FEBRUARY 10, 2020

KNOWLEDGE DOCUMENT

PCG AUTOMATION

JATIN MANCHANDA

Table of Contents

Pre-requisites

- IDE - IntelliJ
- Java - JDK8
- Git

Setting up the workspace

Forking the repository

A fork is the copy of the original repository.

Forking a repository allows to freely work with changes without affecting the original project. Most commonly, forks are used to either propose changes to the original project or to use original project as a base to develop a module.

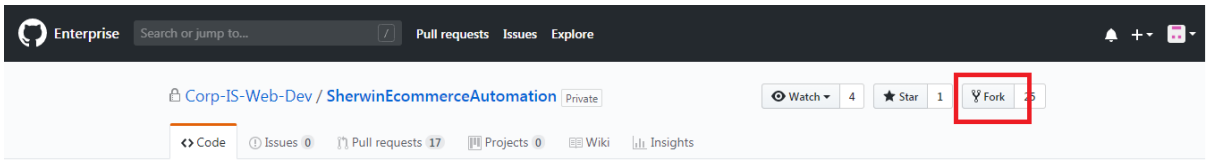
The base project for PCG is **SherwinEcommerceAutomation** and could be found on the following remote repository:

https://github.sherwin.com/Corp-IS-Web-Dev/SherwinEcommerceAutomation

One can access same by logging in with valid credentials (access needs to be provided by admins on user IDs).

Steps to fork the repository:

1. Login to the https://github.sherwin.com
2. Navigate to the **SherwinEcommerceAutomation** repository.
3. Click on the "Fork" option (refer below)



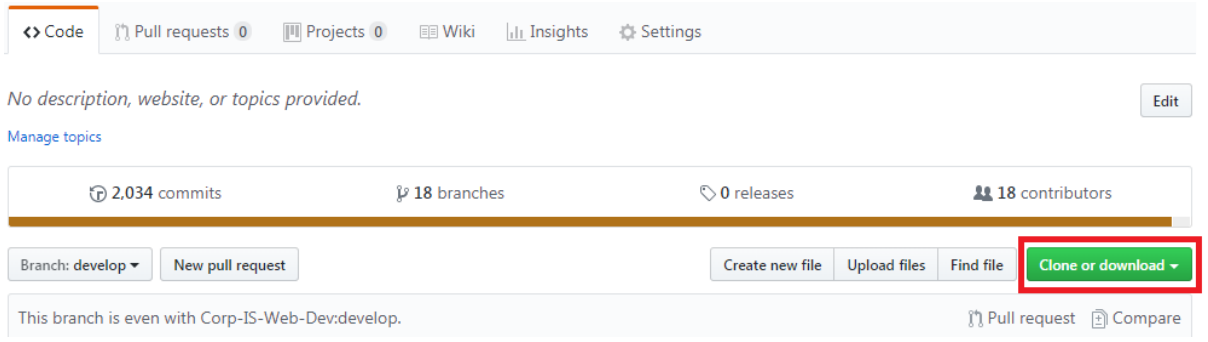
4. Click on the username to fork the repository.
5. Navigate to the forked repository once the forking process is complete.

Cloning the project

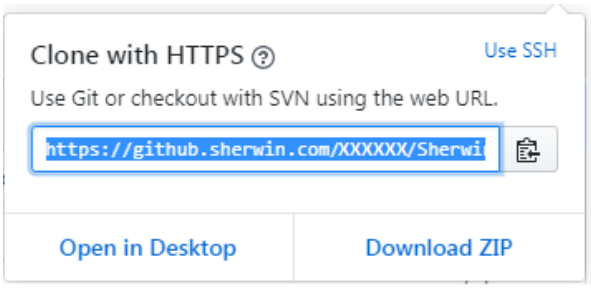
In Git terms, cloning is a process to create a copy of an existing project from remote to local.

Steps to clone the repository:

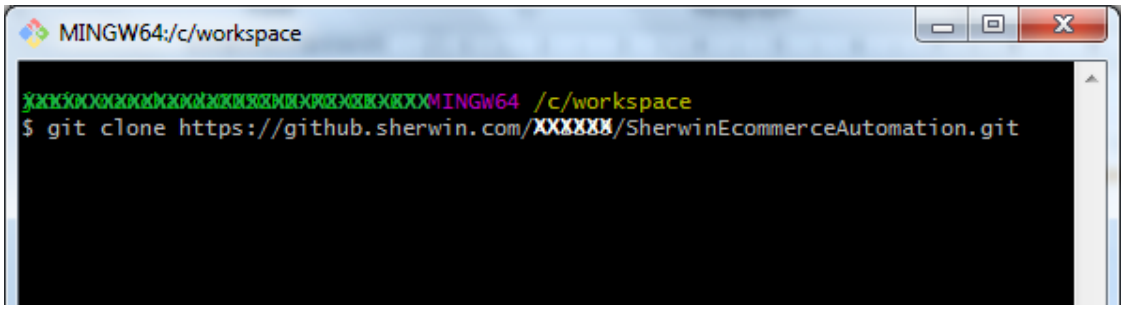
1. Navigate to the forked repository.
2. Click on "Clone or download" button (refer below)



