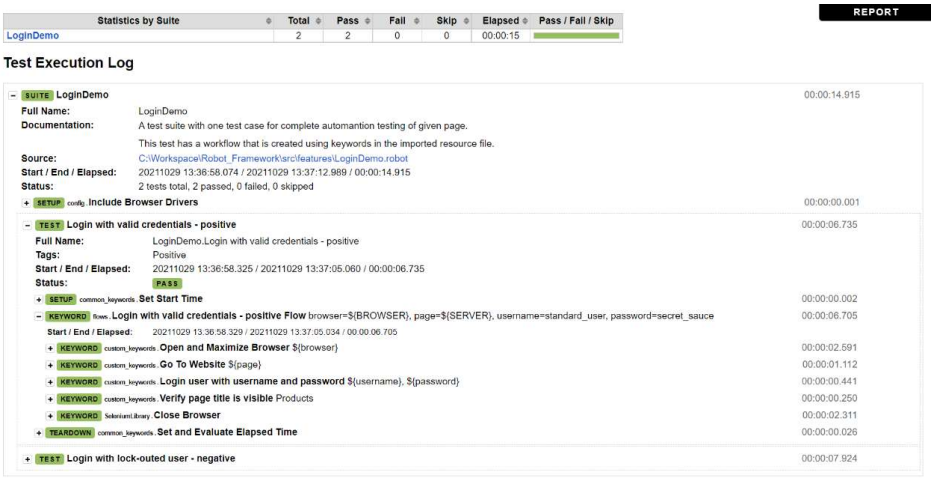
- In Windows explorer, navigate to *report* folder in *robot framework* project (e.g., *C:\Workspace\Robot_Framework\reports* for Windows users, */home/workspace/Robot_Framework/reports* for Linux and macOS users).

- You will see the HTML report folder for the test you ran.



- Open the folder and run *log.html* HTML Document.

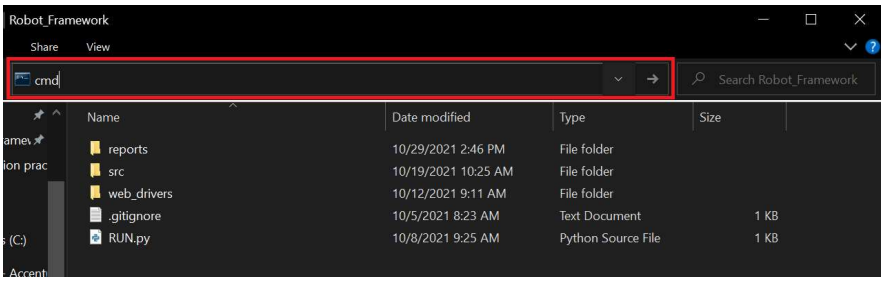
  - The HTML report opens. Screenshots and report features are visible.



7.4 Run test in external console

- Navigate into your robot framework folder In Windows explorer (e.g., *C:\Workspace\Robot_Framework* for Windows users, */home/workspace/Robot_Framework* for Linux and macOS users).

- Type cmd into address bar and press Enter

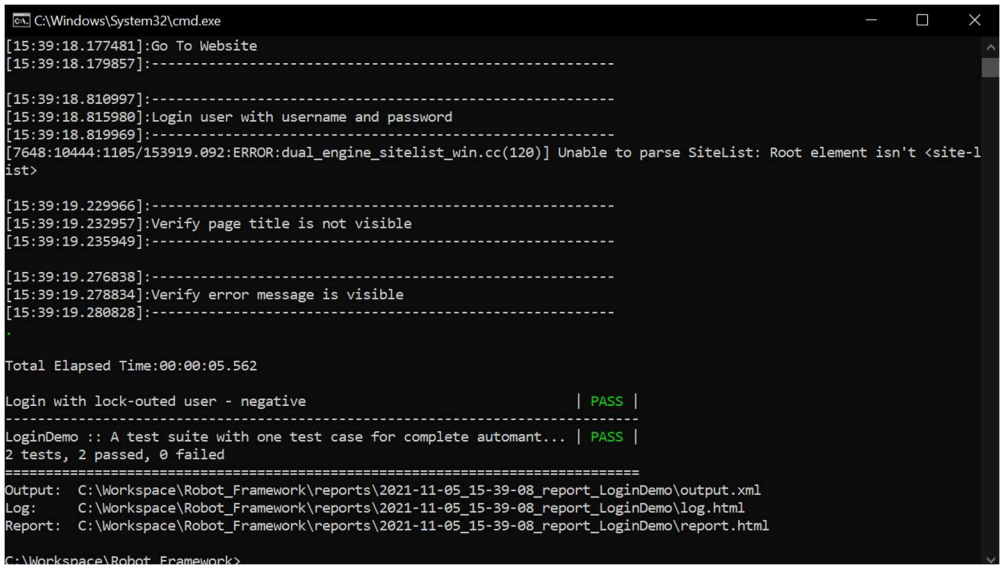


- Command Prompt will open looking like this.

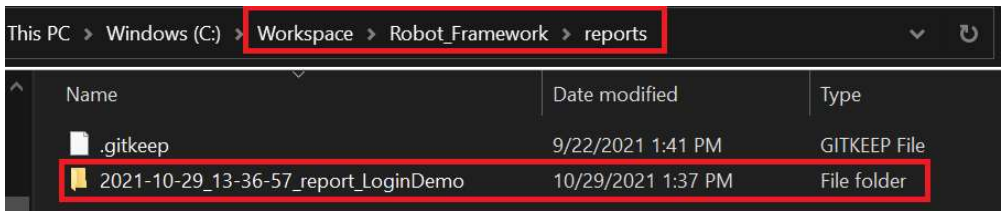
- Run script with command *RUN.exe "name of feature.robot file we want to run"* we can find feature file in *src/features* folder of project, in our case command will look like this

RUN.exe LoginDemo

- and press Enter.
- Sample test (LoginDemo.robot) will run. You will see the following scenario output in the console.



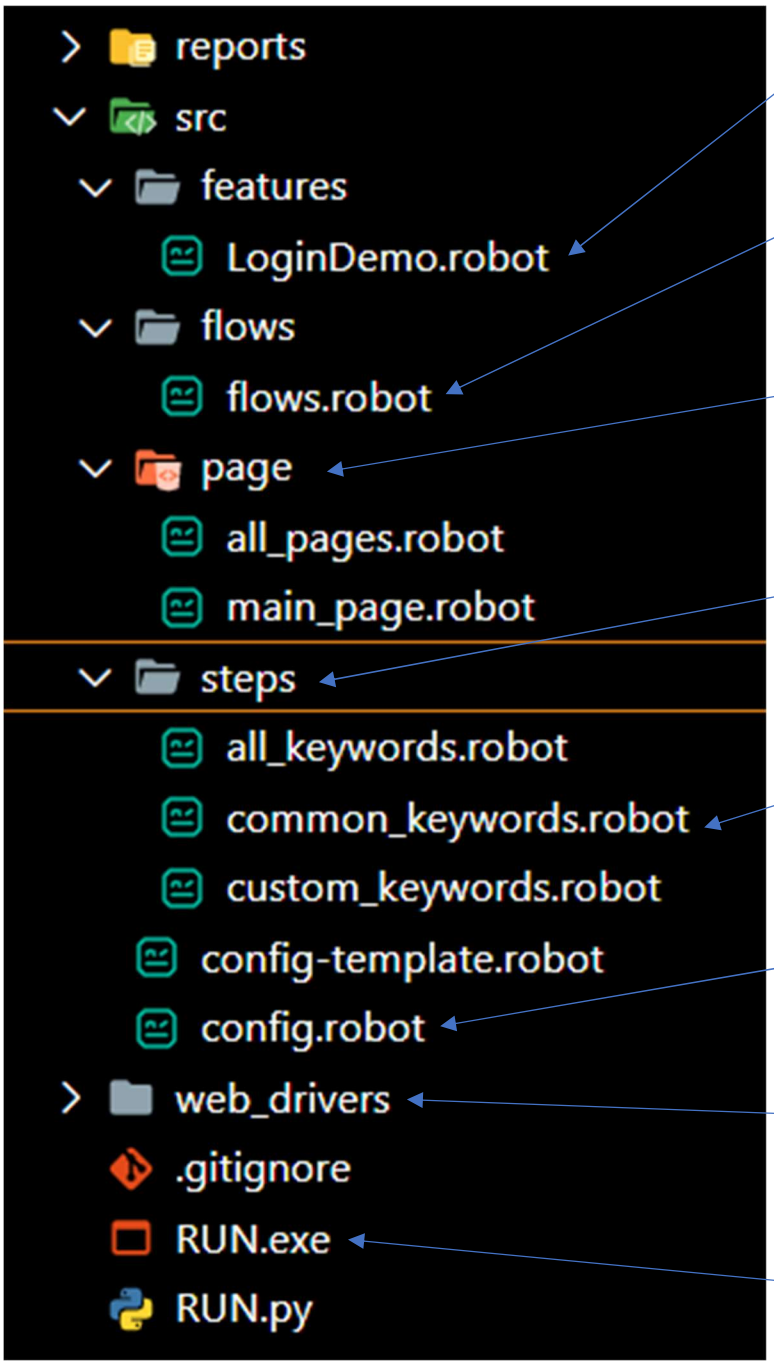
- In Windows explorer, navigate to *report* folder in *robot framework* project (e.g., *C:\Workspace\Robot_Framework\reports* for Windows users, */home/workspace/Robot_Framework/reports* for Linux and macOS users).
  You will see the HTML report folder for the test you ran.



- Open the folder and run *log.html* HTML Document.

- The HTML report opens. Screenshots and report features are visible.

8 Robot Framework project structure



features - file, where all of your scenarios and parameters (arguments) are located. This file is executed.

flows - contains defined order of steps in scenario

page - contains variables with definition of page elements (xpaths) - page object model (POM). *All_pages.robots* is collector of other page files.

steps – contains step definition. It maps the test case steps in the flows file. *All_keywords.robot* is collector file of other step file in case of multiple step files

Common_keywords.robot - contains predefined actions/operations (screenshots, command line logs)

config.robot - used to store project configuration data or settings

web_drivers – folder, used to locate your web drivers such is Chromedriver

runner – Run.exe script is starting point for your tests and contains settings for generating report folders.

9 Robot Framework content

This version of Robot Framework is an upgraded version of robot framework added for some useful features. For detailed Robot documentation, please visit official Robot website: https://robotframework.org

9.1 Features

Features is folder, where all of your feature. robot files are located.

Feature File is an entry point to the Robot Framework tests. Here you will call your flow with arguments you want to run test with. A feature file can contain one or more called flows.

