A screenshot of a project

Description automatically generated with medium confidenceA screenshot of a computer program

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidence

“Control+shift+f” is for formatting.

# Delete the default files created-

A screenshot of a computer

Description automatically generated

# First lets start with pom-

Remove all properties.

Replace with the ones shown below.

A screen shot of a computer code

Description automatically generated with low confidence

Remove existing dependencies and add the below ones-

|  |
| --- |
| <dependencies>  <dependency>  <groupId>io.cucumber</groupId>  <artifactId>cucumber-java</artifactId>  <version>${cucumber.version}</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>io.cucumber</groupId>  <artifactId>cucumber-junit</artifactId>  <version>${cucumber.version}</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>${junit.version}</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.seleniumhq.selenium</groupId>  <artifactId>selenium-java</artifactId>  <version>3.141.59</version>  </dependency>  <dependency>  <groupId>io.github.bonigarcia</groupId>  <artifactId>webdrivermanager</artifactId>  <version>4.2.2</version>  </dependency>  <dependency>  <groupId>tech.grasshopper</groupId>  <artifactId>extentreports-cucumber6-adapter</artifactId>  <version>2.8.0</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.testng</groupId>  <artifactId>testng</artifactId>  <version>6.14.3</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>io.cucumber</groupId>  <artifactId>cucumber-testng</artifactId>  <version>${cucumber.version}</version>  <scope>test</scope>  </dependency>  <!-- https://mvnrepository.com/artifact/org.apache.poi/poi -->  <dependency>  <groupId>org.apache.poi</groupId>  <artifactId>poi</artifactId>  <version>4.1.2</version>  </dependency>  <!-- https://mvnrepository.com/artifact/org.apache.poi/poi-ooxml -->  <dependency>  <groupId>org.apache.poi</groupId>  <artifactId>poi-ooxml</artifactId>  <version>4.1.2</version>  </dependency>  </dependencies> |

Pressing save will rebuild the project and dependencies get added to maven dependencies.

A screenshot of a computer

Description automatically generated with medium confidencesimilarly go to the plugin section and remove everything.

Add the below ones. It is better to copy paste this stuff and remember what we need rather than typing.

|  |
| --- |
| <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>${maven.compiler.version}</version>  <configuration>  <encoding>UTF-8</encoding>  <source>${java.version}</source>  <target>${java.version}</target>  </configuration>  </plugin>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-surefire-plugin</artifactId>  <version>${maven.surefire.version}</version>  <configuration>  <includes>  <include>\*\*/ParallelRun.java</include>  </includes>  <parallel>methods</parallel>  <threadCount>4</threadCount>  <useUnlimitedThreads>false</useUnlimitedThreads>  </configuration>  </plugin>  <!-- <plugin> <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-failsafe-plugin</artifactId>  <version>3.0.0-M3</version> <executions> <execution> <goals>  <goal>integration-test</goal>  </goals> <configuration>  UNCOMMENT - To add any exclusions if required <excludes>  <exclude>\*\*/\*IT\*.java</exclude> </excludes> <includes> UNCOMMENT  BELOW LINE  - To execute feature files with a single runner  <include>\*\*/MyTestRunner.java</include>  UNCOMMENT BELOW LINE -  To execute feature files with multiple runners  <include>\*\*/\*Runner.java</include>  </includes> UNCOMMENT BELOW  3 LINES - To execute using parallel or combination  option  <parallel>methods</parallel> <threadCount>4</threadCount>  <perCoreThreadCount>true</perCoreThreadCount>  UNCOMMENT BELOW  3  LINES - To execute using forking or combination option  <forkCount>2</forkCount> <reuseForks>true</reuseForks>  <reportsDirectory>${project.build.directory}/failsafe-reports\_${surefire.forkNumber}</reportsDirectory>  </configuration> </execution> </executions> </plugin> -->  </plugins>  </build> |

Once pom done, force update all the maven project.

Right click on project and convert to cucumber project.