3. Copy the repository URL (refer below)



4. Navigate to local workspace directory.
5. Open “Git Bash” window.
6. Type clone command and hit enter (refer below)



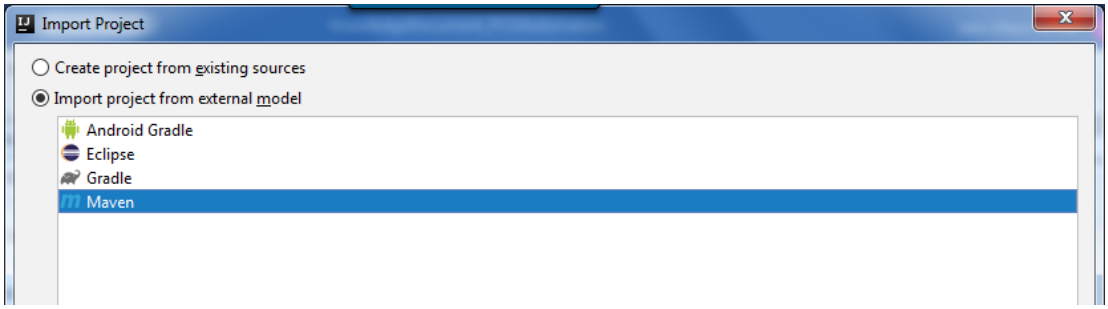
7. Once the clone process is complete, enter the project directory.
8. Checkout to develop branch.

Setting up project in IntelliJ IDE

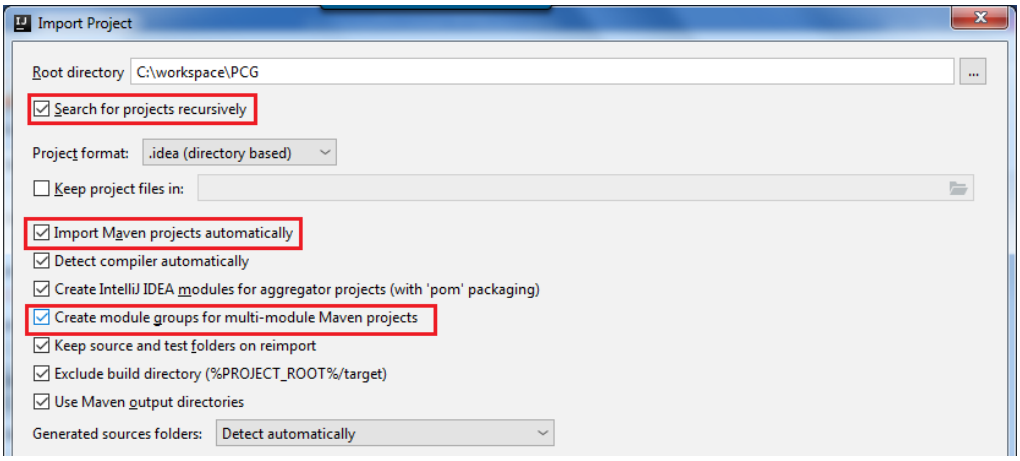
Steps to set up project in IntelliJ IDE:

1. Open IntelliJ Idea community edition.
2. Click on “Import Project” option (refer below)

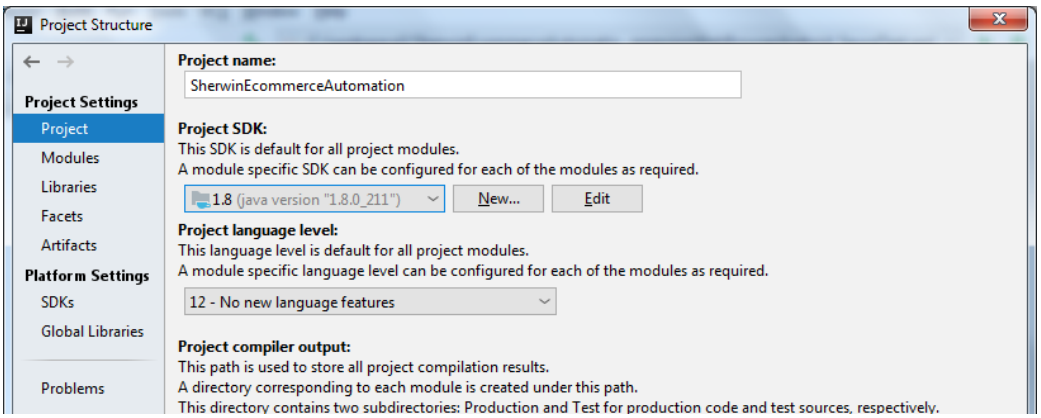
3. Select the project directory path.
4. Select “Import project from external model.”
5. Select “Maven.” (refer below)



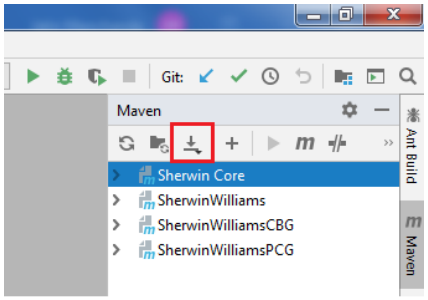
6. Click on the next button.
7. Check following textboxes (if not already checked)



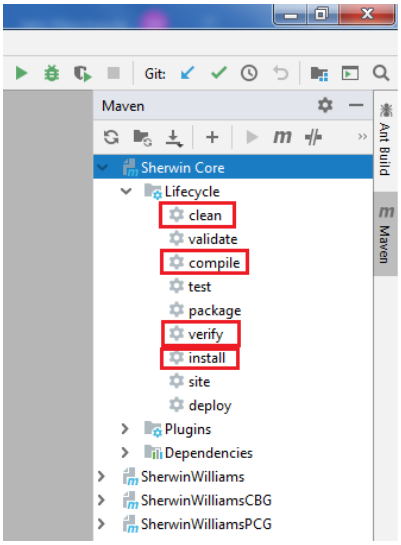
8. Click on Next button.
9. Once the project is loaded, right click on the project and open project settings (or module settings and then click on project settings).
10. Set JDK 1.8 (if not automatically set) in project SDK (refer below)



11. Go to the maven tab and download sources and documentations for all modules (refer below).



12. Once the sources and documents are downloaded, run *clean*, *compile*, *verify* and *install* Maven commands (refer below) for *Sherwin Core* first and then other modules.



Creating branches from forked repository

Branching is required if multiple resources are working on different modules. It is the best practice to branch the code and then make changes to it. To make a new branch on the fork follow these steps:

1. Take latest pull from the develop branch.
2. Run **git checkout -b branch_name** command in GIT bash.

Pulling and Pushing from remote

Pull:

To pull the latest changes from a branch, follow these steps:

1. Checkout onto desired branch.
2. Run **git pull origin branch_name** command in GIT bash.

Push:

To push the changes, follow these steps:

1. Add the changes to stage by running **git add .** (to add all the changes) command in GIT bash.
2. Commit the changes by running **git commit -m commit_message** command
3. Run **git push origin branch_name** command to push the changes to origin.

Updating fork with changes in original project

Since multiple teams can be working on different projects at a time, there can be changes merged into the base code. It is developer's responsibility to pull the latest code from base code and update the fork. Follow below steps to rebase/update the fork:

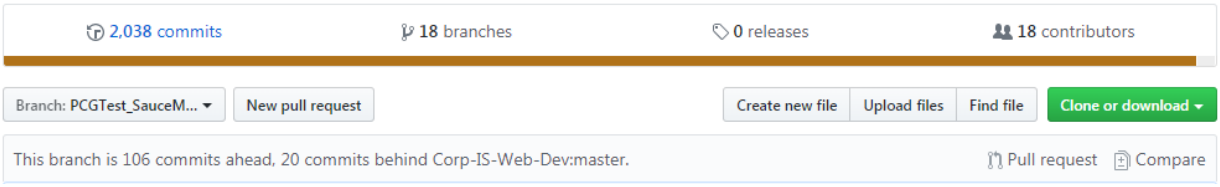
1. Run **git remote add upstream https://github.sherwin.com/Corp-IS-Web-Dev/SherwinEcommerceAutomation** command.
2. Checkout to master branch by running **git checkout master**
3. Run **git fetch upstream**
4. Run **git pull upstream master**
5. Run **git push origin master**
6. Repeat for develop branch

