Selenium Cucumber Maven Hybrid Test Automation Framework

Important note:

- This framework is based on different youtube tutorials. Link you can find below. You can clone this repository and follow the instructions in this documentation.
- After the development is done in your local machine you can push your Project in GitHub. The instruction, you can find in this documentation.
- To run the job in Jenkins, you can find the instruction at the end of this documentation.

Repository: https://github.com/RishikeshShah/JavaCucumberSelenium.git

Autor: Rishikesh Shah (Rishikesh.shah@msg.group)

This Project is based on different Tutorial:
https://www.youtube.com/watch?v=aFlAXLSHbCg&t=5322s&ab_channel=SDET-QA

https://www.youtube.com/watch?v=aFlAXLSHbCg&list=PLUDwpEzHYYLuOleK8iPl6kc2UbXGvllBY&ab_channel=SDET-QA

Prerequisites:

- Java und maven should be installed.
- Maven und Java home should be configured.
- IDE: Eclipse or Intellij (In my case I installed Intellij community version)

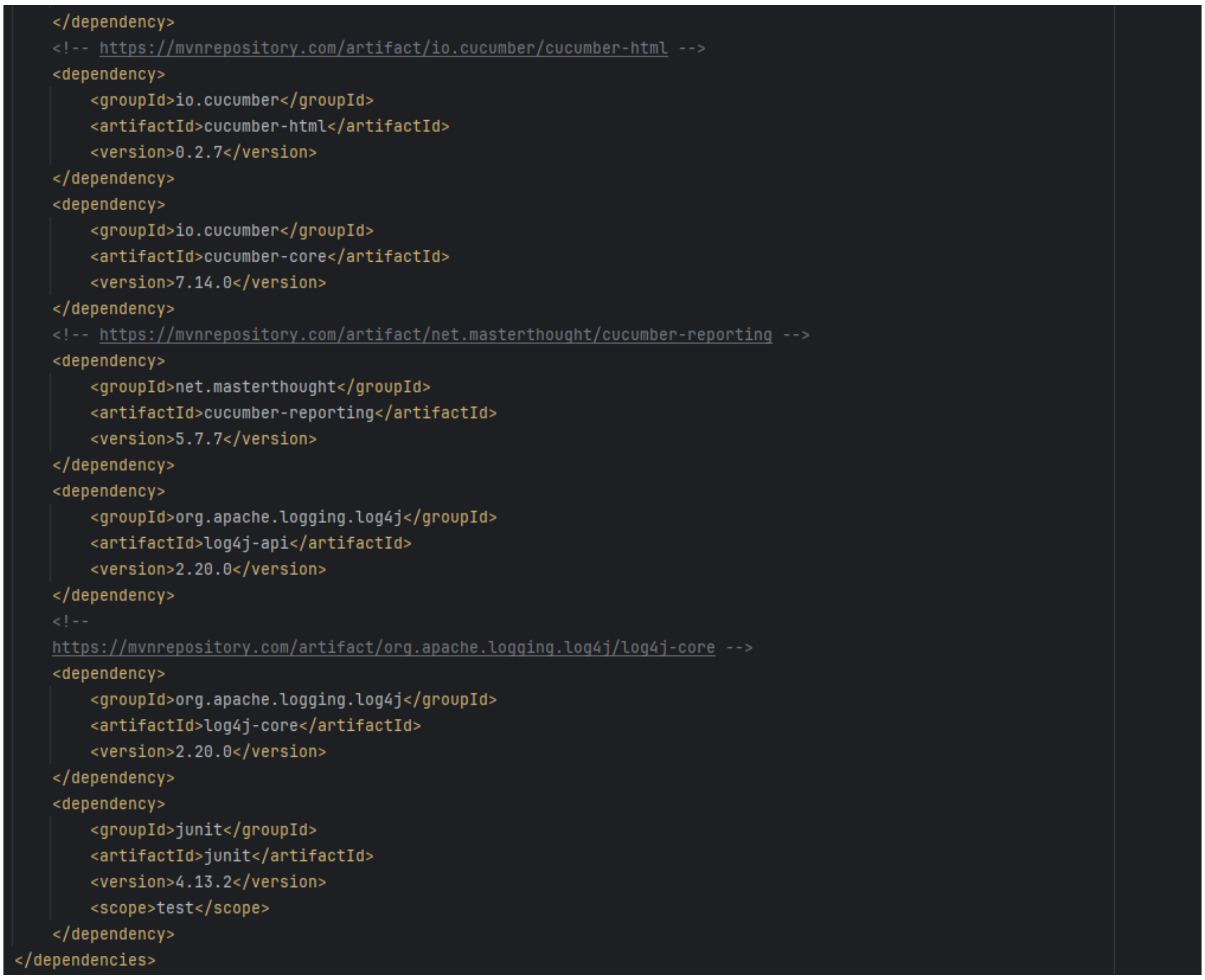
Installation maven dependency:

Add Maven Dependencies:

Create a maven project and search the following dependencies in MVN Repository and paste in pom.xml file. The version of dependencies are new at the time of creating this project. You can install the other version as well. (You can see the version in the repository link at the top)

Link for MVN Repository: https://mvnrepository.com/

- Selenium-java (groupId: org.seleniumhq.selenium)
- Webdrivermanager (groupId: io.github.bonigarcia)
- Cucumber-java (groupId: io.cucumber)
- cucumber-junit (groupId: io.cucumber)
- cucumber-html (groupId; io.cucumber)
- cucumber-core (groupId: io.cucumber)
- cucumber-reporting (groupId: netmasterthought)
- log4j-api (groupId: org.apache.logging.log4j)
- log4j-core (groupId: org.apache.logging.log4j)
- junit (groupId: junit)



```
    </dependency>
    <!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-html -->
    <dependency>
        <groupId>io.cucumber</groupId>
        <artifactId>cucumber-html</artifactId>
        <version>0.2.7</version>
    </dependency>
    <dependency>
        <groupId>io.cucumber</groupId>
        <artifactId>cucumber-core</artifactId>
        <version>7.14.0</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/net.masterthought/cucumber-reporting -->
    <dependency>
        <groupId>net.masterthought</groupId>
        <artifactId>cucumber-reporting</artifactId>
        <version>5.7.7</version>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-api</artifactId>
        <version>2.20.0</version>
    </dependency>
    <!--
    https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-core -->
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-core</artifactId>
        <version>2.20.0</version>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.13.2</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

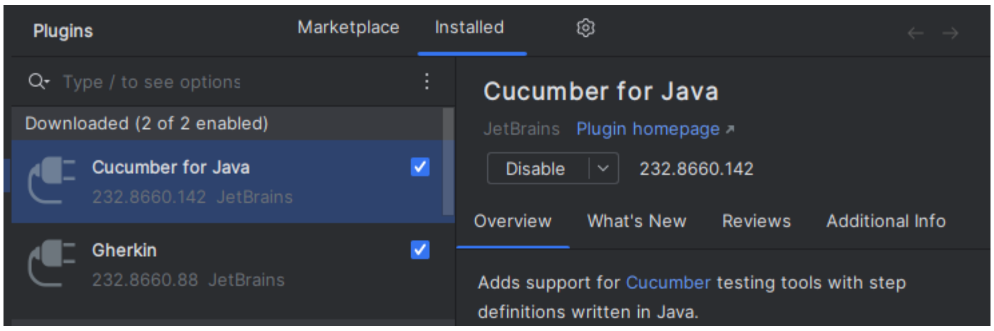
Plugins

You need to install two plugins so that compiler can recognize Gherkins keyword.

- Cucumber for java
- Gherkin

Steps to install plugin.

- Click on IDE and Project setting icon on the top right corner
- Click plugins.
- Click option Marketplace.
- Type in the search field “cucumber for java”
- Click on install.
- Restart IntelliJ
- Similarly install Gherkins also
- To check if they are installed or not: click on Installed button then you will be able to see the tick mark in both plugins like below.


Plugins Marketplace Installed

Type / to see options

Downloaded (2 of 2 enabled)

Cucumber for Java ✓
232.8660.142 JetBrains

Gherkin ✓
232.8660.88 JetBrains

Cucumber for Java

JetBrains Plugin homepage ↗

Disable 232.8660.142

Overview What's New Reviews Additional Info

Adds support for Cucumber testing tools with step definitions written in Java.

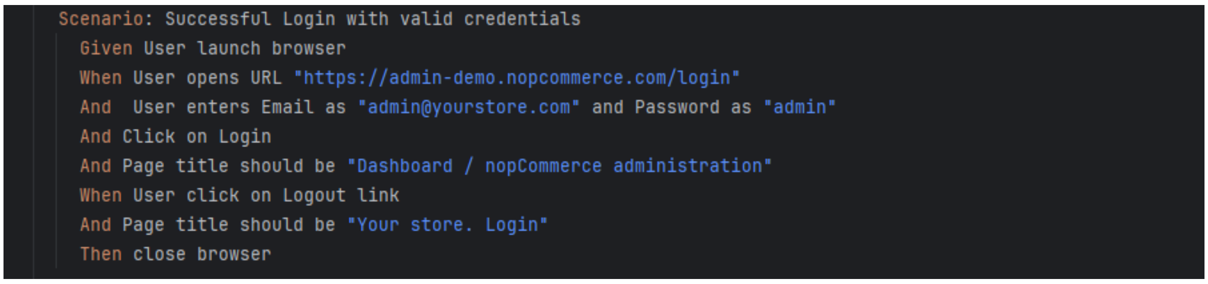
Gherkin Syntax:

Gherkin Syntax is written in plain English language with the following keyword. These Gherkins syntax is written in a file within a project with extension <filename>.feature

- Freature
- Background
- Scenario
- Scenario Outline with Examples
- Given, When, Then, And, But
- <Tags Name>

Now you are ready to write the test script. Let's begin with simple login feature.

- Step1: Create a feature file. You can create this file in Resource folder of src/test/java . I have created a separate folder called Features at project level. In that folder file called login.feature and customers.feature are created.
- Step2: Write the login scenario for test step in Gherkin syntax.



