```java
package com.hrm.base;

public interface AutomationConstants
{

	//Automation constant is an interface were we were maintaining the static
	 //constant variables
	//especially path to specific file like CONFIG_PATH,CHROME_KEY

	public static  final String CONFIG_PATH="./config/config.properties";

	public static  final String REPORT_PATH="./report/";

	public static  final String INPUT_PATH="./data/input.xlsx";

	public static  final String CHROME_KEY="webdriver.chrome.driver";

	public static  final String CHROME_VALUE="./driver/chromedriver.exe";

	public static  final String GECKO_KEY="webdriver.gecko.driver";

	public static  final String GECKO_VALUE="./driver/geckodriver.exe";

	public static  final String SNAP_PATH="./snap/";

	public static  final String IMG_PATH="./testInputImages/";


}
```

```java
package com.hrm.base;

import java.lang.reflect.Method;
import java.util.concurrent.TimeUnit;
import org.apache.log4j.Logger;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.ITestResult;
import org.testng.annotations.AfterClass;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.AfterSuite;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.BeforeSuite;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Listeners;
import org.testng.annotations.Optional;
import org.testng.annotations.Parameters;

import com.hrm.pages.DashboardPage;
import com.hrm.pages.LoginPage;
import com.relevantcodes.extentreports.ExtentReports;
import com.relevantcodes.extentreports.ExtentTest;
import com.relevantcodes.extentreports.LogStatus;

import generics.TestListener;
import generics.Utility;

@Listeners(TestListener.class)
public abstract class BaseTest implements AutomationConstants
{

	//BaseTest is an abstract class which implements AutomationConstants,
	//and it contains various methods which is annotated by testNg
	 //annotation,constructors.

	public WebDriver driver;
	public Logger log;
	public boolean loginlogoutRequired;

	public static ExtentTest eTest;
	public static ExtentReports eReport;

	public static String url;
	public static String un;
	public static String pw;
	public static long iTimeout;
	public static long eTimeout;
	public static String reportFile;
```

```
//Here we were initializing the Log4j and we were specifying Login is required
	public BaseTest()
	{
		log=Logger.getLogger(this.getClass());
		loginlogoutRequired=true;
	}


//Here we can initialize the ExtentReport & initialize the variables like
//URL,UN,PW,Implicit & Explicit wait by taking data from property file

	@BeforeSuite
	public void initFrameWork()
	{
		log.info("Initializing ExtentReport");
		String now=Utility.getFormatedDateTime();
		reportFile=REPORT_PATH+now+".html";
		eReport=new ExtentReports(reportFile);

		log.info("Initialize Global Variables");
		url=Utility.getPropertyValue(CONFIG_PATH,"URL");
		un=Utility.getPropertyValue(CONFIG_PATH,"UN");
		pw=Utility.getPropertyValue(CONFIG_PATH,"PW");

	    iTimeout=Long.parseLong(Utility.getPropertyValue(CONFIG_PATH,"IMPLICIT"));

	    eTimeout=Long.parseLong(Utility.getPropertyValue(CONFIG_PATH,"EXPLICIT"));

	}

	//Here we were setting the path to the driver executables
	@BeforeTest
	public void initGlobalVar()
	{
		System.setProperty(CHROME_KEY,CHROME_VALUE);
		System.setProperty(GECKO_KEY,GECKO_VALUE);
	}


	//It checks & opens browser & sets timeout & maximize the window
	@Parameters({"browser"})
	@BeforeClass
	public void initApplication(@Optional("chrome")String browser)
	{
		log.info("Opening Browser:"+browser);
		if(browser.equals("chrome"))
		{
			driver=new ChromeDriver();
		}
		else
		{
			driver=new FirefoxDriver();
		}
		driver.manage().timeouts().implicitlyWait(iTimeout,TimeUnit.SECONDS);
		driver.manage().window().maximize();
```

```
	}


//Here we were entering the URL,then Login & making the test in Extent report as
started
	@BeforeMethod
	public void preCondition(Method method)
     {
		driver.get(url);
		if(loginlogoutRequired)
		{
			log.info("Auto login");
			new LoginPage(driver).login(un,pw);
		}
		eTest=eReport.startTest(method.getName());
		log.info("Started executing test:"+method.getName());
	}


//It check the status of the test & if the test is fail it will take screen shot &
make test as fail   else make test as pass

	@AfterMethod
	public void postCondition(ITestResult testNGTestResult)
    {
		if(testNGTestResult.getStatus()==ITestResult.FAILURE)
		{
			String imgPath = Utility.getScreenShot(driver,REPORT_PATH);
			String path = eTest.addScreenCapture("."+imgPath);
			eTest.log(LogStatus.FAIL,"Check log for details",path);