A screenshot of a computer

Description automatically generated

You can see “M and C” icon on project which means it’s a maven-cucumber project. This will come after you convert the project to cucumber project.

Even delete the default package which got created-

A screenshot of a computer

Description automatically generated

# When running the new project, you need to select the project also-

A screenshot of a computer

Description automatically generated with medium confidence

“Control+shift+O” for importing everything on page.

# “Threadlocal.get” method returns webdriver-

A screenshot of a computer

Description automatically generated with medium confidence

# See how to check what parameter to pass in takes screenshot method -

A screenshot of a computer

Description automatically generated with medium confidence

See the output type of getscreenshot-

A screenshot of a computer

Description automatically generated with medium confidence

According to pom, we have to create separate pages for every page class.

See how thread local works-

A screenshot of a computer

Description automatically generated with medium confidence

The first picture is parallel execution using thread local. Same chrome browser being accessed by multiple threads at the same time. First test case attacked by first google chrome in first thread, second test case attacked by second google chrome in second thread and so on.

The second picture shows without thread local, at a time the browser can be accessed by only one thread. Other threads will have to wait.

“Threadlocal.get” will get the driver details at run time.

Since we are not using the concept of testng, threadlocal concept needed for parallel execution.

# Note on how to rename the file-

Click on the file name on project explorer.

Press F2.

Then rename.