```
Scenario: Successful Login with valid credentials
  Given User launch browser
  When User opens URL "https://admin-demo.nopcommerce.com/login"
  And  User enters Email as "admin@yourstore.com" and Password as "admin"
  And Click on Login
  And Page title should be "Dashboard / nopCommerce administration"
  When User click on Logout link
  And Page title should be "Your store. Login"
  Then close browser
```

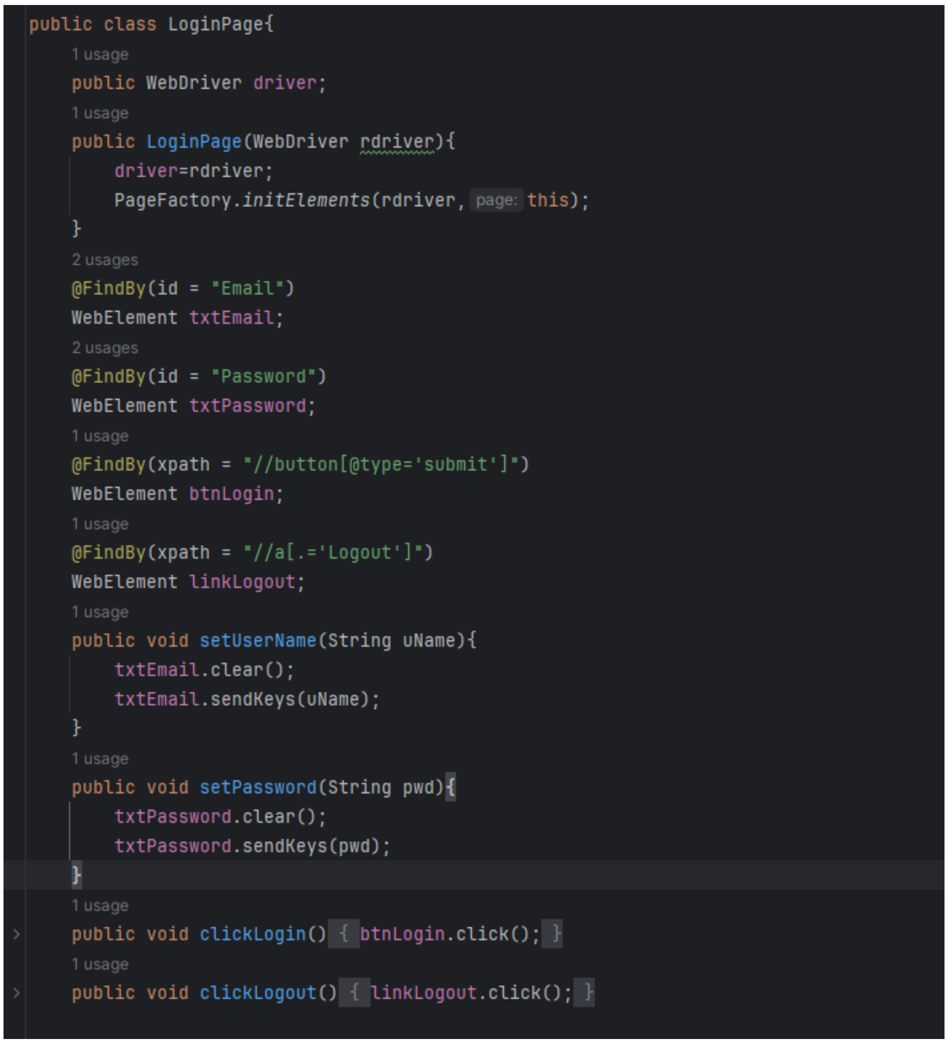
-
- Step3: Create four packages pageObject, utilities, stepDefinitions and testRunner in src/test/java
- Step4: Create a java class in pageObject package. (In my case pageObject/loginPage.java)
- Step5: Create two java classes in stepDefinitions package. (In my case stepDefinitions/BaseClass.java and StepDefinitions/Steps.java)
- Step6: Create a java class in testRunner package. (In my case testRunner/TestRunner.java)

Step7: You can implement this Gherkin syntax in two ways.

- **First approach**: Run the feature file and copy the unimplemented method from console and paste in Step.java. Write the actual code now.
- **Second approach**: In feature file hover over the mouse to any of the yellow underlined sentences
- Click on more actions

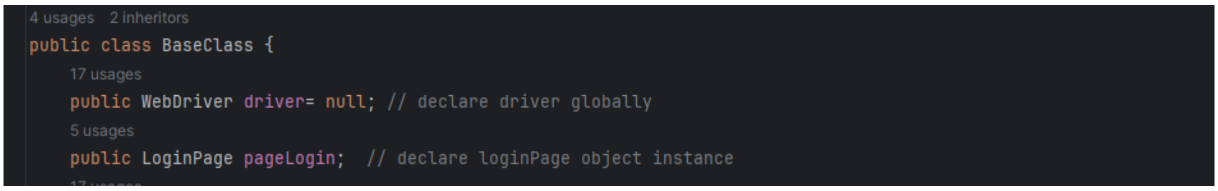
- Click on create all step definitions
- Either create a new class or select steps.java
- Write the actual code now.

Step 8: In LoginPage.java, write the code for page object and action methode on that page object.



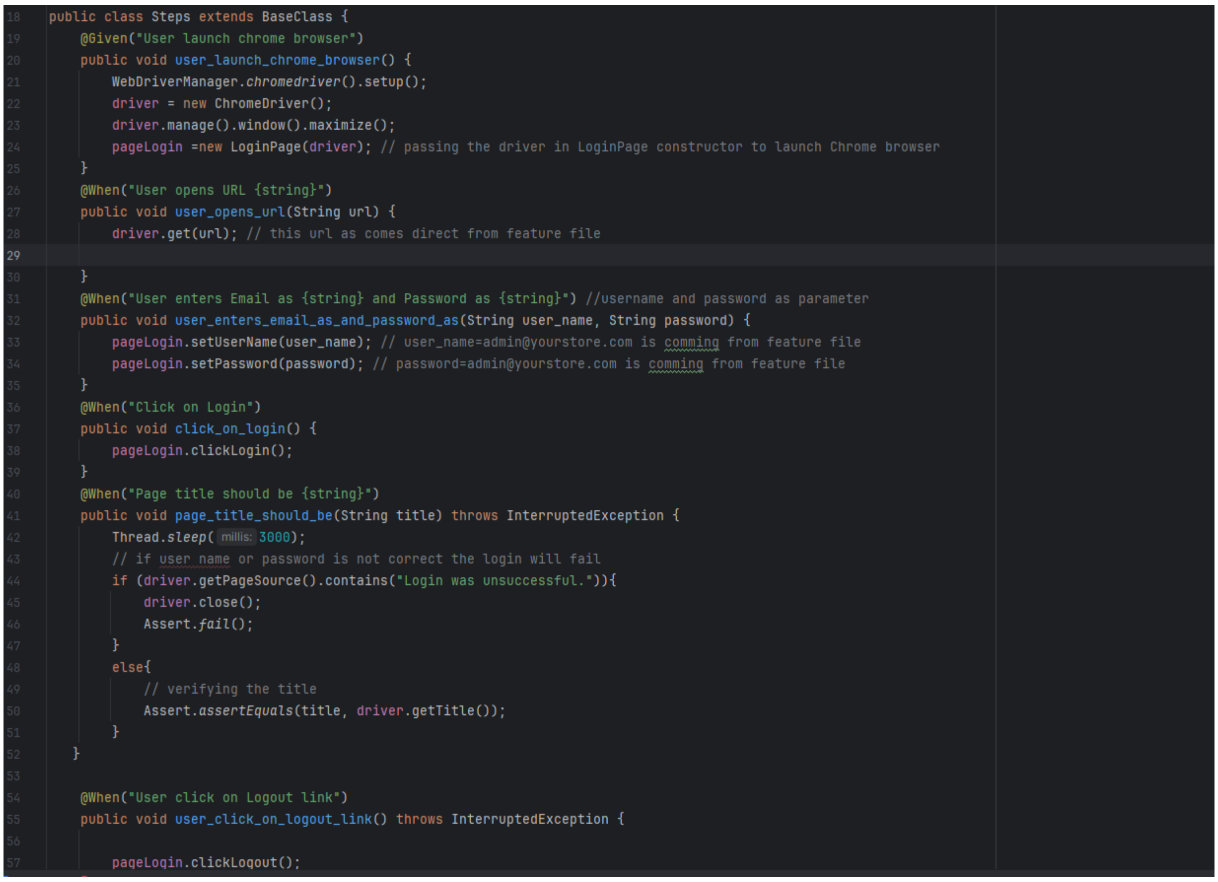
```java
public class LoginPage{
    1 usage
    public WebDriver driver;
    1 usage
    public LoginPage(WebDriver rdriver){
        driver=rdriver;
        PageFactory.initElements(rdriver, page: this);
    }
    2 usages
    @FindBy(id = "Email")
    WebElement txtEmail;
    2 usages
    @FindBy(id = "Password")
    WebElement txtPassword;
    1 usage
    @FindBy(xpath = "//button[@type='submit']")
    WebElement btnLogin;
    1 usage
    @FindBy(xpath = "//a[.='Logout']")
    WebElement linkLogout;
    1 usage
    public void setUserName(String uName){
        txtEmail.clear();
        txtEmail.sendKeys(uName);
    }
    1 usage
    public void setPassword(String pwd){
        txtPassword.clear();
        txtPassword.sendKeys(pwd);
    }
    1 usage
    public void clickLogin() { btnLogin.click(); }
    1 usage
    public void clickLogout() { linkLogout.click(); }
```

Step 9: declare the driver and instance of LoginPage.java in BaseClass.java.



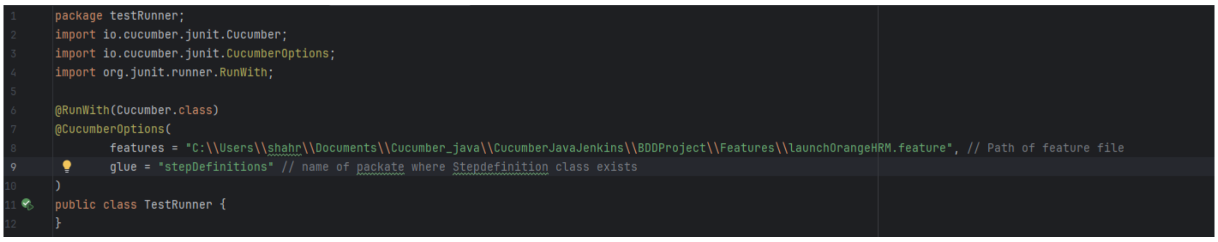
```java
4 usages  2 inheritors
public class BaseClass {
    17 usages
    public WebDriver driver= null; // declare driver globally
    5 usages
    public LoginPage pageLogin;  // declare loginPage object instance
```

Step 9: Extend Step.java with BaseClass.java and implement the code for login.



```java
public class Steps extends BaseClass {
    @Given("User launch chrome browser")
    public void user_launch_chrome_browser() {
        WebDriverManager.chromedriver().setup();
        driver = new ChromeDriver();
        driver.manage().window().maximize();
        pageLogin =new LoginPage(driver); // passing the driver in LoginPage constructor to launch Chrome browser
    }
    @When("User opens URL {string}")
    public void user_opens_url(String url) {
        driver.get(url); // this url as comes direct from feature file

    }
    @When("User enters Email as {string} and Password as {string}") //username and password as parameter
    public void user_enters_email_as_and_password_as(String user_name, String password) {
        pageLogin.setUserName(user_name); // user_name=admin@yourstore.com is comming from feature file
        pageLogin.setPassword(password); // password=admin@yourstore.com is comming from feature file
    }
    @When("Click on Login")
    public void click_on_login() {
        pageLogin.clickLogin();
    }
    @When("Page title should be {string}")
    public void page_title_should_be(String title) throws InterruptedException {
        Thread.sleep( millis: 3000);
        // if user name or password is not correct the login will fail
        if (driver.getPageSource().contains("Login was unsuccessful.")){
            driver.close();
            Assert.fail();
        }
        else{
            // verifying the title
            Assert.assertEquals(title, driver.getTitle());
        }
   }

    @When("User click on Logout link")
    public void user_click_on_logout_link() throws InterruptedException {

        pageLogin.clickLogout();
```

In TestRunner.java write the following code and execute TestRunner.java class. Login implementation is now successful.



```java
package testRunner;
import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(
        features = "C:\\Users\\shahr\\Documents\\Cucumber_java\\CucumberJavaJenkins\\BDDProject\\Features\\launchOrangeHRM.feature", // Path of feature file
        glue = "stepDefinitions" // name of packate where Stepdefinition class exists
)
public class TestRunner {
}
```

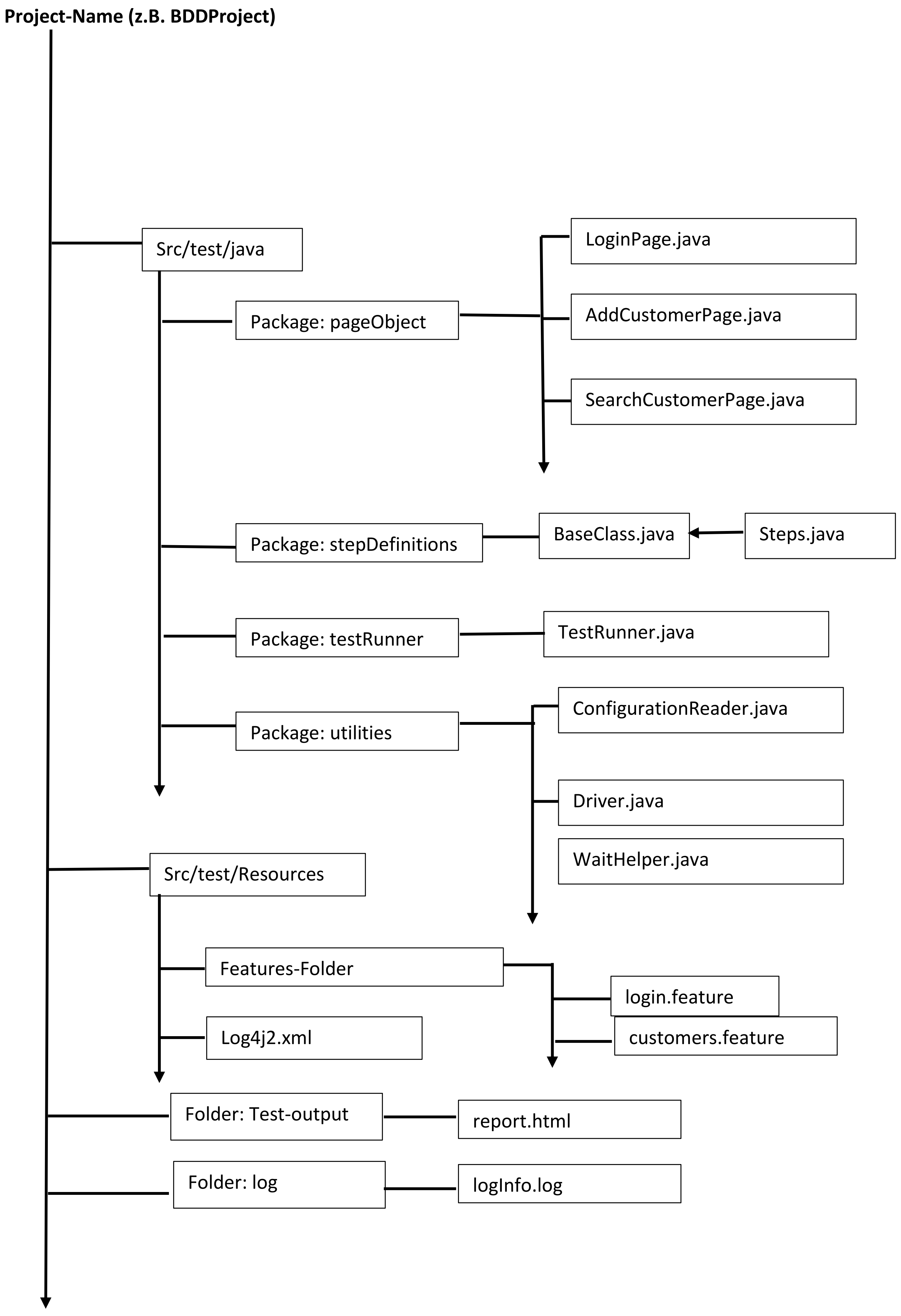
Test Framework with Page Object Model (POM):

Page Object Model (POM):

The Page Object Model (POM) is a design pattern used in test automation for web applications. It is particularly popular in the context of Selenium, a widely used tool for automating web browsers. The main goal of the Page Object Model is to create an abstraction layer that separates the test logic from the details of the user interface (UI).

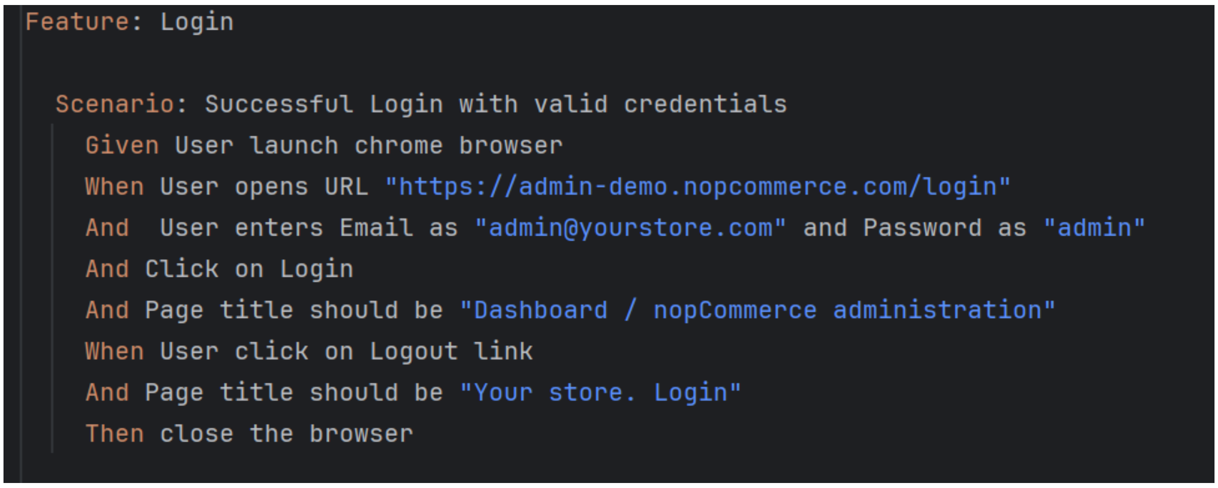
Page Objects: Each web page in the application has a corresponding Page Object. A Page Object represents the services that the page provides and the interactions that can be performed on that page. It encapsulates the details of the UI, such as the HTML structure and element locators. For example, in our project we have three Page Object Classes namely: AddCustorerPage.java, LoginPage.java and SearchCustomerPage.java. We have the defined the Object of that page and action method on that page object which can be called from class Step.java.

Test framework design is in next page.


Project-Name (z.B. BDDProject)
Src/test/java
Package: pageObject
LoginPage.java
AddCustomerPage.java
SearchCustomerPage.java
Package: stepDefinitions
BaseClass.java
Steps.java
Package: testRunner
TestRunner.java
Package: utilities
ConfigurationReader.java
Driver.java
WaitHelper.java
Src/test/Resources
Features-Folder
login.feature
customers.feature
Log4j2.xml
Folder: Test-output
report.html
Folder: log
logInfo.log

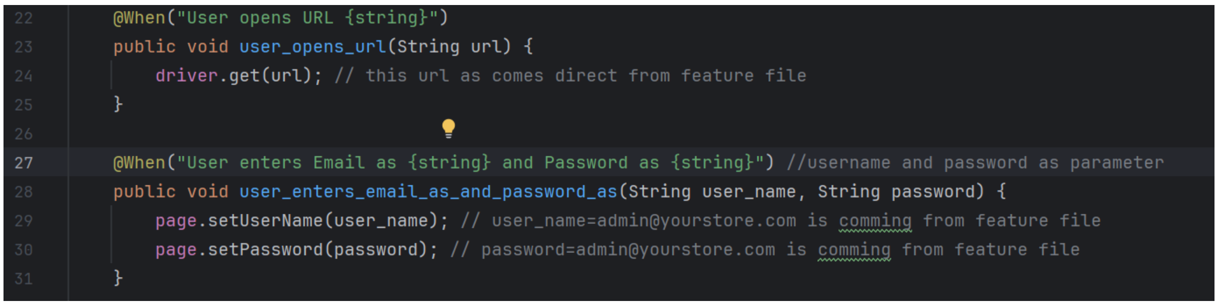
Parameter in feature file:

We can pass the test data directly from the feature file to the step definition java class. Here all the blue coded line are the parameter which will be passed to the step definition directly.



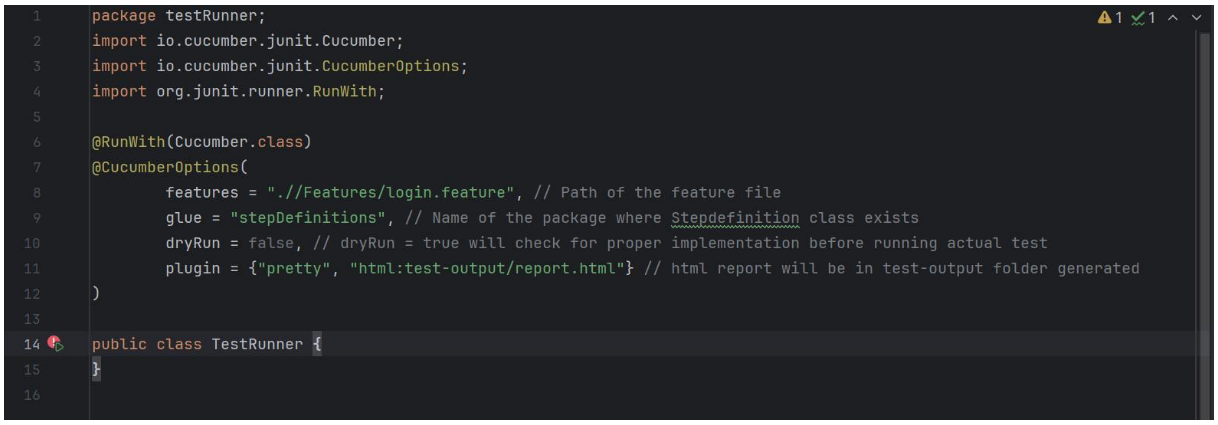
```
Feature: Login