			log.error("Test is FAILED");
		}
		else
		{
			eTest.log(LogStatus.PASS,"Script executed successfully");
			log.info("Test is PASSED");
		}


		if(loginlogoutRequired)
	       {
			log.info("Auto logout");
			new DashboardPage(driver).logout();
		}
		eReport.endTest(eTest);
	}
}

     //Here we were closing the browser
	@AfterClass
	public void closeApplication()
     {
		log.info("Closing Browser");
		driver.quit();
	}
```

```
//Here the user can get the graphical report
@AfterSuite
public void publishReport()
{
	log.info("Publishing ExtentReport:"+reportFile);
	eReport.flush();
}
```

```java
package com.hrm.base;

import generics.Utility;
import org.testng.Assert;
import java.util.ArrayList;
import org.openqa.selenium.By;
import org.apache.log4j.Logger;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.PageFactory;
import org.openqa.selenium.support.ui.WebDriverWait;

public abstract class BasePage
{

	//Base page is an abstract class which contains Base Page constructor
	// Initializing the WebDeriver,
	//Initializing the webelements using pagefactory initElements method,
	//we were setting the Explicit wait time
	//Which inturn contains method like list is duplicate,list is sorted,element
is present.

     protected Logger log = Logger.getLogger(this.getClass());
     protected long timeout =
Long.parseLong(Utility.getPropertyValue(AutomationConstants.CONFIG_PATH,"EXPLICIT"));
	protected WebDriver driver;
	protected WebDriverWait wait;

	protected BasePage(WebDriver driver)
    {
		this.driver = driver;
		PageFactory.initElements(driver, this);
		wait = new WebDriverWait(driver, timeout);
	}

	protected void checkListBoxIsSorted(WebElement listBox, int startIndex)
     {
     ArrayList<String> allText = Utility.getAllTextFromListBox(listBox, startIndex);
		boolean sorted = Utility.checkArrayListIsSorted(allText);
		Assert.assertTrue(sorted, "FAIL:ListBox is Not Sorted");
		log.info("PASS:ListBox is Sorted");
	}

	protected void checkListBoxHasNoDuplicate(WebElement listBox, int startIndex)
    {

    ArrayList<String> allText = Utility.getAllTextFromListBox(listBox, startIndex);
		boolean noDuplicate = Utility.checkArrayListHasNoDuplicate(allText);
		Assert.assertTrue(noDuplicate, "FAIL:ListBox has duplicate");
		log.info("PASS:ListBox has no duplicate");
	}

	protected void checkElementIsPresent(WebElement element)
     {
		boolean present = Utility.verifyElementIsPresent(wait, element);
```

```
            Assert.assertTrue(present, "FAIL:Element is not present");
            log.info("PASS:Element is present");
      }
      protected void checkElementIsPresent(By locator)
    {
            boolean present = Utility.verifyElementIsPresent(wait, locator);
            Assert.assertTrue(present, "FAIL:Element is not present");
            log.info("PASS: Element is present");
      }

      protected void checkElementIsNotPresent(WebElement element)
     {
            boolean present = Utility.verifyElementIsPresent(wait, element);
            Assert.assertFalse(present, "FAIL:Element is present");
            log.info("PASS: Element is Not present");
      }

      protected void checkElementIsNotPresent(By locator)
    {
            boolean present = Utility.verifyElementIsPresent(wait, locator);
            Assert.assertFalse(present, "FAIL:Element is present");
            log.info("PASS: Element is Not present");
      }

}
```

```java
package com.hrm.pages;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;

import com.hrm.base.BasePage;

public class LoginPage extends BasePage{
	@FindBy(id="txtUsername")
	private WebElement txtUsername;

	@FindBy(id="txtPassword")
	private WebElement txtPassword;

	@FindBy(id="btnLogin")
	private WebElement btnLogin;

	public LoginPage(WebDriver driver) {
		super(driver);
	}

	public void login(String un,String pw){
		txtUsername.sendKeys(un);
		txtPassword.sendKeys(pw);
		btnLogin.click();
	}

	public void checkLoginButtonIsPresent(){
		checkElementIsPresent(btnLogin);
	}
}
```

```java
package com.hrm.scripts;
import org.testng.annotations.Test;

import com.hrm.base.BaseTest;

public class LoginLogoutTest extends BaseTest{
	@Test(priority=1,enabled=true)
	public void testAutoLoginLogout1(){
		log.info("This script will automatically login and logout");
	}
}
```

```java
package com.hrm.scripts;

import org.testng.annotations.Test;

import com.hrm.base.BaseTest;
import com.hrm.pages.AddEmpPage;
import com.hrm.pages.AddEmployeePage;
import com.hrm.pages.DashboardPage;

import generics.Utility;

/* LAVANYA R -- 27-02-2018 */