# First big code as part of hybrid framework-

|  |
| --- |
| Driver factory-  **package** com.qa.factory;  **import** org.openqa.selenium.WebDriver;  **import** org.openqa.selenium.chrome.ChromeDriver;  **import** org.openqa.selenium.firefox.FirefoxDriver;  **import** org.openqa.selenium.safari.SafariDriver;  **import** io.github.bonigarcia.wdm.WebDriverManager;  //this class is for initialising the driver and returning it.  //this class will also help in parallel runs because we will use the concept of threadlocal.  //thread local is java 8 feature.  **public** **class** DriverFactory {  **public** WebDriver driver;  // we want to initialize the webdriver with help of threadlocal so we pass  // WebDriver as generics.  **public** **static** ThreadLocal<WebDriver> *tlDriver* = **new** ThreadLocal<>();  // pass the browser and webdriver will open that browser.  // this method should also return webdriver.  /\*\*  \* this method is used to initialize the threadlocal driver on the basis of  \* given browser.  \*  \* **@param** browser  \* **@return** this will return tldriver.  \*/  **public** WebDriver init\_driver(String browser) {  System.***out***.println("browser value is" + browser);  **if** (browser.equals("chrome")) {  WebDriverManager.*chromedriver*().setup();  // tl will give set and get method.  // here chrome driver object is created and it is set with thread local driver.  // local copy of chrome driver object is created and assigned to one thread.  *tlDriver*.set(**new** ChromeDriver());  } **else** **if** (browser.equals("firefox")) {  WebDriverManager.*firefoxdriver*().setup();  *tlDriver*.set(**new** FirefoxDriver());  } **else** **if** (browser.equals("safari")) {  // for safari there is no binary so no web driver manager.  *tlDriver*.set(**new** SafariDriver());  } **else** {  System.***out***.println("please pass the correct browser" + browser);  }  *getDriver*().manage().deleteAllCookies();  *getDriver*().manage().window().maximize();  **return** *getDriver*();  }  //get the driver.  // this will return the threadlocal instance.  // threadlocal has different browsers that driver will invoke.  // we are writing synchronized because multiple threads will call this method at  // same time.  /\*\*  \* this is used to get the driver with thread local.  \*  \* **@return**  \*/  **public** **static** **synchronized** WebDriver getDriver() {  **return** *tlDriver*.get();  }  } |
| Config.properties-  browser = chrome |
| Config reader-  **package** com.qa.util;  **import** java.io.FileInputStream;  **import** java.io.FileNotFoundException;  **import** java.io.IOException;  **import** java.util.Properties;  **public** **class** ConfigReader {    **private** Properties prop;    /\*\*  \* this method is used to load the properties from config.properties file.  \* **@return** it returns properties prop object.  \*/    **public** Properties initProp() {  prop=**new** Properties(); // create object of properties class  //we have to interact with config.properties which is file, so we use file input stream class.  **try** {  FileInputStream ip=**new** FileInputStream("./src/test/resources/config/config.properties");//give path of config.properties.  prop.load(ip); //load the properties for using them.  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  **return** prop;  }  } |
| Hooks-  package appHooks;  import java.util.Properties;  import org.openqa.selenium.OutputType;  import org.openqa.selenium.TakesScreenshot;  import org.openqa.selenium.WebDriver;  import com.qa.factory.DriverFactory;  import com.qa.util.ConfigReader;  import io.cucumber.java.After;  import io.cucumber.java.Before;  import io.cucumber.java.Scenario;  public class ApplicationHooks {    private DriverFactory driverFactory;  private WebDriver driver;  private ConfigReader configReader;  private Properties prop;  private String browserName;    @Before(order=0)  public void getProperty() {  configReader=new ConfigReader();  prop=configReader.initProp();  }    @Before(order=1)  public void launchBrowser() {  browserName=prop.getProperty("browser");  driverFactory=new DriverFactory();  driver=driverFactory.init\_driver(browserName); // we need to assign driver here, else driver.quit will  //give null pointer exception and the webdriver driver declared in step 16 is not pointing anywhere.  }    //note - in after the order works in reverse direction.  //order = 2 is executed before order = 1 and order =0.    @After(order=0)  public void quitBrowser() {  driver.quit();  }    @After(order=1)  public void tearDown(Scenario scenario) {  if(scenario.isFailed()) {  //take screenshot  String screenshotName=scenario.getName().replaceAll(" ", "\_"); // replace all space in file name with \_  byte [] sourcePath=((TakesScreenshot)driver).getScreenshotAs(OutputType.BYTES);// we keep the output type as  //bytes or base64 so that it is compatible with jenkins/ci/cd toold  scenario.attach(sourcePath, "image/png", screenshotName);  }  }      } |