  Scenario: Successful Login with valid credentials
    Given User launch chrome browser
    When User opens URL "https://admin-demo.nopcommerce.com/login"
    And  User enters Email as "admin@yourstore.com" and Password as "admin"
    And Click on Login
    And Page title should be "Dashboard / nopCommerce administration"
    When User click on Logout link
    And Page title should be "Your store. Login"
    Then close the browser
```

Implementation in step definition class.



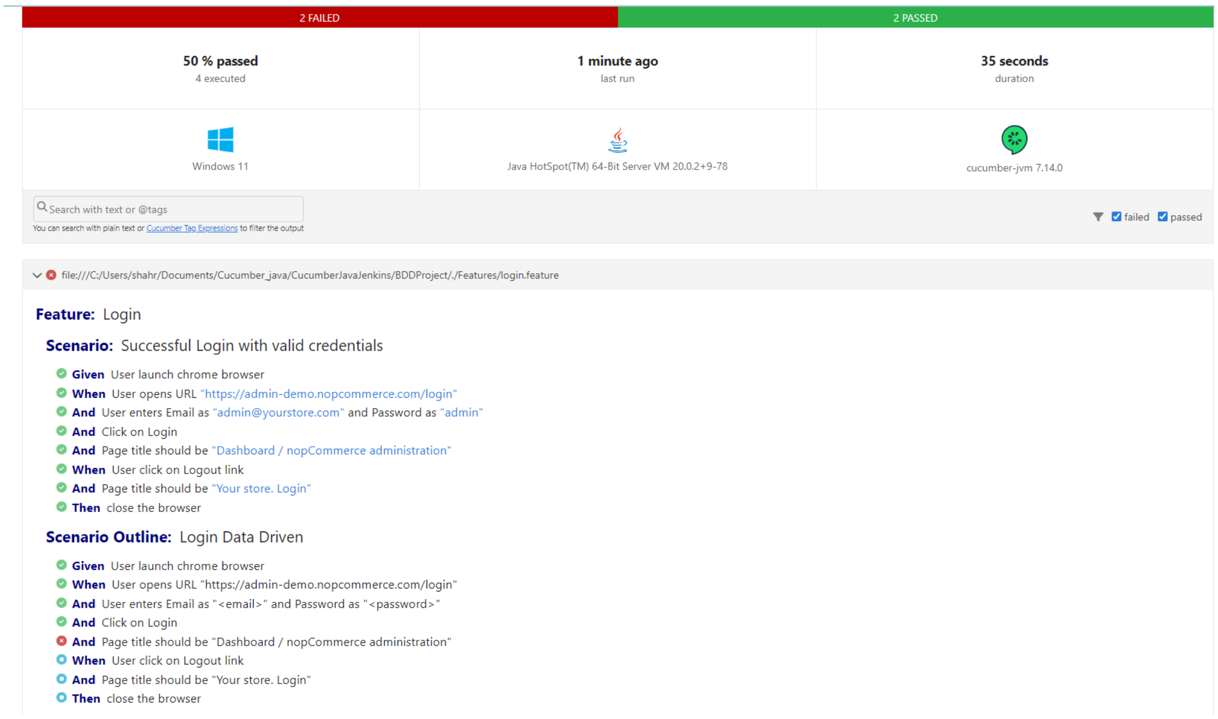
```
    @When("User opens URL {string}")
    public void user_opens_url(String url) {
        driver.get(url); // this url as comes direct from feature file
    }

    @When("User enters Email as {string} and Password as {string}") //username and password as parameter
    public void user_enters_email_as_and_password_as(String user_name, String password) {
        page.setUserName(user_name); // user_name=admin@yourstore.com is comming from feature file
        page.setPassword(password); // password=admin@yourstore.com is comming from feature file
    }
```

Implementation of TestRunner class:



```
package testRunner;
import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(
        features = ".//Features/login.feature", // Path of the feature file
        glue = "stepDefinitions", // Name of the package where Stepdefinition class exists
        dryRun = false, // dryRun = true will check for proper implementation before running actual test
        plugin = {"pretty", "html:test-output/report.html"} // html report will be in test-output folder generated
)

public class TestRunner {
}
```

Html Report:



2 FAILED 2 PASSED

50 % passed
4 executed

1 minute ago
last run

35 seconds
duration

Windows 11

Java HotSpot(TM) 64-Bit Server VM 20.0.2+9-78

cucumber-jvm 7.14.0

Search with text or @tags
You can search with plain text or Cucumber Tag Expressions to filter the output

failed passed

file:///C:/Users/shahr/Documents/Cucumber_java/CucumberJavaJenkins/BDDProject/./Features/login.feature

Feature: Login

Scenario: Successful Login with valid credentials

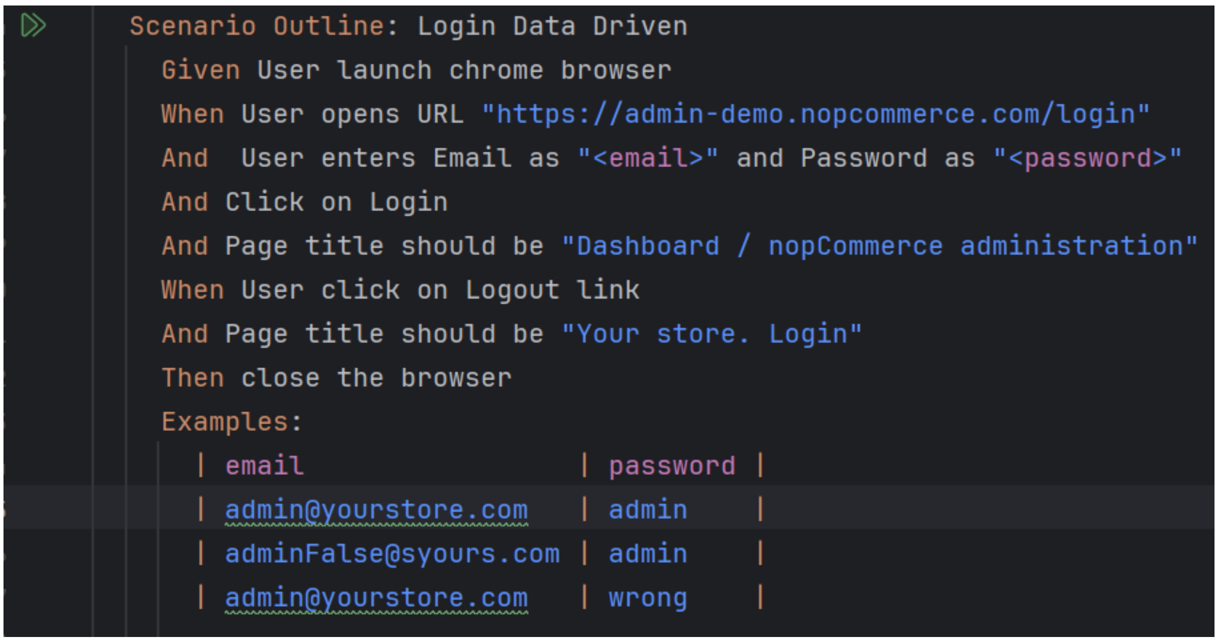
- **Given** User launch chrome browser
- **When** User opens URL "https://admin-demo.nopcommerce.com/login"
- **And** User enters Email as "admin@yourstore.com" and Password as "admin"
- **And** Click on Login
- **And** Page title should be "Dashboard / nopCommerce administration"
- **When** User click on Logout link
- **And** Page title should be "Your store. Login"
- **Then** close the browser

Scenario Outline: Login Data Driven

- **Given** User launch chrome browser
- **When** User opens URL "https://admin-demo.nopcommerce.com/login"
- **And** User enters Email as "<email>" and Password as "<password>"
- **And** Click on Login
- **And** Page title should be "Dashboard / nopCommerce administration"
- **When** User click on Logout link
- **And** Page title should be "Your store. Login"
- **Then** close the browser

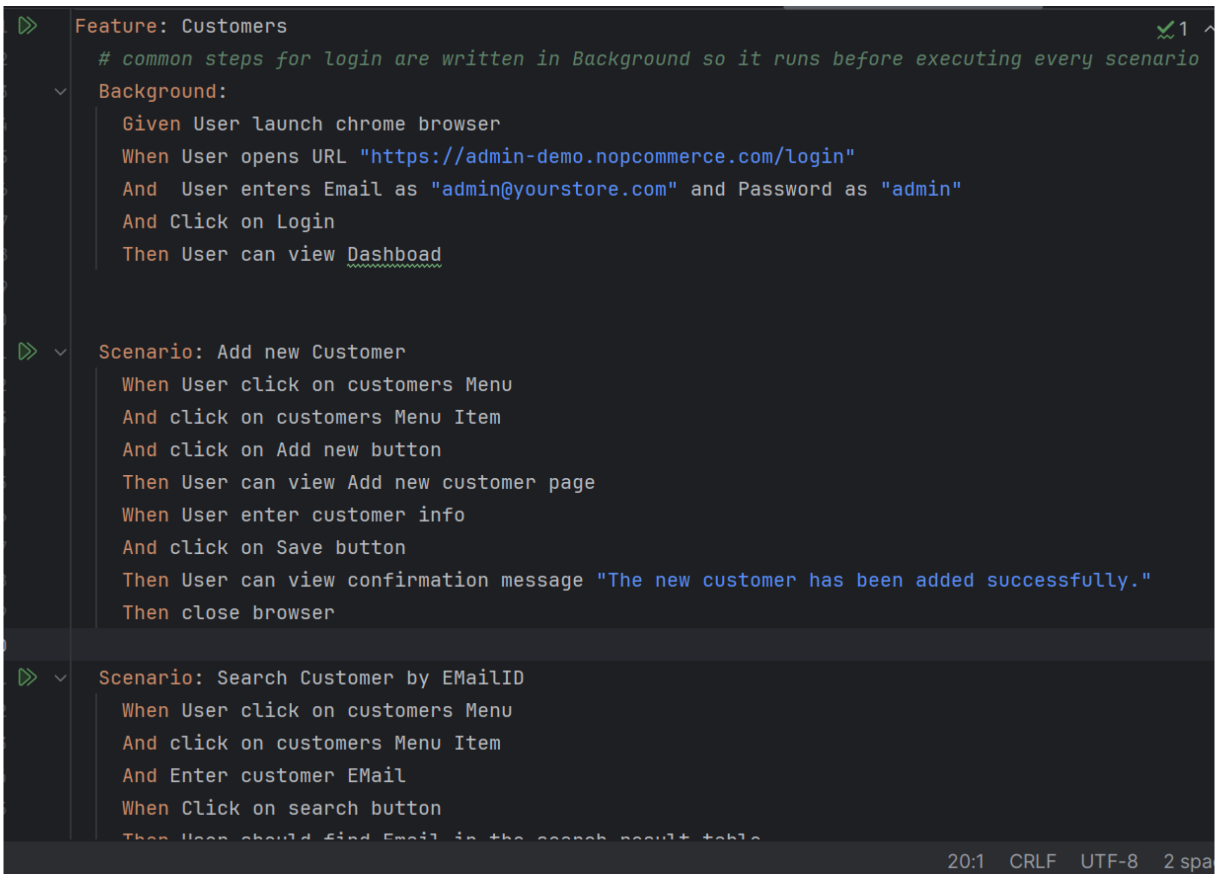
Scenario: A scenario represents a specific test case. It describes a particular set of actions and expected outcomes in a human-readable format.

Scenario Outline: We can perform Data Driven Testing with Scenario outline. A scenario outline is a template for a scenario. It allows you to run the same scenario multiple times with different sets of data. It's particularly useful when you want to test the same functionality with various input values.