public class TestAddEmployee extends BaseTest
{
	@Test(priority=8,enabled=true)
	public void testAddEmployee() throws Exception
	{

		int rc=Utility.getExcelRowCount(INPUT_PATH, "AddEmp");

		for(int i=1;i<=rc;i++){
	   String fn=Utility.getExcelCellValue(INPUT_PATH, "AddEmp",i,0);
	   String mn=Utility.getExcelCellValue(INPUT_PATH, "AddEmp",i,1);
	   String ln=Utility.getExcelCellValue(INPUT_PATH, "AddEmp",i,2);


		String filePath=Utility.getExcelCellValue(INPUT_PATH, "AddEmp",i,3);

		String un=Utility.getExcelCellValue(INPUT_PATH, "AddEmp",i,4);
		String pswd=Utility.getExcelCellValue(INPUT_PATH, "AddEmp",i,5);
		String confirmPswd=Utility.getExcelCellValue(INPUT_PATH, "AddEmp",i,6);


			// go to PIM->add employee
			DashboardPage dbPage=new DashboardPage(driver);
			dbPage.clickPIM_Menu();
			dbPage.click_AddEmp_Menu();
			//enter Fn, LN & Save;
```

```
                    AddEmployeePage addEmp=new AddEmployeePage(driver);

                    addEmp.setFirstName(fn);
                    addEmp.setMiddleName(mn);
                    addEmp.setLastName(ln);

                    addEmp.setphotofile(filePath);

                    //Thread.sleep(10000);

                    Object a = addEmp.setchkLogin();

                    System.out.println("BBBBBBBB:::"+a);

                    addEmp.setuserName(un);

                    addEmp.setUserPassword(pswd);
                    addEmp.setConfirmPassword(confirmPswd);

                    addEmp.clickSave();


            }
        }


}
```

```
package generics;

import org.testng.ITestContext;
import org.testng.ITestListener;
import org.testng.ITestResult;

import com.hrm.base.AutomationConstants;
```

```
public class TestListener implements ITestListener{


	public void onTestStart(ITestResult result) {
	}

	public void onTestSuccess(ITestResult result) {
	}

	public void onTestFailure(ITestResult result) {
		Utility.getScreenShot(AutomationConstants.SNAP_PATH);
	}

	public void onTestSkipped(ITestResult result) {
    }

	public void onTestFailedButWithinSuccessPercentage(ITestResult result) {
	}

	public void onStart(ITestContext context) {
	}

	public void onFinish(ITestContext context) {
	}

}
```

```
package generics;

import java.awt.Dimension;
import java.awt.Rectangle;
import java.awt.Robot;
import java.awt.Toolkit;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileInputStream;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Date;
import java.util.HashSet;
import java.util.List;
import java.util.Properties;
import java.util.Set;

import javax.imageio.ImageIO;

import org.apache.commons.io.FileUtils;
import org.apache.poi.ss.usermodel.WorkbookFactory;
import org.openqa.selenium.By;
```

```
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.interactions.Actions;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.Select;
import org.openqa.selenium.support.ui.WebDriverWait;

public class Utility {

	public static String getFormatedDateTime(){
		SimpleDateFormat simpleDate = new
SimpleDateFormat("dd_MM_yyyy_hh_mm_ss");
		return simpleDate.format(new Date());
	}

	public static String getScreenShot(WebDriver driver, String imageFolderPath){
		String imagePath=imageFolderPath+"/"+getFormatedDateTime()+".png";
		TakesScreenshot page=(TakesScreenshot) driver;
		try{
			FileUtils.copyFile(page.getScreenshotAs(OutputType.FILE), new
File(imagePath));
		}catch(Exception e){

		}
		return imagePath;

	}

	public static String getScreenShot(String imageFolderPath){
		String imagePath=imageFolderPath+"/"+getFormatedDateTime()+".png";

		try{
			Dimension
desktopSize=Toolkit.getDefaultToolkit().getScreenSize();
			BufferedImage image = new Robot().createScreenCapture(new
Rectangle(desktopSize));
			ImageIO.write(image, "png", new File(imagePath));
		}
		catch(Exception e){
		}

		return imagePath;

	}
	public static String getPropertyValue(String filePath,String key)
	{
		String value="";
		Properties ppt=new Properties();
		try{
			ppt.load(new FileInputStream(filePath));
			value=ppt.getProperty(key);
		}
```

```
		catch(Exception e){
		}
		return value;
	}
	public static int getExcelRowCount(String path,String sheet)
	{
		int r=0;
		try{

			r=WorkbookFactory.create(new
FileInputStream(path)).getSheet(sheet).getLastRowNum();
		}
		catch(Exception e)
		{
			e.printStackTrace();
		}
		return r;
	}

	public static String getExcelCellValue(String path,String sheet,int r,int c)
	{
		String v="";
		try{

			v=WorkbookFactory.create(new
FileInputStream(path)).getSheet(sheet).getRow(r).getCell(c).toString();
		}
		catch(Exception e)
		{
			e.printStackTrace();
