AUTOMATION FRAMEWORK USING CUCUMBER (BDD)

Description: -

This is a web application testing automation Framework using cucumber and selenium web driver. This framework follows Behavioral Driven Development approach. This is a hybrid framework, which follows Page Factory Model that is the enhanced version of Page Object Model. Any team involved in Web Application automation project can use this framework.

Prerequisites: -

Following are the prerequisites to use this framework in any system:

1. JDK (Java Development Kit) should be installed and declared in environment variable.
2. IDE (e.g. Eclipse, IntelliJ Idea) is needed.
    a. For this assignment, Eclipse is used
3. MAVEN should be installed.
4. Cucumber plugin should be installed.
5. Should have connection to global or local Maven repository.
6. Driver of the browser in which the web application will run.
    a. Due to time constraint framework is tested on only chrome but the framework is designed for cross browser. The code is in place for the same.
7. TestNG and Maven plugins should be installed in IDE

Advantages: -

1. This framework is developed in JAVA language and hence absolutely platform independent. It is supported on all operating systems (for e.g. Windows, Linux, Mac).
2. This framework supports cross browser testing. The tester can choose any of the leading browsers in the market i.e. Firefox, Chrome, Internet Explorer, Edge etc.
3. Advance concept of page factory model is used
4. testNG features like extent reporting etc. are used.
    a. Extent reporting provides graphical and detailed report
    b. Screenshot captured in report for failed steps which helps in analysis
5. Excel execution status report which will give consolidated status of execution
6. Screenshot capture of all important steps by calling a single function
7. Common functions concept is used to increase maintainability
8. Maven build automation has been used in this framework. So external jar files need not to be attached or downloaded by the testers. During execution all the dependencies and plugins will be added automatically.
9. This framework can be integrated with CI tools like Jenkins for automatic triggering and scheduling the execution.

Execution Procedure: -

- Following are the steps to execute test cases using this framework.

1. Import the project in Eclipse IDE
2. Ensure all pre-requisite as explained above is met
3. Update the config.properties file with correct browser and driver location
4. Open TestRunner.java class
5. Right click and Select Run As > TestNG Test
6. Once execution is complete, refresh the project and expand the report folder for results