Creating Pull requests

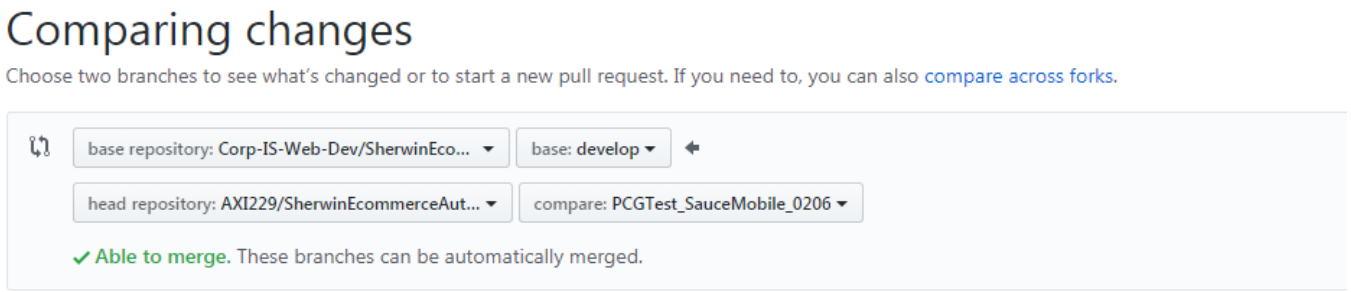
Once the code has been pushed to the forked remote repository, next step is to create a Pull Request to have the reviewers review the code and merge in the base code.

Follow below steps to create the Pull Request:

1. Visit github UI on a browser.
2. Login with valid credentials (same as of the fork)
3. Navigate to the forked repository.
4. Select the branch for which PR is to be raised.
5. Click on “New pull request” button (refer below)



6. Select develop as base branch (refer below)



7. Verify that there are no merge conflicts (resolve if there are some). With the latest code pulled before raising the PR, the chances are less for conflicts.
8. Add comments describing the changes done in the code.
9. Attach test reports, test coverage sheet and developer’s checklist and click on raise a PR.

Sherwin PCG Tests Design

Presently, there are four directories in the project named; *SherwinCBGTests*, *SherwinEcommerceCore*, *SherwinEcommerceTests,* and *SherwinPCGTests.*
Our focus is however on *SherwinEcommerceCore* and *SherwinPCGTests.*
SherwinEcommerceCore contains the base code for all the projects.
SherwinPCGTests contains code specific to PCG tests only.
The dependencies have already been defined in corresponding pom.xml files for all four modules.

Toolset

- Java
- TestNG
- Selenium WebDriver
- ExtentReports
- SauceLabs
- AppliTools

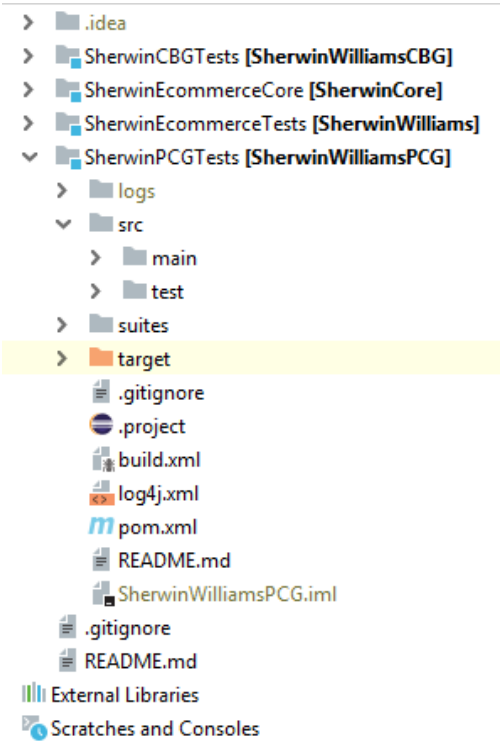
Overview

The project follows *Page Object Model* and deploys *Maven* and *TestNG* framework with *ExtentReports* reporting. Selenium scripts are being written in Java.

Package Structure

SherwinPCGTests contains four packages named *logs, src, suites* and *target*.