| Login page steps-  package stepDefinitions;  import com.pages.LoginPageObject;  import com.qa.factory.DriverFactory;  import io.cucumber.java.en.Given;  import io.cucumber.java.en.Then;  import io.cucumber.java.en.When;  import static org.junit.Assert.\*;  import org.junit.Assert;  @SuppressWarnings("deprecation")  public class LoginPageSteps {    private static String title;  private LoginPageObject loginPageObject = new LoginPageObject(DriverFactory.getDriver());    @Given("user is on login page")  public void user\_is\_on\_login\_page() {  DriverFactory.getDriver().get("http://www.automationpractice.pl/index.php?controller=authentication&back=my-account");  //getdriver from driver factory returns web driver and with this web driver we can use all the  //browser controls and actions  }  @When("user gets the title of the page")  public void user\_gets\_the\_title\_of\_the\_page() throws InterruptedException {  Thread.sleep(10000);  title=loginPageObject.getLoginPageTitle();  System.out.println("page title is " + title);  }  @Then("page title should be {string}")  public void page\_title\_should\_be(String expectedTitleName) {  Assert.assertTrue(title.contains(expectedTitleName)); // assert has to come from junit and not testng else we get  //deprecated error.  }  @Then("forgot your password link should be displayed")  public void forgot\_your\_password\_link\_should\_be\_displayed() {  Assert.assertTrue(loginPageObject.isForgotPasswordLinkPresent());  }  @When("user enters username as {string}")  public void user\_enters\_username\_as(String userName) {  loginPageObject.enterUserName(userName);  }  @When("user enters password as {string}")  public void user\_enters\_password\_as(String password) {  loginPageObject.enterPassword(password);  }  @When("user clicks on login button")  public void user\_clicks\_on\_login\_button() {  loginPageObject.clickOnSignInButton();  }  //we always write assertions in then  //when is for test actions  //given is pre condition  } |
| Login page object-  **package** com.pages;  **import** org.openqa.selenium.By;  **import** org.openqa.selenium.WebDriver;  **public** **class** LoginPageObject {  // every page object will have by locators, constructor and page actions.  **private** WebDriver driver; // every class will have this webdriver.  // 1. by locators.  //by locators are also known as object repositories.  **private** By emailID = By.*id*("email");  **private** By password = By.*id*("passwd");  **private** By signInButton = By.*id*("SubmitLogin");  **private** By forgotPasswordLink = By.*linkText*("Forgot your password?");  // 2.constructor of the page class  **public** LoginPageObject(WebDriver driver) {  **this**.driver = driver;  }  // page classes should not have assertion.  // assertion should be written in test class or step def class.  // 3. page actions: features (behaviour) of the page in the form of methods.  **public** String getLoginPageTitle() {  **return** driver.getTitle();  }  **public** **boolean** isForgotPasswordLinkPresent() {  **return** driver.findElement(forgotPasswordLink).isDisplayed();  }  **public** **void** enterUserName(String userName) {  driver.findElement(emailID).sendKeys(userName);  }  **public** **void** enterPassword(String pwd) {  driver.findElement(password).sendKeys(pwd);  }  **public** **void** clickOnSignInButton() {  driver.findElement(signInButton).click();  }  // in step def do not maintain by locators and page methods. its ugly  // programming.  // in page object, selenium code should be written in page class.  } |
| Test runner-  package com.myTestRunner;  import org.junit.runner.RunWith;  import io.cucumber.junit.Cucumber;  import io.cucumber.junit.CucumberOptions;  @RunWith(Cucumber.class)  @CucumberOptions(  features = { "src/test/resources/appFeatures" },  glue = { "stepDefinitions", "appHooks" },  plugin = { "pretty" }  )  public class MyTestRunner {  } |
| Login feature-  Feature: login page feature  Scenario: login page title  Given user is on login page  When user gets the title of the page  Then page title should be "Login - My Shop"  Scenario: forgot password link  Given user is on login page  Then forgot your password link should be displayed  Scenario: login with correct credentials  Given user is on login page  When user enters username as "cucumbertesting1960@gmail.com"  And user enters password as "Malaravi@123"  And user clicks on login button  Then user gets the title of the page  And page title should be "My account - My Shop" |