```
Scenario Outline: Login Data Driven
  Given User launch chrome browser
  When User opens URL "https://admin-demo.nopcommerce.com/login"
  And  User enters Email as "<email>" and Password as "<password>"
  And Click on Login
  And Page title should be "Dashboard / nopCommerce administration"
  When User click on Logout link
  And Page title should be "Your store. Login"
  Then close the browser
  Examples:
    | email                  | password |
    | admin@yourstore.com    | admin    |
    | adminFalse@syours.com  | admin    |
    | admin@yourstore.com    | wrong    |
```

Background: The background keyword allows you to define steps that should be run before each scenario in the feature file. This is useful for setting up common preconditions for all scenarios in a feature. Here the steps in Background are common for both scenario Add new Customer and Search Customer by EMailID.



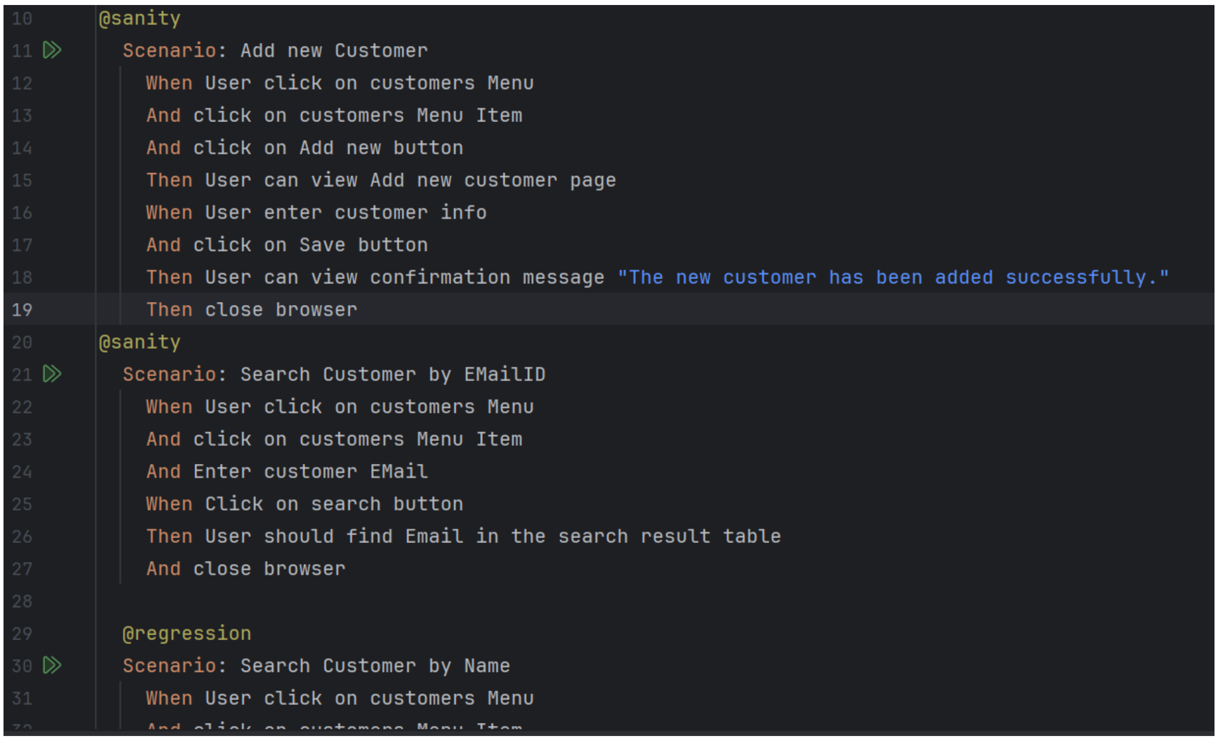
```
Feature: Customers
  # common steps for login are written in Background so it runs before executing every scenario
  Background:
    Given User launch chrome browser
    When User opens URL "https://admin-demo.nopcommerce.com/login"
    And  User enters Email as "admin@yourstore.com" and Password as "admin"
    And Click on Login
    Then User can view Dashboad


  Scenario: Add new Customer
    When User click on customers Menu
    And click on customers Menu Item
    And click on Add new button
    Then User can view Add new customer page
    When User enter customer info
    And click on Save button
    Then User can view confirmation message "The new customer has been added successfully."
    Then close browser

  Scenario: Search Customer by EMailID
    When User click on customers Menu
    And click on customers Menu Item
    And Enter customer EMail
    When Click on search button
```

20:1 CRLF UTF-8 2 spa

Tags:

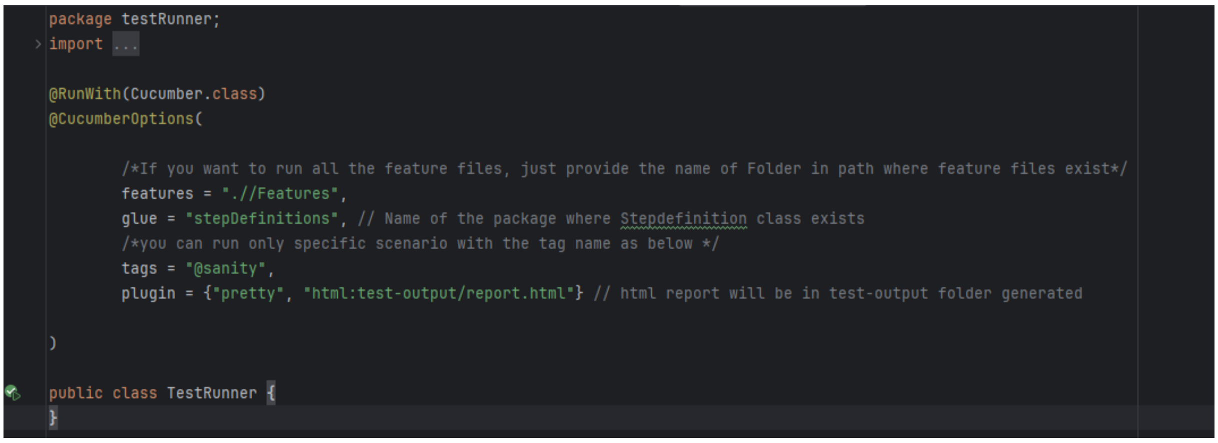
In Cucumber, tags are annotations that you can add to scenarios or features in the feature files. Tags are used to categorize and organize your scenarios, allowing you to run specific subsets of your scenarios based on these tags. Tags are preceded by the "@" symbol and are typically placed above the feature, scenario, or scenario outline in the feature file.

For example: we added two different tags @regression and @sanity for three different scenario. We can run only the scenario with regression or sanity tag from TestRunner.java.



```
@sanity
  Scenario: Add new Customer
    When User click on customers Menu
    And click on customers Menu Item
    And click on Add new button
    Then User can view Add new customer page
    When User enter customer info
    And click on Save button
    Then User can view confirmation message "The new customer has been added successfully."
    Then close browser
@sanity
  Scenario: Search Customer by EMailID
    When User click on customers Menu
    And click on customers Menu Item
    And Enter customer EMail
    When Click on search button
    Then User should find Email in the search result table
    And close browser

  @regression
  Scenario: Search Customer by Name
    When User click on customers Menu
    And click on customers Menu Item
```

Implementation:



```
package testRunner;
import ...

@RunWith(Cucumber.class)
@CucumberOptions(

        /*If you want to run all the feature files, just provide the name of Folder in path where feature files exist*/
        features = ".//Features",
        glue = "stepDefinitions", // Name of the package where Stepdefinition class exists
        /*you can run only specific scenario with the tag name as below */
        tags = "@sanity",
        plugin = {"pretty", "html:test-output/report.html"} // html report will be in test-output folder generated

)

public class TestRunner {
}
```

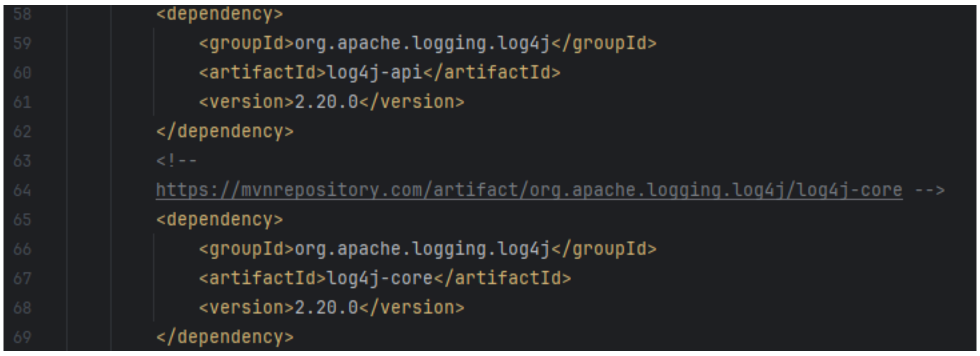
Logging with Log4j2:

Logging with Log4j refers to the process of incorporating the Apache Log4j library into your Java application to facilitate logging of messages at various levels of severity. Logging is an essential aspect of software development, helping developers to track the execution flow, identify issues, and gather information for troubleshooting and debugging.

Log4j provides a flexible and efficient logging framework for Java applications. It allows developers to control the output of log statements by configuring various aspects such as the log level, log format, and destination.

Steps to configure log4j2 in project.

- Add the log4j-core/2.20.0 and log4j-api/2.20.0 dependency to your pom.xml



```
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.20.0</version>
</dependency>
<!--
https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-core -->
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.20.0</version>
</dependency>
```

- Create a xml file with name log4j2.xml in resources Folder. (**If you create in any other folder the program will not be able to find log4j2.xml**)
- Paste the following text in log4j2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Extra logging related to initialization of Log4j.
 Set to debug or trace if log4j initialization is failing. -->
<Configuration status="WARN">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} %-5level -
%msg%n"/>
        </Console>
        <!-- location of file where the log file is to be placed-->
        <File name="File"
fileName="C:\Users\shahr\Documents\Cucumber_java\CucumberJavaJenkins\
BDDProject\log\logInfo.log" append="true">
            <!-- Log pattern looks something like; 16:32:35.416
[main] INFO  - Chrome browser launched successfully-->
            <PatternLayout>
                <Pattern>%d{HH:mm:ss.SSS} [%t] %-5level -
%msg%n</Pattern>
            </PatternLayout>
        </File>
    </Appenders>
    <Loggers>
        <Logger name="base" level="info" additivity="true">
            <AppenderRef ref="Console"/>
        </Logger>
        <!-- you can change the log level according to your need-->
        <Root level="info">
            <AppenderRef ref="File"/>
        </Root>
    </Loggers>
</Configuration>
```

Edit `<File name="File" fileName="<Location of folder to be saved>\logInfo.log" append="true">`

- Edit: <Logger name="<Package name>" level="info" additivity="true">
- In BaseClass.java LogManager and Logger

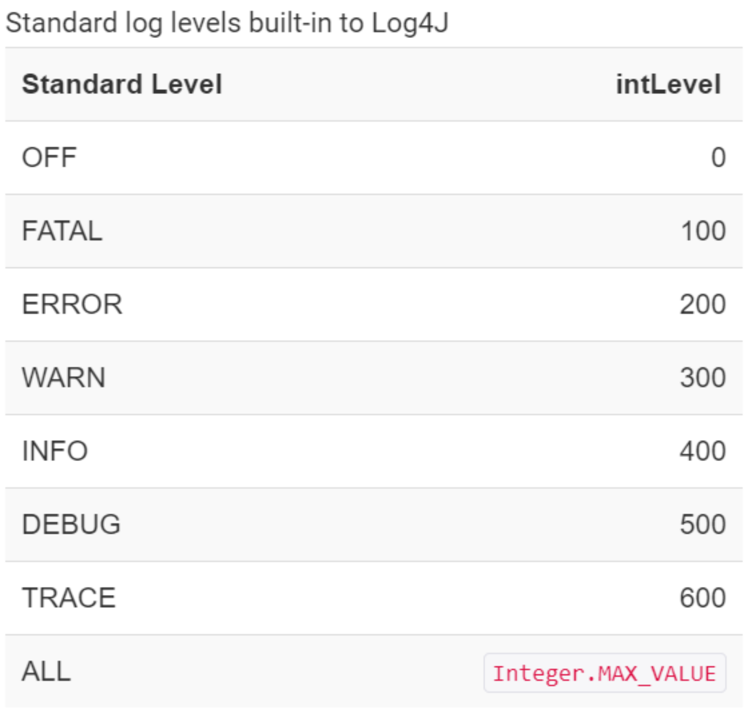
```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
```

At global level create variable for logger `public static Logger loger;`
In Setup method add line: `logger= LogManager.getLogger(BaseClass.class);`

- logger.info("Info message")

➢ Now your setup is complete. You can use logger.info(“”), logger.error(“”) etc. in any child class.

