NIMBAL

Test Automation Made Easy

USER MANUAL OF WEB IDE

Accelerate, Improve and Plan success with an economical code

https://www.nimbal.io/

INDEX

INTRODUCTION

A **web integrated development environment** (Web IDE), also known as an Online IDE or Cloud IDE, is a browser-based IDE. **Web testing** is software testing that focuses on web applications. Complete testing of a web-based system before going live can help address issues before the system is revealed to the public.

Issues may include the security of the web application, the basic functionality of the site, its accessibility to handicapped users and fully able users, its ability to adapt to the multitude of desktops, devices, and operating systems, as well as readiness for expected traffic and number of users and the ability to survive a massive spike in user traffic, both of which are related to load testing.

PRE-REQUISITE

1. Knowledge of XPath
2. Basics of Testing concepts
3. Access to Nimbal Web IDE i.e., its URL, such as nimbal-webide.getskills.co.nz
4. Before cloning the repository on Web IDE, get ready with the following steps.
    - Create Bitbucket account
    - Clone Repository on Bitbucket
    - Create an App password for your bitbucket account
5. Install 7-Zip
6. Your browser should have the following programs installed
    a. Selector Hub : it is a helper browser extension, which gives us prebuilt XPath.
    b. ChroPath: it is an alternative extension for XPath.

A. Clone the Repository on WEB IDE

Copy the URL cloned from Bitbucket. This URL is used to clone the project in Web IDE.

1. Go to the Web IDE link provided to you through an email and open it in web browser.

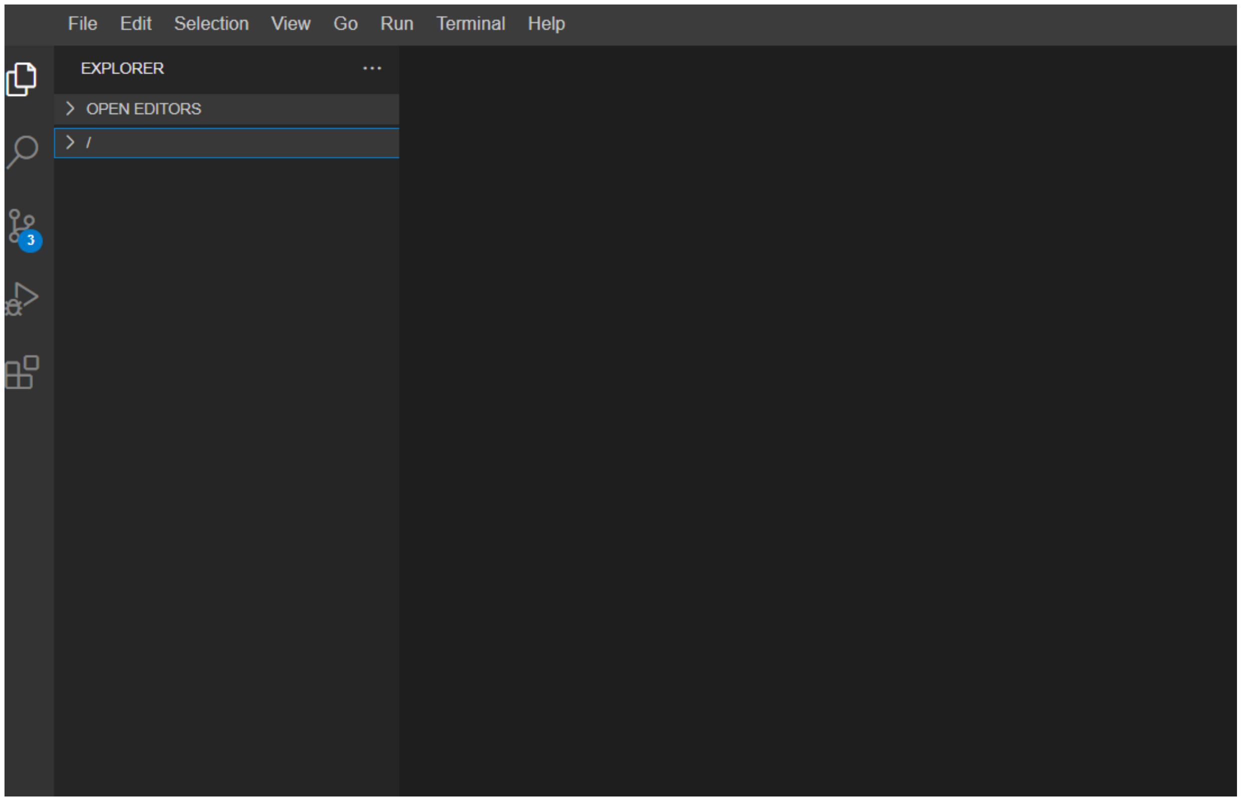
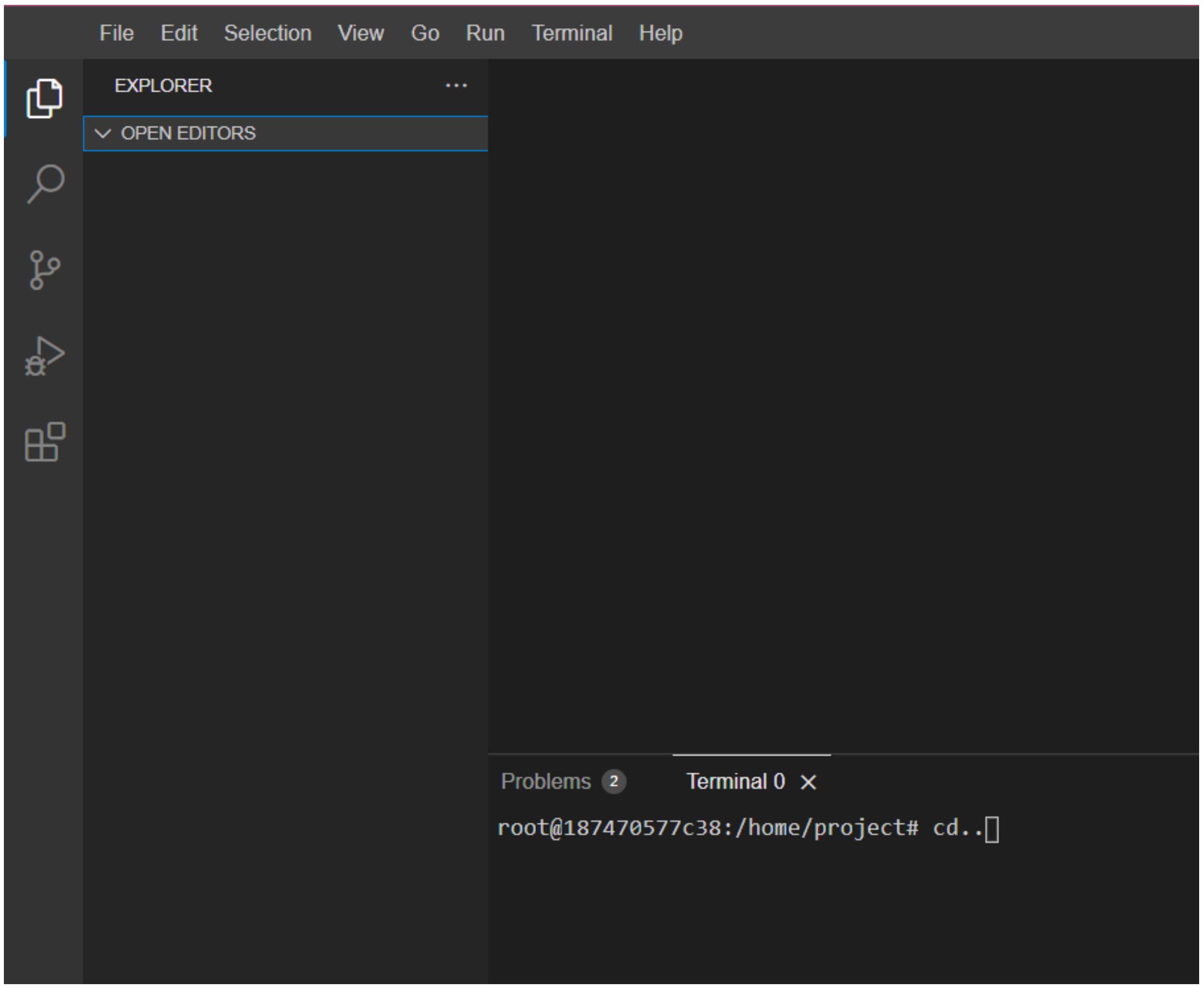


Figure 1: Open web IDE link on web browser

2. Click on Terminal, type below command to change directory from **/home/project** to **/home**.

   $cd ..

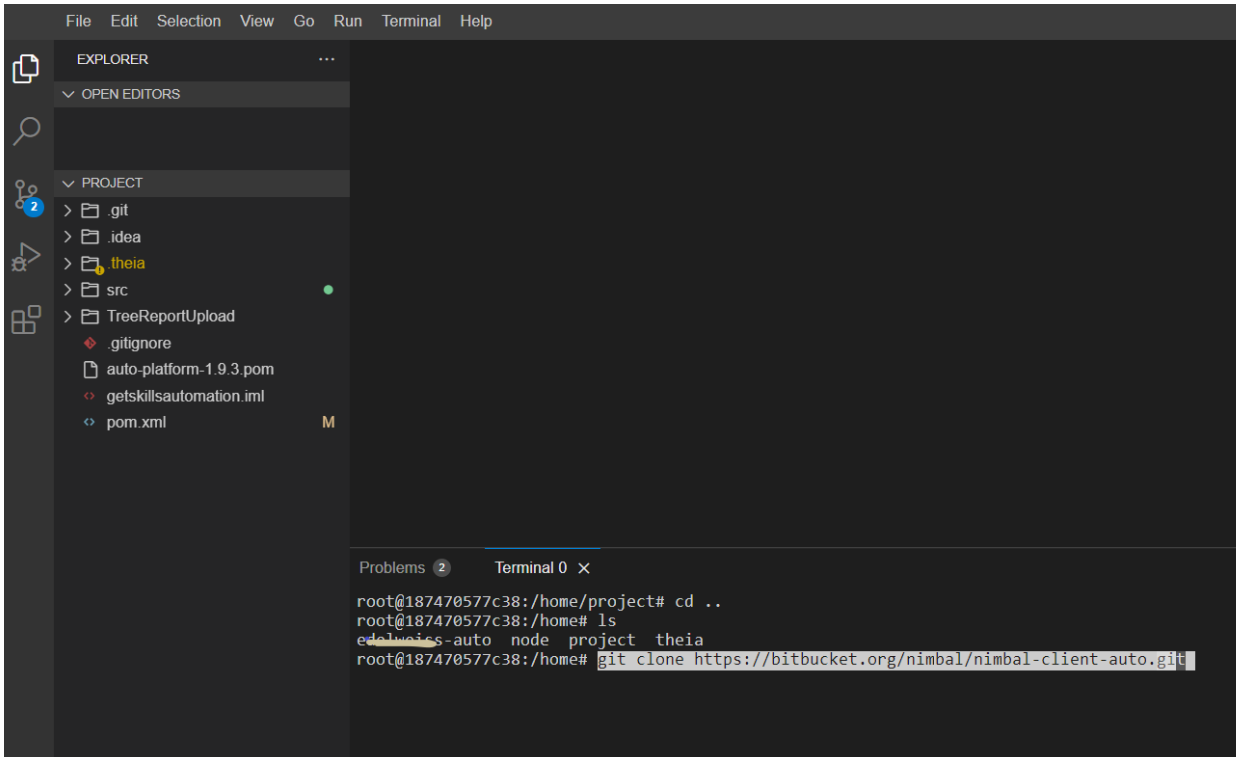


```
root@187470577c38:/home/project# cd..
```

Figure 2: home directory

3. Copy the clone command from bitbucket and enter in terminal as shown below (screenshot)

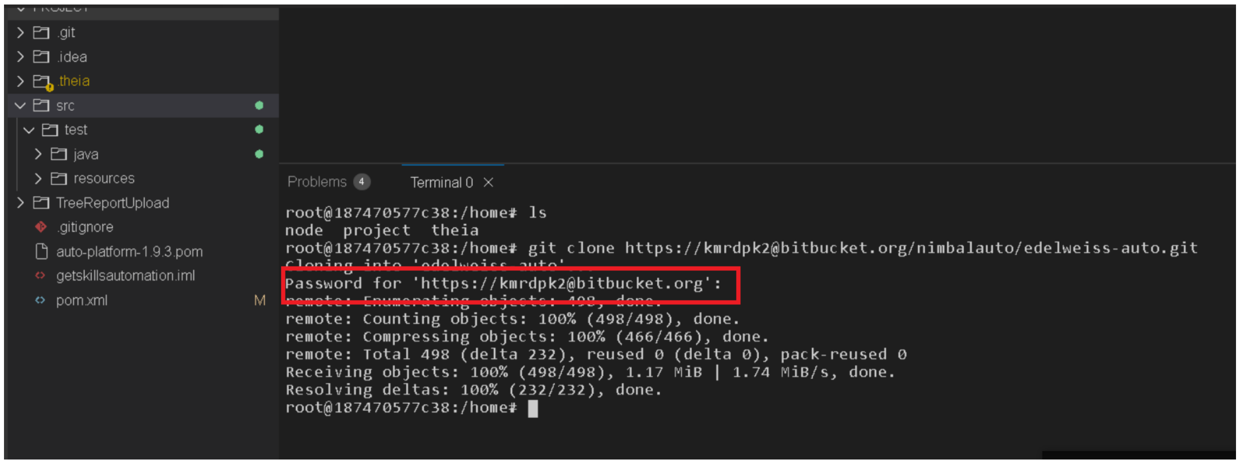
   $ git clone https://bitbucket.org/nimbal/nimbal-client-auto.git.



```
root@187470577c38:/home/project# cd ..
root@187470577c38:/home# ls
e[redacted]s-auto  node  project  theia
root@187470577c38:/home# git clone https://bitbucket.org/nimbal/nimbal-client-auto.git
```