		}
		return v;
	}

	public static boolean verifyElementIsPresent(WebDriverWait wait,WebElement
element) {
		try {
			wait.until(ExpectedConditions.visibilityOf(element));
			return true;
		} catch (Exception E) {
			return false;
		}
	}

	public static boolean verifyElementIsPresent(WebDriverWait wait, By locator) {
		try {

	wait.until(ExpectedConditions.visibilityOfElementLocated(locator));
			return true;
		} catch (Exception e) {
			//e.printStackTrace();
			return false;
		}
	}
```

```
	public static boolean verifyElementIsNotPresent(WebDriverWait wait,WebElement
element) {
		try {
			wait.until(ExpectedConditions.invisibilityOf(element));
			return true;
		} catch (Exception e) {
			return false;
		}
	}

	public static boolean verifyElementIsNotPresent(WebDriverWait wait, By
locator) {
		try {

	wait.until(ExpectedConditions.invisibilityOfElementLocated(locator));
			return true;
		} catch (Exception e) {
			//e.printStackTrace();

			return false;
		}
	}
	public static void clickUsingJS(WebDriver driver, WebElement e) {
		JavascriptExecutor j = (JavascriptExecutor) driver;
		j.executeScript("arguments[0].click()", e);

	}
	public static ArrayList<String> getAllTextFromListBox(WebElement listBox, int
startIndex) {
		Select select = new Select(listBox);
		List<WebElement> allOptions = select.getOptions();
		ArrayList<String> allText = new ArrayList<String>();
		for (int i = startIndex; i < allOptions.size(); i++) {
			String text = allOptions.get(i).getText();
			allText.add(text);
		}
		return allText;
	}
	public static boolean checkArrayListHasNoDuplicate(ArrayList<String> allText)
{
		HashSet<String> clone = new HashSet<String>();
		for (String s : allText) {
			if (clone.add(s)) {
				System.out.println(s + "->Not Duplicate");
			} else {
				System.out.println(s + "->Is Duplicate");
				return false;// means it has duplicate
			}
		}
		return true; // means it has no duplicate
	}



	public static boolean checkArrayListIsSorted(ArrayList<String> allText) {
```

```
		ArrayList<String> clone = new ArrayList<String>(allText);
		Collections.sort(clone, String.CASE_INSENSITIVE_ORDER);
		// for debugging purpose
		System.out.println("-----------------------------------");
		for (int i = 0; i < clone.size(); i++) {
			System.out.println(allText.get(i) + "==" + clone.get(i));
		}
		System.out.println("-----------------------------------");
		return allText.equals(clone);
	}

	public static boolean switchBrowser(WebDriver driver, String eTitle) {
	String currentWH = "";
		try
		{
			currentWH = driver.getWindowHandle();
		} catch (Exception e)
		{
		}
	Set<String> allWH = driver.getWindowHandles();
	for (String wh : allWH) {
	String title = driver.switchTo().window(wh).getTitle();
		if (title.equals(eTitle)) {
			System.out.println("Browser Found");
			return true;
		}
	}
	driver.switchTo().window(currentWH);
	return false;
}

	public static boolean switchBrowser(String eURL, WebDriver driver) {
	String currentWH = "";
		try {
			currentWH = driver.getWindowHandle();
		}
		catch (Exception e)
		{
		}
		Set<String> allWH = driver.getWindowHandles();
		for (String wh : allWH) {
			String url = driver.switchTo().window(wh).getCurrentUrl();
				if (url.contains(eURL)) {
					System.out.println("Browser Found");
					return true;
				}
		}
		driver.switchTo().window(currentWH);
		return false;
	}



	public static void moveToElement(WebDriver driver, WebElement element) {
		Actions ac = new Actions(driver);
```

```
			ac.moveToElement(element).perform();
		}

		public static boolean verifyURLhas(WebDriverWait wait,String
expectedPartialUrl) {
			try{
				wait.until(ExpectedConditions.urlContains(expectedPartialUrl));
				System.out.println("URL contains:"+expectedPartialUrl);
				return true;
			}
			catch(Exception e){
				System.out.println("URL doest not contains:"+expectedPartialUrl);
				return false;
			}
			}

		public static boolean verifyURLIs(WebDriverWait wait,String
expectedCompleteUrl) {
				try{

		wait.until(ExpectedConditions.urlToBe(expectedCompleteUrl));
					System.out.println("URL is:"+expectedCompleteUrl);
					return true;
				}
				catch(Exception e){
					System.out.println("URL is not:"+expectedCompleteUrl);
					return false;
				}
			}


}
log4j.rootLogger = INFO,FILE,CONSOLE
log4j.appender.FILE=org.apache.log4j.FileAppender
log4j.appender.FILE.File=./report/result.log
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.conversionPattern=%c %d %l %m %n

log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
log4j.appender.CONSOLE.layout.ConversionPattern=%m%n
```