For more detail about log level:
https://logging.apache.org/log4j/2.x/manual/customloglevels.html



Standard log levels built-in to Log4J

Standard Level	intLevel
OFF	0
FATAL	100
ERROR	200
WARN	300
INFO	400
DEBUG	500
TRACE	600
ALL	`Integer.MAX_VALUE`

Implementation:

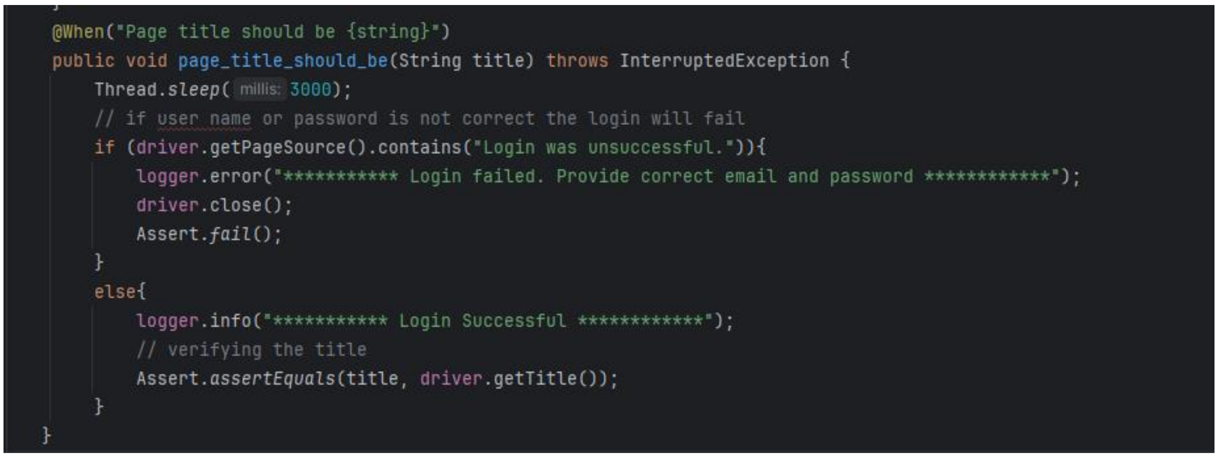
Declare logger in base class:

```
public Logger logger; // logger variable declaration
```

Initialize logger in setup() method of steps.java class.

```
logger = LogManager.getLogger(BaseClass.class); // initializing logger to
call different log levels
```

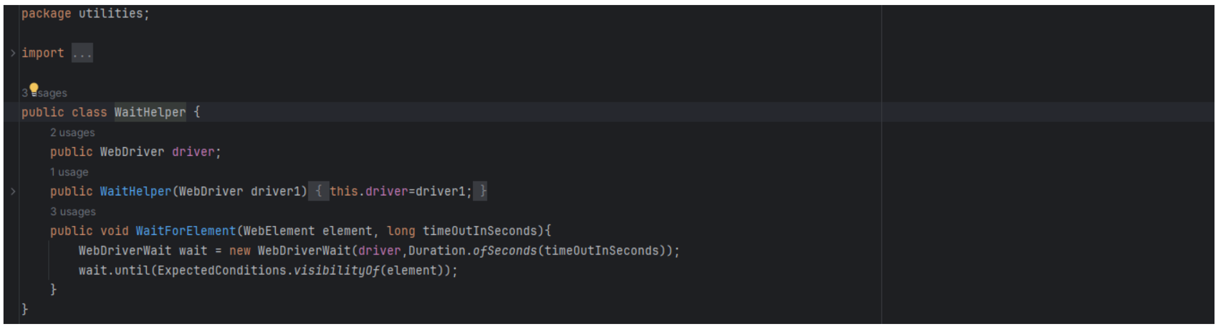
using logger in teststeps.



```
    @When("Page title should be {string}")
    public void page_title_should_be(String title) throws InterruptedException {
        Thread.sleep( millis: 3000);
        // if user name or password is not correct the login will fail
        if (driver.getPageSource().contains("Login was unsuccessful.")){
            logger.error("*********** Login failed. Provide correct email and password ************");
            driver.close();
            Assert.fail();
        }
        else{
            logger.info("*********** Login Successful ************");
            // verifying the title
            Assert.assertEquals(title, driver.getTitle());
        }
    }
```

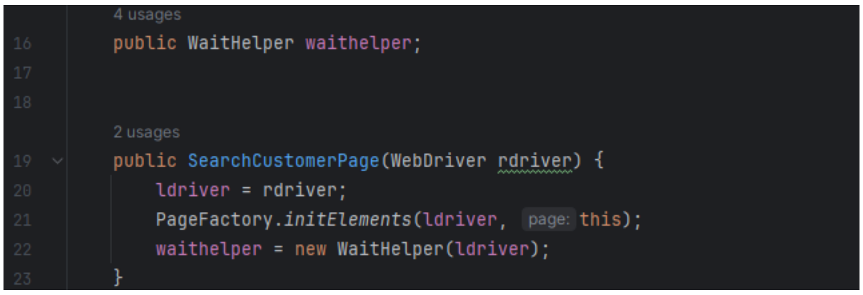
Dynamic waits:

You can create a WaitHelper.java class in utilities package and implement as below.

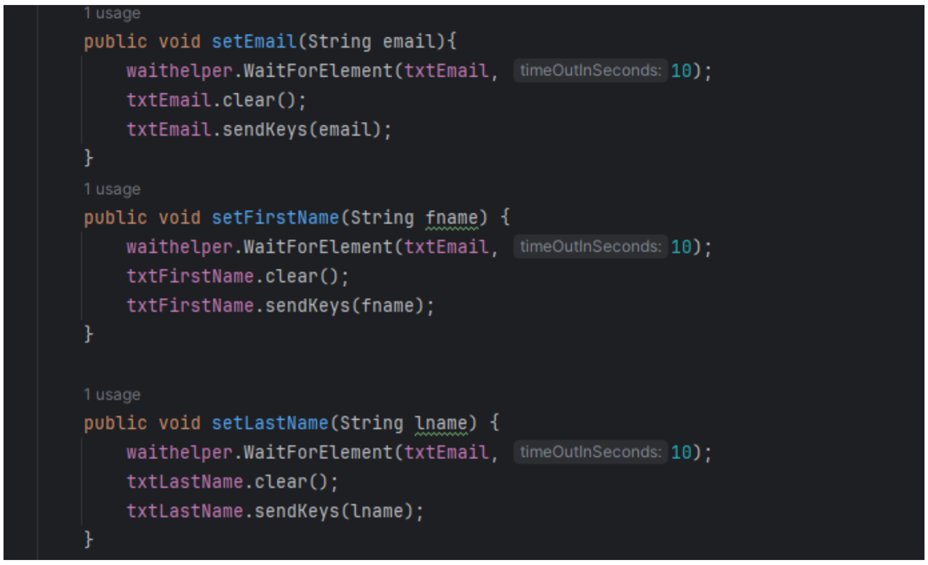


```java
package utilities;

import ...

3 usages
public class WaitHelper {
    2 usages
    public WebDriver driver;
    1 usage
    public WaitHelper(WebDriver driver1) { this.driver=driver1; }
    3 usages
    public void WaitForElement(WebElement element, long timeOutInSeconds){
        WebDriverWait wait = new WebDriverWait(driver,Duration.ofSeconds(timeOutInSeconds));
        wait.until(ExpectedConditions.visibilityOf(element));
    }
}
```

You can use it in action method where you have to wait explicitly for certain web element. See its application in SearchCustomerPage.java



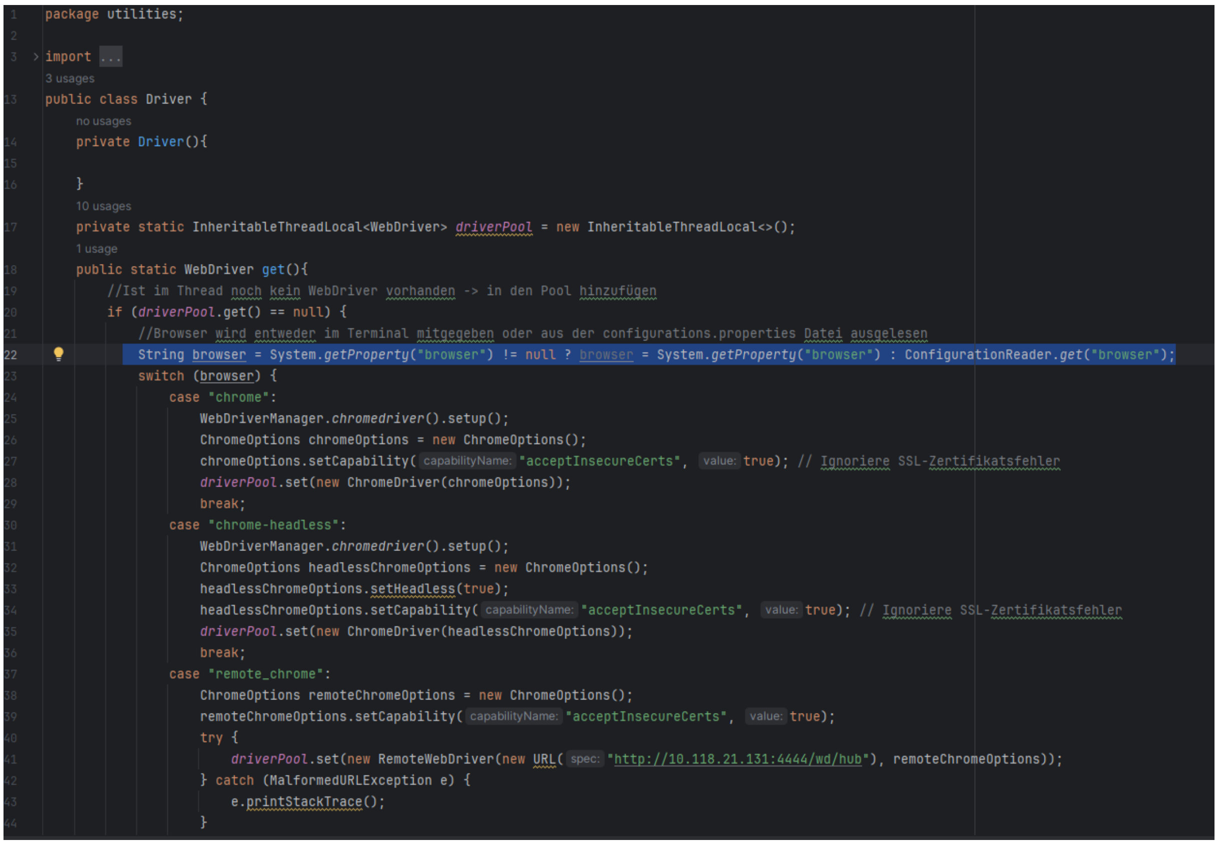
```java
    4 usages
    public WaitHelper waithelper;


    2 usages
    public SearchCustomerPage(WebDriver rdriver) {
        ldriver = rdriver;
        PageFactory.initElements(ldriver, page: this);
        waithelper = new WaitHelper(ldriver);
    }
```



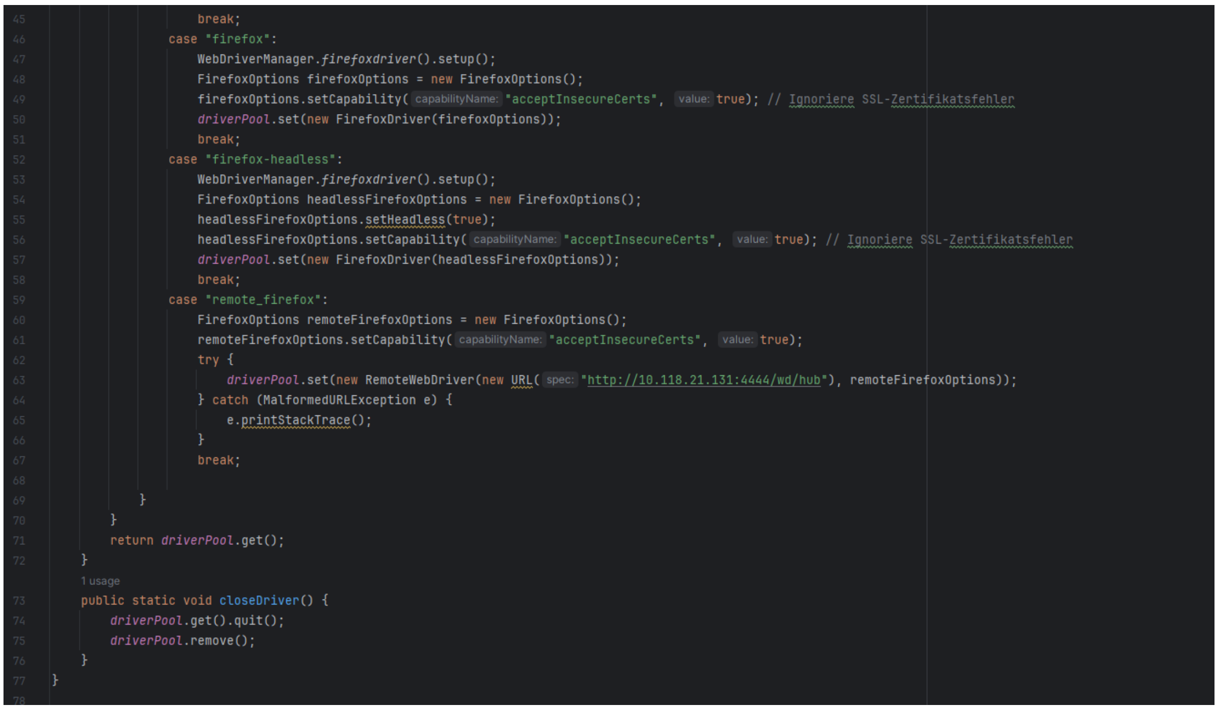
```java
1 usage
public void setEmail(String email){
    waithelper.WaitForElement(txtEmail, timeOutInSeconds: 10);
    txtEmail.clear();
    txtEmail.sendKeys(email);
}
1 usage
public void setFirstName(String fname) {
    waithelper.WaitForElement(txtEmail, timeOutInSeconds: 10);
    txtFirstName.clear();
    txtFirstName.sendKeys(fname);
}

1 usage
public void setLastName(String lname) {
    waithelper.WaitForElement(txtEmail, timeOutInSeconds: 10);
    txtLastName.clear();
    txtLastName.sendKeys(lname);
}
```