It consists of following components:

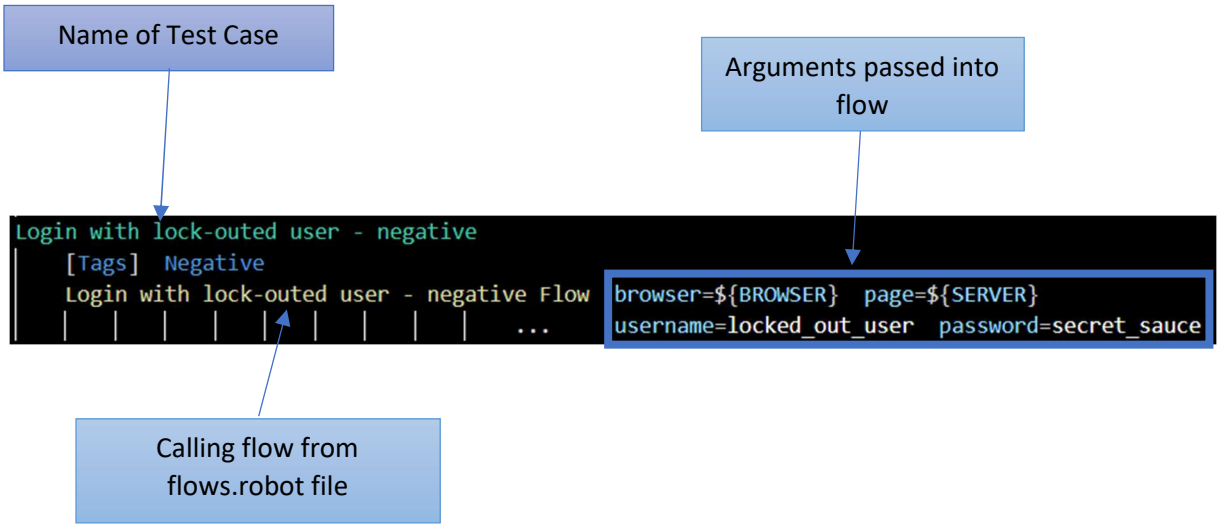
- ***Feature***
- ***Arguments***
- ***Tags***

Feature would describe the current test flow with arguments which will be executed.

Arguments are data that are passed to flow, and test is run with them. They are defined next to the flow you want to execute. You can use several data (e.g., usernames and passwords) if you want to, you just need to call the same flow again but this time with different entry data. Data are next to called flow and separated by TAB. If we want to continue write our input data on next line, we need to specify with … that new line is continuation of line before. Again, data on this new line are separated by TAB. We can specify them by names defined in flow file for better readability. For example, one for *locked_out_user*, second for *problem_user* and third for *performance_glitch_user*. It can look like the following:

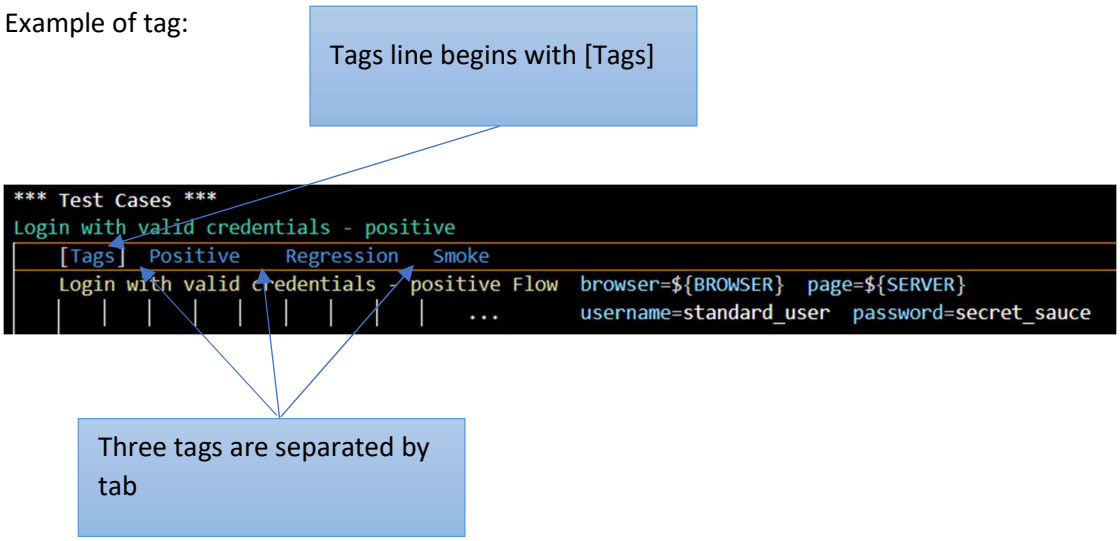
```
*** Test Cases ***
Login with valid credentials - positive
    [Tags]  Positive
    Login with valid credentials - positive Flow  browser=${BROWSER}  page=${SERVER}
    |   |   |   |   |   |   |   |   |   ...       username=standard_user  password=secret_sauce
    |
    Login with valid credentials - positive Flow  browser=${BROWSER}  page=${SERVER}
    |   |   |   |   |   |   |   |   |   ...       username=performance_glitch_user  password=secret_sauce

Login with lock-outed user - negative
    [Tags]  Negative
    Login with lock-outed user - negative Flow  browser=${BROWSER}  page=${SERVER}
    |   |   |   |   |   |   |   |   |   ...     username=locked_out_user  password=secret_sauce
```

As you can see, we have two test cases, “Login with valid credentials - positive” has two identical flows with different usernames as input parameters. Our second test case has a different flow with different usernames.



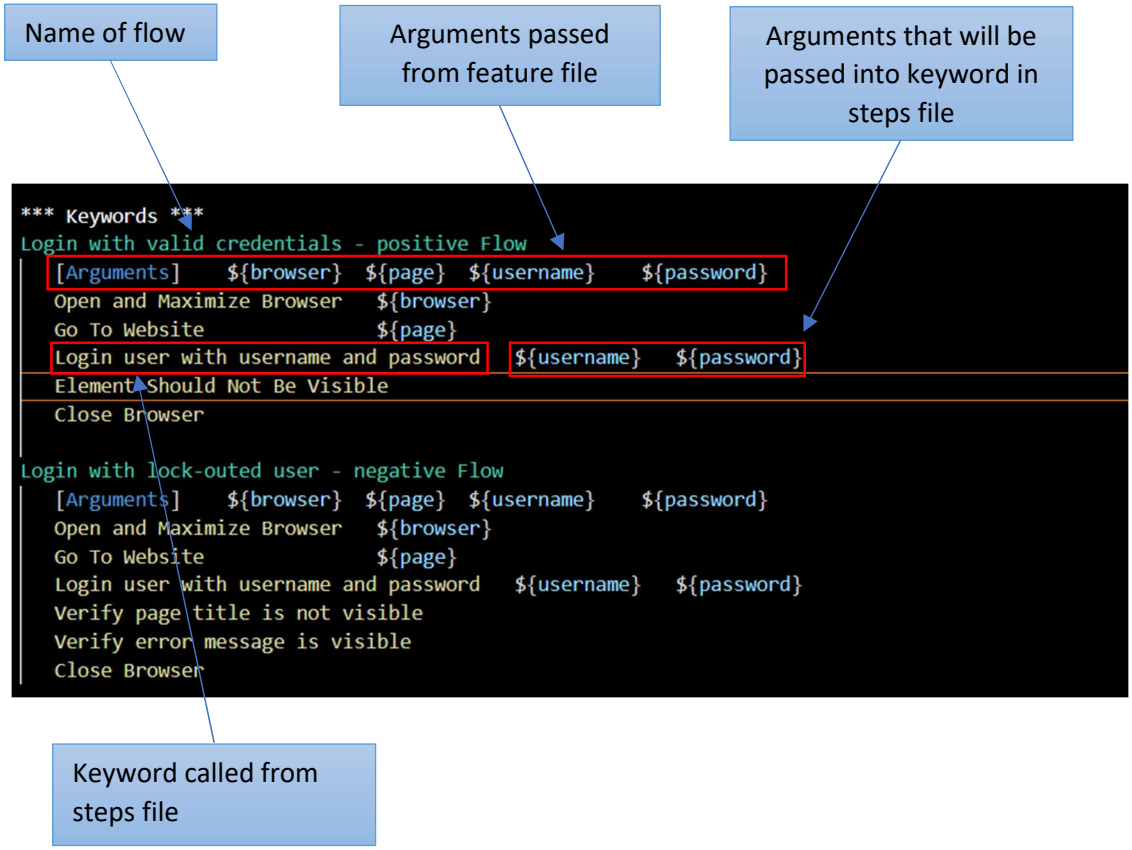
Tags are the special part of feature file**.** In Robot Framework, they are used to categorize a test with particular scenario (e.g., smoke, regression, etc.). Tags are an optional feature. You don't have to use them, but you can also use more tags in one scenario.

Example of tag:



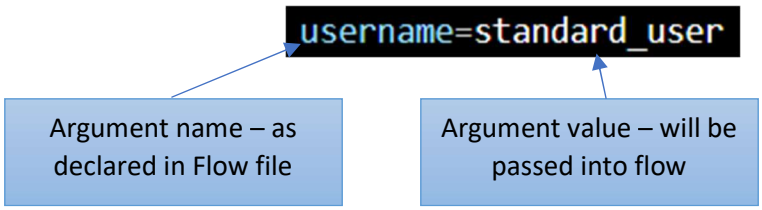
9.2 Flows

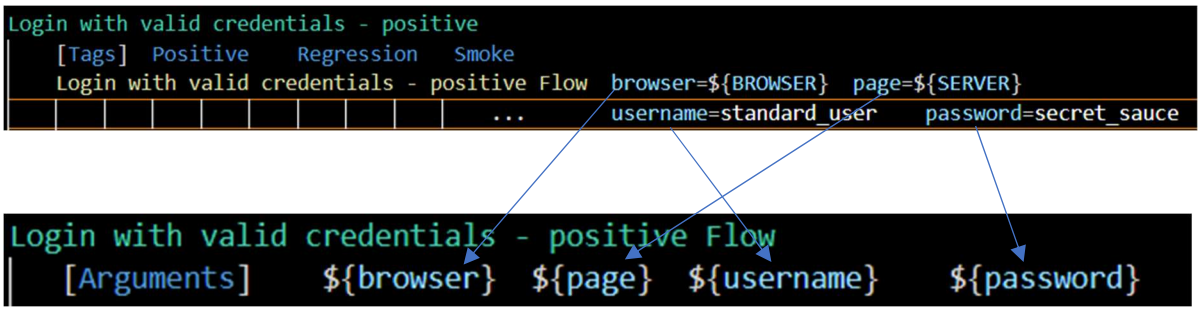
Flows. robot file contains our test flows (order of steps with input parameters). This is where input data from feature file is passed to. There can be multiple flows in one file. However, there can only be one flows file. Below you can see a flow file with two flows in it.