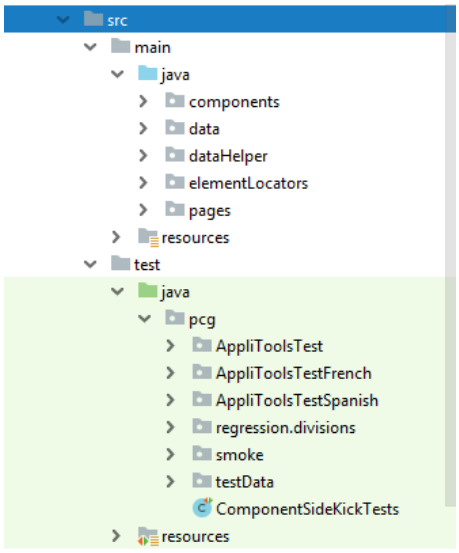
logs package contains saved console logs.
src package contains the source code for PCG tests.
suites package contains the xml files with information of test suites.
target package is the output folder and one can find test reports under it.



Under *src* package, there are two packages named *main* and *test.*

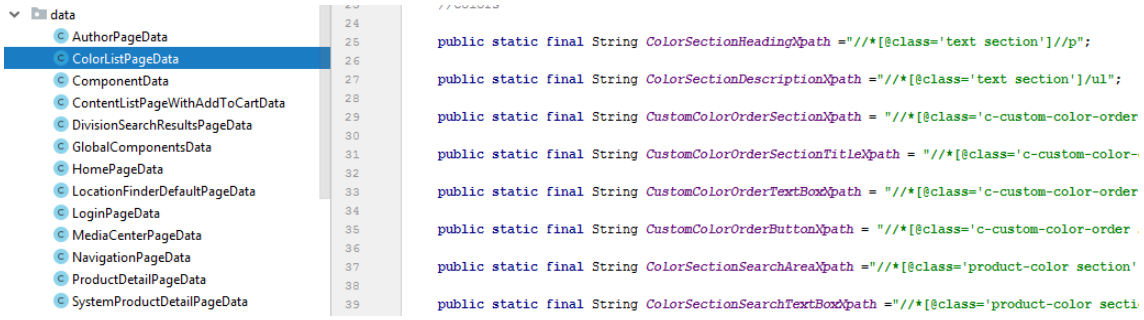
Main package contains component classes, data classes, data helper classes, element locators and page classes in corresponding packages.

Whereas, **Test** package contains all of the test classes in packages per sub modules.



Element locator data:

All the information about the page locators such as xpath, id, linkText, css selector etc is stored under classes in **data** package.

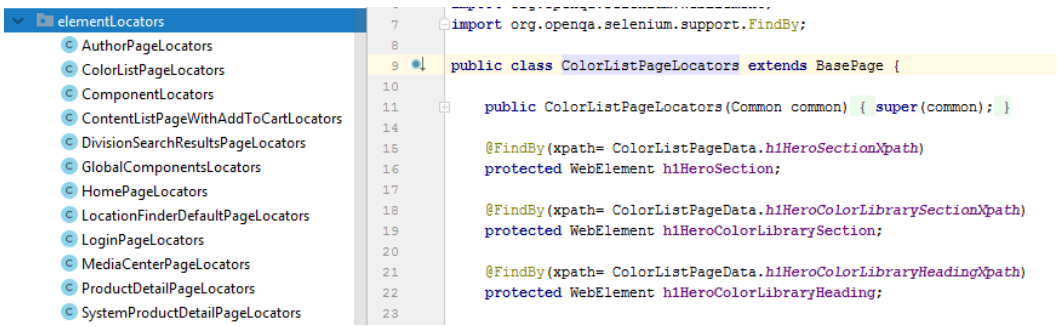


One needs to follow the coding standard to name the element locator; each element location should have the exact name of the field as shown in the application suffixed with the locator type,

For e.g. userNamexpath or passwordcss or signInXPATH or continueCSS

Element Locators:

All the page objects are stored in the classes under elementLocators package. The classes are categorised based on pages same as data classes and are extended from BasePage class.



Pages:

All the actions that are being performed on the application pages are written as code in the classes under pages package.

Components:

All the actions that are being performed on the application components are written in the classes under pages package.

DataHelper

Contains AppDataFromDB class having Database connection code.

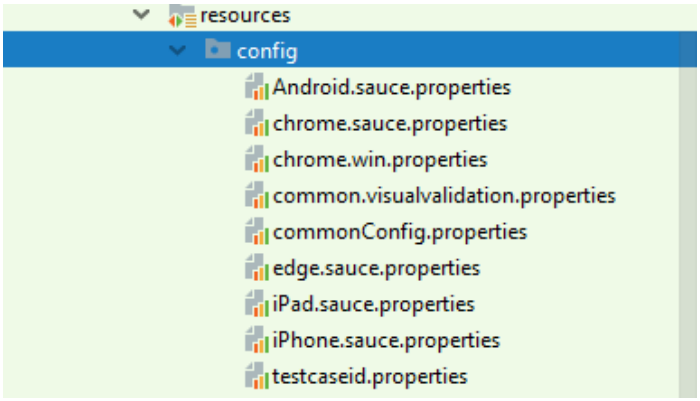
Test

The test package has all the test classes categorised according to the sub module. UI automation test classes extends from EcommerceCoreBaseTest class and Applitools automation test classes extends from ApplitoolsCoreBaseTest class which are in core on the following path SherwinEcommerceCore\src\main\java\com\ecommerce\store\testsuites.