Run as junit and check console-

|  |
| --- |
|  |

# Lets create ui report-

|  |
| --- |
| Created a properties file for reports.  Cucumber.properties in src/test/resources.  cucumber.publish.enabled=true |

There should not be any space in between or after true else we get error as “cannot parse the property cucumber.enabled=true.”

So remove all spaces and update maven project and then run and you should be good.

Report link is opened.

A screenshot of a computer

Description automatically generated

# Lets fail a scenario and then see if screenshot is taken-

A screen shot of a computer code

Description automatically generated with low confidence

Console-

A screenshot of a computer

Description automatically generated with medium confidence

Report-

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Screenshot also taken-

A screenshot of a website

Description automatically generated with medium confidence

# Project structure-

A screenshot of a computer

Description automatically generated with medium confidence

# Codes used-

|  |
| --- |
| Driver factory-  **package** com.qa.factory;  **import** org.openqa.selenium.WebDriver;  **import** org.openqa.selenium.chrome.ChromeDriver;  **import** org.openqa.selenium.firefox.FirefoxDriver;  **import** org.openqa.selenium.safari.SafariDriver;  **import** io.github.bonigarcia.wdm.WebDriverManager;  //this class is for initialising the driver and returning it.  //this class will also help in parallel runs because we will use the concept of threadlocal.  //thread local is java 8 feature.  **public** **class** DriverFactory {  **public** WebDriver driver;  // we want to initialize the webdriver with help of threadlocal so we pass  // WebDriver as generics.  **public** **static** ThreadLocal<WebDriver> *tlDriver* = **new** ThreadLocal<>();  // pass the browser and webdriver will open that browser.  // this method should also return webdriver.  /\*\*  \* this method is used to initialize the threadlocal driver on the basis of  \* given browser.  \*  \* **@param** browser  \* **@return** this will return tldriver.  \*/  **public** WebDriver init\_driver(String browser) {  System.***out***.println("browser value is" + browser);  **if** (browser.equals("chrome")) {  WebDriverManager.*chromedriver*().setup();  // tl will give set and get method.  // here chrome driver object is created and it is set with thread local driver.  // local copy of chrome driver object is created and assigned to one thread.  *tlDriver*.set(**new** ChromeDriver());  } **else** **if** (browser.equals("firefox")) {  WebDriverManager.*firefoxdriver*().setup();  *tlDriver*.set(**new** FirefoxDriver());  } **else** **if** (browser.equals("safari")) {  // for safari there is no binary so no web driver manager.  *tlDriver*.set(**new** SafariDriver());  } **else** {  System.***out***.println("please pass the correct browser" + browser);  }  *getDriver*().manage().deleteAllCookies();  *getDriver*().manage().window().maximize();  **return** *getDriver*();  }  //get the driver.  // this will return the threadlocal instance.  // threadlocal has different browsers that driver will invoke.  // we are writing synchronized because multiple threads will call this method at  // same time.  /\*\*  \* this is used to get the driver with thread local.  \*  \* **@return**  \*/  **public** **static** **synchronized** WebDriver getDriver() {  **return** *tlDriver*.get();  }  } |
| Config properties-  browser = chrome |
| Config reader-  **package** com.qa.util;  **import** java.io.FileInputStream;  **import** java.io.FileNotFoundException;  **import** java.io.IOException;  **import** java.util.Properties;  **public** **class** ConfigReader {    **private** Properties prop;    /\*\*  \* this method is used to load the properties from config.properties file.  \* **@return** it returns properties prop object.  \*/    **public** Properties initProp() {  prop=**new** Properties(); // create object of properties class  //we have to interact with config.properties which is file, so we use file input stream class.  **try** {  FileInputStream ip=**new** FileInputStream("./src/test/resources/config/config.properties");//give path of config.properties.  prop.load(ip); //load the properties for using them.  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  **return** prop;  }  } |
| Hooks-  package appHooks;  import java.util.Properties;  import org.openqa.selenium.OutputType;  import org.openqa.selenium.TakesScreenshot;  import org.openqa.selenium.WebDriver;  import com.qa.factory.DriverFactory;  import com.qa.util.ConfigReader;  import io.cucumber.java.After;  import io.cucumber.java.Before;  import io.cucumber.java.Scenario;  public class ApplicationHooks {    private DriverFactory driverFactory;  private WebDriver driver;  private ConfigReader configReader;  private Properties prop;  private String browserName;    @Before(order=0)  public void getProperty() {  configReader=new ConfigReader();  prop=configReader.initProp();  }    @Before(order=1)  public void launchBrowser() {  browserName=prop.getProperty("browser");  driverFactory=new DriverFactory();  driver=driverFactory.init\_driver(browserName); // we need to assign driver here, else driver.quit will  //give null pointer exception and the webdriver driver declared in step 16 is not pointing anywhere.  }    //note - in after the order works in reverse direction.  //order = 2 is executed before order = 1 and order =0.    @After(order=0)  public void quitBrowser() {  driver.quit();  }    @After(order=1)  public void tearDown(Scenario scenario) {  if(scenario.isFailed()) {  //take screenshot  String screenshotName=scenario.getName().replaceAll(" ", "\_"); // replace all space in file name with \_  byte [] sourcePath=((TakesScreenshot)driver).getScreenshotAs(OutputType.BYTES);// we keep the output type as  //bytes or base64 so that it is compatible with jenkins/ci/cd toold  scenario.attach(sourcePath, "image/png", screenshotName);  }  }      } |