Name of flow - name by which we can later call it in our feature file.

Arguments – first we need to define what argument flow should be expecting when we call him. We do that by specifying line with [Arguments] descriptor and then we define names for all arguments that we will get from our feature file. Later these arguments are passed further into specific keywords if needed. Below is shown how are arguments passed from feature into flow file.



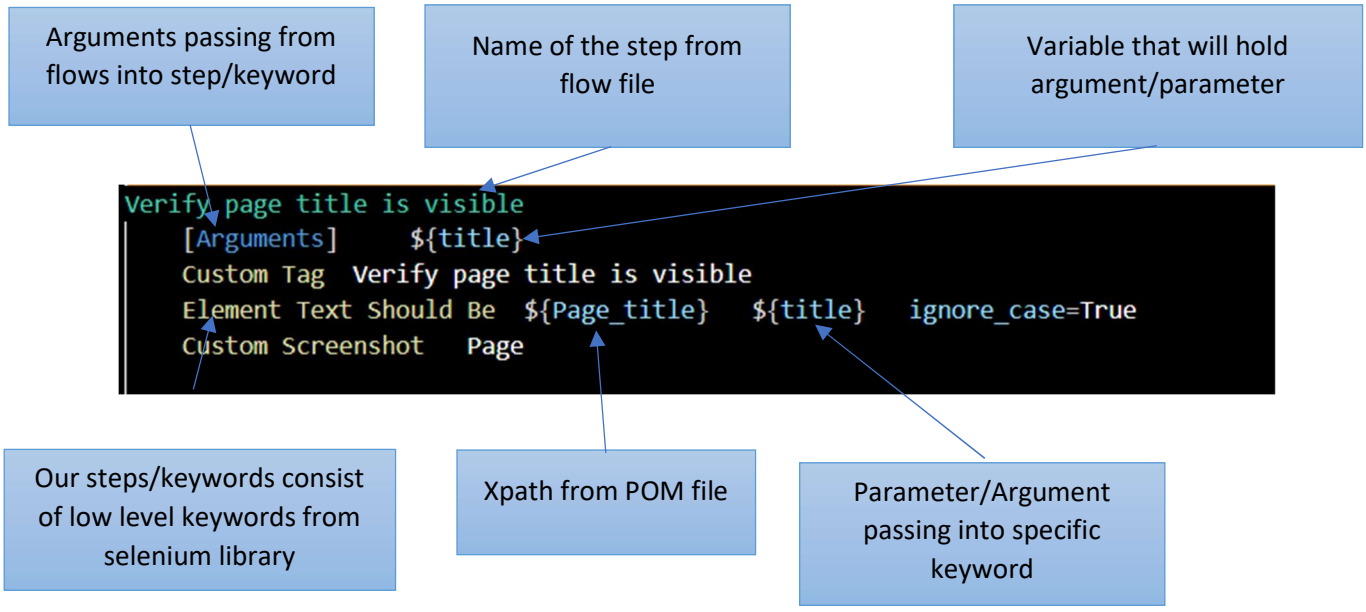


Keywords called from steps folder. Entire flow consist of called keyword in order we want them to execute.

9.3 Steps

Step definition maps the flow steps in the flow file . Name of the step in flow file has to equal the name in the steps file. Step definition itself contains the actual code to execute the test scenario in the feature file. The code is composed mostly of steps from selenium library.

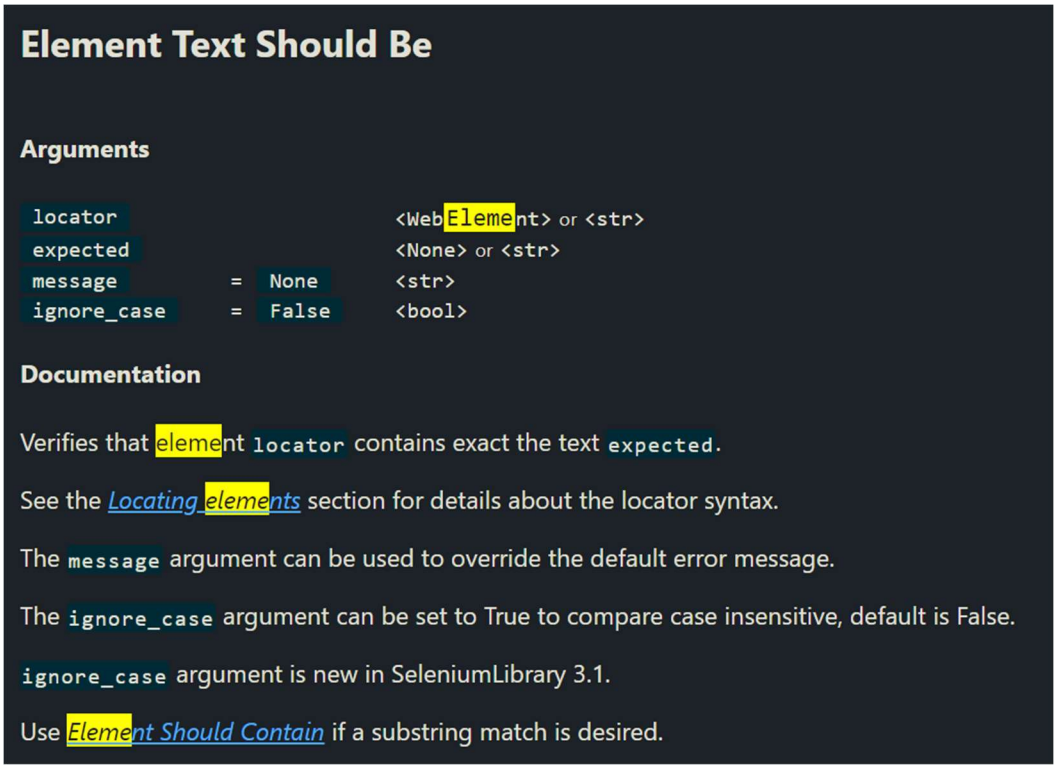
Sample step definition example:



Arguments **-** Data passed into steps from flow file. Later used as parameter for specific selenium keyword.

Custom Tag , Custom Screenshot – Custom keywords defined in common_keywords.robot file. More about them can be found in chapter 12

Keyword Parameters **-** Parameters passed into specific keyword , can be Xpath, text, or many others, details about what parameter keyword need can be found in documentation of selenium library. If order of parameters is preserved as in documentation there is no need to specify which parameter is which, however if order is for some reason not preserved you need to specify what is parameter you are passing. Example:



Element Text Should Be

Arguments

locator			<WebElement> or <str>
expected			<None> or <str>
message	=	None	<str>
ignore_case	=	False	<bool>

Documentation

Verifies that element `locator` contains exact the text `expected`.

See the *Locating elements* section for details about the locator syntax.

The `message` argument can be used to override the default error message.

The `ignore_case` argument can be set to True to compare case insensitive, default is False.

`ignore_case` argument is new in SeleniumLibrary 3.1.

Use *Element Should Contain* if a substring match is desired.

If keyword is written like this – ***Element Text Should Be ${Xpath_example} ${text_example}*** there is no need to specify argument name, but when order is different I must be specified

Element Text Should Be expected=${text_example} locator=${Xpath_example}.

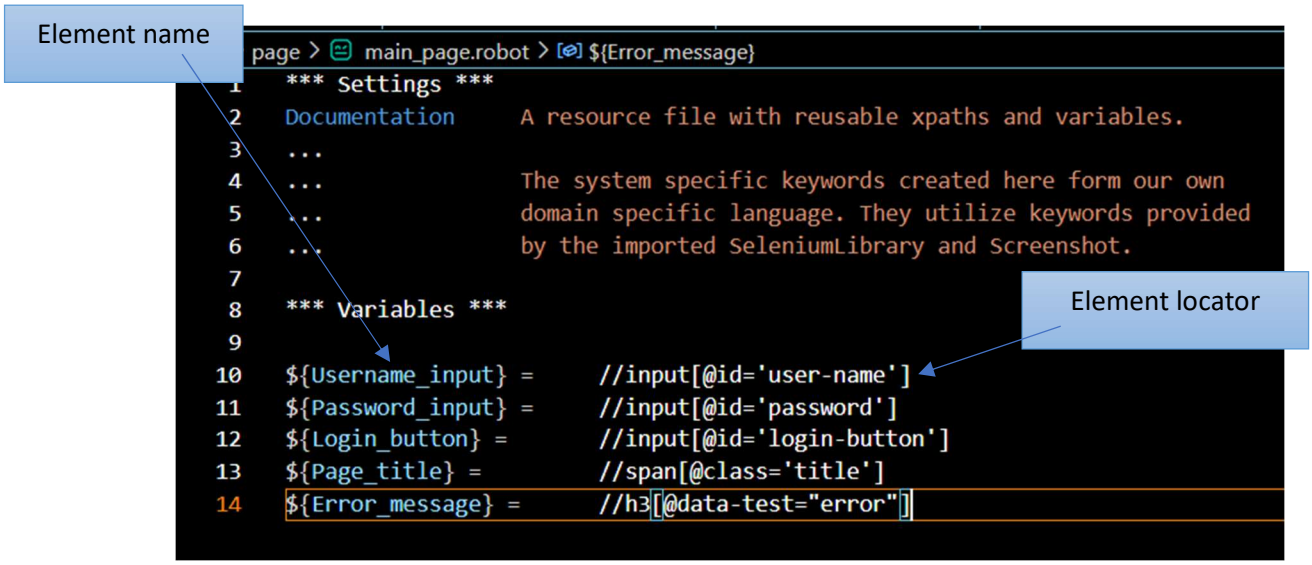
Only exception to this are arguments in Feature file, where name of argument is always written for better overall readability.

All_keywords.robot – Is file that collect all other steps files, every time you will create new step file is important to add it as resource into All_keywords.robot . If not done framework will not be able to use keywords from that file.