Figure 3: creating clone repository

4. Enter the App Password to clone the Repository in local. (Create an App password for your bitbucket account).

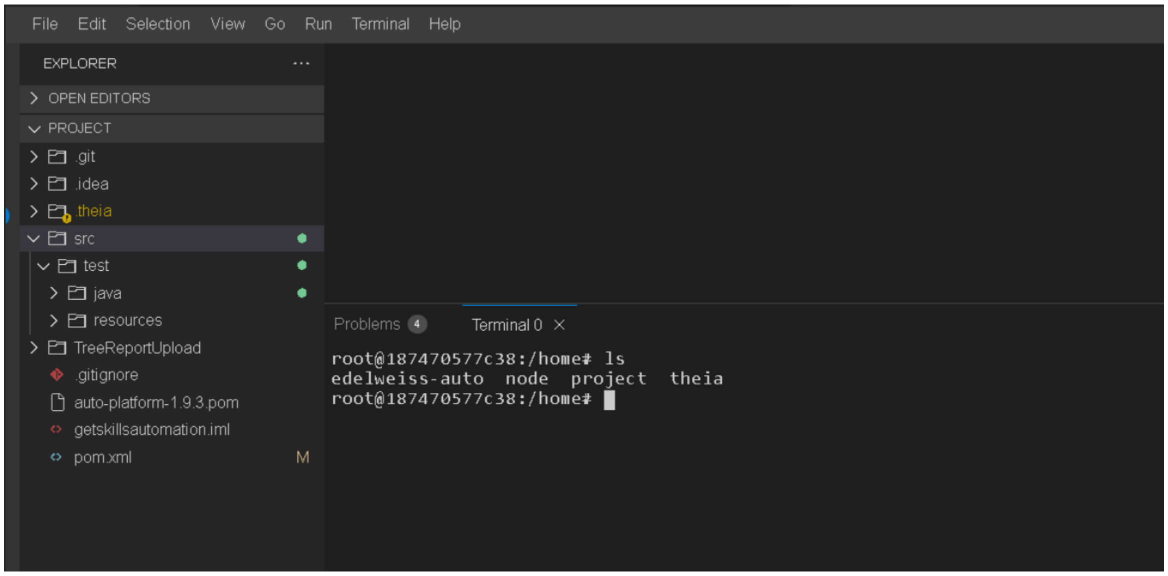


```
root@187470577c38:/home# ls
node  project  theia
root@187470577c38:/home# git clone https://kmrdpk2@bitbucket.org/nimbalauto/edelweiss-auto.git
Cloning into 'edelweiss-auto'...
Password for 'https://kmrdpk2@bitbucket.org':
remote: Enumerating objects: 498, done.
remote: Counting objects: 100% (498/498), done.
remote: Compressing objects: 100% (466/466), done.
remote: Total 498 (delta 232), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (498/498), 1.17 MiB | 1.74 MiB/s, done.
Resolving deltas: 100% (232/232), done.
root@187470577c38:/home#
```

Figure 4: App Password to clone repository

5. Run the command in the terminal to see the list where the project is created

   $ ls



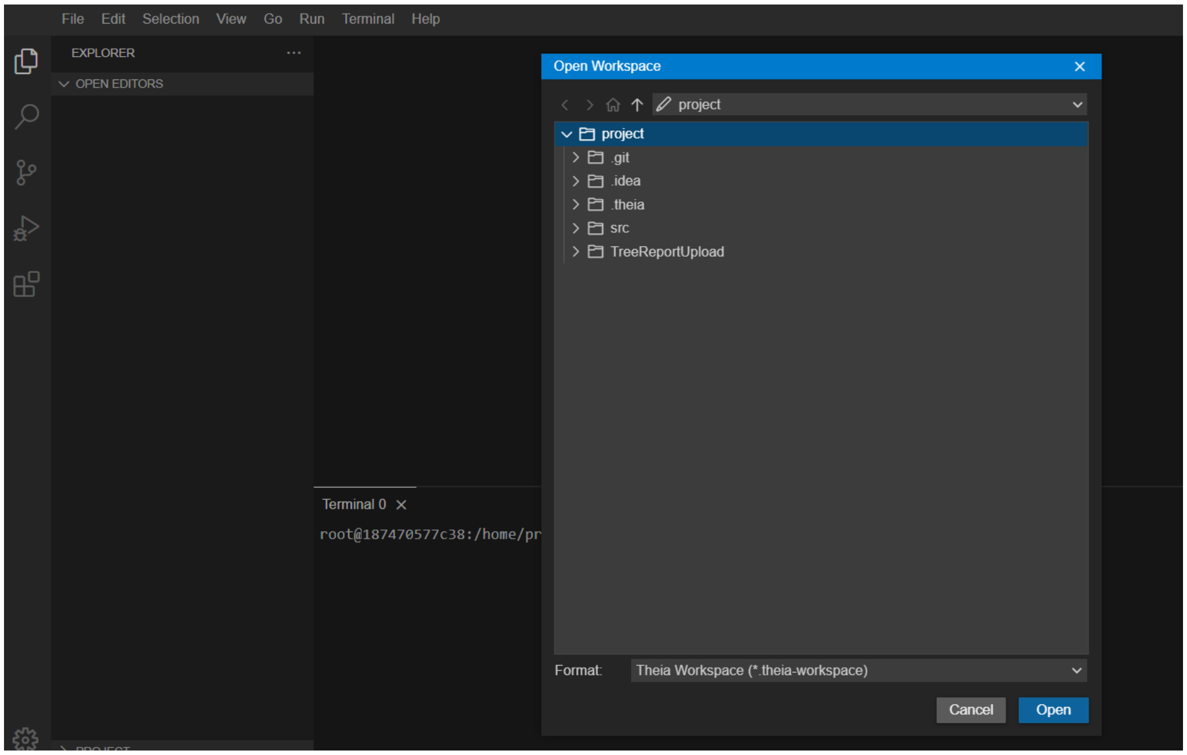
```
root@187470577c38:/home# ls
edelweiss-auto  node  project  theia
root@187470577c38:/home# 
```

Figure 5: Listing the project

After following the above steps, a project is created.

6. Check if the dependencies being created under **M2** Folder.

   a. Navigate to your cloned folder by clicking on **File** >> **Open Workspace**, then choose the project name which you have just cloned as shown below

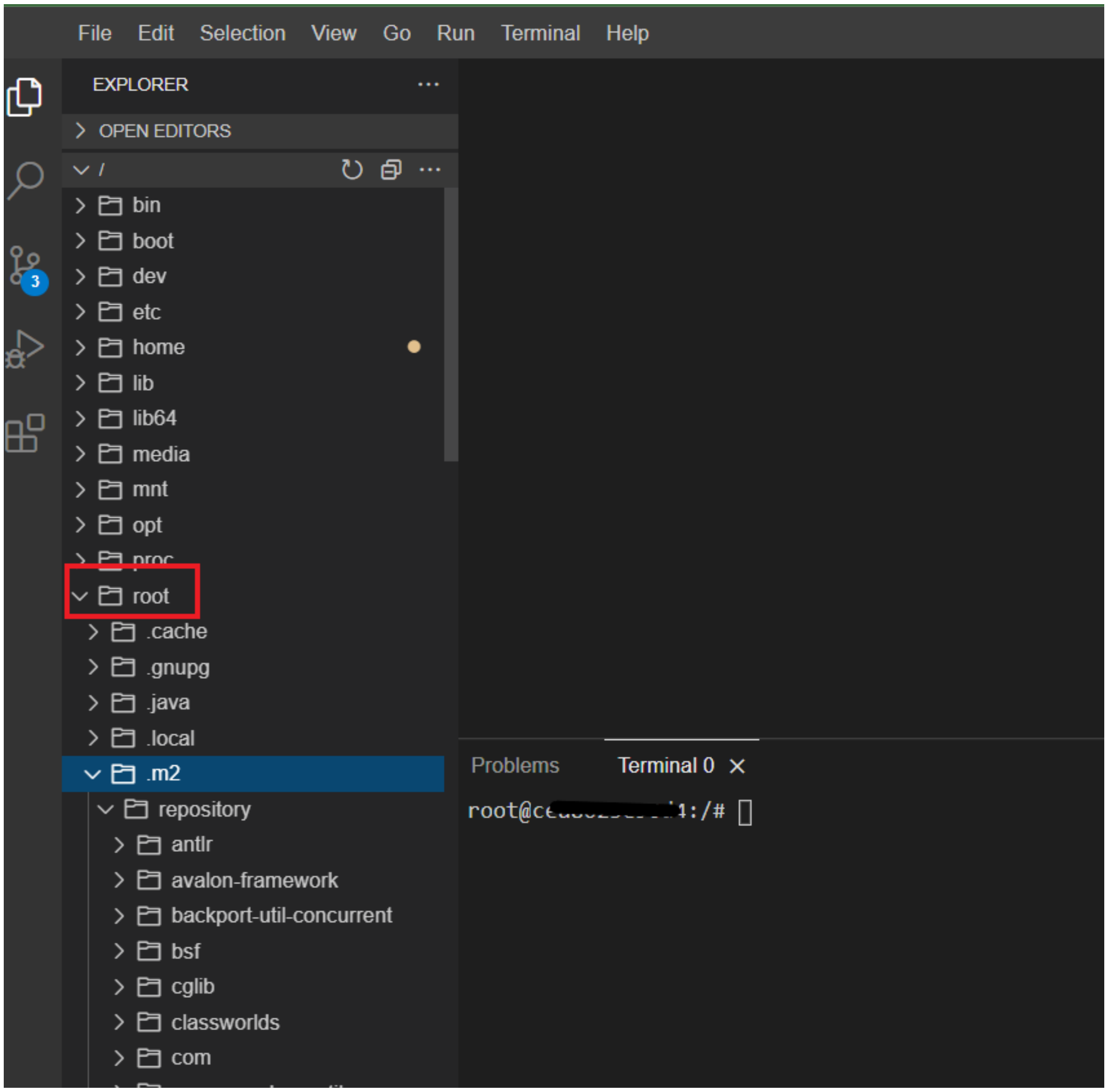


```
root@187470577c38:/home/pr
```

Format: Theia Workspace (*.theia-workspace)

Figure 6: Open Workspace

b. Go to M2 under root directory to see if Maven Dependencies are available.

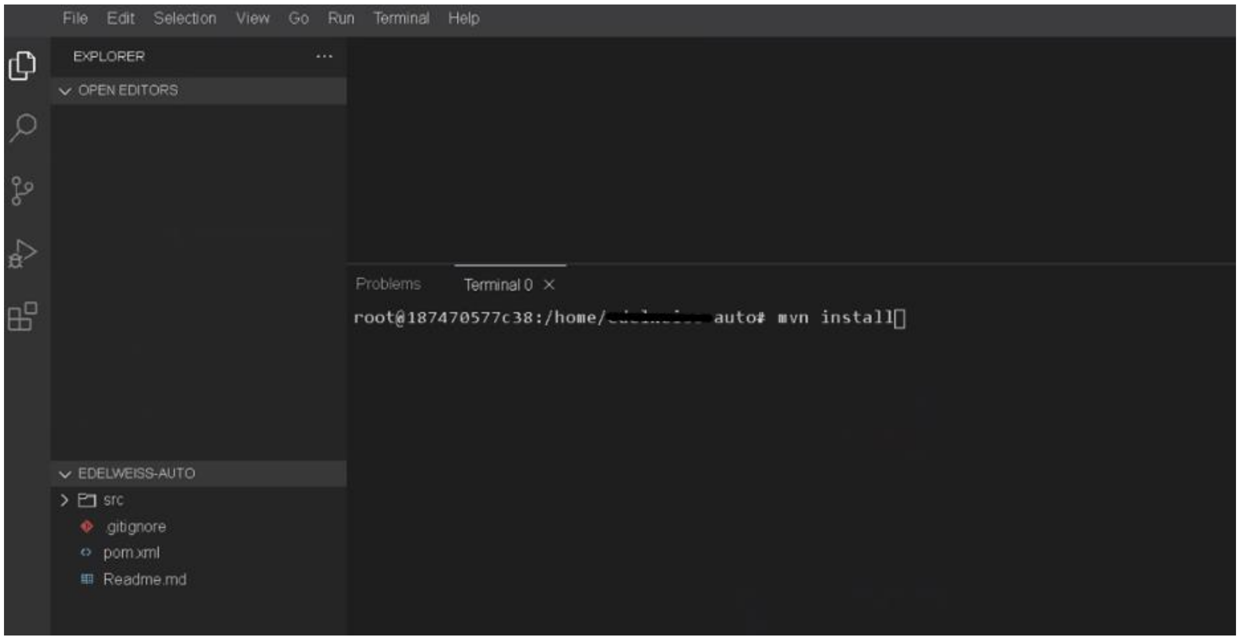


```
root@ce[redacted]1:/#
```

Figure 7: Look for Maven dependencies

c. Select the path /home/client-auto/, run the Command shown below to this install the dependencies related to Maven, Apache etc

$ **mvn install**



```
root@187470577c38:/home/[redacted]auto# mvn install
```

Figure 8: Running Command to install Dependencies

d. Notice new files are created under **M2** folders which means dependencies has been installed.