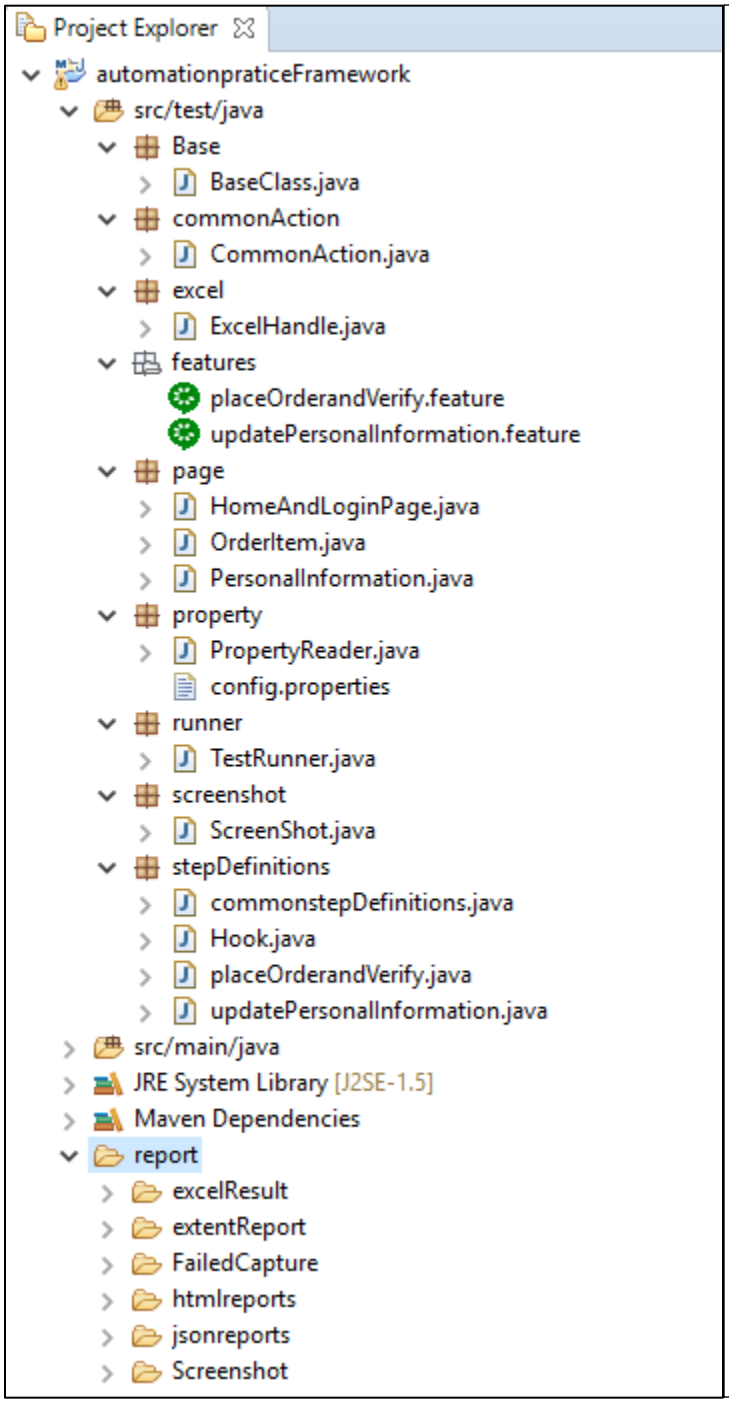
Alternate execution using POM.xml

1. Import the project in Eclipse IDE
2. Ensure all pre-requisite as explained above is met
3. Update the config.properties file with correct browser and driver location
4. Open POM.xml and update the JDK path

```
<configuration>
    <source>1.8</source>
    <target>1.8</target>
    <fork>true</fork>
    <executable>C:\Program Files\Java\jdk1.8.0_162\bin\javac</executable>
</configuration>
```

5. Right click POM.xml and Select Run As > Maven test

Folder Structure: -



- **automationpracticeFramework** is the parent folder which contains the entire framework
- Inside parent folder under **/src/test/java/** all the packages are present.
- **Report** folder contains all different types of reports

Functionalities of the files: -

- **Base Package:** Web driver, TestNG extent reporting is defined in this base class. This class will be extended in other classes so that we can use the web driver and extent reporting features across classes
- **Common Action:** This contains critical or common functions that are common to all the pages. For e.g. clickonElement, enterText etc. If testers want, they can include simple common functions also inside this class. All the methods are static, hence methods are directly called with class name

```
1  package commonAction;
2
3  import org.openqa.selenium.WebDriver;
7
8  public class CommonAction {
9
10     public static void clickOnElement(WebElement element) throws InterruptedException{
11         element.click();
12         Thread.sleep(3000);
13     }
14
15     public static void enterText(WebElement element, String data){
16         element.sendKeys(data);
17     }
18
19     public static String getText(WebElement element){
20         return element.getText();
21     }
22
23     public static void waitFor(WebElement element, WebDriver driver){
24         WebDriverWait wait = new WebDriverWait(driver, 30);
25         wait.until(ExpectedConditions.visibilityOf(element));
26     }
27 }
```