9.4 Page/POM

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

POM example:

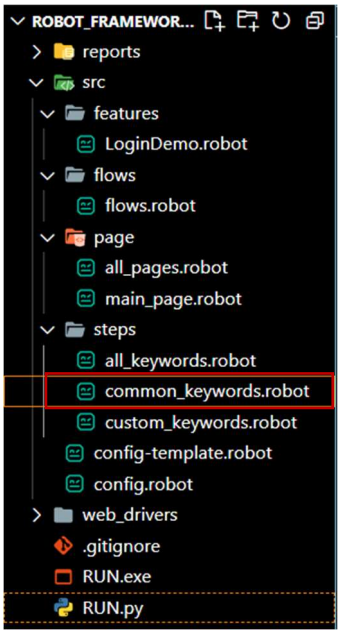


As you can see in the example, there are some patterns witch is well adhere to.

Element name:
Snake case with first letter capital, Name of element + Type of element
(e.g. Login_button, Page_title, Error_message,…)

TestStepActions.java

To add some useful features into Robot Framework ***Common_keywords.robot*** was created. It is located in steps package:



For detailed information about keywords in *Common_keywords.robot* check chapter 12

10 HTML Reports

HTML reports are one of the simplest ways to quickly see the test results and share it with others. Default reports in *Robot Framework* are generated in HTML and XML files.

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

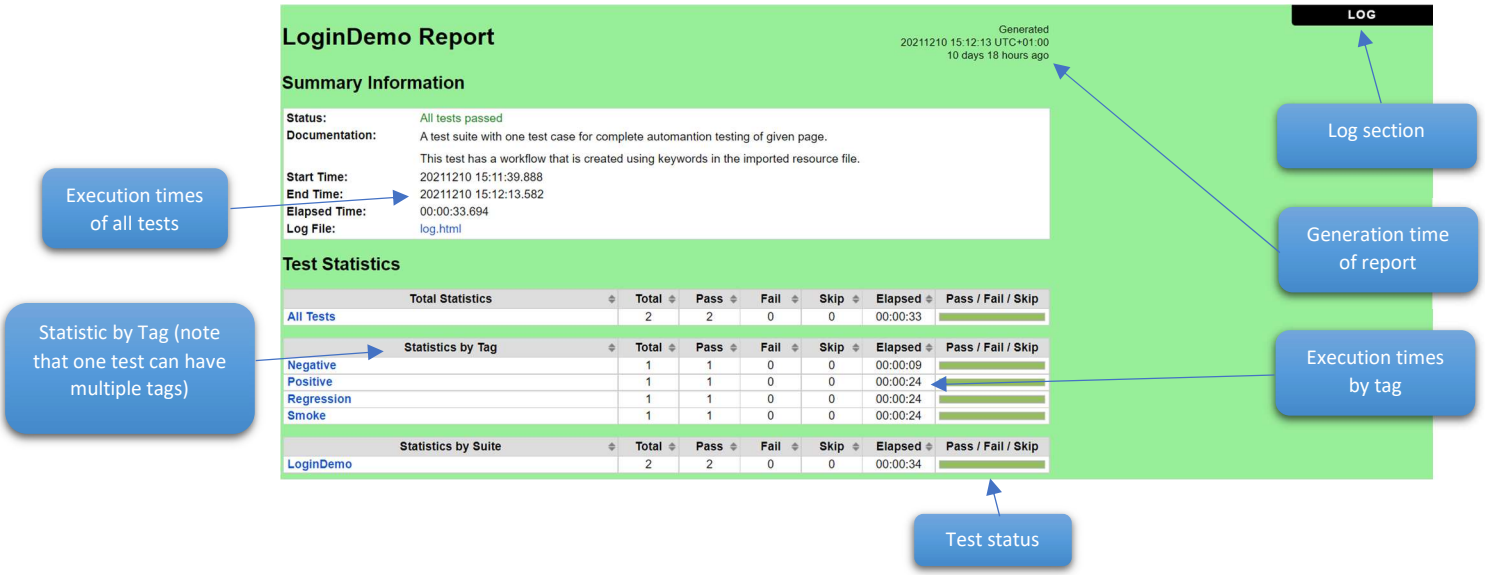
It consists of two sections:

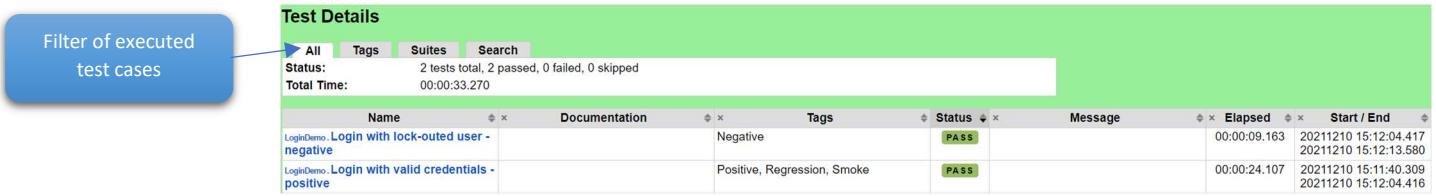
- ***Report section***
- ***Log section***

10.1 Report Section

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

You can also see some other information, e.g. when execution started and ended , from what test suite.





10.2 Log Section

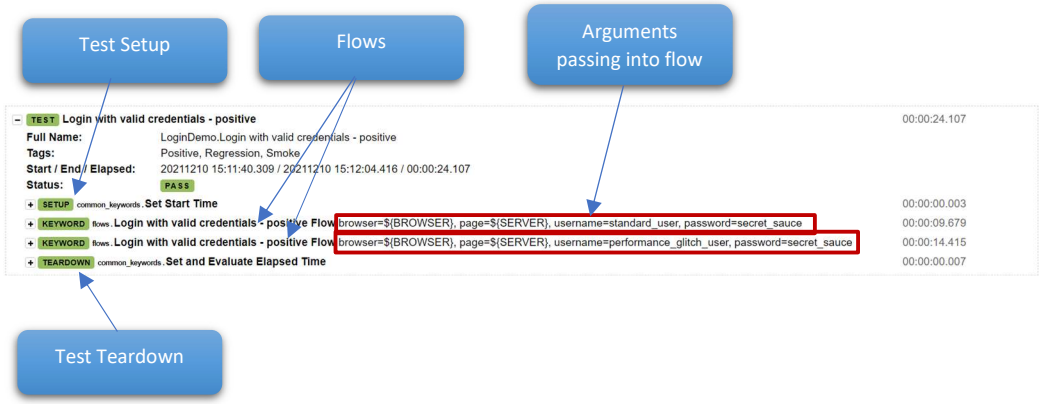
In log section with test details you can see detailed information of executed test run.

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

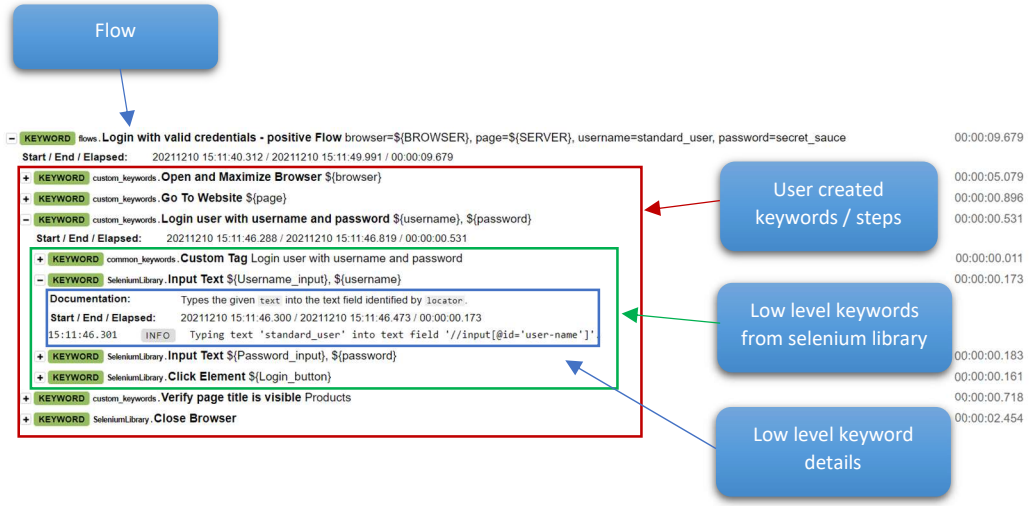
- ***Actual status*** of executed test run (Pass/Fail/Warning)
- ***Background processes*** as Suite Setup
- ***Test Cases*** - action performed over web element with more information in step details (Action, description, locator, timing)
- ***Report*** – link to report file

After clicking on Test case following will roll out:

- **Background processes** as Test Setup and Test Teardown
- **Flows –** flows executed in Test case.
- **Arguments –** arguments passed into flow.

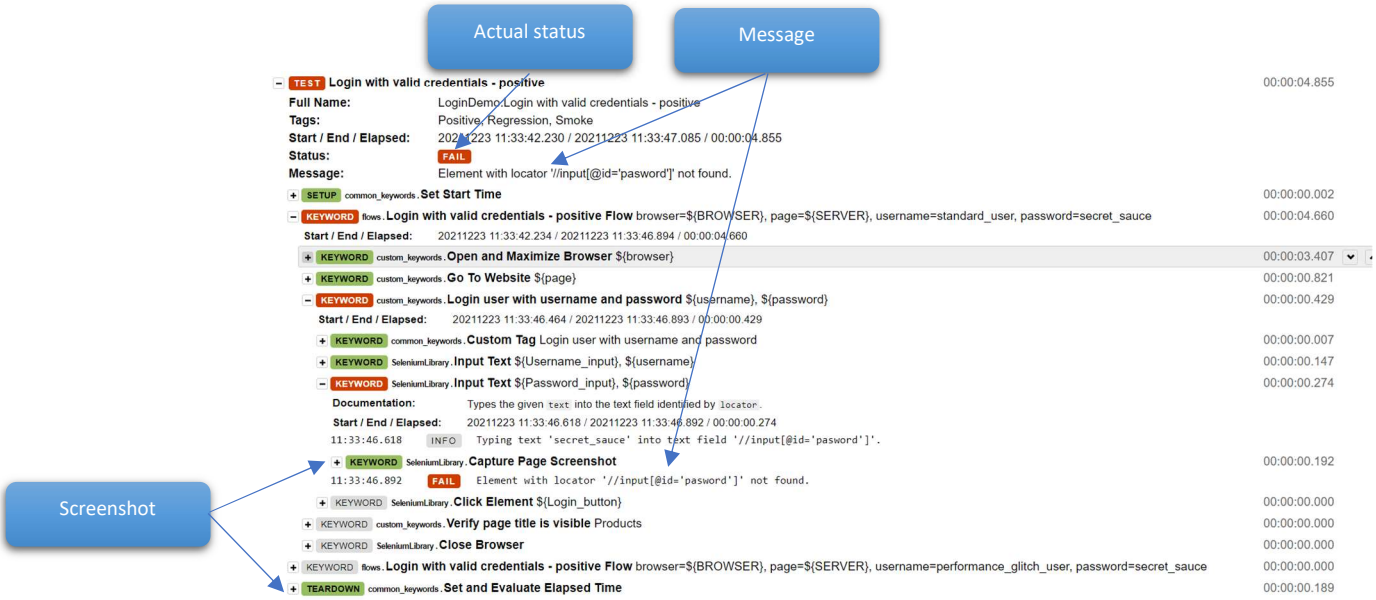


- When flow section is open , following will roll out.