Here is your framework Explanation

In our selenium project we were using java language.

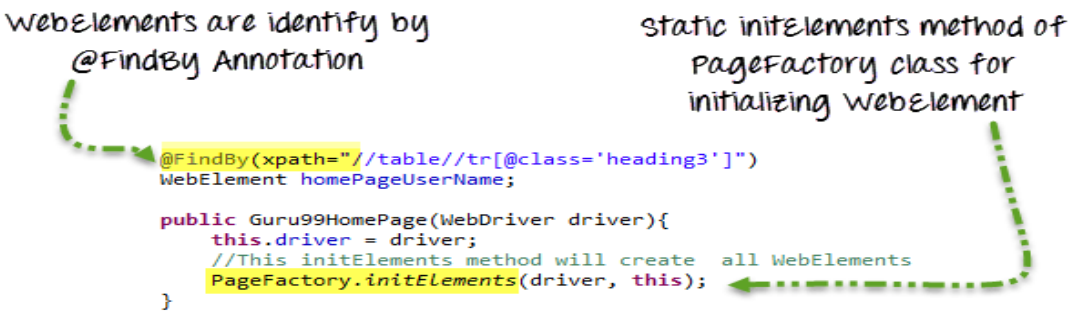
In our project we were using Hybrid framework by using page object model design pattern with page factory.