- **Excel:** The execution status of test cases is saved in excel sheet. This class contains the functions which creates the excel sheet, updates execution status of every executed test cases.

```java
public class ExcelHandle {

    public static String excelresultName;

    public static void createExcelFile(String excelName) throws IOException{
        String fileName = System.getProperty("user.dir") + "\\report\\excelResult\\" + excelName + ".xlsx";
        excelresultName = fileName;
        XSSFWorkbook workBook = new XSSFWorkbook();
        XSSFSheet workSheet = workBook.createSheet("Result");
        XSSFRow rowHead = workSheet.createRow((short)0);
        rowHead.createCell(0).setCellValue("Scenario");
        rowHead.createCell(1).setCellValue("Execution_Status");

        FileOutputStream fileOutput = new FileOutputStream(fileName);
        workBook.write(fileOutput);
        workBook.close();
        fileOutput.close();
    }

    public static void updateResultInExcelSheet(String scenario, String result) throws IOException{
        File myFile = new File(excelresultName);
        FileInputStream myXLSX = new FileInputStream(myFile);
        XSSFWorkbook resultWorkbook = new XSSFWorkbook(myXLSX);
        XSSFSheet resultSheet = resultWorkbook.getSheetAt(0);
        int lastRowNum = resultSheet.getLastRowNum();
        XSSFRow row = resultSheet.createRow(++lastRowNum);
        row.createCell(0).setCellValue(scenario);
        row.createCell(1).setCellValue(result);
        FileOutputStream fileOut = new FileOutputStream(myFile);
        resultWorkbook.write(fileOut);

        fileOut.close();
        resultWorkbook.close();
        myXLSX.close();
    }
}
```

- **Features:** This contains the features files
- **Page:** As we are using page factory module, this will contain the list of all objects on a page. Each file corresponds to one web page.

```java
package page;

import org.openqa.selenium.WebDriver;

public class HomeAndLoginPage {

    public HomeAndLoginPage(WebDriver driver){
        PageFactory.initElements(driver, this);
    }

    @FindBy(how = How.XPATH, using = "//a[@title='Log in to your customer account']" )
    public WebElement linkSignIn;

    @FindBy(how = How.XPATH, using = "//input[@id='email']" )
    public WebElement txtUsernameToLogin;

    @FindBy(how = How.XPATH, using = "//input[@id='passwd']" )
    public WebElement txtPasswordToLogin;

    @FindBy(how = How.XPATH, using = "//button[@id='SubmitLogin']" )
    public WebElement btnSubmitToLogin;

    @FindBy(how = How.XPATH, using = "//div[@id='columns']/div[1]/span[2]")
    public WebElement myAccount;

    @FindBy(how = How.XPATH, using = "//*[@id='center_column']/div/div[1]/ul/li[4]/a/span")
    public WebElement btnUpdatePersonalInformation;

    @FindBy(how = How.XPATH, using = "//a[@title='Log me out']" )
    public WebElement linkSignOut;
}

```

- **Property:** This class contains the function to retrieve the value of the key from the config.properties file. The values like URL, Web driver etc. can be stored here. This framework is designed to execute on multiple browser and the configuration is done through this file

Class file

```java
package property;

import java.io.FileInputStream;

public class PropertyReader {

    Properties properties = new Properties();
    InputStream inputStream = null;

    public PropertyReader() {
        loadProperties();
    }
    private void loadProperties(){
        try{
            inputStream = new FileInputStream("src/test/java/property/config.properties");
            properties.load(inputStream);
        }catch(Exception e){
            e.printStackTrace();
        }

    }
        public String readProperty(String key){
            return properties.getProperty(key);
        }

}
```

Property file

```
url = http://www.automationpractice.com
browser = chrome
webDriverPath = D:/Driver/chromedriver.exe
```

- **Runner:** This class will execute the BDD tests

```java
package runner;

import cucumber.api.CucumberOptions;

@CucumberOptions(
        features = "src/test/java/features/",
        //tags = "@Order_Tshirt", //This will be useful when we want to divide the overall suite in sub divisions like regression, smoke, functional etc
        format = {"pretty","json:report/jsonreports/ExecutionReportInJson.json","html:report/htmlreports"},
        glue = "stepDefinitions",
        monochrome = true,
        strict = true,
        dryRun = false)
public class TestRunner extends AbstractTestNGCucumberTests{
}
```