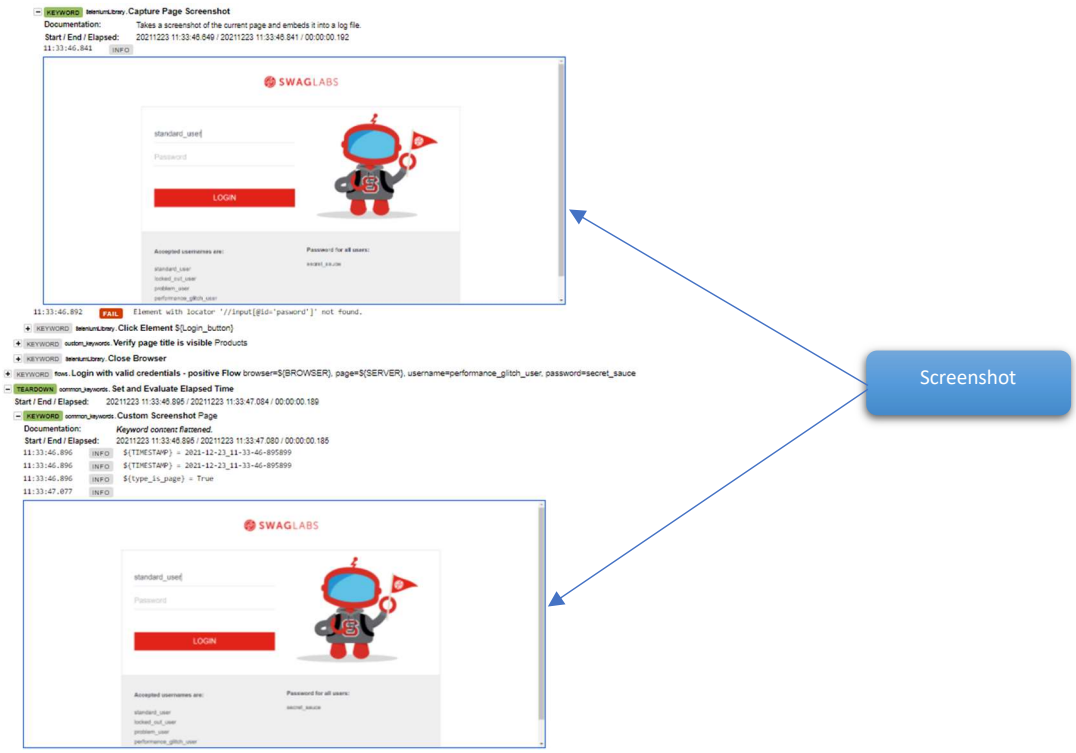


If test is failed, there are also recorded some additional information and log is automatically expanded to failed keyword:

- **Message**
- **Screenshot** - created right after error appears, so we know what exactly happened. In case that failed keyword is not from selenium library Screenshot of fail will be only in test teardown



- After expanding screenshot keywords screenshots will be seen as on picture



11 Integrations

11.1 BrowserStack

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 *Test Setup ${Mode} ${DEVICE} ${OS} ${OS_VERSION} ${BROWSER} ${BROWSER_VERSION} ${RESOLUTION} ${APPIUM_VERSION}* with mode BROWSERSTACK and required parameters as seen on picture below.

```
#Test Setup    Mode=BROWSERSTACK    DEVICE=Samsung Galaxy M32    OS_VERSION=11.0    BROWSER=chrome
Test Setup    Mode=BROWSERSTACK    OS=windows    OS_VERSION=11    BROWSER=chrome    BROWSER_VERSION=latest    RESOLUTION=1920x1080
```

- *Test Setup* keyword is defined in common_keywords.robot file

```
Test Setup
    [Arguments]    ${Mode}    ${DEVICE}=${EMPTY}    ${OS}=${EMPTY}    ${OS_VERSION}=${EMPTY}    ${BROWSER}=${EMPTY}    ${BROWSER_VERSION}=${EMPT
    # This keyword is used at the beggining of test to choose mode of browser.
    # Correct $Mode values are LOCAL ,BROWSERSTACK and SAUCELAB

    Run Keyword If    '${Mode}'=='LOCAL'          Open and Maximize Browser    ${BROWSER}
    ...    ELSE IF    '${Mode}'=='BROWSERSTACK'    Setup Browserstack    ${OS}    ${OS_VERSION}    ${DEVICE}    ${BROWSER}    ${BROWSER_VERSION}
    ...    ELSE IF    '${Mode}'=='SAUCELAB'        Setup Saucelab    ${OS}    ${OS_VERSION}    ${DEVICE}    ${BROWSER}    ${BROWSER_VERSION}
    ...    ELSE       Log       testSetup went wrong. Check the value of the variable
```

- In *feature file* in your test case call *Test Setup* keyword with mode BROWSERSTACK and wanted parameters like on pictures below.
    - Desktop version

```
Test Setup    Mode=BROWSERSTACK    OS=windows    OS_VERSION=11    BROWSER=chrome    BROWSER_VERSION=latest    RESOLUTION=1920x1080
```

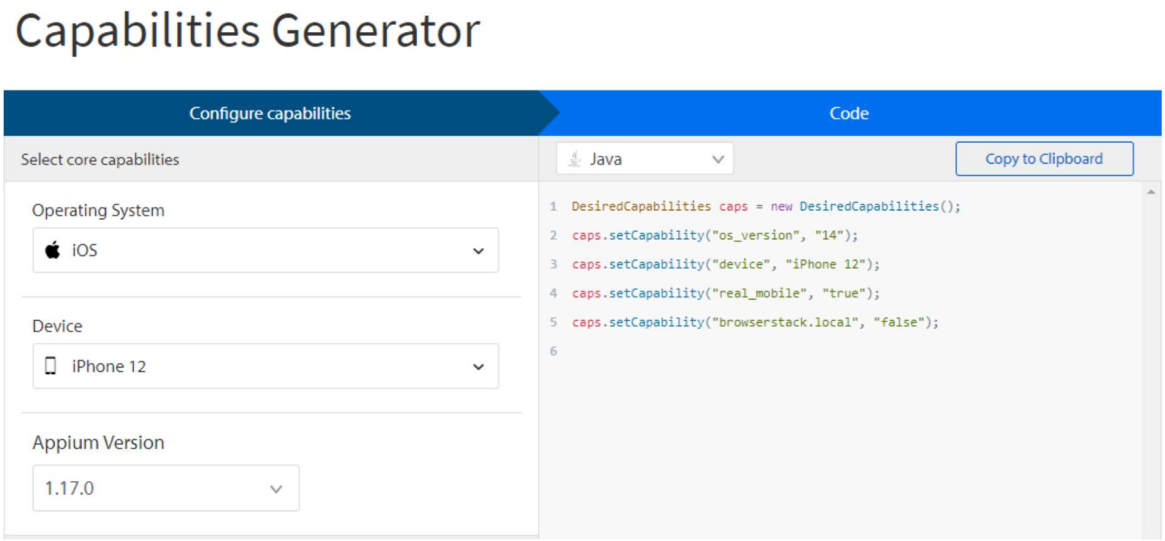
    - Mobile version

```
Test Setup    Mode=BROWSERSTACK    DEVICE=Samsung Galaxy M32    OS_VERSION=11.0    BROWSER=chrome
```

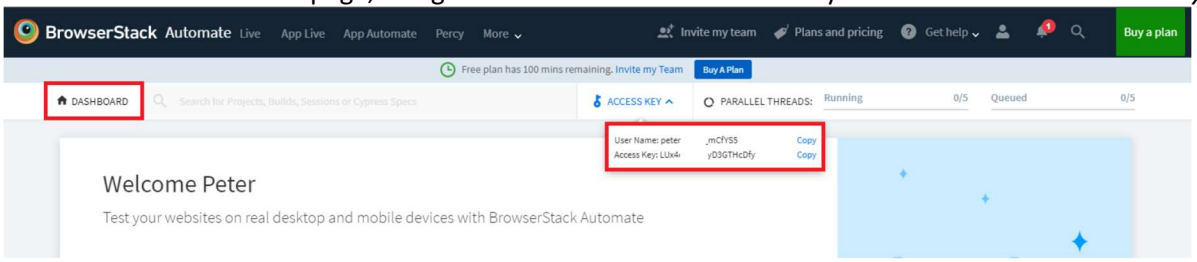
- In *Keywords | common_keywords.robot* in *Setup Browserstack* keyword you can see all possible parameters. There are different parameters for desktop app and mobile app.

```
Setup Browserstack
    [Arguments]    ${OS}    ${OS_VERSION}    ${DEVICE}    ${BROWSER}    ${BROWSER_VERSION}    ${RESOLUTION}    ${APPIUM_VERSION}
```

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



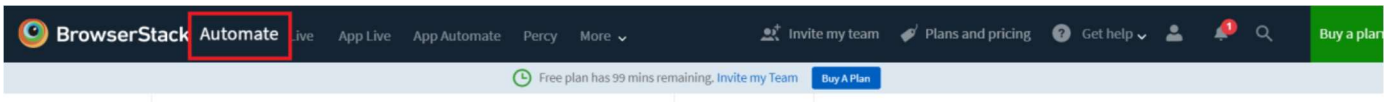
- Create your BrowserStack account. https://www.browserstack.com/users/sign_up
- On BrowserStack page, navigate to Dashboard and check for your *User Name* and *Access Key.*