POM:

As per the Page Object Model, we have maintained a class for every web page. Each web page has a separate class and that class holds the functionality and members of that web page. Separate classes for every individual test.

Page Factory is an inbuilt Page Object Model concept for Selenium WebDriver but it is very optimized.

Here as well, we follow the concept of separation of Page Object Repository and Test Methods. Additionally, with the help of PageFactory class, we use annotations **@FindBy** to find WebElement. We use initElements method to initialize web elements



@**FindBy** can accept **tagName, partialLinkText, name, linkText, id, css, className, xpath** as attributes.

Packages:

We maintained different packages like base, pages, scripts, generics.

In **base package** we were having Automation constant interface,abstract BasePage class, abstract base Test class, abstract Home page class.

Coming to **pages package** we have maintained the separate .java class for every page

Which contains/includes the locators and methods of the respective page.

We were writing and maintaining all our test scripts in **Script package**.

Since were using Test Ng we were prioritizing, enable & disable of test cases were done here.

In **generic pakage** we were having the Utility class and TestListner class.

We were having different folders like **config** which contains config.properties ,

data which contains input.xlsx from were we can pass the data,

driver contains chromedriver.exe,

test Input images for pasing images,

Report folder which contains result.log and extent reports generated.

test-output folder of testing.

Screenshots: Screenshots will be captured and stored in a separate folder and also the screenshots of a failed test cases will be added in the extent reports.

Test Data: All the historical test data will be kept in excel sheet (*controller.xlsx*). By using *'controller.xlsx'*, we pass test data and handle data driven testing. We use Apache POI to handle excel sheets.

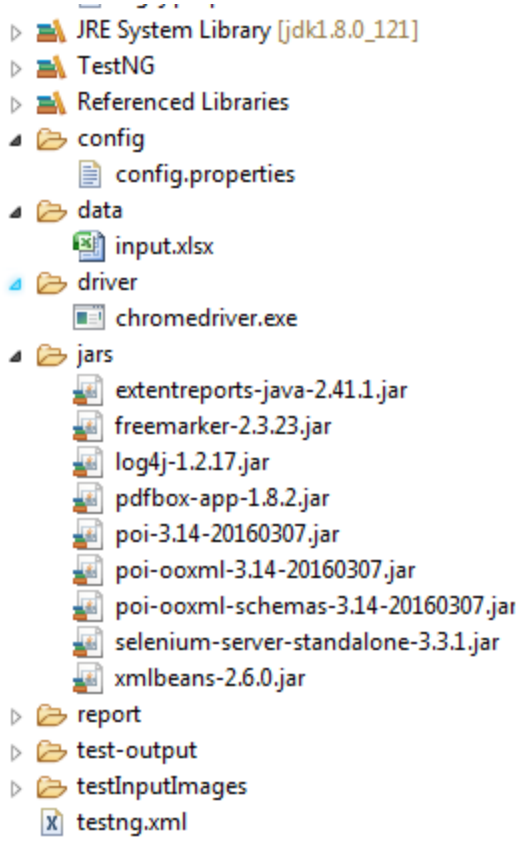
TestNG: Using TestNG for Assertions, Grouping and Parallel execution.