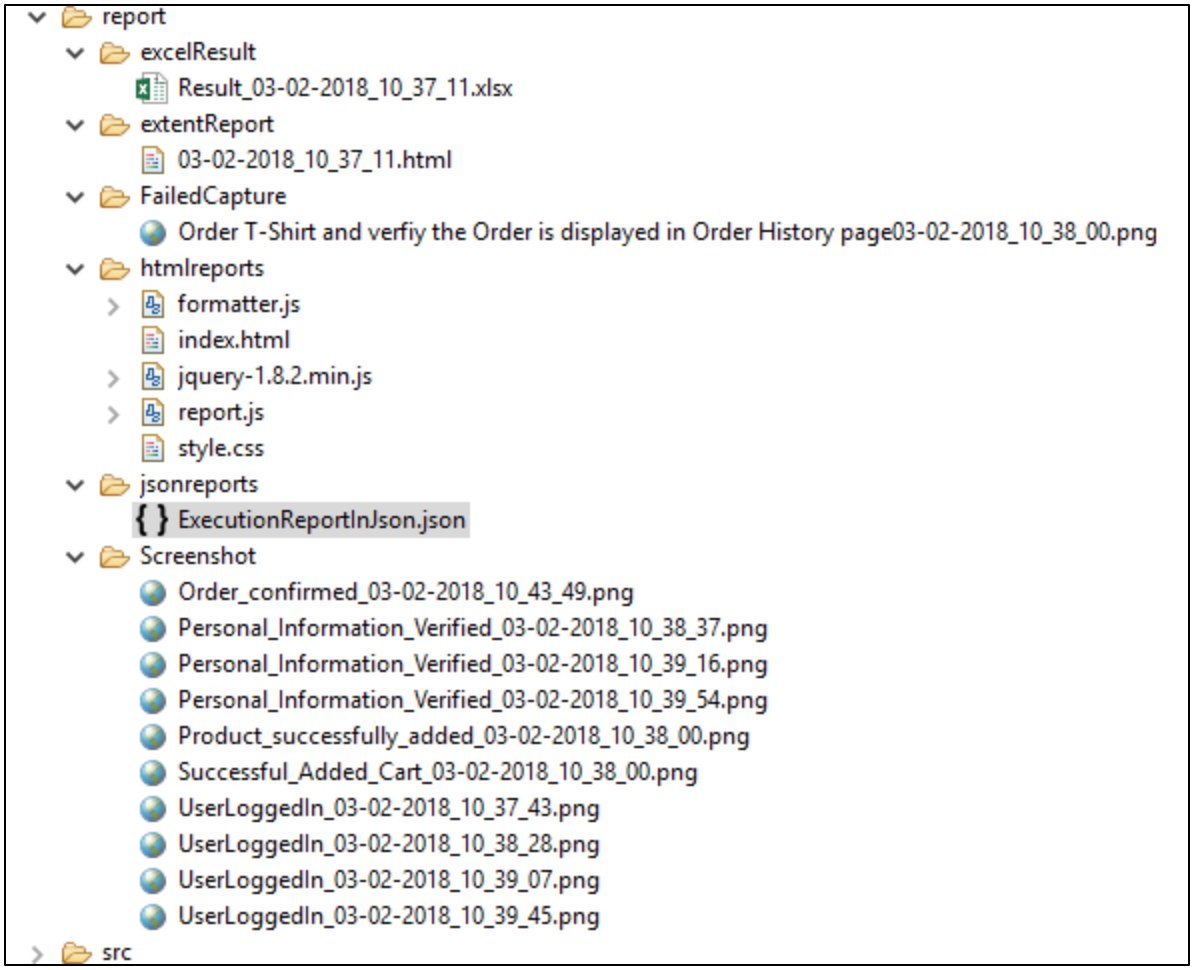
- **Screenshot:** This class has functions to capture screenshots. The framework captures all important screenshots during execution. We just need to call a function to capture it

```java
package screenshot;

import org.apache.commons.io.FileUtils;
public class ScreenShot {

    public static String screenCapture(WebDriver driver, String screenShotName) throws IOException{
    TakesScreenshot ts = (TakesScreenshot)driver;
    File source = ts.getScreenshotAs(OutputType.FILE);
    String path = (System.getProperty("user.dir")+"/report/FailedCapture/");
    String dest = path+screenShotName+".png";
    File destination = new File(dest);
    FileUtils.copyFile(source, destination);

    return dest;
    }

    public static void captureScreen(WebDriver driver, String screenShotName) throws IOException{
        TakesScreenshot ts = (TakesScreenshot)driver;
        File source = ts.getScreenshotAs(OutputType.FILE);
        Calendar calendar = Calendar.getInstance();
        SimpleDateFormat formater = new SimpleDateFormat("dd-MM-yyyy_hh_mm_ss");
        String path = (System.getProperty("user.dir")+"/report/Screenshot/");
        String dest = path+screenShotName+"_"+formater.format(calendar.getTime())+".png";
        File destination = new File(dest);
        FileUtils.copyFile(source, destination);
    }
}