7. Install the Cucumber Plugin for steps Intellisense.

a. Open Web IDE Command Palette using **Ctrl + Shift + P**. You will see input box popup up as shown below

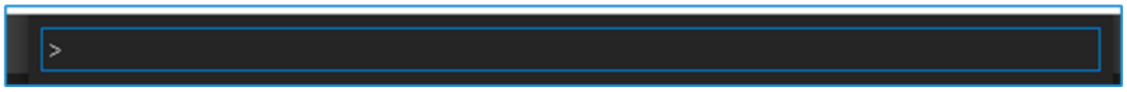


Figure 9: Open Command Palette

b. Type "**Plugin: Deploy**" as shown below in screenshot

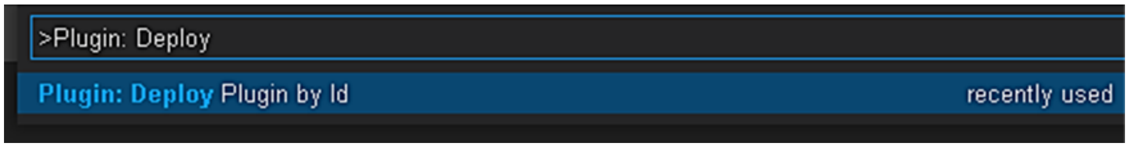


Figure 10: Search Plugin Deploy

c. Press **Enter**, and you will see input box as shown below in screenshot

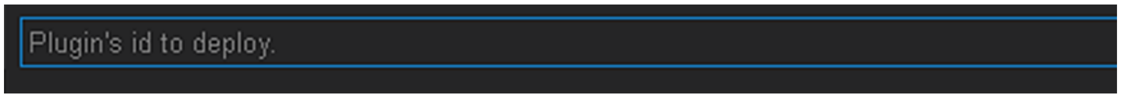
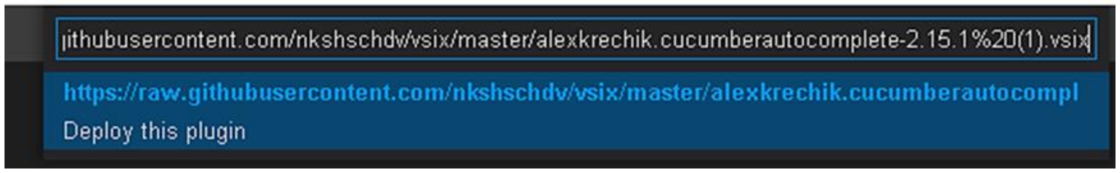

Plugin's id to deploy.

Figure 11: Select Plugin ID

d. Enter below URL in input box and press enter again as shown below in screenshot.

https://raw.githubusercontent.com/nkshschdv/vsix/master/alexkrechik.cucumberautocomplete-2.15.1%20(1).vsix

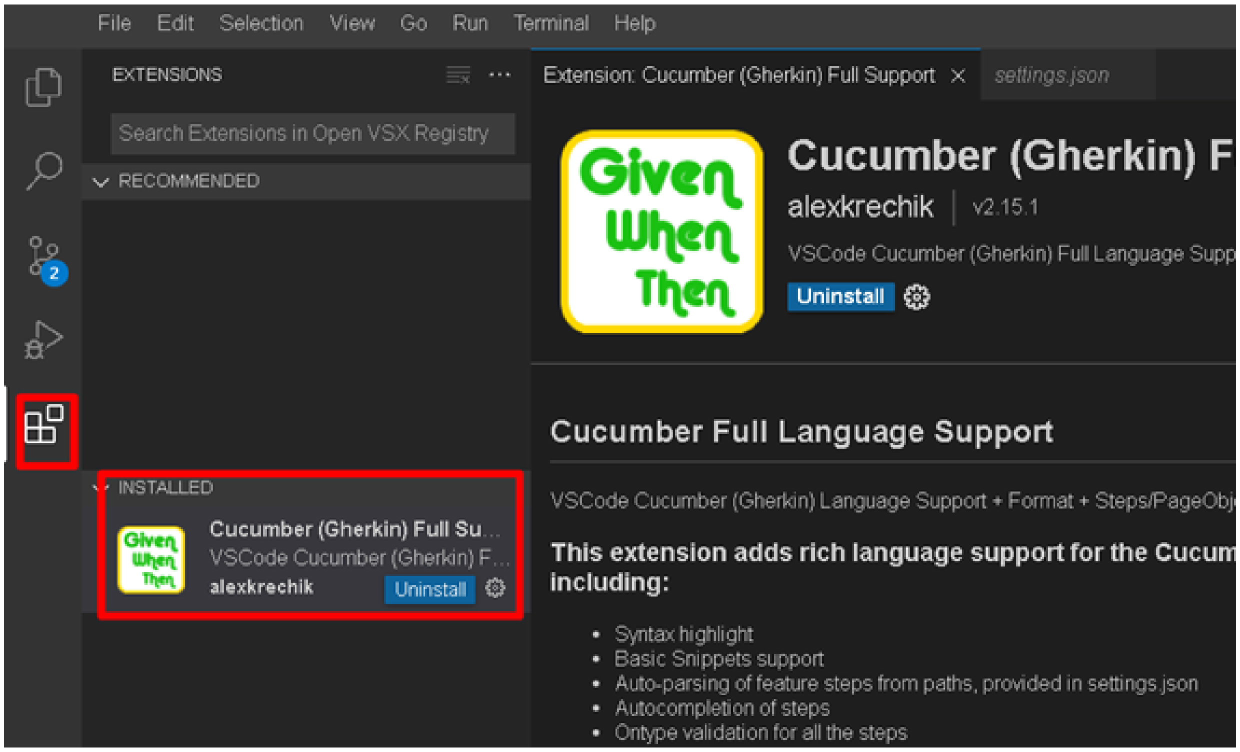

ithubusercontent.com/nkshschdv/vsix/master/alexkrechik.cucumberautocomplete-2.15.1%20(1).vsix

https://raw.githubusercontent.com/nkshschdv/vsix/master/alexkrechik.cucumberautocompl

Deploy this plugin

Figure 12: paste URL to deploy

e. You will see a plugin is installed in Plugin section as shown below in screenshot


File
Edit
Selection
View
Go
Run
Terminal
Help
EXTENSIONS
Search Extensions in Open VSX Registry
RECOMMENDED
INSTALLED
Cucumber (Gherkin) Full Su...
VSCode Cucumber (Gherkin) F...
alexkrechik
Uninstall
Extension: Cucumber (Gherkin) Full Support
settings.json
Cucumber (Gherkin) F
alexkrechik
v2.15.1
VSCode Cucumber (Gherkin) Full Language Supp
Uninstall

Cucumber Full Language Support

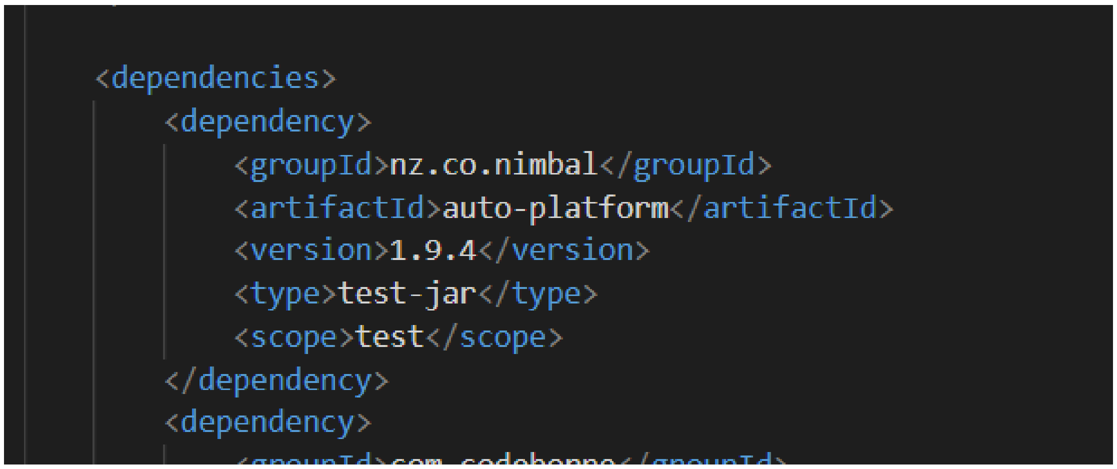
VSCode Cucumber (Gherkin) Language Support + Format + Steps/PageObj

This extension adds rich language support for the Cucum including:

- Syntax highlight
- Basic Snippets support
- Auto-parsing of feature steps from paths, provided in settings.json
- Autocompletion of steps
- Ontype validation for all the steps

Figure 13: Cucumber Plugin Installed

8. Check if the **auto-platform.jar** file is available. (If not, please contact Nimbal team). Follow the steps to install the auto-platform.jar with version.
   a. Go to **pom.xml** >> copy the version.



```
<dependencies>
    <dependency>
        <groupId>nz.co.nimbal</groupId>
        <artifactId>auto-platform</artifactId>
        <version>1.9.4</version>
        <type>test-jar</type>
        <scope>test</scope>
    </dependency>
    <dependency>
```

Figure 14: Copy the auto-platform version 1.9.4

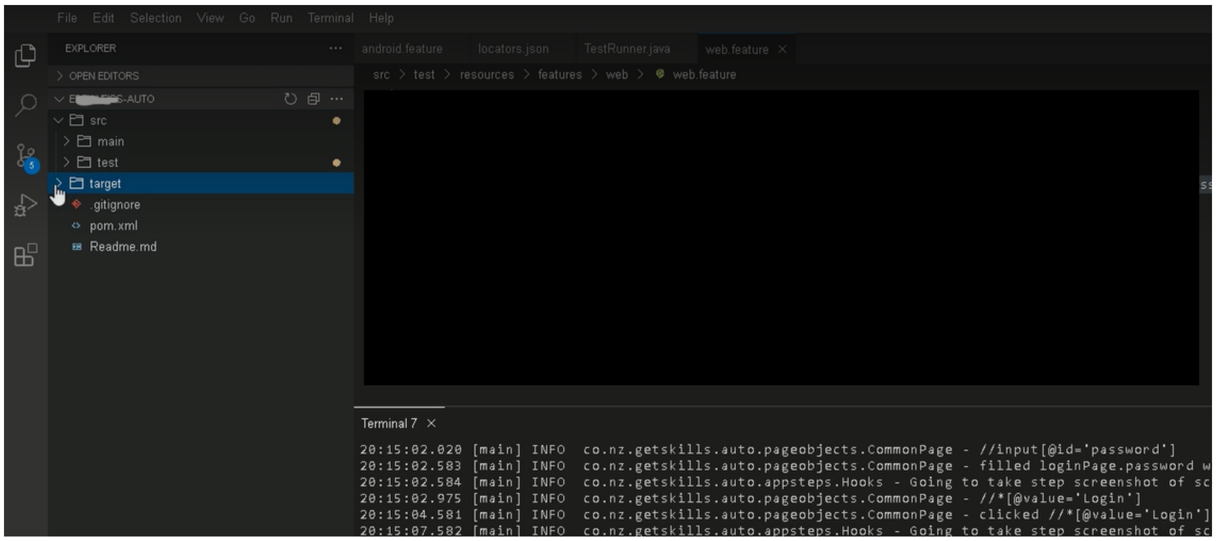
   b. Run the command

   $ mvn install:install-file -Dfile=mvn install:install-file -Dfile='/home/project/src/test/resources/1.9.3/auto-platform-1.9.3-tests.jar' -DgroupId='nz.co.nimbal' -DartifactId='auto-platform' -Dversion='1.9.3' -Dpackaging='test-jar'

   *Note: You can also find this command in **Readme***

   c. Run the command to delete the Target folder created from running **mvn install** for changing the version for auto-platform jar file as shown above.