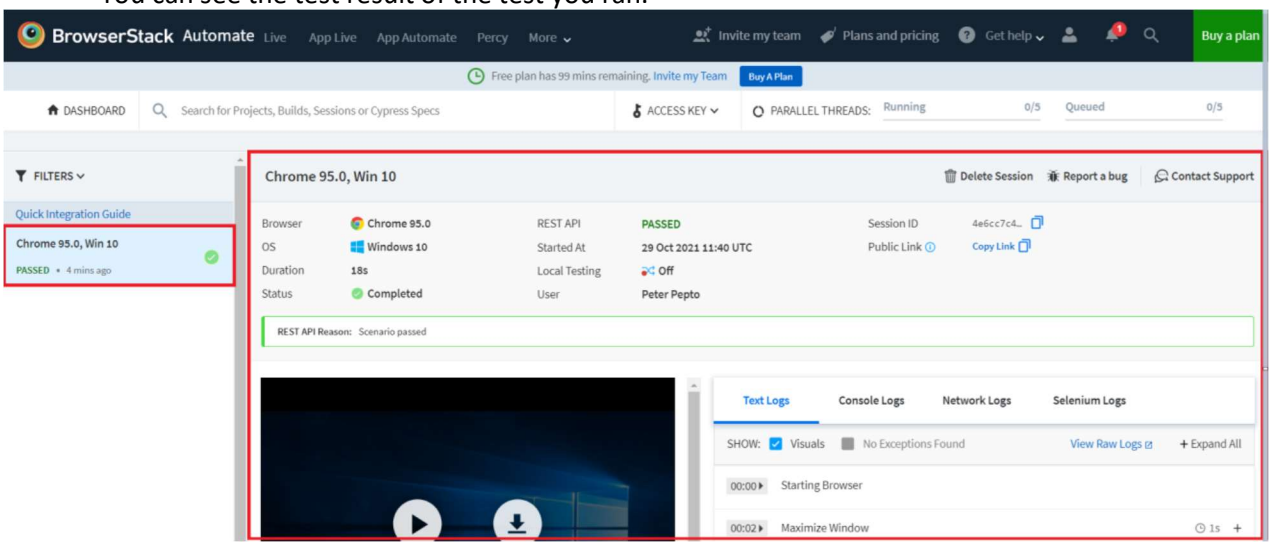
- In config.properties you have to set *${BROWSERSTACK_USERNAME}* (your BrowserStack *User Name*) and *${BROWSERSTACK_ACCESS_KEY}* (your BrowserStack *Access Key*).

```
#    Browserstack settings

${BROWSERSTACK_USERNAME}        userMarek_Ao0DZR
${BROWSERSTACK_ACCESS_KEY}      Qhdghjs53GDYUgjhoypsXn
```

- 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.



11.2 Saucelab

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 *Test Setup ${Mode} ${DEVICE} ${OS} ${BROWSER} ${BROWSER_VERSION} ${RESOLUTION} ${APPIUM_VERSION}* with mode SAUCELAB and required parameters as seen on picture below.

```
#Test Setup    Mode=SAUCELAB    OS=Android    OS_VERSION=12.0    BROWSER=Chrome    DEVICE=Google Pixel 6 GoogleAPI Emulator
Test Setup     Mode=SAUCELAB    OS=Windows 11    BROWSER=Chrome    BROWSER_VERSION=104
```

- *Test Setup* keyword is defined in common_keywords.robot file

```
Test Setup
    [Arguments]    ${Mode}    ${DEVICE}=${EMPTY}    ${OS}=${EMPTY}    ${OS_VERSION}=${EMPTY}    ${BROWSER}=${EMPTY}    ${BROWSER_VERSION}=${EMPT
    # This keyword is used at the beggining of test to choose mode of browser.
    # Correct $Mode values are LOCAL ,BROWSERSTACK and SAUCELAB

    Run Keyword If    '${Mode}'=='LOCAL'          Open and Maximize Browser    ${BROWSER}
    ...     ELSE IF   '${Mode}'=='BROWSERSTACK'   Setup Browserstack   ${OS}    ${OS_VERSION}    ${DEVICE}    ${BROWSER}    ${BROWSER_VERSION}
    ...     ELSE IF   '${Mode}'=='SAUCELAB'       Setup Saucelab     ${OS}    ${OS_VERSION}    ${DEVICE}    ${BROWSER}    ${BROWSER_VERSION}
    ...     ELSE      Log       testSetup went wrong. Check the value of the variable
```

- In *feature file* in your test case call *Test Setup* keyword with mode SAUCELAB and wanted parameters like on pictures below.
    - Desktop version

```
Test Setup    Mode=SAUCELAB    OS=Windows 11    BROWSER=Chrome    BROWSER_VERSION=104
```

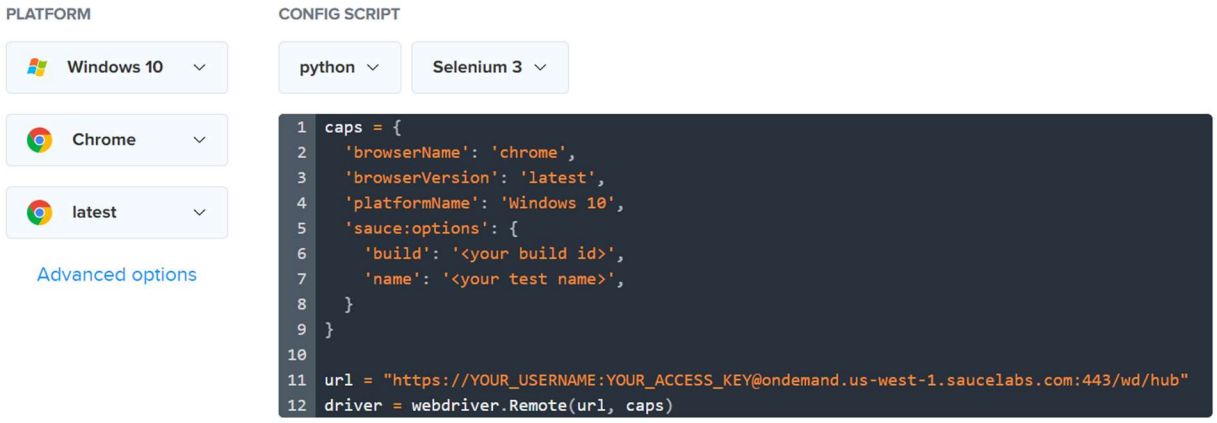
    - Mobile version

```
Test Setup    Mode=SAUCELAB    OS=Android    OS_VERSION=12.0    BROWSER=Chrome    DEVICE=Google Pixel 6 GoogleAPI Emulator
```

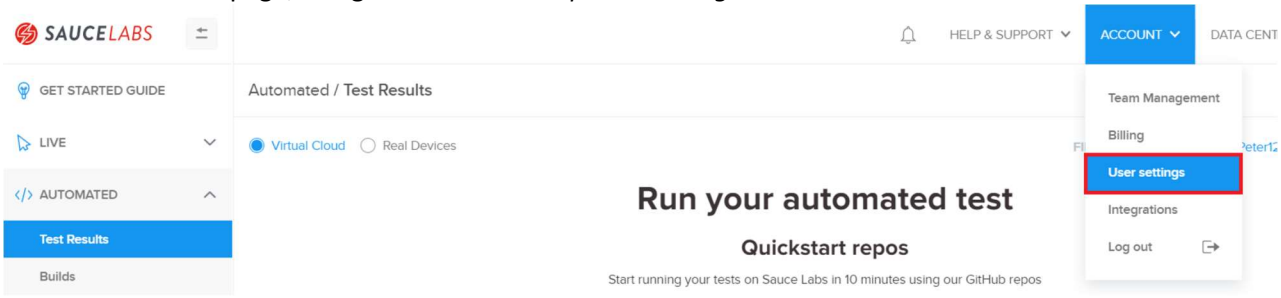
- In *Keywords | common_keywords.robot* in *Setup Saucelab* keyword you can see all possible parameters. There are different parameters for desktop app and mobile app.

```
Setup Saucelab
    [Arguments]    ${OS}    ${OS_VERSION}    ${DEVICE}    ${BROWSER}    ${BROWSER_VERSION}    ${RESOLUTION}    ${APPIUM_VERSION}
```

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

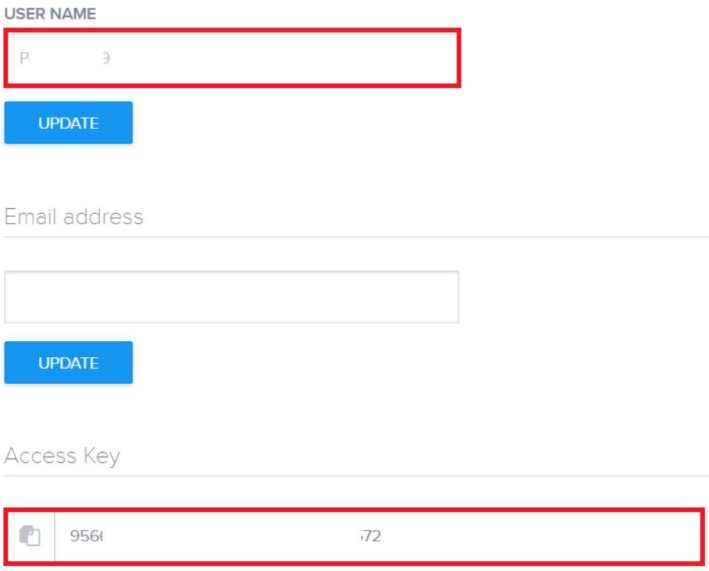


- Create your Sauce Labs account https://saucelabs.com/sign-up.
- On Sauce Labs page, navigate to *ACCOUNT | User settings.*