```

- **StepDefinition:** This class will contain the actual code for the BDD scenarios.

Test Reporting: -

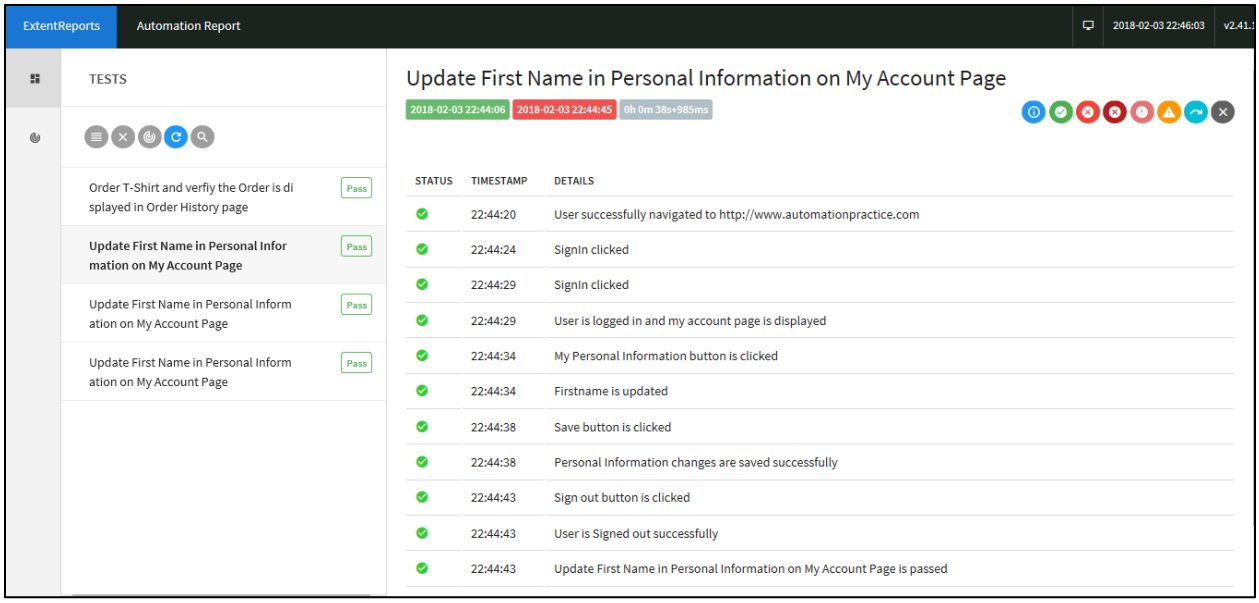
The report folder will contain below subfolders:

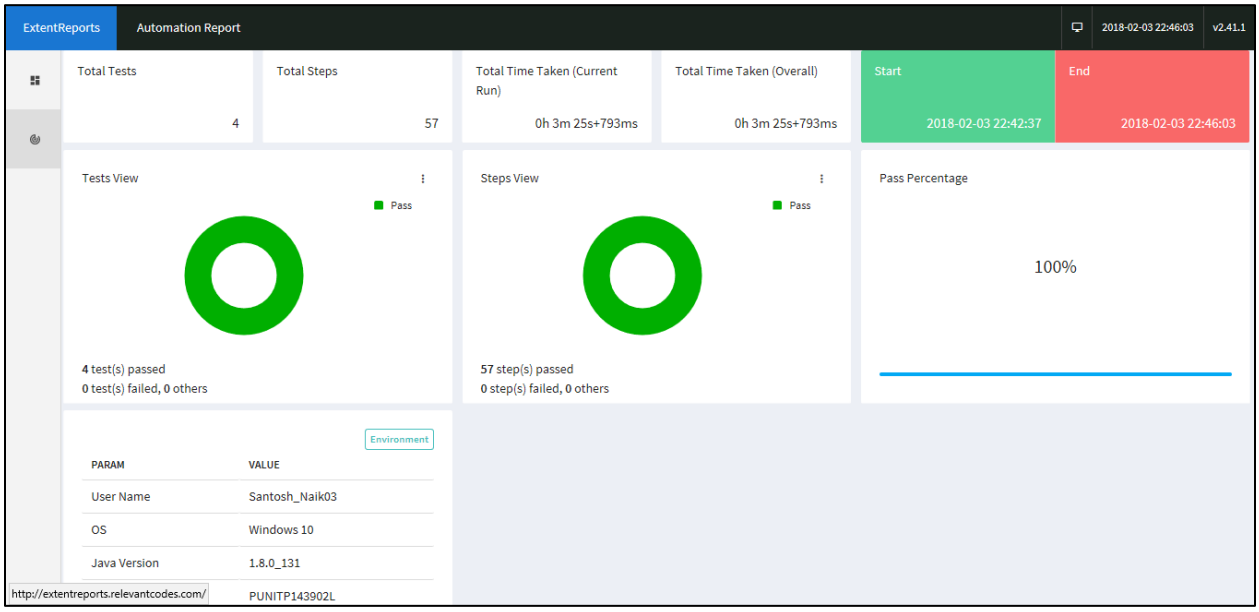


- excelReport – The status of each test case will be stored in this folder. Time stamp will be used in filename to avoid overwriting the file
- extentReport – testNG extent report will be stored in this folder with timestamp
- FailedCapture – The screenshot for failed steps will be stored in this location for reference
- htmlreports – This folder contains the cucumber reports
- jsonreports – This folder contains json reports
- Screenshot – All important steps screenshots will be stored here

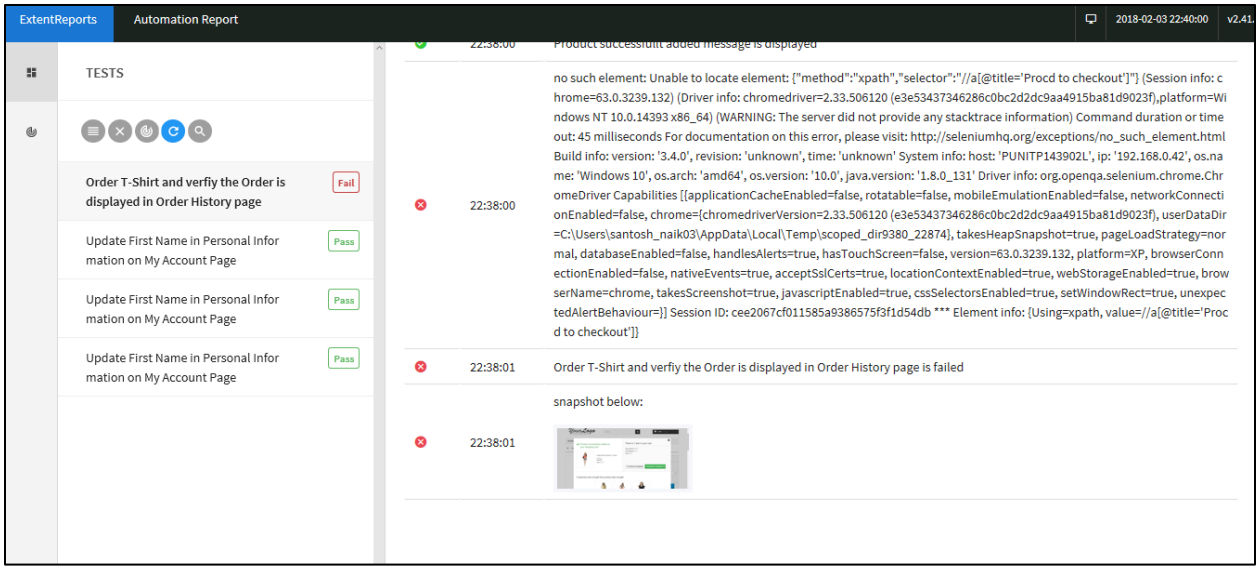
This framework has three different types of reporting. User can select any or all three based on their requirement.