   $ **mvn clear**



```
20:15:02.020 [main] INFO  co.nz.getskills.auto.pageobjects.CommonPage - //input[@id='password']
20:15:02.583 [main] INFO  co.nz.getskills.auto.pageobjects.CommonPage - filled loginPage.password w
20:15:02.584 [main] INFO  co.nz.getskills.auto.appsteps.Hooks - Going to take step screenshot of sc
20:15:02.975 [main] INFO  co.nz.getskills.auto.pageobjects.CommonPage - //*[@value='Login']
20:15:04.581 [main] INFO  co.nz.getskills.auto.pageobjects.CommonPage - clicked //*[@value='Login']
20:15:07.582 [main] INFO  co.nz.getskills.auto.appsteps.Hooks - Going to take step screenshot of sc
```

Figure 15: Delete Target Folder

B. CODES AND TEST CASES

In WEB IDE testing, user create/update 4 major files to automate testing. The files are mentioned below:

1. **config.dev.properties**
2. **.feature File.**
3. **locators.json**
4. **TestRunner.java**

1. **config.dev.properties** : .**properties** file is used to store the configurable parameters of an application. Config file defines the parameters, options, settings and preferences applied to operating systems (OSes), infrastructure devices and applications used to run the test.

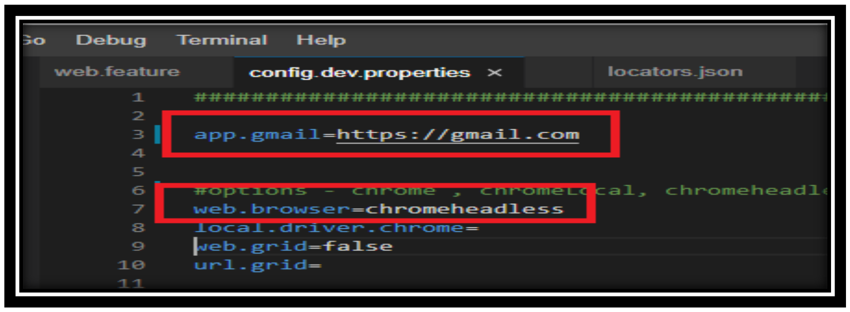
   Located at **/home/project/src/test/resources/env/config.dev.properties**. Add the following values as shown below in Figure 4

   a. app.gmail = gmail.com

   instead of gmail, you can replace it with any constant value of your concern. For example, app.website = www.website.com

   b. web.browser=chromeheadless

Specified which browser to the user for running the test. other values possible are firefox



```
#################################################

app.gmail=https://gmail.com


#options - chrome , chromelocal, chromeheadle
web.browser=chromeheadless
local.driver.chrome=
web.grid=false
url.grid=
```

Figure 16: config.dev.properties

2. **locators.json**

   Located at **/home/project/src/test/resources/locators.json** . This file is used to add XPath key pair values, while key will be user understandable keyword and value will be an XPath, which is used to locate an HTML element within a webpage. For example, XPath for inputting email id in gmail.com page will be **//input[@id='identifierId']** as shown below in Figure

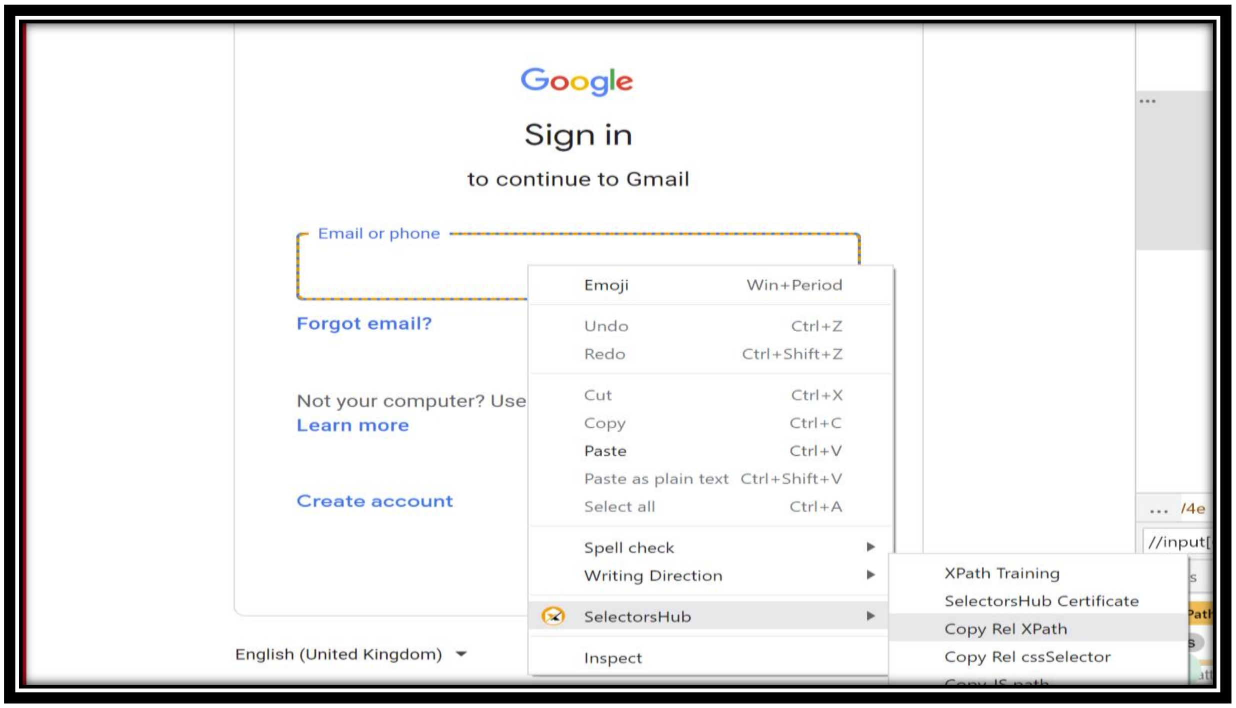
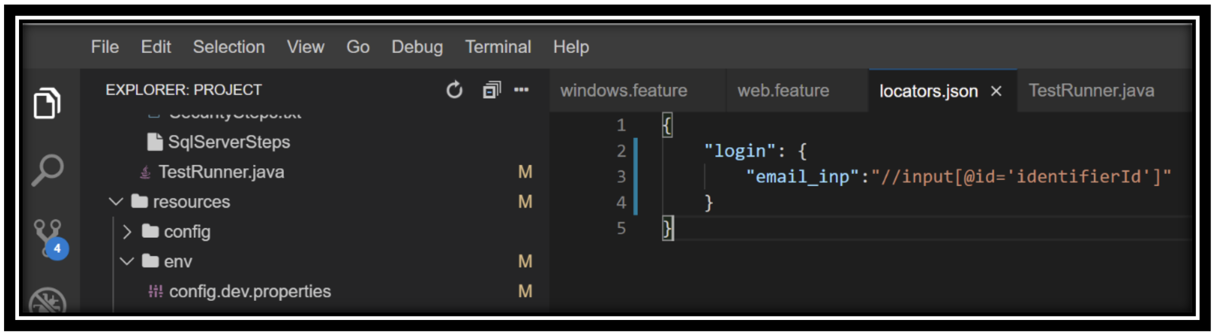


Figure 17: illustrates how to choose the XPath through Selectors Hub

Therefore, the locators.json file will look something like the below after making changes



```
{
    "login": {
        "email_inp":"//input[@id='identifierId']"
    }
}
```

Figure 18: locator.json file after adding XPath for Gmail input location from Gmail.com

3. .feature File

New feature file is created to test the steps in scenarios. Follow the steps below to create a new file.

a. Click on the file's icon.

b. Navigate to src > java > feature.

c. A dialog box will appear and will ask for the name of the new file enter any name for example, login.feature extension

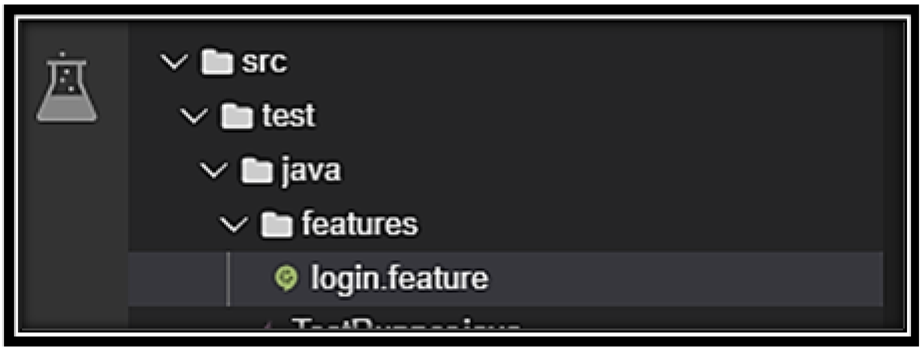
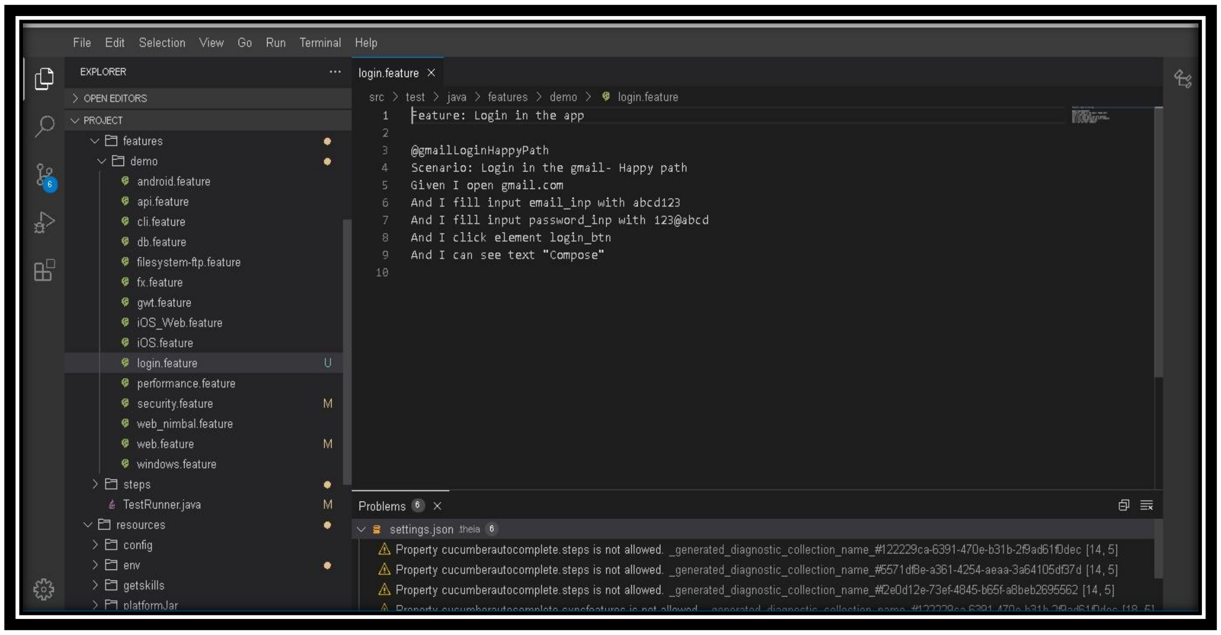


Figure 19: Path to the file

d. login. feature for login related feature test scenarios as shown in the below screenshot



```
Feature: Login in the app

@gmailLoginHappyPath
Scenario: Login in the gmail- Happy path
Given I open gmail.com
And I fill input email_inp with abcd123
And I fill input password_inp with 123@abcd
And I click element login_btn
And I can see text "Compose"
```

```
Property cucumberautocomplete.steps is not allowed. _generated_diagnostic_collection_name_#122229ca-6391-470e-b31b-2f9ad61f0dec [14, 5]
Property cucumberautocomplete.steps is not allowed. _generated_diagnostic_collection_name_#5571df8e-a361-4254-aeaa-3a64105df37d [14, 5]
Property cucumberautocomplete.steps is not allowed. _generated_diagnostic_collection_name_#f2e0d12e-73ef-4845-b65f-a8beb2695562 [14, 5]
```