Configurations

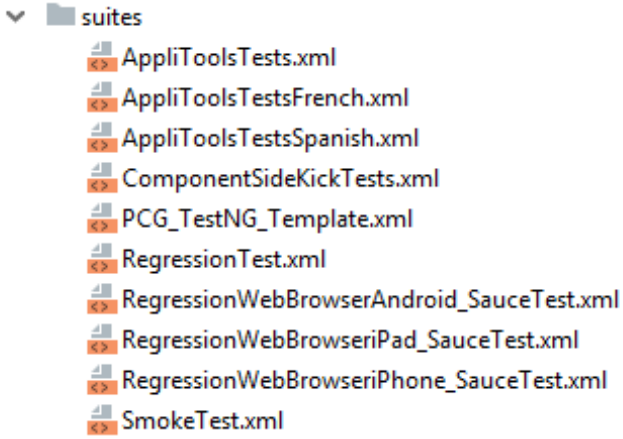
The property configuration files are under *resources > config* package and contains the information about browser, webdriver, environment, versions and base URLs.

resources > config > testcaseid.properties contains test case ids that are stored in the database as well.



Suites

Suites have the testNG xml files to execute scripts according to suites and groups.



Smoke Suite

Smoke suite is aligned with the original framework.

Element locator data is in data classes and element locators are in elementLocators classes categorised based on the 8 pages that are currently available in the smoke suite.

The action methods are written in the page classes directly by consuming page objects. Page classes extends from the corresponding element locator classes.

Test classes with information of test steps, their priorities and URLs are on the following path:

SherwinPCGTests\src\test\java\pcg\smoke

Test classes contains TestNG annotations like @beforeMethod, @Test, and @AfterMethod and make use of dataProvider class.

Regression Suite

Regression suite uses twisted approach to the original framework. The model is however similar to the original framework but the action methods are written in component classes which consumes page objects from *ComponentLocators* class which uses *ComponentData* class for the element locator information.

The page classes calls the methods from component classes based on the division category (refer below example).

```
/**
 * Checks Breadcrumbs.
 *
 * @throws Exception
 */
public AislePage verifyBreadcrumbs(String DivisionName) throws Exception {
    switch (DivisionName.toUpperCase()) {
        case "GENERALINDUSTRIAL":
            breadcrumbs
                    .verifyBreadCrumb()
                    .verifyBreadCrumbNavigation();
            break;
        default:
            common.htmlReporter( result: false,  description: "Method not defined for <b>" + DivisionName + "</b> Division.");
            throw new Exception("Method not defined for <b>" + DivisionName + "</b> Division");
    }
    return this;
}
```

The test classes are categorised on basis of divisions unlike pages in smoke suite. Each test is a method with all the applicable methods on a page i.e. each test is for different page which calls the page methods based on division passed as a parameter.

Applitools Tests

Applitools is an application visual management and AI-powered visual UI testing and monitoring software.