Configuration for different browser:

In this Project there are browser setting for chrome, chrome-headless, remote_chrome, firefox, firefox-headless and remote_firefox. The browser related setting is done in a extra class called Driver in utilities package.



```java
package utilities;

import ...
3 usages
public class Driver {
    no usages
    private Driver(){

    }
    10 usages
    private static InheritableThreadLocal<WebDriver> driverPool = new InheritableThreadLocal<>();
    1 usage
    public static WebDriver get(){
        //Ist im Thread noch kein WebDriver vorhanden -> in den Pool hinzufügen
        if (driverPool.get() == null) {
            //Browser wird entweder im Terminal mitgegeben oder aus der configurations.properties Datei ausgelesen
            String browser = System.getProperty("browser") != null ? browser = System.getProperty("browser") : ConfigurationReader.get("browser");
            switch (browser) {
                case "chrome":
                    WebDriverManager.chromedriver().setup();
                    ChromeOptions chromeOptions = new ChromeOptions();
                    chromeOptions.setCapability( capabilityName: "acceptInsecureCerts",  value: true); // Ignoriere SSL-Zertifikatsfehler
                    driverPool.set(new ChromeDriver(chromeOptions));
                    break;
                case "chrome-headless":
                    WebDriverManager.chromedriver().setup();
                    ChromeOptions headlessChromeOptions = new ChromeOptions();
                    headlessChromeOptions.setHeadless(true);
                    headlessChromeOptions.setCapability( capabilityName: "acceptInsecureCerts",  value: true); // Ignoriere SSL-Zertifikatsfehler
                    driverPool.set(new ChromeDriver(headlessChromeOptions));
                    break;
                case "remote_chrome":
                    ChromeOptions remoteChromeOptions = new ChromeOptions();
                    remoteChromeOptions.setCapability( capabilityName: "acceptInsecureCerts",  value: true);
                    try {
                        driverPool.set(new RemoteWebDriver(new URL( spec: "http://10.118.21.131:4444/wd/hub"), remoteChromeOptions));
                    } catch (MalformedURLException e) {
                        e.printStackTrace();
                    }
```



```java
                    break;
                case "firefox":
                    WebDriverManager.firefoxdriver().setup();
                    FirefoxOptions firefoxOptions = new FirefoxOptions();
                    firefoxOptions.setCapability( capabilityName: "acceptInsecureCerts",  value: true); // Ignoriere SSL-Zertifikatsfehler
                    driverPool.set(new FirefoxDriver(firefoxOptions));
                    break;
                case "firefox-headless":
                    WebDriverManager.firefoxdriver().setup();
                    FirefoxOptions headlessFirefoxOptions = new FirefoxOptions();
                    headlessFirefoxOptions.setHeadless(true);
                    headlessFirefoxOptions.setCapability( capabilityName: "acceptInsecureCerts",  value: true); // Ignoriere SSL-Zertifikatsfehler
                    driverPool.set(new FirefoxDriver(headlessFirefoxOptions));
                    break;
                case "remote_firefox":
                    FirefoxOptions remoteFirefoxOptions = new FirefoxOptions();
                    remoteFirefoxOptions.setCapability( capabilityName: "acceptInsecureCerts",  value: true);
                    try {
                        driverPool.set(new RemoteWebDriver(new URL( spec: "http://10.118.21.131:4444/wd/hub"), remoteFirefoxOptions));
                    } catch (MalformedURLException e) {
                        e.printStackTrace();
                    }
                    break;

            }
        }
        return driverPool.get();
    }
    1 usage
    public static void closeDriver() {
        driverPool.get().quit();
        driverPool.remove();
    }
}
```

Initializing browser variable: If the browser option is passed from command window then initialize browser variable with the given option otherwise initialize it with the value from configurations.properties file.

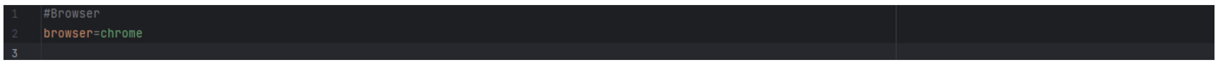
```
String browser = System.getProperty("browser") != null ? browser =
System.getProperty("browser") : ConfigurationReader.get("browser");
```

You can get the value of browser variable form cofigurations.properties file with method ConfigurationReader.get("browser").

ConfigurationReader class implementation:

```
public class ConfigurationReader {
    private static Properties properties;
    static {
        try {
            String path = "configurations.properties";
            FileInputStream input = new FileInputStream(path);
            properties = new Properties();
            properties.load(input);
            input.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public static String get(String keyName) {
        return properties.getProperty(keyName);
    }
}
```

configurations.properties file implementation: You can change the value of variable browser in this file which is default setting.



```
#Browser
browser=chrome
```

Depending on your requirement you can set the following option:
chrome, chrome-headless, remote_chrome, firefox, firefox-headless or remote_firefox

```
browser = chrome   // chrome option runs the script locally on your system
```

```
browser = chrome-headless (// chrome-headless option runs the script in
headless mode locally on your system
```

```
browser = remote_chrome // romote_chrome option runs the script on chrome
browser in selenium grid (url: http://10.118.21.131:4444/wd/hub
```

```
browser = firefox // firefox option runs script on firefox browser on your
local machine
```

```
browser = firefox-headless // firefox option runs script on firefox browser
in headless mode on your local machine
```

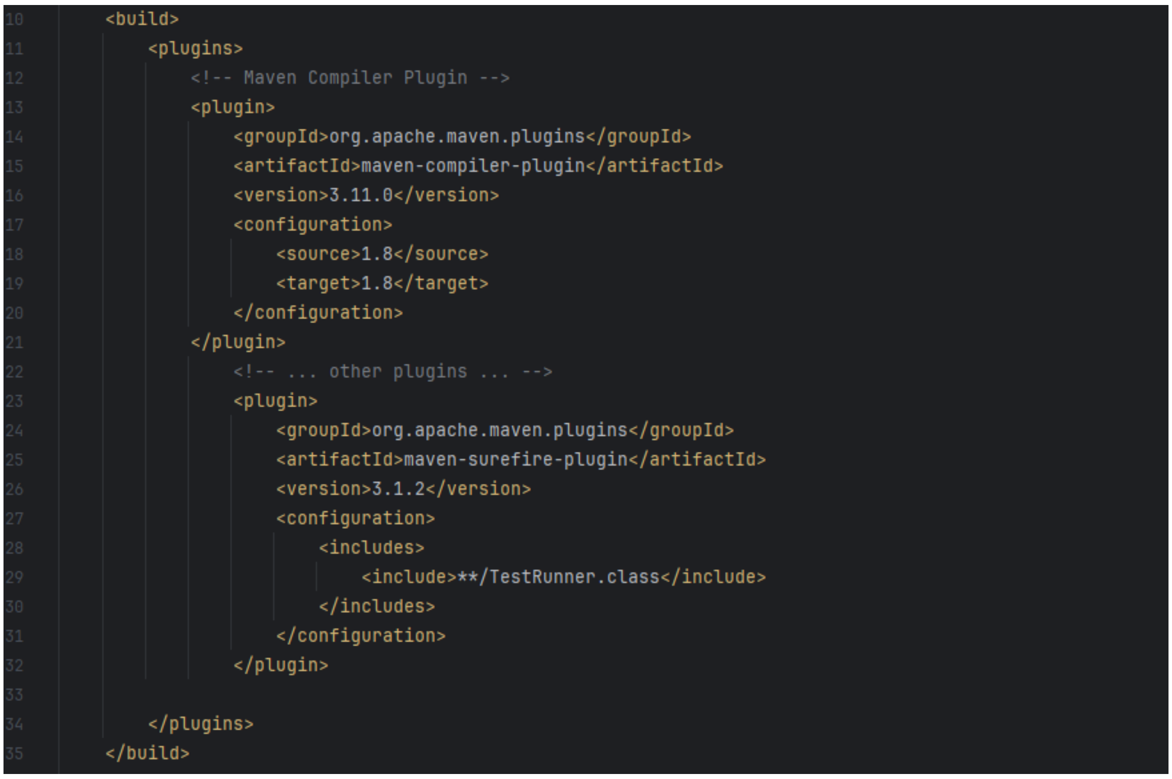
```
browser = remote_firefox // firefox option runs script in firefox browser
on selenium grid (url: http://10.118.21.131:4444/wd/hub
```

Running the tests through command prompt:

The reason behind running the tests through command prompt is that the test can be triggered through Jenkins.

Step1: We need to add some extra plugins in pom.xml file to execute the test through command prompt:

- maven-compiler-plugin
- maven-surefire-plugin



```xml
    <build>
        <plugins>
            <!-- Maven Compiler Plugin -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.11.0</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
                <!-- ... other plugins ... -->
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-surefire-plugin</artifactId>
                    <version>3.1.2</version>
                    <configuration>
                        <includes>
                            <include>**/TestRunner.class</include>
                        </includes>
                    </configuration>
                </plugin>

        </plugins>
    </build>
```

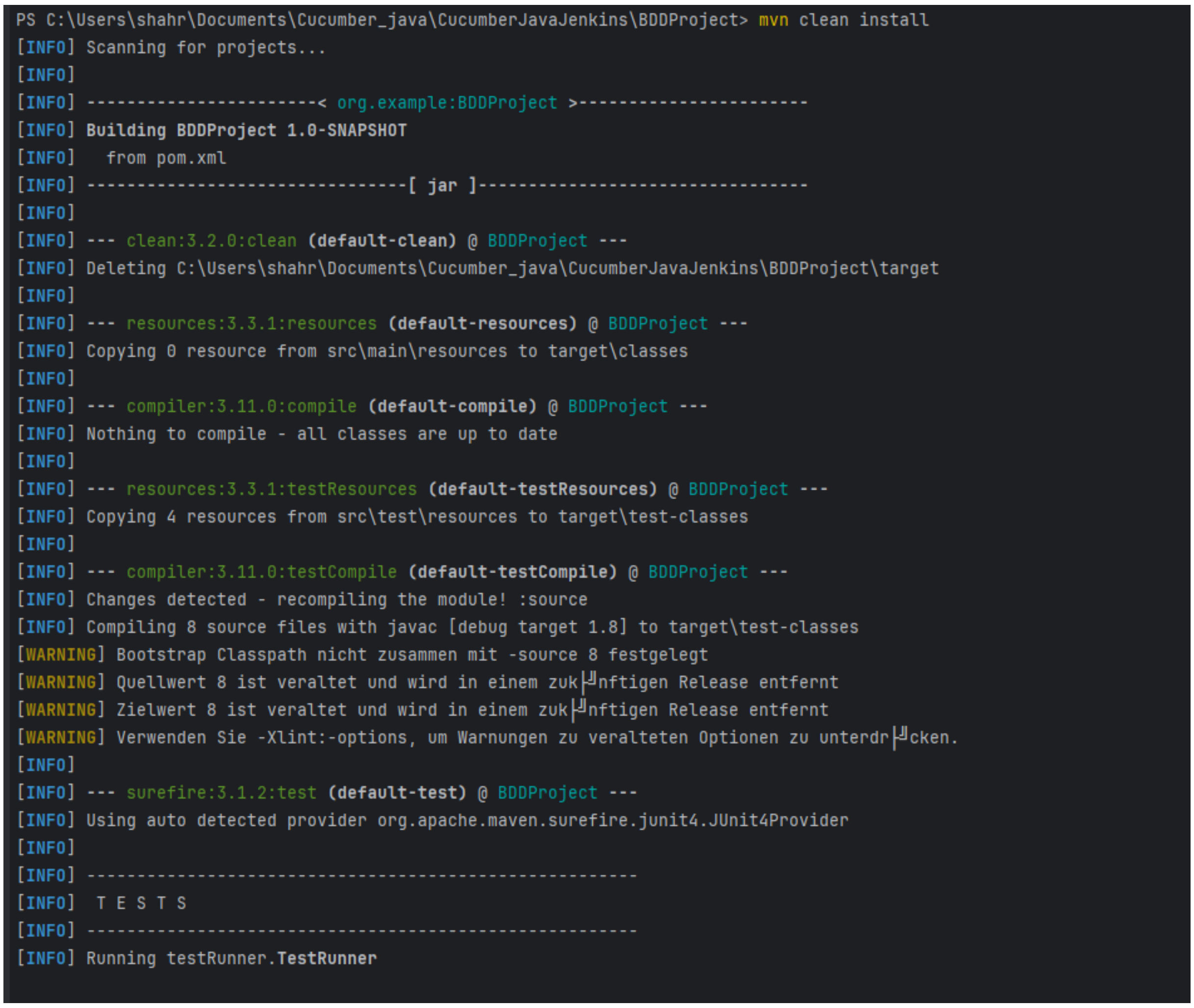
Step2: After adding the plugins in pom.xml. Open cmd windows in project directory and run the mvn clean install: This will take the browser option from configurations.properties file.

```
mvn clean install
```