Figure 20: feature file with test Cases for login

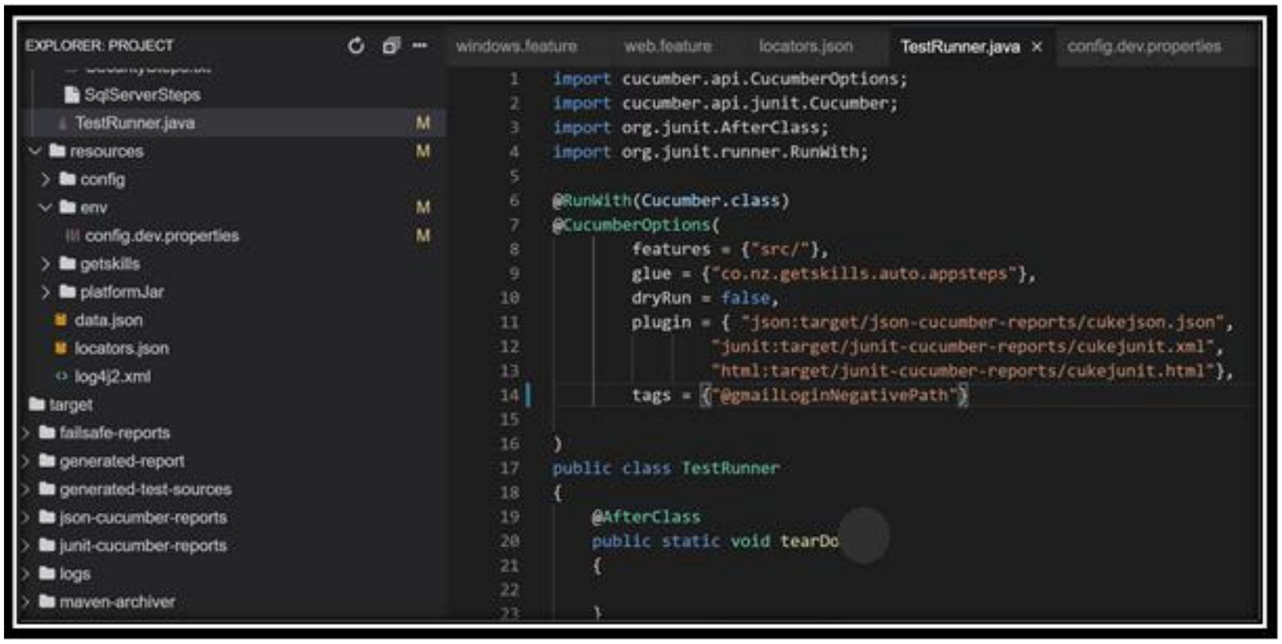
4. **TestRunner.java**:
   TestRunner.java file is located at **/home/project/src/test/java/TestRunner.java**. This file indicates which tests need to be run using tags. A tag can be placed on a scenario/test or a feature in a .feature file. It usually starts **"@"** keyword.
   For example,
   a. Figure 6 depicts the functioning of the **TestRunner.java** file and the tag provided in the tags section written at line number 14 as

      tags**={**"**@gmailLoginNegativePath**"}

      Later we will replace this tag to run the tests we want to execute.

   b. If tags are empty then the TestRunner.java file will run all the feature files present in the project as shown below

      tags={""}



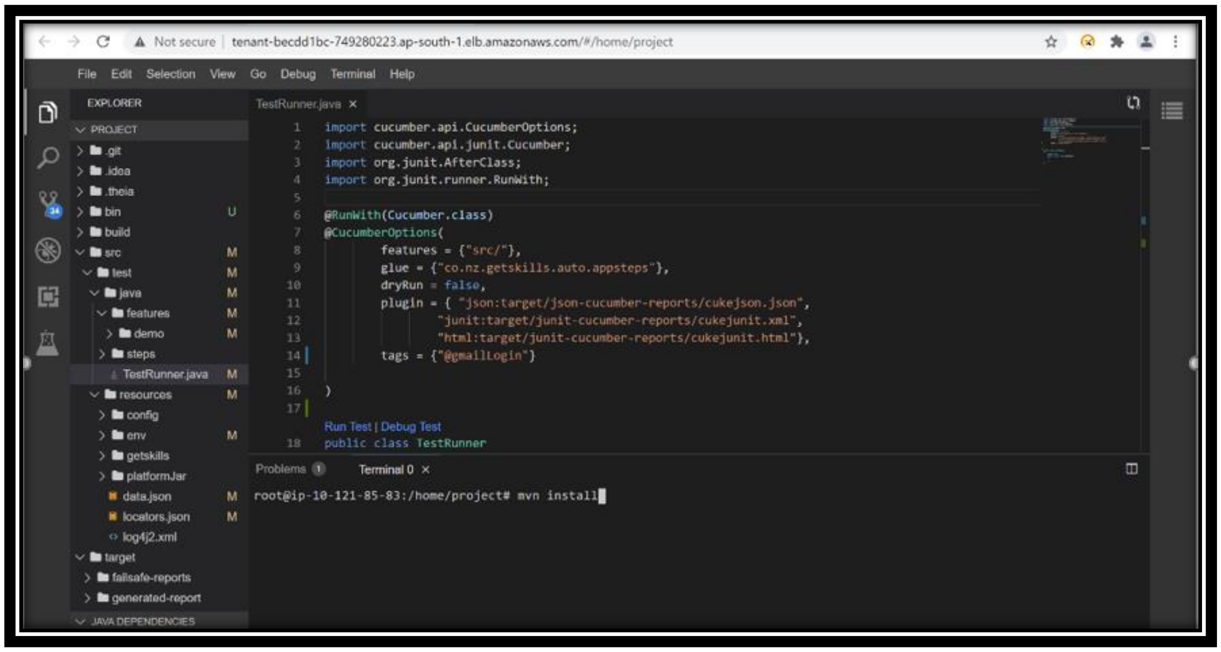
```
import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;
import org.junit.AfterClass;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(
        features = {"src/"},
        glue = {"co.nz.getskills.auto.appsteps"},
        dryRun = false,
        plugin = { "json:target/json-cucumber-reports/cukejson.json",
                "junit:target/junit-cucumber-reports/cukejunit.xml",
                "html:target/junit-cucumber-reports/cukejunit.html"},
        tags = {"@gmailLoginNegativePath"}

)
public class TestRunner
{
    @AfterClass
    public static void tearDo
    {

    }
```

Figure 21: TestRunner.java file

C. Run Test Case in WEB IDE

Follow the steps involved in running the test cases:

In the terminal, enter **mvn install** command and it will show the results in the terminal.



```
import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;
import org.junit.AfterClass;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(
        features = {"src/"},
        glue = {"co.nz.getskills.auto.appsteps"},
        dryRun = false,
        plugin = { "json:target/json-cucumber-reports/cukejson.json",
                "junit:target/junit-cucumber-reports/cukejunit.xml",
                "html:target/junit-cucumber-reports/cukejunit.html"},
        tags = {"@gmailLogin"}
)

public class TestRunner
```

```
root@ip-10-121-85-83:/home/project# mvn install
```

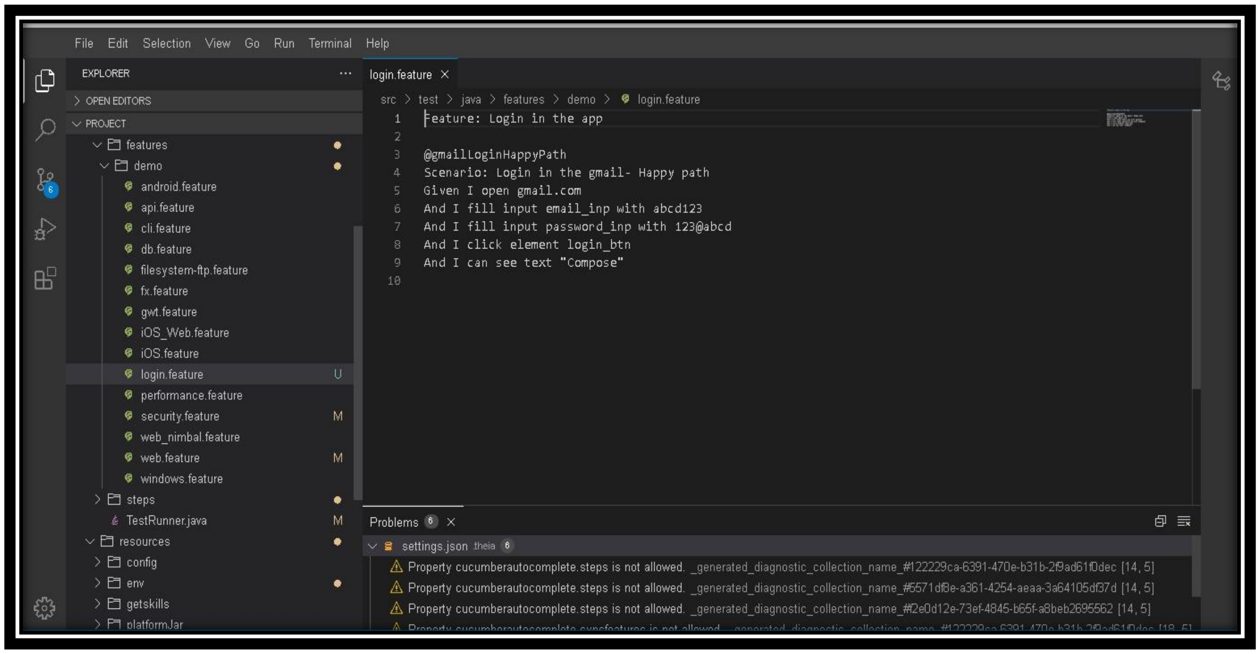
Figure 22: TestRunner.java

Run 1: Running Happy Path Scenario

Following is the code is written for the Happy path as shown in code snippet 1. Happy path test is a well-defined test case using known input, which executes without exception and produces an expected output.

```
Feature: Login in the app

@gmailLoginHappyPath
Scenario: Login in the Gmail- Happy path
Given I open gmail.com
And I fill input email_inp with abcd123
And I fill input password_inp with 123@abcd
And I click element login_btn
And I can see the text "Compose"
```

Code Snippet 1: Happy Path for Gmail Login



```
Feature: Login in the app

@gmailLoginHappyPath
Scenario: Login in the gmail- Happy path
Given I open gmail.com
And I fill input email_inp with abcd123
And I fill input password_inp with 123@abcd
And I click element login_btn
And I can see text "Compose"
```

```
Property cucumberautocomplete.steps is not allowed. _generated_diagnostic_collection_name_#122229ca-6391-470e-b31b-2f9ad61f0dec [14, 5]
Property cucumberautocomplete.steps is not allowed. _generated_diagnostic_collection_name_#5571df8e-a361-4254-aeaa-3a64105df37d [14, 5]
Property cucumberautocomplete.steps is not allowed. _generated_diagnostic_collection_name_#2e0d12e-73ef-4845-b65f-a8beb2695562 [14, 5]
```

Figure 23: Reference image in Web IDE Happy Path

Explanation of the Code

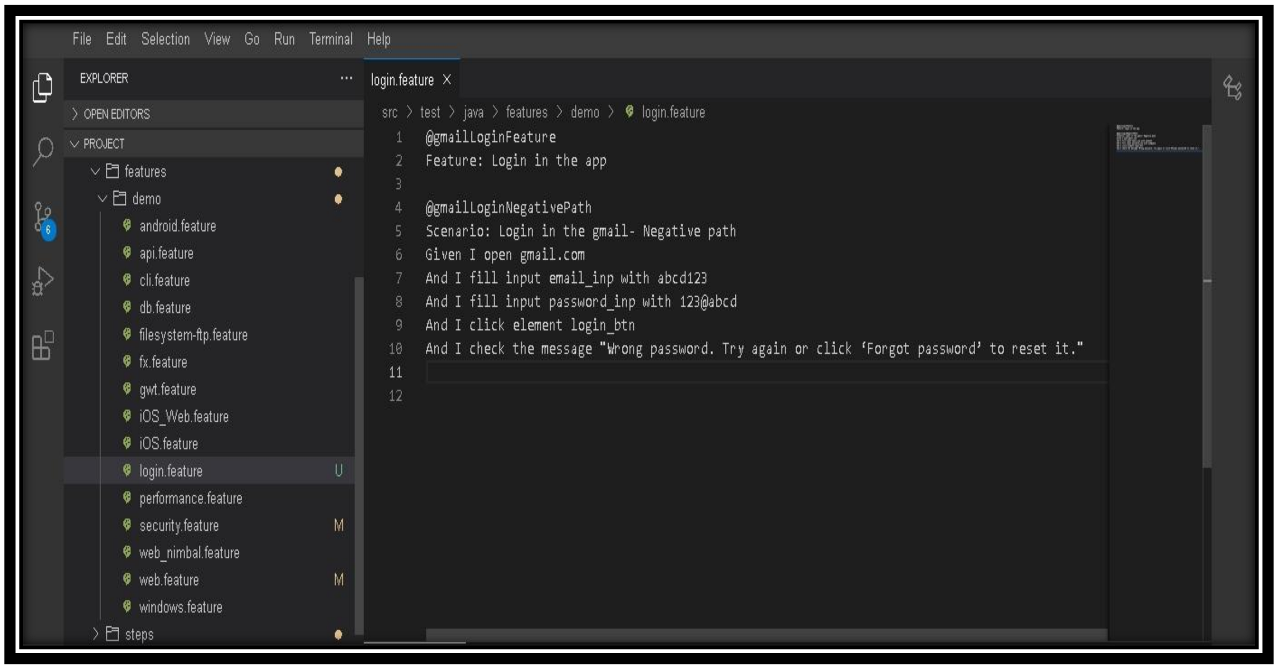
1. Title of the Feature, always start with Feature Keyword
2. The keyword of the whole case (No need to write the complete code for testing just enter the keyword)
3. Description of the code
4. The user opens the URL gmail.com
5. Then users enter the data as abcd123 in the email field (fill input is a keyword for giving input)
6. Then users enter the data as 123@abcd in the password field (fill input is a keyword for giving input)
7. Then by clicking the login_btn (Button named Login)
8. User login successful can view the compose button