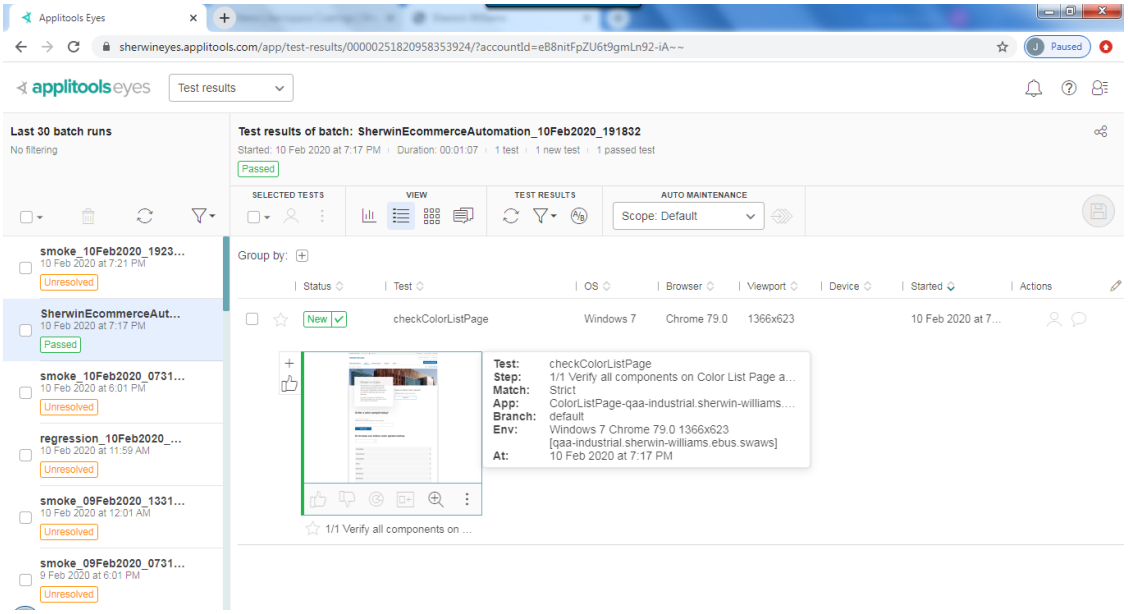
Applitools take the snapshot of the window/component and stores it as a baseline. Whenever the test is re-run, the new snapshot is compared with the baseline and changes are reported on Applitools dashboard based on user preference of severity of change.

In PCG test, applitoolsTest package has page test classes with the testNG annotations which extends from ApplitoolsCoreBaseTest. The test method takes screenshot of the window to baseline and compare with.

Same visual tests can be run with different application languages to check if there is diversion from expected.

```
@Test(groups = {"smoke"}, description = "Verify Color List Page components.")
public void checkColorListPage() {
    checkWindow( checkTagName: "Verify all components on Color List Page are as expected",  change_stitchMode_to_scroll: false);
    Assert.assertTrue(isApplitoolsTestPassed(),  message: "Applitools content validation has either failed and/or unresolved steps.
}
```

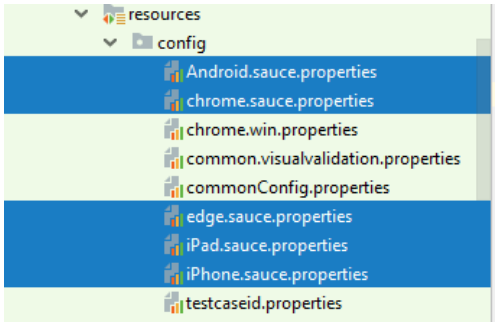
The results are published on Applitools dashboard.



Saucelabs execution

Saucelabs is cloud based platform for cross browser – cross platform execution.

In PCG tests, the configuration has been added in the property files under *resources > config* with following names <browser/platform>.sauce.properties



Sherwin PCG Tests Execution

Smoke Suite

Executing whole suite:

Whole smoke suite can be executed using the xml file located at

SherwinEcommerceAutomation\SherwinPCGTests\suites\SmokeTest.xml

By simply right clicking and selecting Run (or alternatively, using CTRL + SHIFT + F10 in intelliJ)

Executing individual pages:

Open the test class, right click and select Run (or alternatively, using CTRL + SHIFT + F10 in intelliJ)

Executing individual method:

Open the test class, move pointer to the method to run, right click and select Run.

Regression Suite

Executing whole suite:

Whole regression suite can be executed using the xml file located at

SherwinEcommerceAutomation\SherwinPCGTests\suites\RegressionTest.xml

By simply right clicking and selecting Run (or alternatively, using CTRL + SHIFT + F10 in intelliJ)

Executing individual pages:

Open the test class, move pointer to the page method to run, right click and select Run.

AppliTools Test

Executing whole suite:

Whole smoke and regression suite can be executed using the xml file located at

SherwinEcommerceAutomation\SherwinPCGTests\suites\AppliToolsTests.xml or AppliToolsTestsFrench.xml or AppliToolsTestsSpanish.xml based on the language to run

By simply right clicking and selecting Run (or alternatively, using CTRL + SHIFT + F10 in intelliJ)

Executing individual pages:

Open the test class, right click and select Run (or alternatively, using CTRL + SHIFT + F10 in intelliJ)

Saucelabs Test

Executing whole suite:

Whole smoke and regression suites can be executed using the xml file located at

SherwinEcommerceAutomation\SherwinPCGTests\suites

By simply right clicking and selecting Run (or alternatively, using CTRL + SHIFT + F10 in intelliJ)

Executing individual pages:

Since the applicable capabilities are passed from the xml itself, therefore to execute single page, other pages are to be commented out within the xml.