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

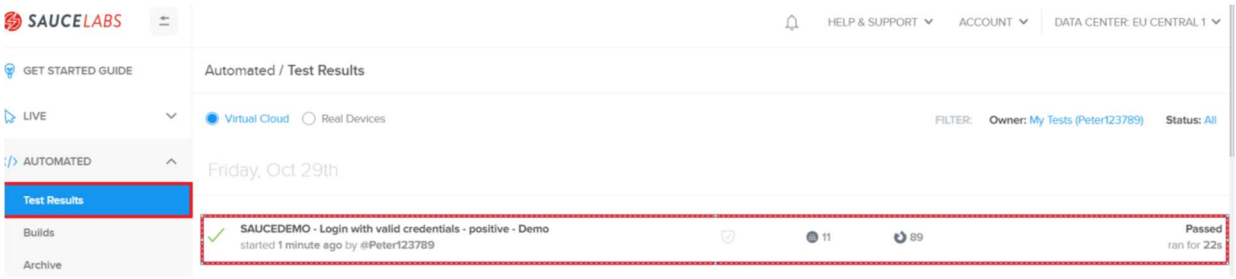
-



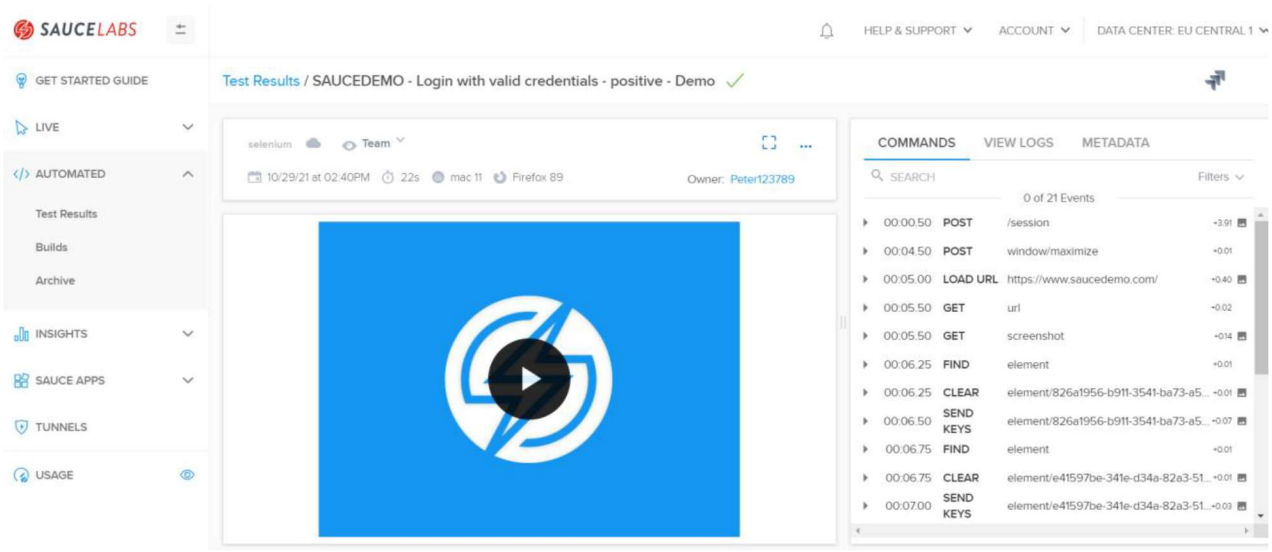
- In config.robot you have to set SAUCELAB_USERNAME (your Sauce Labs *USER NAME*) and SAUCELAB_ACCESS_KEY (your Sauce Labs *Access Key*).

```
#    Saucelab Credentials

${SAUCELAB_USERNAME}          oauchjkh57dhlsbsgsd4563d
${SAUCELAB_ACCES_KEY}         486d864f8-f4d56h-5ru8-b358-475464628e3b
```

- 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.


SAUCELABS
HELP & SUPPORT
ACCOUNT
DATA CENTER: EU CENTRAL 1
GET STARTED GUIDE
LIVE
AUTOMATED
Test Results
Builds
Archive
INSIGHTS
SAUCE APPS
TUNNELS
USAGE
Test Results / SAUCEDEMO - Login with valid credentials - positive - Demo
selenium
Team
10/29/21 at 02:40PM
22s
mac 11
Firefox 89
Owner: Peter123789
COMMANDS
VIEW LOGS
METADATA
SEARCH
Filters
0 of 21 Events
00:00.50 POST /session
00:04.50 POST window/maximize
00:05.00 LOAD URL https://www.saucedemo.com/
00:05.50 GET url
00:05.50 GET screenshot
00:06.25 FIND element
00:06.25 CLEAR element/826a1956-b911-3541-ba73-a5...
00:06.50 SEND KEYS element/826a1956-b911-3541-ba73-a5...
00:06.75 FIND element
00:06.75 CLEAR element/e41597be-341e-d34a-82a3-51...
00:07.00 SEND KEYS element/e41597be-341e-d34a-82a3-51...

12 First Test

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

1. **Add URL to *config.robot***
2. ***Create unique feature file , if already created add new test case***
3. ***Add new flow into the flow file***
4. ***Create unique Page file***
5. ***Create Step file with definitions***

12.1 Add URL to config.robot

- Create new record in config.robot and add URL required for the test

```
#   Global Variable with site you want to test. (URL site)
#   Feel free to add more URL sites
${SAUCEDEMO}            https://www.saucedemo.com/
${GOOGLE_EXAMPLE}       https://www.google.com/
${RAHUL}                https://www.rahulshettyacademy.com/AutomationPractice/
```

12.2 Create unique feature file

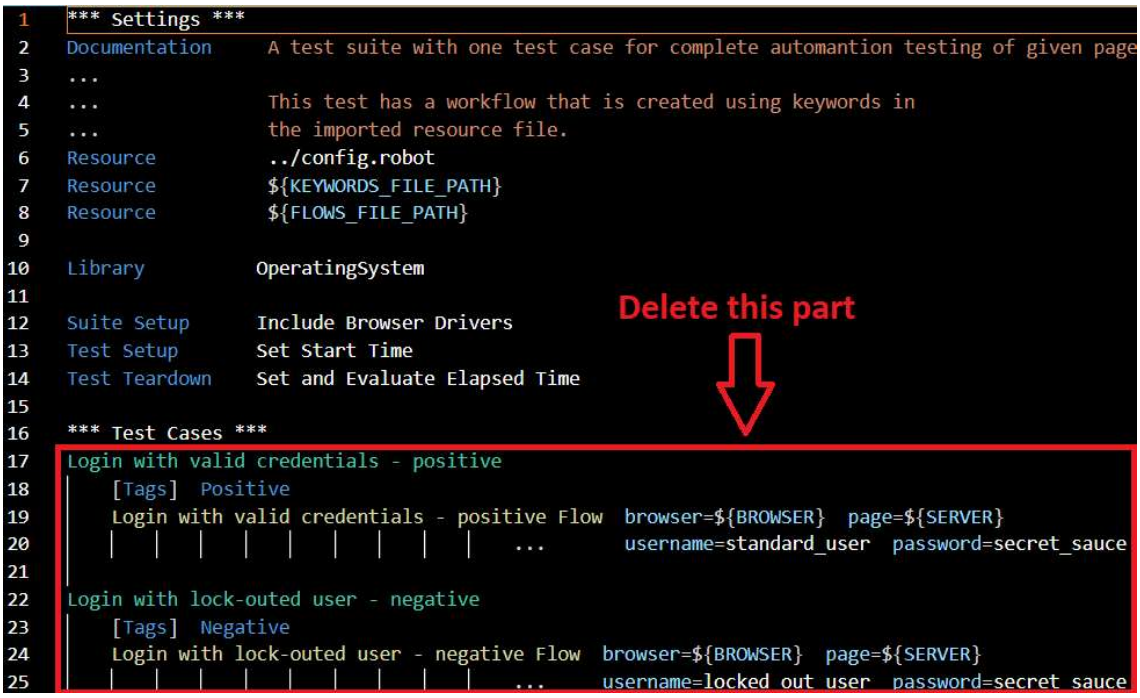
- Create copy of LoginDemo.robot feature file in ***features*** folder.

Name	Date modified	Type	Size
LoginDemo - Copy.robot	11/30/2021 9:51 AM	ROBOT File	1 KB
LoginDemo.robot	11/30/2021 9:51 AM	ROBOT File	1 KB

- Rename it with unique name for example AutomationPractice.robot.

Name	Date modified	Type	Size
AutomationPractice.robot	11/30/2021 9:51 AM	ROBOT File	1 KB
LoginDemo.robot	11/30/2021 9:51 AM	ROBOT File	1 KB

- You can delete unnecessary lines from created file and start to write the test cases and flows you will define in flows folder later. Don't forget to add new ULR as parameter, as you set in config.properties. Feature file would look like the following:




```
*** Settings ***
Documentation     A test suite with one test case for complete automantion testing of given page
...
...               This test has a workflow that is created using keywords in
...               the imported resource file.
Resource          ../config.robot
Resource          ${KEYWORDS_FILE_PATH}
Resource          ${FLOWS_FILE_PATH}

Library           OperatingSystem

Suite Setup       Include Browser Drivers
Test Setup        Set Start Time
Test Teardown     Set and Evaluate Elapsed Time

*** Test Cases ***
Login with valid credentials - positive
    [Tags]  Positive
    Login with valid credentials - positive Flow  browser=${BROWSER}  page=${SERVER}
    |   |   |   |   |   |   |   |   |   ...       username=standard_user  password=secret_sauce

Login with lock-outed user - negative
    [Tags]  Negative
    Login with lock-outed user - negative Flow  browser=${BROWSER}  page=${SERVER}
    |   |   |   |   |   |   |   |   |   ...     username=locked_out_user  password=secret_sauce
```

- Start writing test case with flow and basic arguments in it. You can always come back and add more arguments if you need.

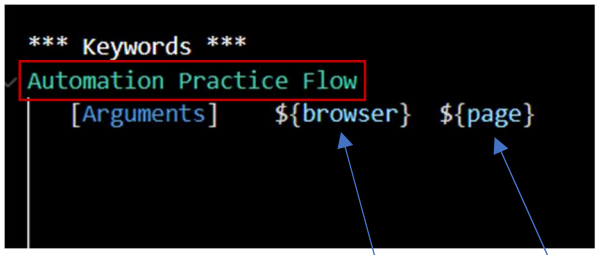
```
*** Settings ***
Documentation     A test suite with one test case for complete automantion testing of given page.
...
...               This test has a workflow that is created using keywords in
...               the imported resource file.
Resource          ../config.robot
Resource          ${KEYWORDS_FILE_PATH}
Resource          ${FLOWS_FILE_PATH}

Library           OperatingSystem

Suite Setup       Include Browser Drivers
Test Setup        Set Start Time
Test Teardown     Set and Evaluate Elapsed Time

*** Test Cases ***

Automation Practice
    [Tags]  Full-test
    Automation Practice Flow  browser=${BROWSER}  page=${RAHUL}
```

- Implement the scenario sequentially, one command 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.

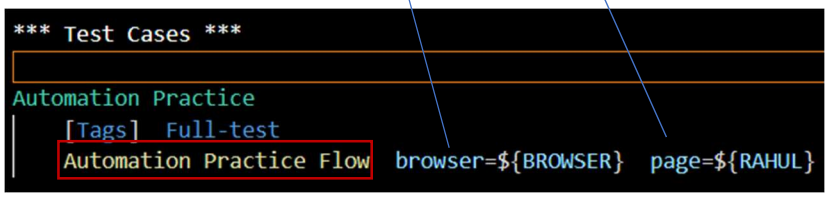
12.3 Break flow into steps in flows file

In flows.robot file we need to add our new flow and break it on smaller steps. First we write our flow header with parameters. Take notice that flow name must be exactly same as written in feature file, same with parameter names from feature file.