Run 2: Running Negative Path Scenario

Following is the code is written for the Negative path as shown in code snippet 2. Negative testing ensures that your application can gracefully handle invalid input or unexpected user behaviour.

```
@gmailLoginFeature
Feature: Login in the app

@gmailLoginNegativePath
Scenario: Login in the Gmail- Negative path
Given I open gmail.com
And I fill input email_inp with abcd123
And I fill input password_inp with 123@abcd
And I click element login_btn
And I check the message "Wrong password. Try again or click ‘Forgot password’ to
 reset it."
```

Code Snippet 2: Negative Path for Gmail Login



```
@gmailLoginFeature
Feature: Login in the app

@gmailLoginNegativePath
Scenario: Login in the gmail- Negative path
Given I open gmail.com
And I fill input email_inp with abcd123
And I fill input password_inp with 123@abcd
And I click element login_btn
And I check the message "Wrong password. Try again or click ‘Forgot password’ to reset it."
```

Figure 24: Reference image in Web IDE for Negative path

Explanation of the Code

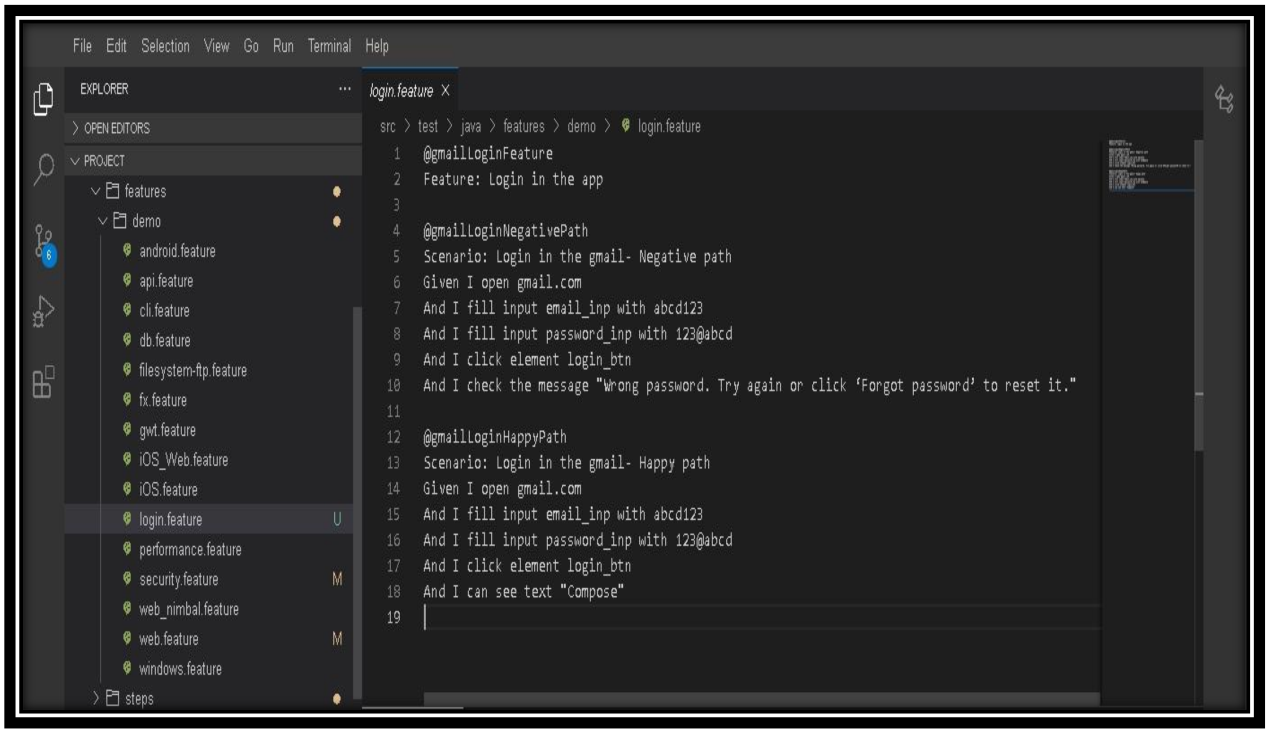
1. The keyword of the whole case (No need to write the complete code for testing just enter the keyword)
2. Title
3. The keyword of case 2(No need to write the complete code for testing just enter the keyword)
4. Description of the code
5. The user opens the URL gmail.com
6. Then users enter the data as abcd123 in the email field (fill input is a keyword for giving input)
7. Then users enter the data as 123@abcd in the password field (fill input is a keyword for giving input)
8. Then by clicking the login_btn (Button named Login) User login successful, can view the message "Wrong password. Try again or click ‘Forgot password’ to reset it."

Run 3: Running the whole feature

Following is the complete code for our feature. Which can be executed by the adding tag **@gmailLoginFeature** in the **TestRunner.java** file.

```
@gmailLoginFeature

Feature: Login in the app

@gmailLoginHappyPath
Scenario: Login in the Gmail- Happy path
Given I open gmail.com
And I fill input email_inp with abcd123
And I fill the input password with 123@abcd
And I click element login_btn
And I can see the text "Compose"

@gmailLoginNegativePath
Scenario: Login in the Gmail- Negative path
Given I open gmail.com
And I fill input email_inp with abcd123
And I fill input password_inp with 123@abcd
And I click element login_btn
And I check the message "Wrong password. Try again or click ‘Forgot password’ to
 reset it."
```

Code Snippet 3: complete feature file code



```
@gmailLoginFeature
Feature: Login in the app

@gmailLoginNegativePath
Scenario: Login in the gmail- Negative path
Given I open gmail.com
And I fill input email_inp with abcd123
And I fill input password_inp with 123@abcd
And I click element login_btn
And I check the message "Wrong password. Try again or click 'Forgot password' to reset it."

@gmailLoginHappyPath
Scenario: Login in the gmail- Happy path
Given I open gmail.com
And I fill input email_inp with abcd123
And I fill input password_inp with 123@abcd
And I click element login_btn
And I can see text "Compose"
```

Figure 25: Reference image in Web IDE for complete code

D. RUNNING TEST USING TAGS

There are two components to understand how to run tests (features/scenarios) by tags

1. **Tags:**

   Various tags are used in feature files. We can identify a tag as a group of features or a group of scenarios or a group of both features and tags. Depending on a tester.

   For example, in login.feature file above files we have used the following tags

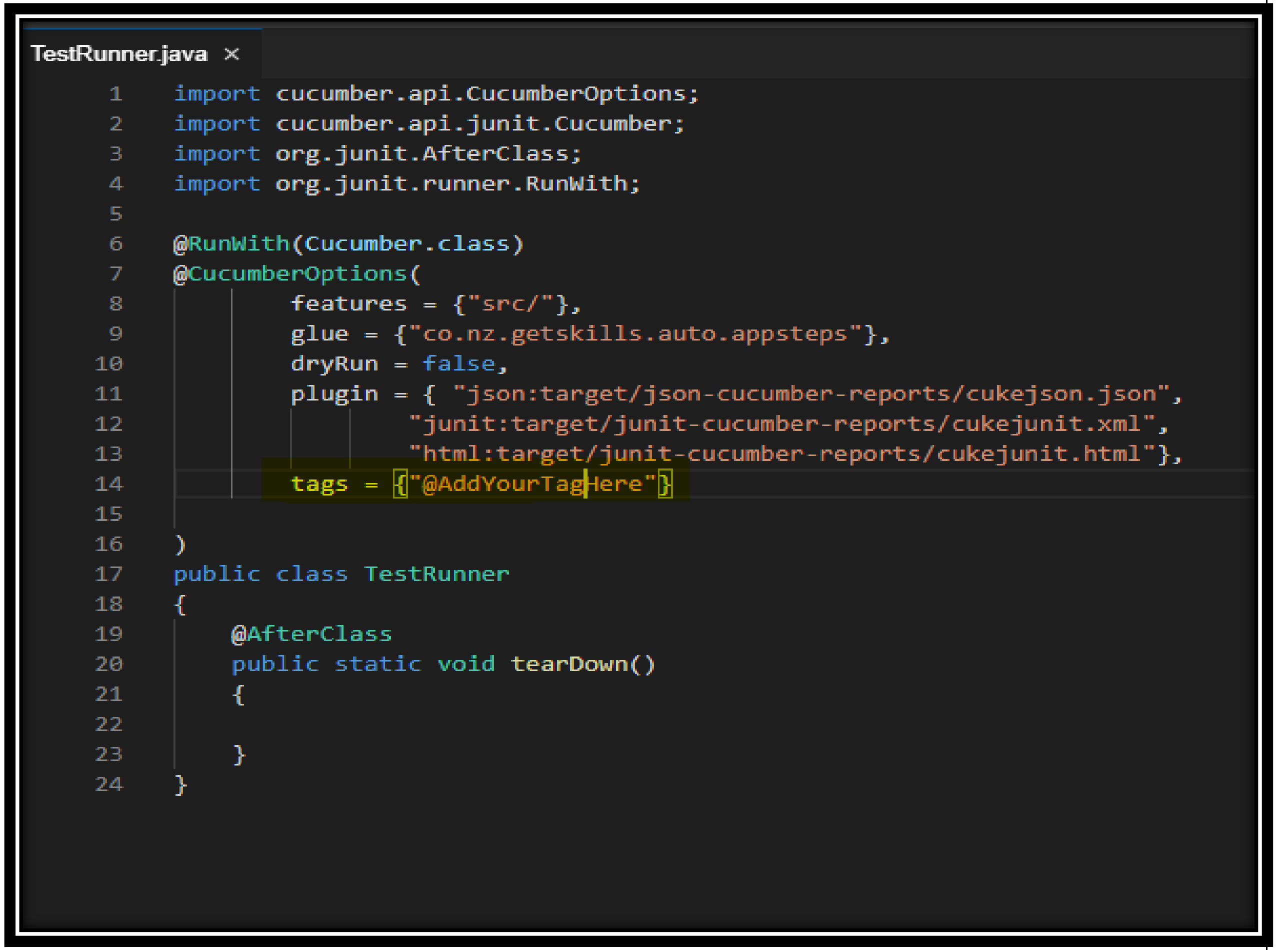
   - **@gmailLoginFeature** – To execute complete feature
   - **@gmailLoginHappyPath** – To execute the happy path scenario
   - **@gmailLoginNegativePath** – To execute the negative path scenario

2. **TestRunner.java file:**

   It is used for running the test with the help of tags. Every feature and scenario have a tag associated with it. We will insert this tag in the TestRunner.java file and run the program to get the desired outcome of the test using reports.

Following is the way to utilize the TestRunner.java file to run the tests.