| Login page steps-  package stepDefinitions;  import com.pages.LoginPageObject;  import com.qa.factory.DriverFactory;  import io.cucumber.java.en.Given;  import io.cucumber.java.en.Then;  import io.cucumber.java.en.When;  import static org.junit.Assert.\*;  import org.junit.Assert;  @SuppressWarnings("deprecation")  public class LoginPageSteps {    private static String title;  private LoginPageObject loginPageObject = new LoginPageObject(DriverFactory.getDriver());    @Given("user is on login page")  public void user\_is\_on\_login\_page() {  DriverFactory.getDriver().get("http://www.automationpractice.pl/index.php?controller=authentication&back=my-account");  //getdriver from driver factory returns web driver and with this web driver we can use all the  //browser controls and actions  }  @When("user gets the title of the page")  public void user\_gets\_the\_title\_of\_the\_page() throws InterruptedException {  Thread.sleep(10000);  title=loginPageObject.getLoginPageTitle();  System.out.println("page title is " + title);  }  @Then("page title should be {string}")  public void page\_title\_should\_be(String expectedTitleName) {  Assert.assertTrue(title.contains(expectedTitleName)); // assert has to come from junit and not testng else we get  //deprecated error.  }  @Then("forgot your password link should be displayed")  public void forgot\_your\_password\_link\_should\_be\_displayed() {  Assert.assertTrue(loginPageObject.isForgotPasswordLinkPresent());  }  @When("user enters username as {string}")  public void user\_enters\_username\_as(String userName) {  loginPageObject.enterUserName(userName);  }  @When("user enters password as {string}")  public void user\_enters\_password\_as(String password) {  loginPageObject.enterPassword(password);  }  @When("user clicks on login button")  public void user\_clicks\_on\_login\_button() {  loginPageObject.clickOnSignInButton();  }  //we always write assertions in then  //when is for test actions  //given is pre condition  } |
| Login page object-  **package** com.pages;  **import** org.openqa.selenium.By;  **import** org.openqa.selenium.WebDriver;  **public** **class** LoginPageObject {  // every page object will have by locators, constructor and page actions.  **private** WebDriver driver; // every class will have this webdriver.  // 1. by locators.  //by locators are also known as object repositories.  **private** By emailID = By.*id*("email");  **private** By password = By.*id*("passwd");  **private** By signInButton = By.*id*("SubmitLogin");  **private** By forgotPasswordLink = By.*linkText*("Forgot your password?1111");  // 2.constructor of the page class  **public** LoginPageObject(WebDriver driver) {  **this**.driver = driver;  }  // page classes should not have assertion.  // assertion should be written in test class or step def class.  // 3. page actions: features (behaviour) of the page in the form of methods.  **public** String getLoginPageTitle() {  **return** driver.getTitle();  }  **public** **boolean** isForgotPasswordLinkPresent() {  **return** driver.findElement(forgotPasswordLink).isDisplayed();  }  **public** **void** enterUserName(String userName) {  driver.findElement(emailID).sendKeys(userName);  }  **public** **void** enterPassword(String pwd) {  driver.findElement(password).sendKeys(pwd);  }  **public** **void** clickOnSignInButton() {  driver.findElement(signInButton).click();  }  // in step def do not maintain by locators and page methods. its ugly  // programming.  // in page object, selenium code should be written in page class.  } |
| Test runner-  package com.myTestRunner;  import org.junit.runner.RunWith;  import io.cucumber.junit.Cucumber;  import io.cucumber.junit.CucumberOptions;  @RunWith(Cucumber.class)  @CucumberOptions(  features = { "src/test/resources/appFeatures" },  glue = { "stepDefinitions", "appHooks" },  plugin = { "pretty" }  )  public class MyTestRunner {  } |
| Login page feature-  Feature: login page feature  Scenario: login page title  Given user is on login page  When user gets the title of the page  Then page title should be "Login - My Shop"  Scenario: forgot password link  Given user is on login page  Then forgot your password link should be displayed  Scenario: login with correct credentials  Given user is on login page  When user enters username as "cucumbertesting1960@gmail.com"  And user enters password as "Malaravi@123"  And user clicks on login button  Then user gets the title of the page  And page title should be "My account - My Shop" |
| Cucumber properties-  cucumber.publish.enabled=true |