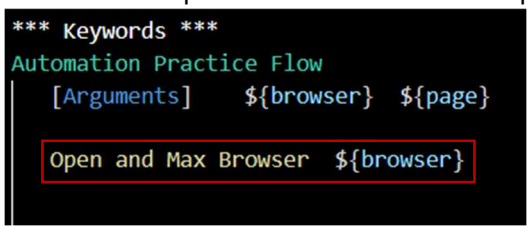
- Flows.robot



- Feature file



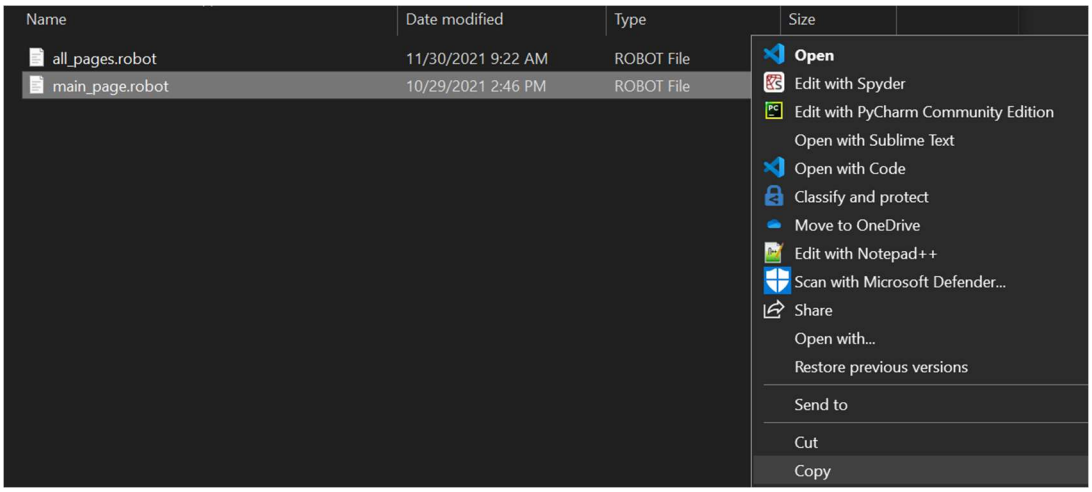
- As next step we will start to declare steps in our flow.



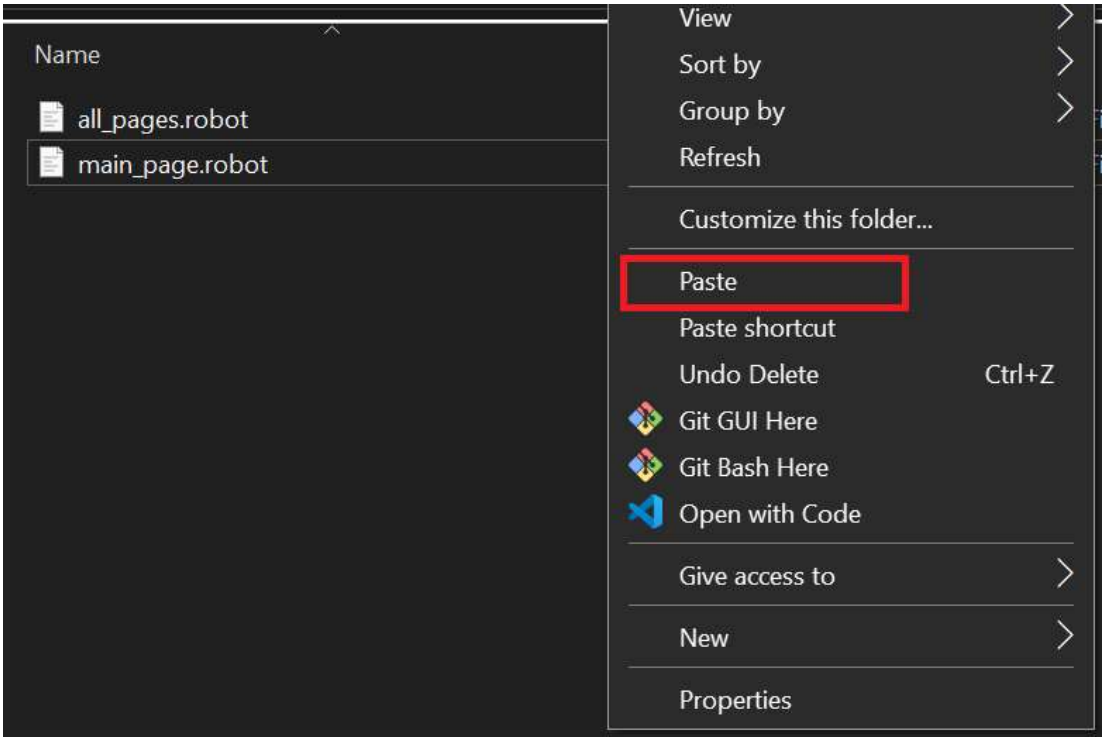
12.4 Create unique Page file

- Create unique page file in page project folder. The easiest way to do that is to copy and paste existing main_page.robot and rename it how you see fit. Don't forget that after renaming file you must also add it into all_pages.robot to be visible for framework.

- Right-click *page* file and select *copy*



- Right-click again and select *paste*



- Rename copy file, e.g. *RAHUL_AutomationPracticePage*. Always use naming convention: ***PORTALNAME*** *in upper case + _ +* ***ScenarioName*** *in camel case +* ***Page*** **(**e.g.SAUCEDEMO_LoginPage, ALZA_OrderProductPage, GOOGLE_PhoneSearchPage,…)

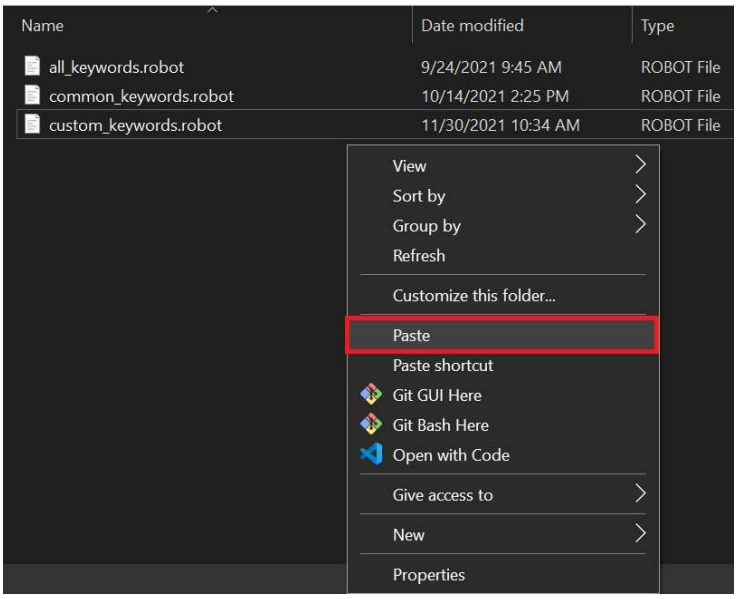
- You can now remove old xpaths and add xpath variables required for the test in step definition. More about steps in next section. Don't forget to add this new file into all_pages.robot

```
*** Variables ***

${Page_title} =     //h1
```

12.5 Create Step definition file

- **Step definition**
  - Create step definition. The easiest way to do that is to copy and paste existing file custom_keywords.robot or rewrite into existing file. Right-click existing *custom_keywords.robot* file in *steps* folder and select *Copy*.

  - Right-click anywhere in Steps folder and select *Paste*



  - Rename copy file, e.g. *RAHUL_AutomationPracticeSteps*. Always use naming convention: ***PORTALNAME*** *in upper case + _ +* ***ScenarioName*** *in camel case +* ***Steps*** **(**e.g.SAUCEDEMO_LoginSteps, ALZA_OrderProductSteps, GOOGLE_PhoneSearchSteps,…)

  - Delete all steps copied from original class. It should looks like following:

```
1   *** Settings ***
2   Documentation     A resource file with reusable keywords.
3   ...
4   ...               The system specific keywords created here form our own
5   ...               domain specific language. They utilize keywords provided
6   ...               by the imported SeleniumLibrary and Screenshot.
7   Library           SeleniumLibrary
8
9   Resource          ${CONFIG_FILE_PATH}
10  Resource          ${KEYWORDS_FILE_PATH}
11  Resource          ${POM_FILES_PATH}
12
13  *** Keywords ***
14
```

  - You can now implement your step definitions. Copy-paste the step name from flow file to step file and add needed arguments.

- Flow file

```
Resource          ${CONFIG_FILE_PATH}
Resource          ${KEYWORDS_FILE_PATH}
Resource          ${POM_FILES_PATH}

*** Keywords ***
Automation Practice Flow
    [Arguments]    ${browser}  ${page}

    Open and Max Browser  ${browser}
```

- Keyword/step file

```
*** Keywords ***
Open and Max Browser
    [Arguments]    ${browser}

    //// SELENIUM KEYWORDS COME HERE //////
```

- Now you can start to implement steps with selenium keywords found in documentation. Finished "Open and Max Browser" will look like this:

```
*** Keywords ***
Open and Max Browser
    [Arguments]     ${browser}

    Custom Tag  Open Browser
    Open Browser  browser=${browser}
    Maximize Browser Window
```

- Don't forget to add this new steps file into all_keywords.robot

```
1   *** Settings ***
2   Documentation     A resource file with reusable xpaths
3   ...
4   ...               The system specific keywords created
5   ...               domain specific language. They utili
6   ...               by the imported SeleniumLibrary and
7   #   List keywords files
8   Resource    common_keywords.robot
9   Resource    custom keywords.robot
10  Resource    RAHUL_AutomationPracticeSteps.robot
```

- Continue to write more steps into flow file and define them in steps file. For how to run tests see Chapter 7 in this guide.

13 Common_keywords.robot

Custom Screenshot ${type_of_screenshot} ${locator_of_element}=NONE

Parameters:
@param type_of_screenshot - [string] – Page, Alert or Element.
@param locator_of_element - [xpath] – default NONE, only need to specify when type of screenshot is set to element.

Description:
Takes screenshot of entire page, element or alert based on value in type_of_screenshot

Example:
Custom Screenshot Page

${Div_example} = //div[@class="example"] - in POM file

Custom Screenshot Element ${Div_example}

Custom tag ${module_name}

Parameters:
@param module_name - [string] – Name of currently executed module/keywords

Description:
Print timestamp into console for better orientation in what keyword is executed.
Best practice is to use this command at the start of every keywords in steps folder.

Example:
Custom tag Open and Maximize Browser