1. Click on the file's icon. Navigate to src > java > TestRunner.java



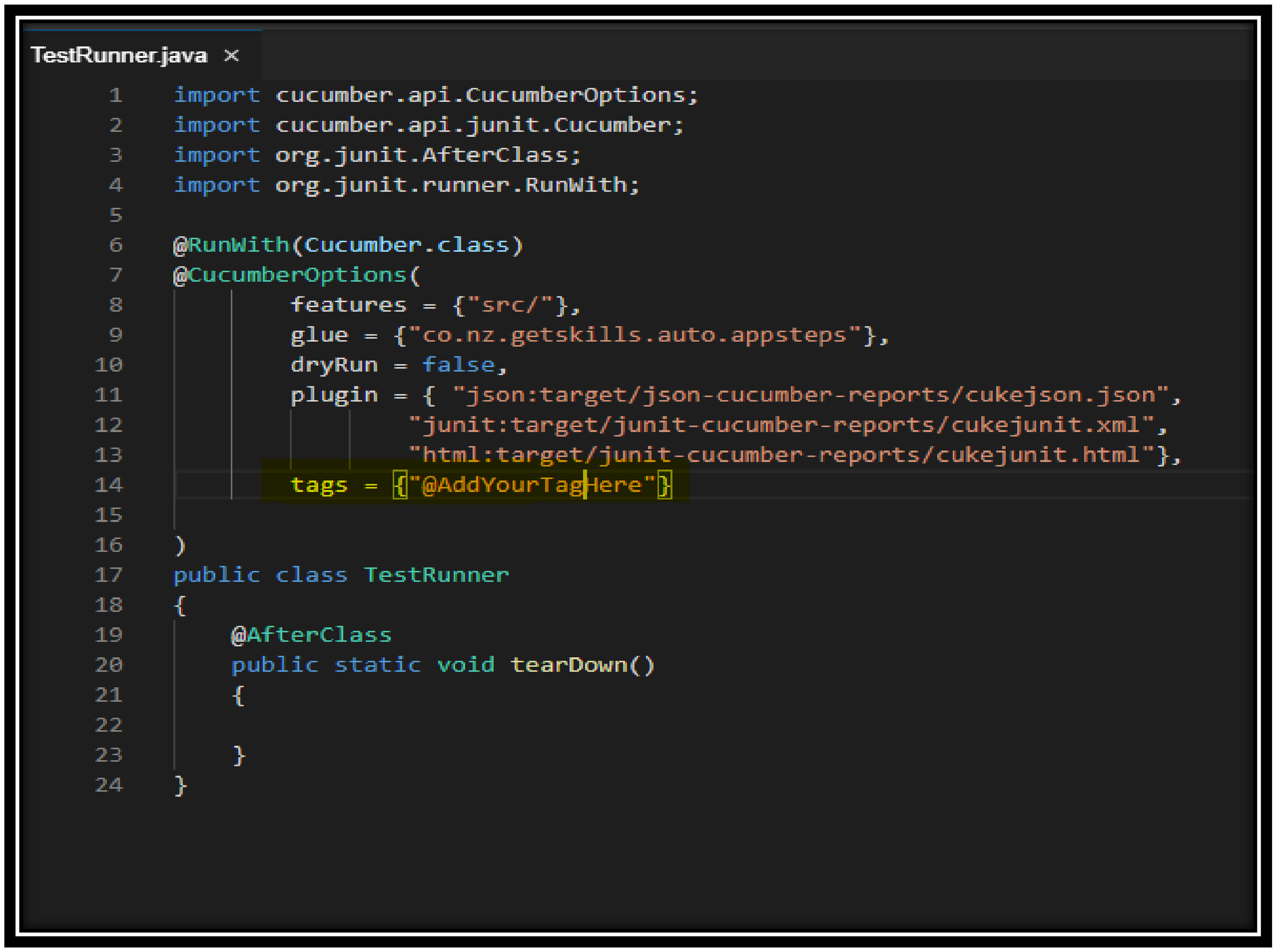
```
TestRunner.java ×
import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;
import org.junit.AfterClass;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(
        features = {"src/"},
        glue = {"co.nz.getskills.auto.appsteps"},
        dryRun = false,
        plugin = { "json:target/json-cucumber-reports/cukejson.json",
                "junit:target/junit-cucumber-reports/cukejunit.xml",
                "html:target/junit-cucumber-reports/cukejunit.html"},
        tags = {"@AddYourTagHere"}

)
public class TestRunner
{
    @AfterClass
    public static void tearDown()
    {

    }
}
```

Figure 26: Add the tag

2. In any pre-created tag can be used for the test by taking any Tag from an existing code then writing it in the place of **"@AddYourTagHere"**

For example tags = {"@gmailLoginFeature"}

Here the tag @gmailLoginFeature is used.



TestRunner.java ×

```java
import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;
import org.junit.AfterClass;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(
        features = {"src/"},
        glue = {"co.nz.getskills.auto.appsteps"},
        dryRun = false,
        plugin = { "json:target/json-cucumber-reports/cukejson.json",
                "junit:target/junit-cucumber-reports/cukejunit.xml",
                "html:target/junit-cucumber-reports/cukejunit.html"},
        tags = {"@AddYourTagHere"}

)
public class TestRunner
{
    @AfterClass
    public static void tearDown()
    {

    }
}
```

Figure 27: Enter the tag to run the test case

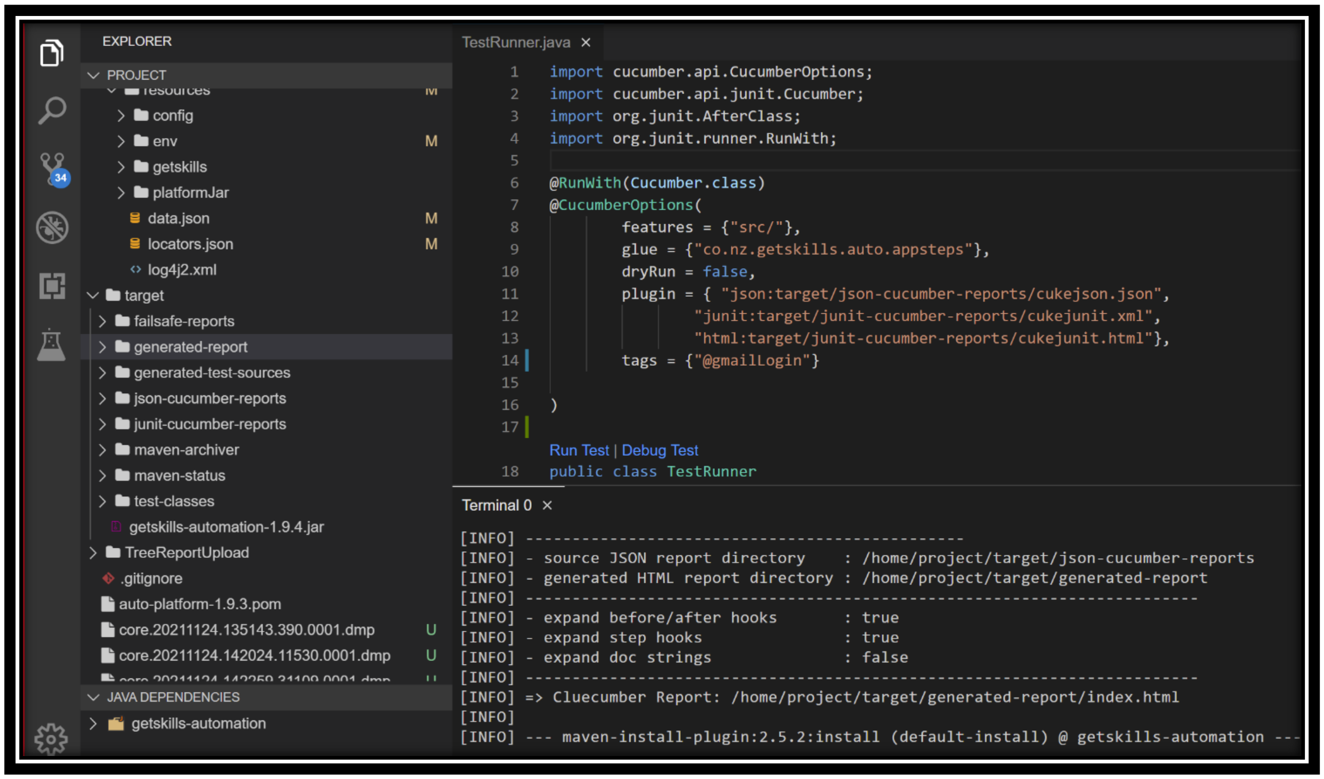
3. Run the command to execute the test case with the specific scenario with tag.
4. Target Folder gets generated where reports are created with passed or failed status.

E. REPORTS

The report is generated each time when the user runs a tag using the maven command or by running right click on the TestRunner.java file.

1. The report generated can be seen under the targets folder that is located at /home/project/target.

2. The json reports are found at /home/project/target/json-cucumber-reports.



```java
import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;
import org.junit.AfterClass;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(
        features = {"src/"},
        glue = {"co.nz.getskills.auto.appsteps"},
        dryRun = false,
        plugin = { "json:target/json-cucumber-reports/cukejson.json",
                "junit:target/junit-cucumber-reports/cukejunit.xml",
                "html:target/junit-cucumber-reports/cukejunit.html"},
        tags = {"@gmailLogin"}

)

public class TestRunner
```

```
[INFO] ------------------------------------------------
[INFO] - source JSON report directory    : /home/project/target/json-cucumber-reports
[INFO] - generated HTML report directory : /home/project/target/generated-report
[INFO] ---------------------------------------------------------------------
[INFO] - expand before/after hooks       : true
[INFO] - expand step hooks               : true
[INFO] - expand doc strings              : false
[INFO] ---------------------------------------------------------------------
[INFO] => Cluecumber Report: /home/project/target/generated-report/index.html
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ getskills-automation ---
```

Figure 28: Generated Report -JSON

3. The generated HTML report is found at /home/project/target/generated-report.
4. If any test fails the auto-generated report then render's clarity where it is failing, one can see the captured screenshots at **/home/project/target/generated-report/attachments**

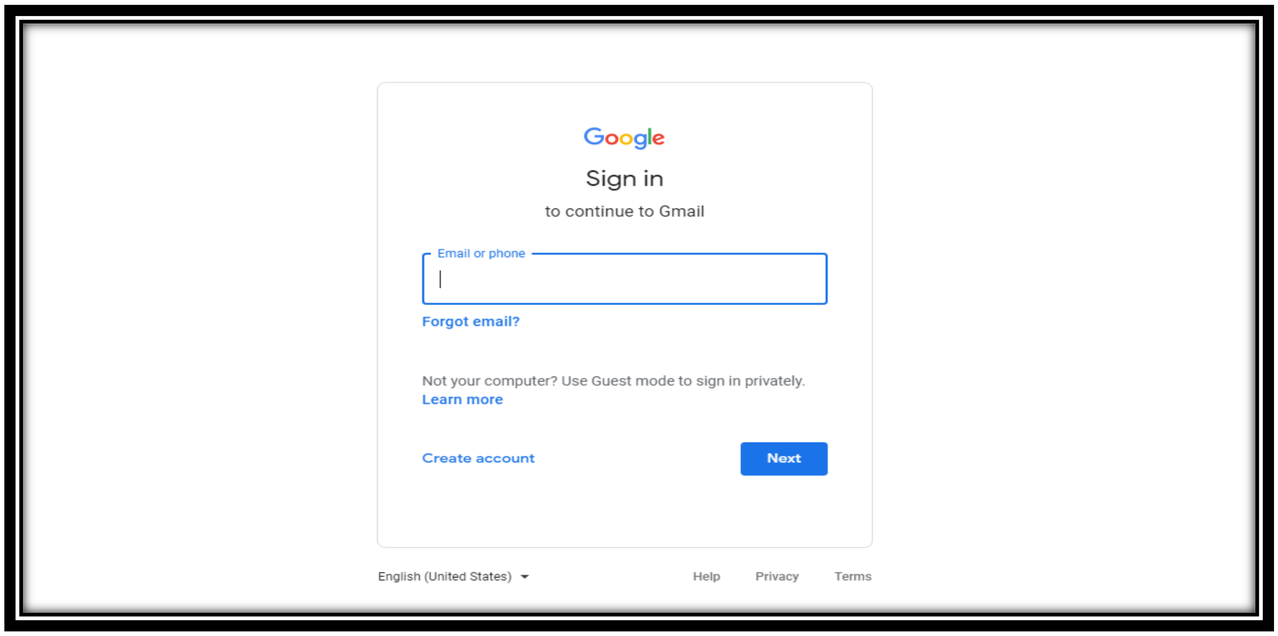


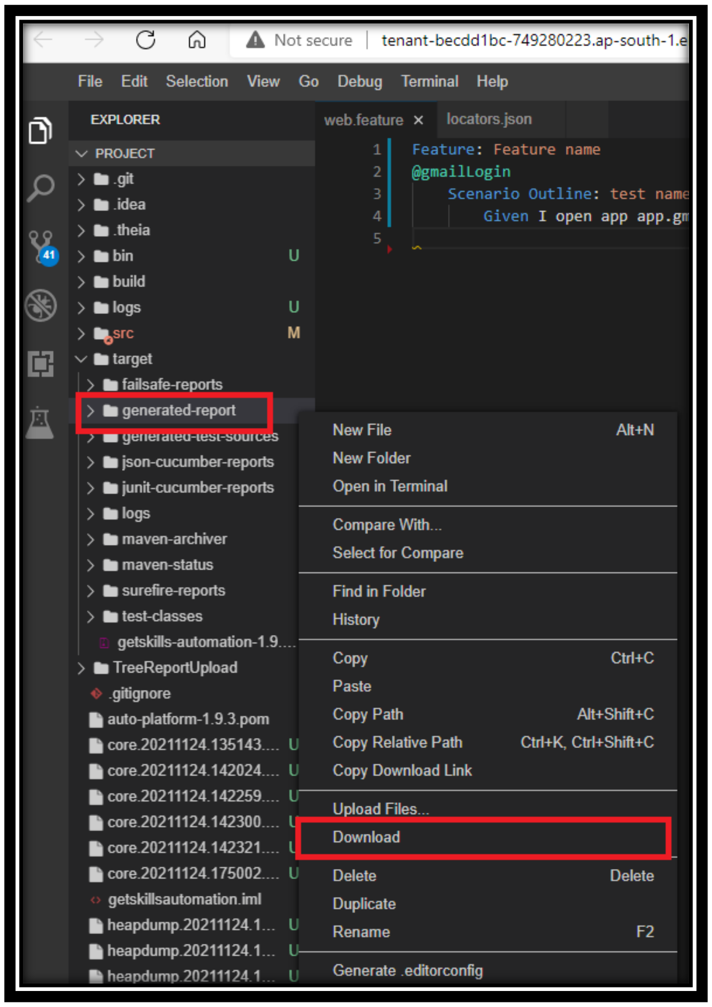
Figure 29: Screenshots in auto-generated report

It shows the page that we are getting on login.