Passing the browser option from command window as argument: Passing the browser option form command window will overwrite the browser option from configurations.properties file**. If you choose option remote_chrome or remote_firefox you need to turn your vpn on so that your local machine can connect to browser in selenium grid.**

```
mvn clean install -Dbrowser=chrome

mvn clean install -Dbrowser=chrome-headless

mvn clean install -Dbrowser=remote_chrome

mvn clean install -Dbrowser=firefox

mvn clean install -Dbrowser=firefox-headless

mvn clean install -Dbrowser=remote_firefox
```

If the test runs successfully, it seems something like this.



```
PS C:\Users\shahr\Documents\Cucumber_java\CucumberJavaJenkins\BDDProject> mvn clean install
[INFO] Scanning for projects...
[INFO]
[INFO] -----------------------< org.example:BDDProject >-----------------------
[INFO] Building BDDProject 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- clean:3.2.0:clean (default-clean) @ BDDProject ---
[INFO] Deleting C:\Users\shahr\Documents\Cucumber_java\CucumberJavaJenkins\BDDProject\target
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ BDDProject ---
[INFO] Copying 0 resource from src\main\resources to target\classes
[INFO]
[INFO] --- compiler:3.11.0:compile (default-compile) @ BDDProject ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ BDDProject ---
[INFO] Copying 4 resources from src\test\resources to target\test-classes
[INFO]
[INFO] --- compiler:3.11.0:testCompile (default-testCompile) @ BDDProject ---
[INFO] Changes detected - recompiling the module! :source
[INFO] Compiling 8 source files with javac [debug target 1.8] to target\test-classes
[WARNING] Bootstrap Classpath nicht zusammen mit -source 8 festgelegt
[WARNING] Quellwert 8 ist veraltet und wird in einem zuk├╝nftigen Release entfernt
[WARNING] Zielwert 8 ist veraltet und wird in einem zuk├╝nftigen Release entfernt
[WARNING] Verwenden Sie -Xlint:-options, um Warnungen zu veralteten Optionen zu unterdr├╝cken.
[INFO]
[INFO] --- surefire:3.1.2:test (default-test) @ BDDProject ---
[INFO] Using auto detected provider org.apache.maven.surefire.junit4.JUnit4Provider
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running testRunner.TestRunner
```

Running tests through bat files:

Step 1: Create a run.bat file at project level and add the following lines in that file.

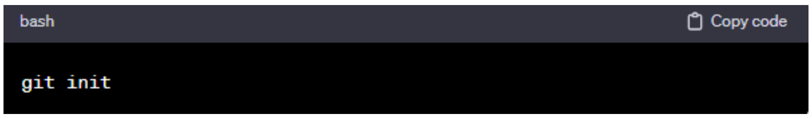
```
Step 2: add the cd <path to your project Folder> Example:
cd C:\Users\shahr\Documents\Cucumber_java\CucumberJavaJenkins\BDDProject
Step 3: add the command:
mvn clean install
```

Push Local project to GitHub:

Step 1: Set Up a GitHub Repository

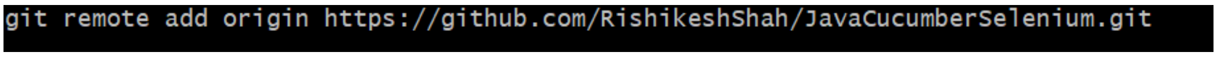
- **Create a GitHub Account:** If you don't have a GitHub account, sign up at GitHub.
- **Create a New Repository:** On GitHub, click the "+" sign in the upper right corner and select "New repository."
- Fill in the repository name, add a description, choose public or private visibility, and click "Create repository."

Step 2: Initialize Git in Your Maven Project

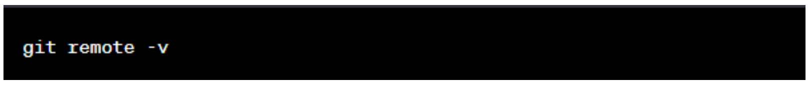
- **Navigate to your Project Directory:** Open a terminal of command prompt and go to your Maven project’s directory.
- **Initialize Git:** Run the following commands to initialize Git in your project directory
- 

```
git init
```

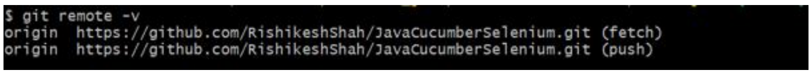
Step 3: Connect your local project to the GitHub Repository

- **Add a Remote Repository:** Run the following command to add a remote repository (replace <repository_url> with the URL of your GitHub repository):


```
git remote add origin https://github.com/RishikeshShah/JavaCucumberSelenium.git
```

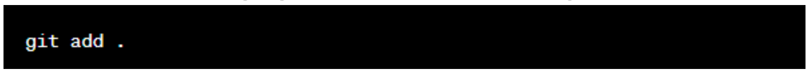
- Verify the Remote Repository: To verify that the remote repository is correctly added, you can run:
- 

```
git remote -v
```

- You will see something like this.
- 

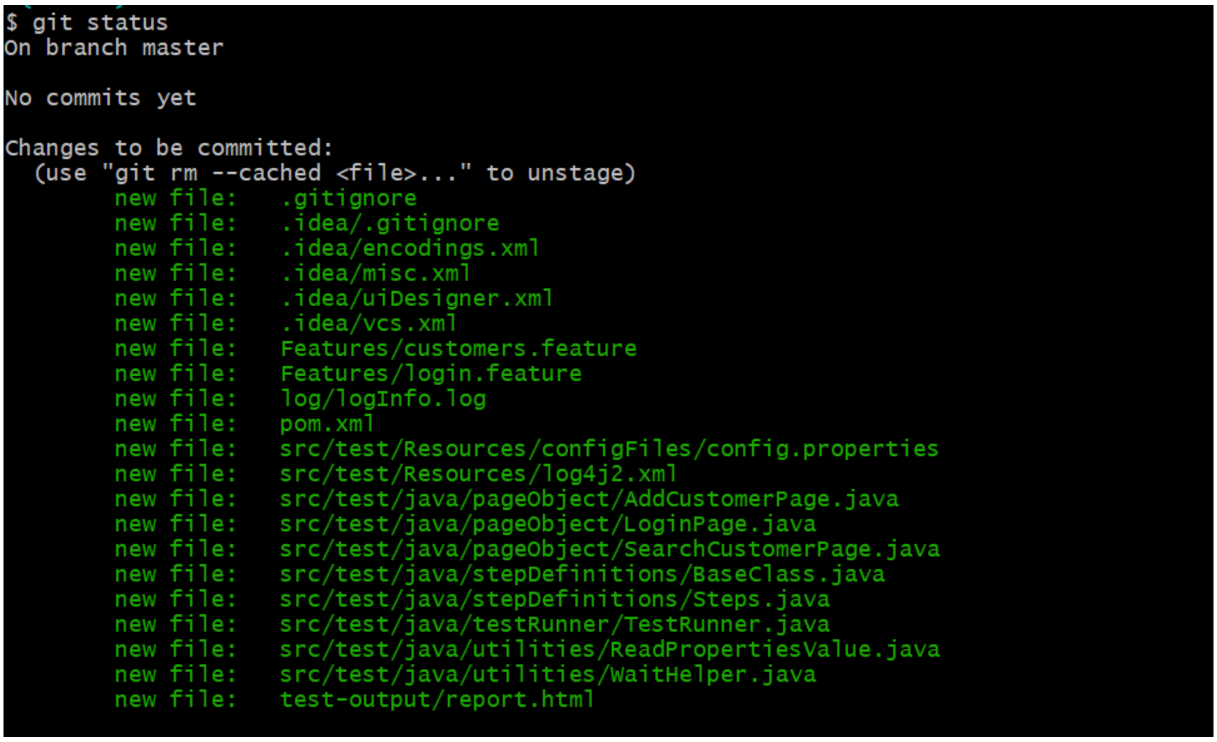
```
$ git remote -v
origin  https://github.com/RishikeshShah/JavaCucumberSelenium.git (fetch)
origin  https://github.com/RishikeshShah/JavaCucumberSelenium.git (push)
```

Step 4: Commit your Maven Project

- **Add Files to the Staging Area:** Use the following command to add all the files to staging area:
- 

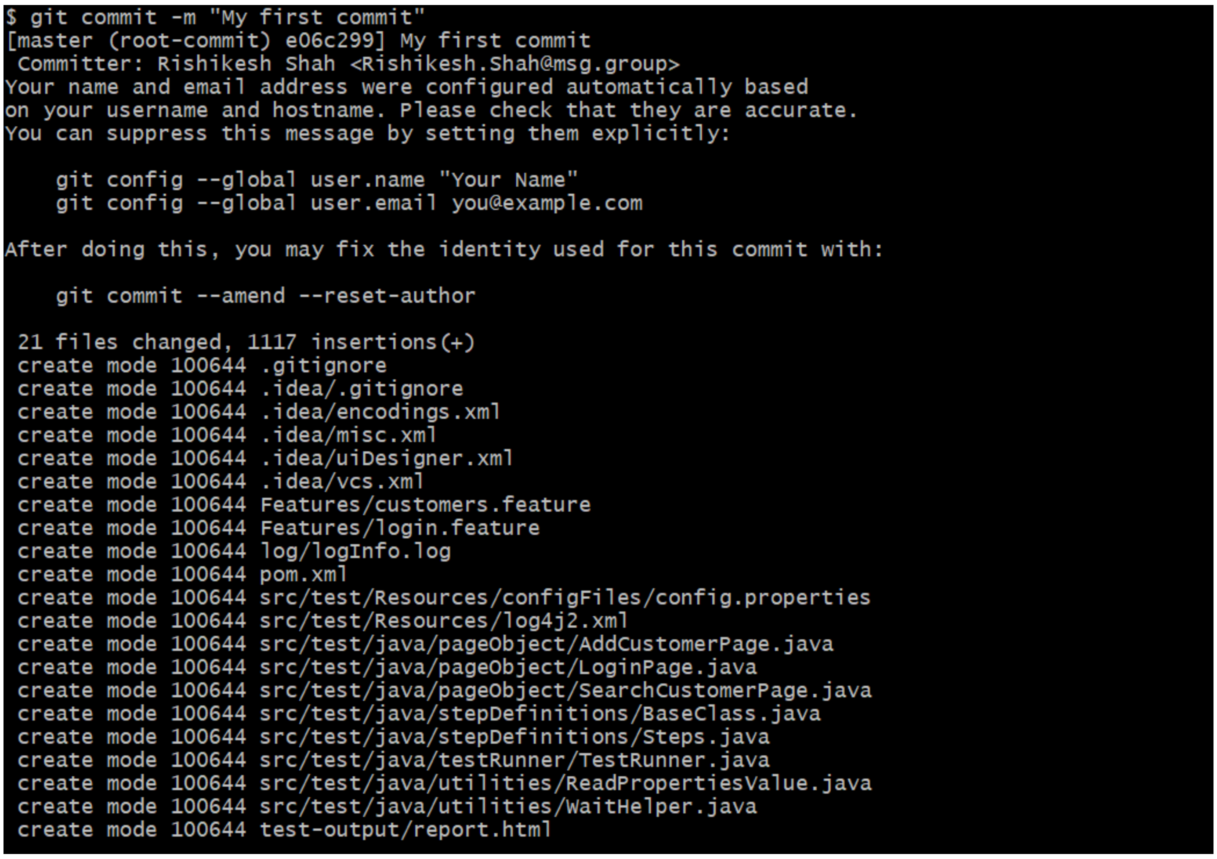
```
git add .
```

- To check, if the files added to staging or not type git status in command line you will see the staged file in green.

- 

```
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   .gitignore
        new file:   .idea/.gitignore
        new file:   .idea/encodings.xml
        new file:   .idea/misc.xml
        new file:   .idea/uiDesigner.xml
        new file:   .idea/vcs.xml
        new file:   Features/customers.feature
        new file:   Features/login.feature
        new file:   log/logInfo.log
        new file:   pom.xml
        new file:   src/test/Resources/configFiles/config.properties
        new file:   src/test/Resources/log4j2.xml
        new file:   src/test/java/pageObject/AddCustomerPage.java
        new file:   src/test/java/pageObject/LoginPage.java
        new file:   src/test/java/pageObject/SearchCustomerPage.java
        new file:   src/test/java/stepDefinitions/BaseClass.java
        new file:   src/test/java/stepDefinitions/Steps.java
        new file:   src/test/java/testRunner/TestRunner.java
        new file:   src/test/java/utilities/ReadPropertiesValue.java
        new file:   src/test/java/utilities/WaitHelper.java
        new file:   test-output/report.html
```

- Commit Changes: Commit the changes to your local repository.
- `git commit -m "My first commit"`
- Console looks something like this

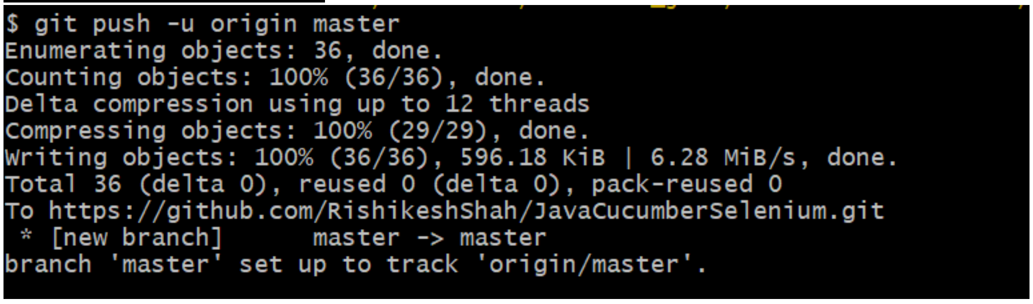


```
$ git commit -m "My first commit"
[master (root-commit) e06c299] My first commit
 Committer: Rishikesh Shah <Rishikesh.Shah@msg.group>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 21 files changed, 1117 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 .idea/.gitignore
 create mode 100644 .idea/encodings.xml
 create mode 100644 .idea/misc.xml
 create mode 100644 .idea/uiDesigner.xml
 create mode 100644 .idea/vcs.xml
 create mode 100644 Features/customers.feature
 create mode 100644 Features/login.feature
 create mode 100644 log/logInfo.log
 create mode 100644 pom.xml
 create mode 100644 src/test/Resources/configFiles/config.properties
 create mode 100644 src/test/Resources/log4j2.xml
 create mode 100644 src/test/java/pageObject/AddCustomerPage.java
 create mode 100644 src/test/java/pageObject/LoginPage.java
 create mode 100644 src/test/java/pageObject/SearchCustomerPage.java
 create mode 100644 src/test/java/stepDefinitions/BaseClass.java
 create mode 100644 src/test/java/stepDefinitions/Steps.java
 create mode 100644 src/test/java/testRunner/TestRunner.java
 create mode 100644 src/test/java/utilities/ReadPropertiesValue.java
 create mode 100644 src/test/java/utilities/WaitHelper.java
 create mode 100644 test-output/report.html
```

Step 5: Push your Maven project to GitHub

- Push your committed changes to the GitHub repository
- The -u option sets the upstream branch for the master branch, so you can simply use git push in the future.