And by right clicking and selecting Run (or alternatively, using CTRL + SHIFT + F10 in intelliJ)

Reporting

html reports

Test reports can be found as html file under target folder at following path:

SherwinEcommerceAutomation\SherwinPCGTests\target

Note:-

1. Report is generated at the end of the execution.
2. The existing report needs to be renamed or copied to another directory to prevent being overwritten by the report of latest run.

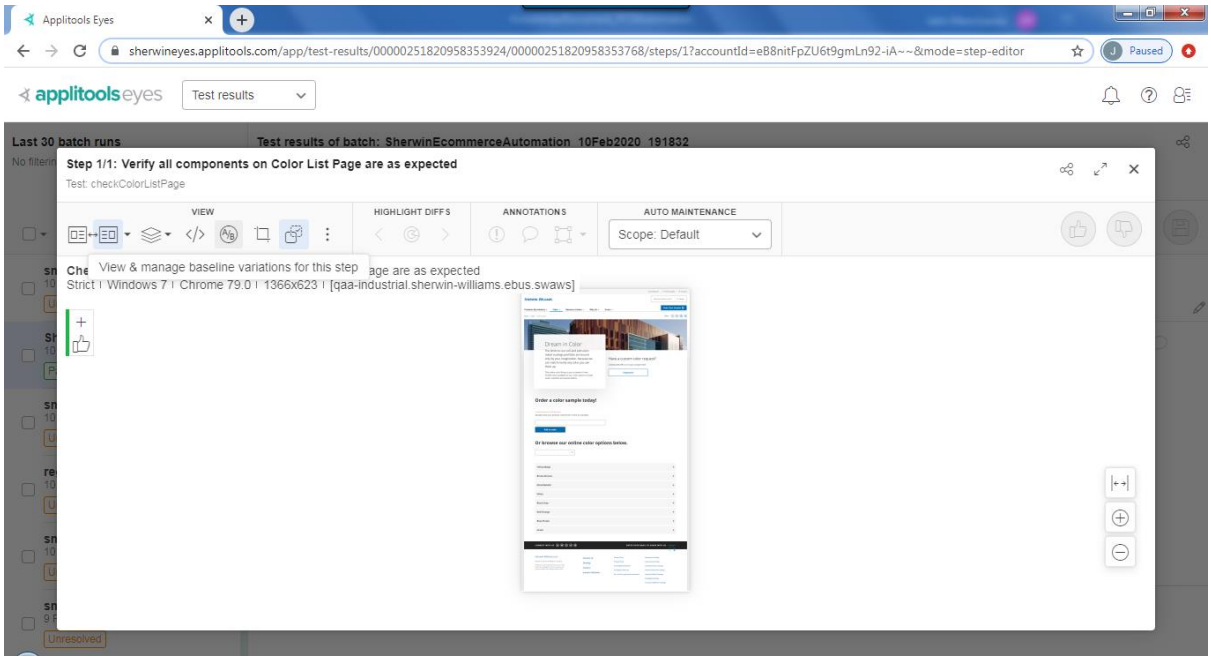
AppliTools reports

Applitools reports can be found on the applitools dashboard at the following location:

https://sherwineyes.applitools.com/app/test-results

Note:-

Users must manage the baseline variations by using applitools UI tools for consistent and reliable results (refer below screenshot)



Possible Issues and preventive measures

Verify that there are no compilation and runtime errors in the script. There can be chances that besides these errors, following issues could be seen:

Tests being skipped

It has been observed that there are chances of tests getting skipped. Check following if it happens:

- ✓ Make sure that the test case ids are added to the property file and to the database.
- ✓ Make sure that the system can connect to the database.
- ✓ Make sure supported versions of browsers, webdriver and testNG are being used.
- ✓ In case, connection is not built, comment out following line of code from AppDataFromDB class before executing (make sure to uncomment before submitting the code for review)
  results.setTestCaseDescription(getNewTestCaseFROMDB(ecommerceTestCaseID).getTestCaseDescription())

Execution not starting

- ✓ Make sure you're connected to Sherwin Willams network and have correct proxy settings.
- ✓ Make sure supported versions of browsers, webdriver and testNG are being used.
- ✓ Make sure maven commands are run after taking pull from remote.
- ✓ Comment eyes.setProxy(new ProxySettings("http://proxy.proxysherwin.com:39000")); from ApplitoolsCoreBaseTest class, if connected through VPN to disable setting proxy to run applitools tests
- ✓ Comment insertResults.execute(); from EcommerceCoreBaseTest.java class, if connected through VPN and running tests on saucelabs.