Reviewing Report

To review the reports please follow the following steps:

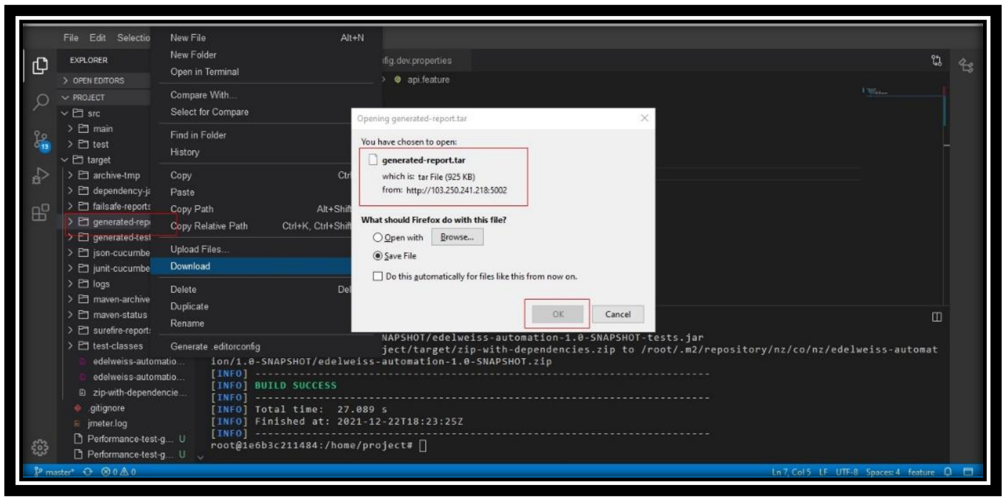
1. Right-click on target/generated-report as shown below



```
Feature: Feature name
@gmailLogin
    Scenario Outline: test name
        Given I open app app.gm
```

Figure30: Download Auto-generated report

2. Open generated report.tar and Save file and then Click on ok button



Opening generated-report.tar

You have chosen to open:

generated-report.tar

which is: tar File (925 KB)

from: http://103.250.241.218:5002

What should Firefox do with this file?

( ) Open with [Browse...]

(●) Save File

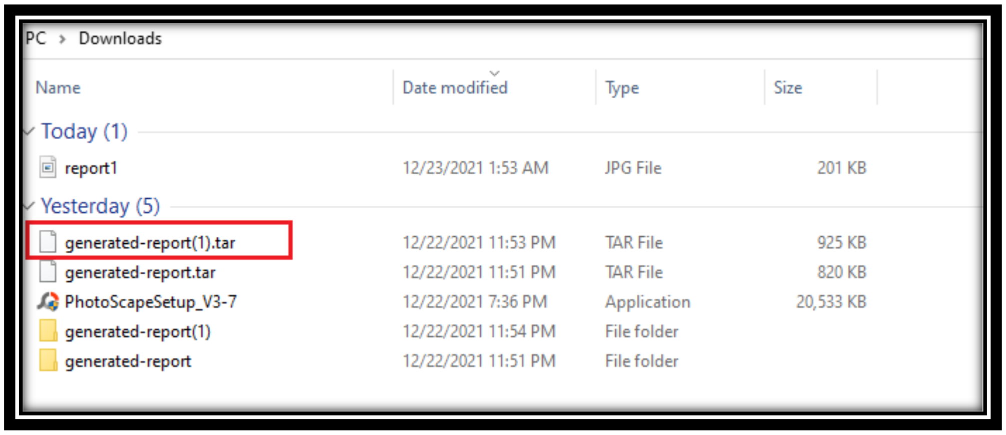
[ ] Do this automatically for files like this from now on.

[OK] [Cancel]

```
NAPSHOT/edelweiss-automation-1.0-SNAPSHOT-tests.jar
ject/target/zip-with-dependencies.zip to /root/.m2/repository/nz/co/nz/edelweiss-automat
ion/1.0-SNAPSHOT/edelweiss-automation-1.0-SNAPSHOT.zip
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  27.089 s
[INFO] Finished at: 2021-12-22T18:23:25Z
[INFO] ------------------------------------------------------------------------
root@1e6b3c211484:/home/project# 
```

Figure31: Auto-generated report

3. A file will be downloaded as shown below



PC › Downloads

Name	Date modified	Type	Size
Today (1)			
report1	12/23/2021 1:53 AM	JPG File	201 KB
Yesterday (5)			
generated-report(1).tar	12/22/2021 11:53 PM	TAR File	925 KB
generated-report.tar	12/22/2021 11:51 PM	TAR File	820 KB
PhotoScapeSetup_V3-7	12/22/2021 7:36 PM	Application	20,533 KB
generated-report(1)	12/22/2021 11:54 PM	File folder	
generated-report	12/22/2021 11:51 PM	File folder	

Figure32: Downloaded auto-generated report

4. Extract this file in your folder as shown below

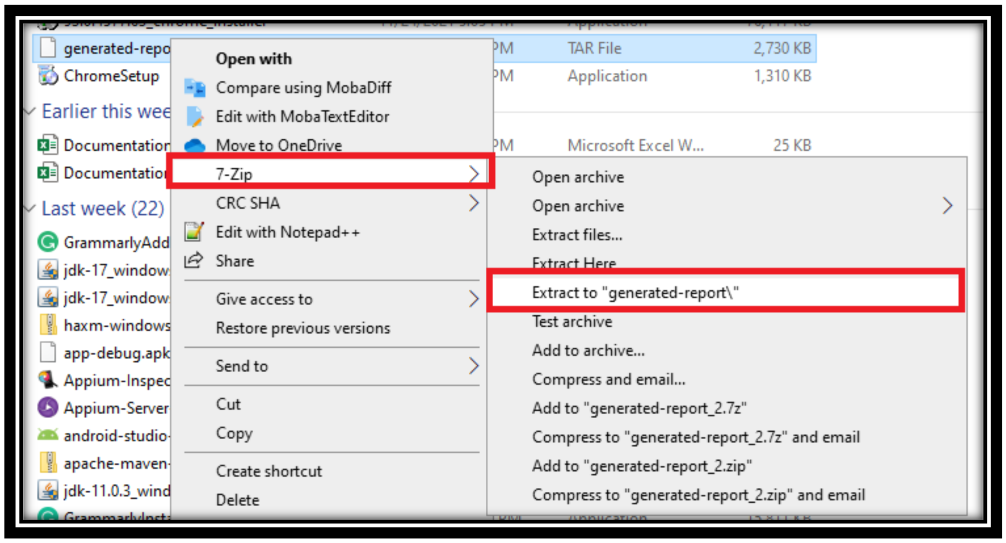
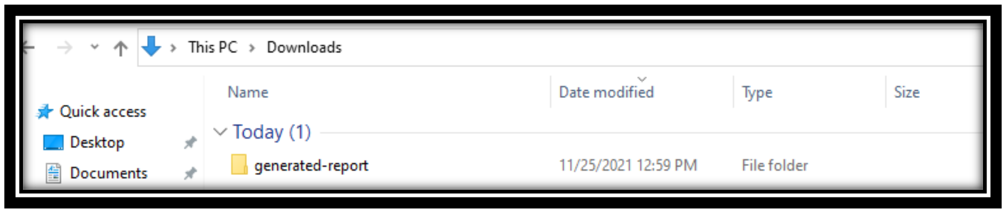


Figure33: Extract to auto-generated report

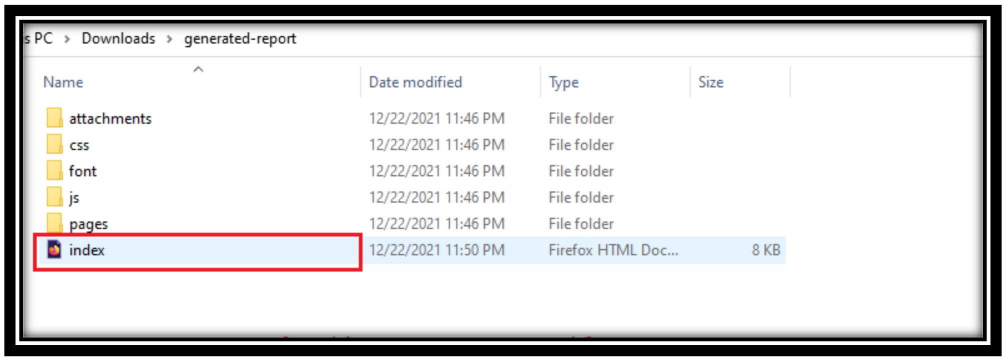
5. After the extraction, a folder name **generated-repor**t will appear in the same directory as shown below,



Name	Date modified	Type	Size
generated-report	11/25/2021 12:59 PM	File folder	

Figure 34: Downloaded file

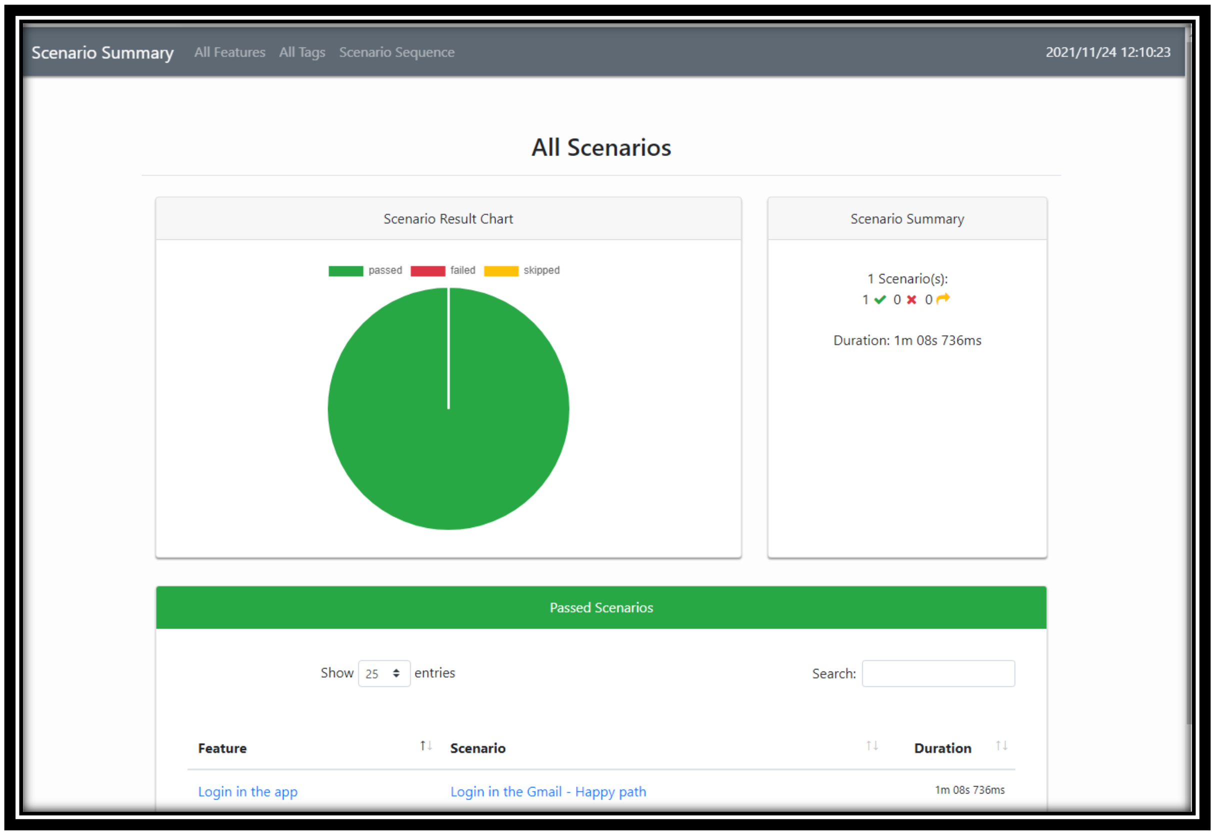
6. Inside this folder open the index file in the browser to see the report.



Name	Date modified	Type	Size
attachments	12/22/2021 11:46 PM	File folder	
css	12/22/2021 11:46 PM	File folder	
font	12/22/2021 11:46 PM	File folder	
js	12/22/2021 11:46 PM	File folder	
pages	12/22/2021 11:46 PM	File folder	
index	12/22/2021 11:50 PM	Firefox HTML Doc...	8 KB

Figure 35: Screenshots in auto-generated report

7. Here is example of the report shown below



All Scenarios

Scenario Result Chart

Scenario Summary

1 Scenario(s):
1 ✓ 0 ✗ 0

Duration: 1m 08s 736ms

Passed Scenarios

Show 25 entries Search:

Feature	Scenario	Duration
Login in the app	Login in the Gmail - Happy path	1m 08s 736ms

Figure 36: Scenario Report

REFERENCES

www.javatpoint.com

www.w3schools.com

www.geeksforgeeks.com