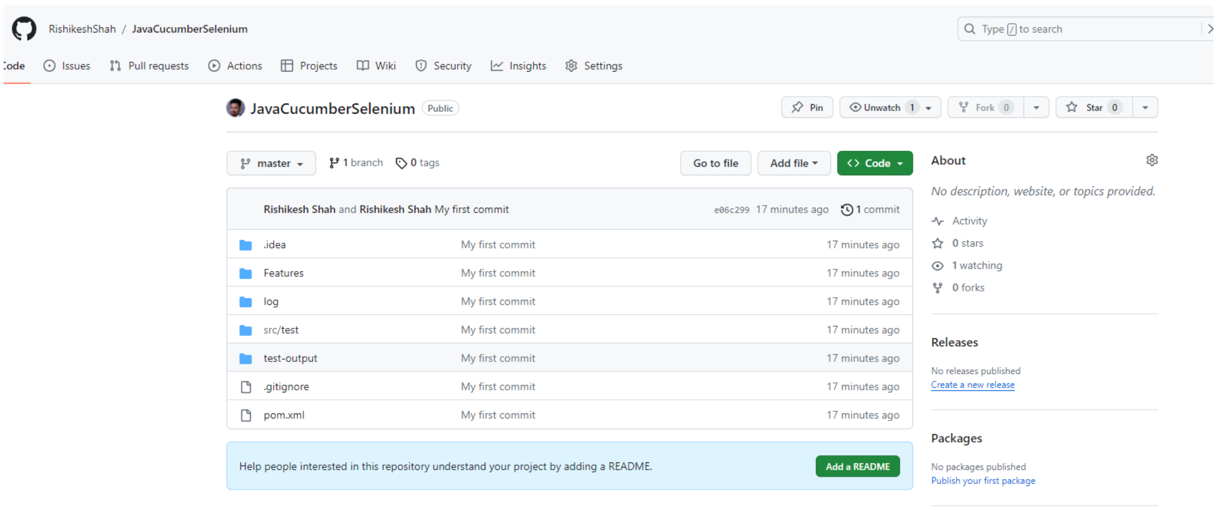
-
  ```
  git push -u origin master
  ```



```
$ git push -u origin master
Enumerating objects: 36, done.
Counting objects: 100% (36/36), done.
Delta compression using up to 12 threads
Compressing objects: 100% (29/29), done.
Writing objects: 100% (36/36), 596.18 KiB | 6.28 MiB/s, done.
Total 36 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/RishikeshShah/JavaCucumberSelenium.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

Step 6: Verify on GitHub

- **Visit your GitHub Repository:** Open your GitHub repository in a web browser to verify that your Maven project has been pushed successfully.
- It looks something like this:



RishikeshShah / JavaCucumberSelenium

JavaCucumberSelenium Public

master · 1 branch · 0 tags

Rishikesh Shah and Rishikesh Shah My first commit · e06c299 17 minutes ago · 1 commit

.idea	My first commit	17 minutes ago
Features	My first commit	17 minutes ago
log	My first commit	17 minutes ago
src/test	My first commit	17 minutes ago
test-output	My first commit	17 minutes ago
.gitignore	My first commit	17 minutes ago
pom.xml	My first commit	17 minutes ago

Help people interested in this repository understand your project by adding a README.

About

No description, website, or topics provided.

Activity
0 stars
1 watching
0 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

- Now your Project is on GitHub. Congratulations!

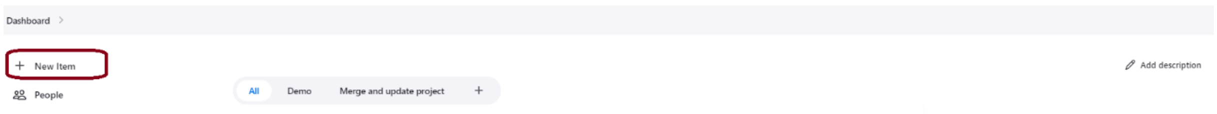
Running Maven project in Jenkins:

Prerequisites:

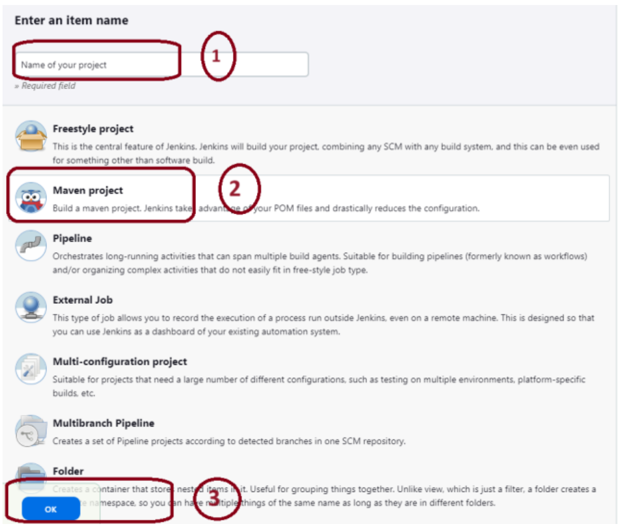
1. Maven project should be executable from command window with command: mvn clean install (see chapter: Running the tests through command prompt:)
2. To run the tests on browser in Selenium grid. Browser setting for remote browser should be configured. (See chapter: Configuration for different browser)
3. The Project should be on GitHub or GitLab. (To push your project to GitHub, See chapter Push Local project to GitHub)
4. Java-sdk, maven, browser and browser driver should be installed. In our case all the software are already installed.

Steps to configure the Jenkins-job:

- Login to Jenkins with your credentials
- Click on Dashboard and then on +New Item



- Give a meaningful name of project then select option Maven project and click on ok.



Enter an item name

Name of your project

» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

External Job
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system.

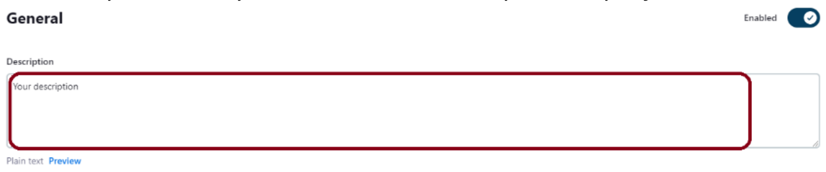
Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Multibranch Pipeline
Creates a set of Pipeline projects according to detected branches in one SCM repository.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

OK

- In Description field you can add the description of project



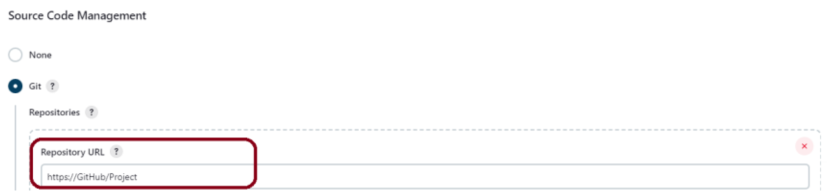
General Enabled

Description

Your description

Plain text Preview

- In Source Code Management section click on option Git.
- Enter repository URL auf your GitHub or GitLab. Leave all the option as default. Build will be triggered from master branch.



Source Code Management

None

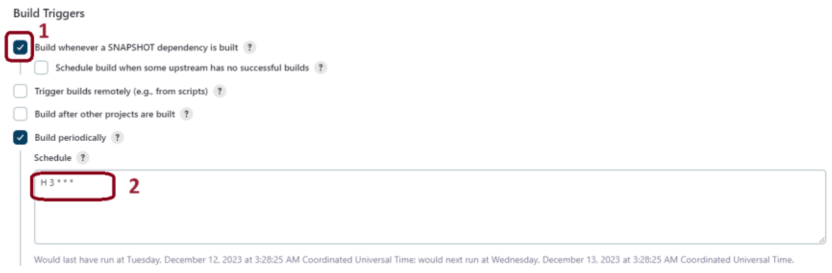
[✓ Git]

Repositories

Repository URL

https://GitHub/Project

- Build Triggers section: click on option Build periodically. If you want to schedule your Job. I for example run my Job at 03:00 am. Setting for that you can see in screenshot below.



Build Triggers

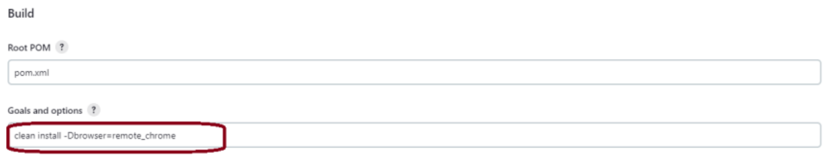
- [x] Build whenever a SNAPSHOT dependency is built
  - [ ] Schedule build when some upstream has no successful builds
- [ ] Trigger builds remotely (e.g., from scripts)
- [ ] Build after other projects are built
- [x] Build periodically

Schedule

```
H 3 * * *
```

Would last have run at Tuesday, December 12, 2023 at 3:28:25 AM Coordinated Universal Time; would next run at Wednesday, December 13, 2023 at 3:28:25 AM Coordinated Universal Time.

- In Build section: in Goals and options section enter the command below so that Jenkins runs the script and use the chrome browser in selenium grid.
- clean install -Dbrowser=remote_chrome



Build

Root POM

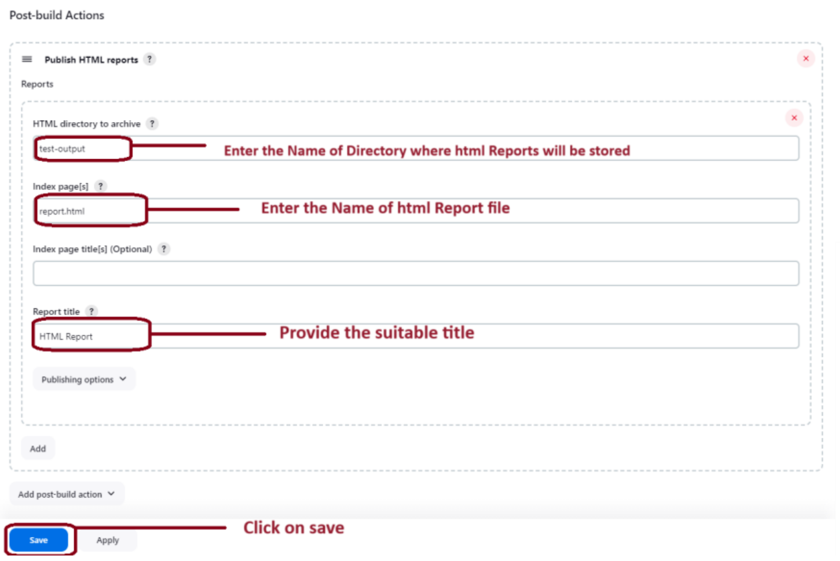
pom.xml

Goals and options

clean install -Dbrowser=remote_chrome

- **Post-build Action**: To generate the html report in Jenkins, you need to install HTML Publisher plugin, if not already installed.
- Click on Add post-build action, select publish HTML reports from dropdown menu. Note if you don’t see publish HTML reports in dropdown menu, you need to install HTML Publisher plugin.

- Then click on Add button. Fill the following fields and save the setting.



Post-build Actions

Publish HTML reports

Reports

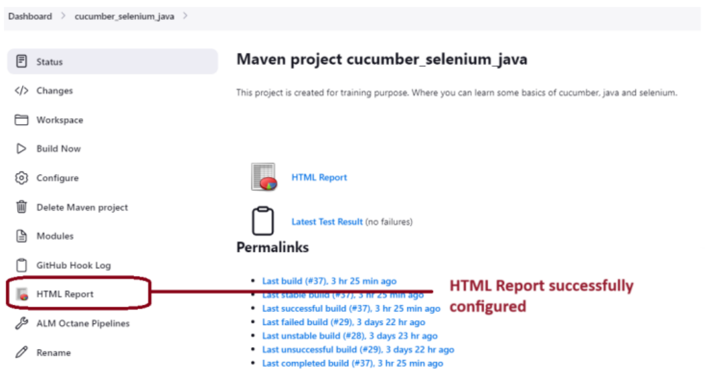
HTML directory to archive: test-output — Enter the Name of Directory where html Reports will be stored

Index page[s]: report.html — Enter the Name of html Report file

Index page title[s] (Optional):

Report title: HTML Report — Provide the suitable title

If everything is configured well then you will be able to see HTML Report on lefthand side. With that you have successfully configured Jenkins job.



Dashboard > cucumber_selenium_java >

Maven project cucumber_selenium_java

This project is created for training purpose. Where you can learn some basics of cucumber, java and selenium.

HTML Report

Latest Test Result (no failures)

Permalinks

- Last build (#37), 3 hr 25 min ago
- Last stable build (#37), 3 hr 25 min ago
- Last successful build (#37), 3 hr 25 min ago
- Last failed build (#29), 3 days 22 hr ago
- Last unstable build (#28), 3 days 23 hr ago
- Last unsuccessful build (#29), 3 days 22 hr ago
- Last completed build (#37), 3 hr 25 min ago

HTML Report successfully configured