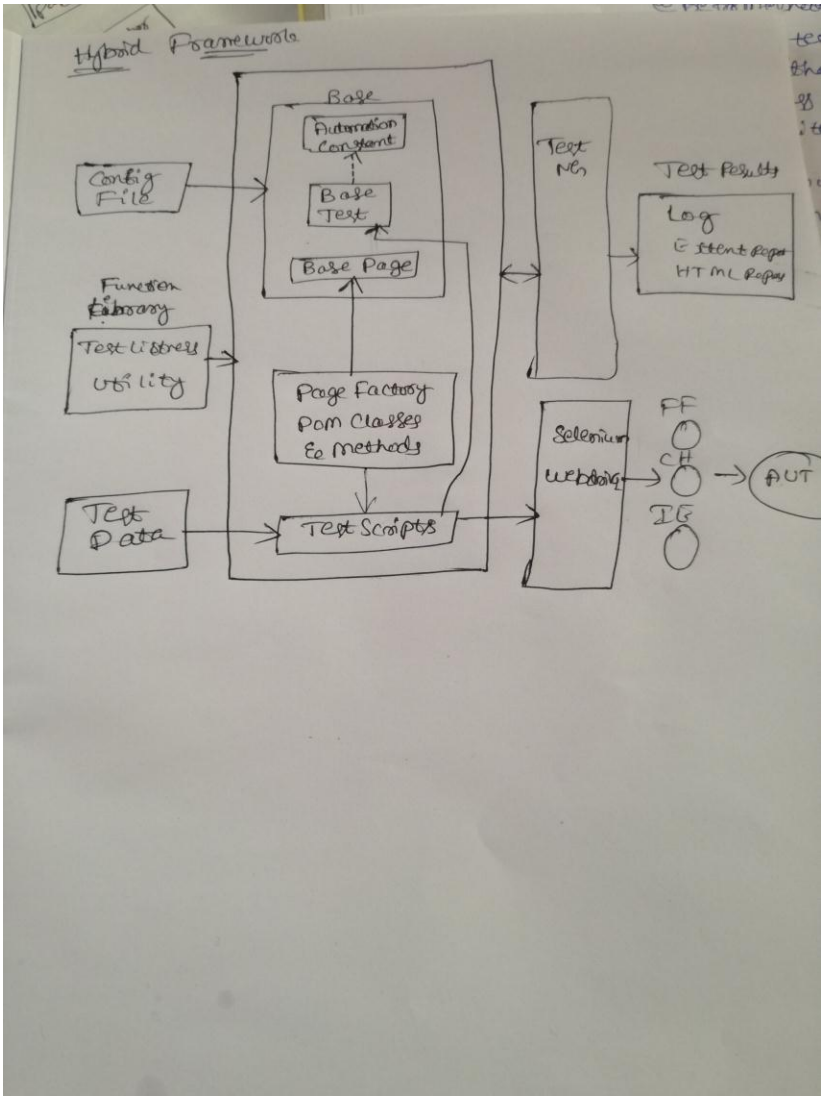
Maven: Using Maven for build, execution and dependency purpose. Integrating the TestNG dependency in POM.xml file and running this POM.xml file using Jenkins.

Version Control Tool: We use Git as a repository to store our test scripts.

Jenkins: By using Jenkins CI (Continuous Integration) Tool, we execute test cases on daily basis and also for nightly execution based on the schedule. Test Result will be sent to the peers using Jenkins.

Extent Reports: For the reporting purpose, we are using Extent Reports. It generates beautiful HTML reports. We use the extent reports for maintaining logs and also to include the screenshots of failed test cases in the Extent Report and log4j.


JRE System Library [jdk1.8.0_121]
TestNG
Referenced Libraries
config
config.properties
data
input.xlsx
driver
chromedriver.exe
jars
extentreports-java-2.41.1.jar
freemarker-2.3.23.jar
log4j-1.2.17.jar
pdfbox-app-1.8.2.jar
poi-3.14-20160307.jar
poi-ooxml-3.14-20160307.jar
poi-ooxml-schemas-3.14-20160307.jar
selenium-server-standalone-3.3.1.jar
xmlbeans-2.6.0.jar
report
test-output
testInputImages
testng.xml


Hybrid Framework
Base
Automation Constant
Base Test
Base Page
Config File
Function Library
Test Utility
Page Factory
POM Classes & Methods
Test Data
Test Scripts
Test NG
Test Results
Log
Extent Report
HTML Report
Selenium Webdriver
FF
CH
IE
AUT