- **testNG Reporting:** After every execution one testNg report will be generated. It will contain the number of test cases passed, failed and skipped. It will also contain the details of different iteration
- **Extent Reporting:** After every execution Extent report will generate one html report containing all the steps information and result of every test case iteration wise. This report will be generated with time stamp inside **report** folder under parent directory.





If test case fails, then the extent report will show the exact cause of the failure and screenshot of the page at which the test case was failed.



- **Cucumber Reports:** These reports are generated by cucumber

▼ **Feature**: Place Order and verify if the order is displayed in Order history
Description : This feature will place the order and verify if the order reference number is displayed in Order History page
▼ **Background**: User is Logged in
Given User is on home page
When User Navigate to Sign in Page
And User enters Username and Password on Sign in Page

username	password
someone@example.com	Password123

Then User is logged in and is on My Account page
▶ **Scenario**: Order T-Shirt and verfiy the Order is displayed in Order History page
▼ **Feature**: Update Personal Information on My Account Page
Description: This feature will test if user is able to update the Personal Information on My Account page
▼ **Scenario Outline**: Update First Name in Personal Information on My Account Page
Given User navigates to home page
When User clicks on Sign in link
And User enters "<username>" and "<password>"
Then User is logged in and My Account page is displayed
When User click on My Personal Information button
And User update "<firstname>" with valid allowed name and enter valid "<reenter_password>"
And User clicks on Save
Then Your personal information has been successfully updated message is displayed
When User clicks on Sign out
And User is logged out
▼ **Examples**:

username	password	firstname	reenter_password
someone@example.com	Password123	changedName	Password123
someone@example.com	Password123	nameChanged	Password123
someone@example.com	Password123	NewName	Password123

▶ **Scenario Outline**: Update First Name in Personal Information on My Account Page
▶ **Scenario Outline**: Update First Name in Personal Information on My Account Page
▶ **Scenario Outline**: Update First Name in Personal Information on My Account Page

- **Excel Reporting:** This excel will store the execution status of the test case

	A	B	C
1	Scenario	Execution_Status	
2	Order T-Shirt and verfiy the Order is displayed in Order History page	Failed	
3	Update First Name in Personal Information on My Account Page	Passed	
4	Update First Name in Personal Information on My Account Page	Passed	
5	Update First Name in Personal Information on My Account Page	Passed	
6			

- **Json reports:**


```
[
  {
    "line": 1,
    "elements": [
      {
        "line": 4,
        "name": "User is Logged in",
        "description": "",
        "type": "background",
        "keyword": "Background",
        "steps": [
          {
            "result": {
              "duration": 10402405246,
              "status": "passed"
            },
            "line": 5,
            "name": "User is on home page",
            "match": {
              "location": "placeOrderandVerify.Given_User_is_on_home_page()"
            },
            "keyword": "Given "
          },
          {
            "result": {
              "duration": 4899222340,
              "status": "passed"
            },
            "line": 6,
            "name": "User Navigate to Sign in Page",
            "match": {
              "location": "placeOrderandVerify.When_User_Navigate_to_Sign_in_Page()"
            },
            "keyword": "When "
          },
          {
            "result": {
              "duration": 4833890294,
```