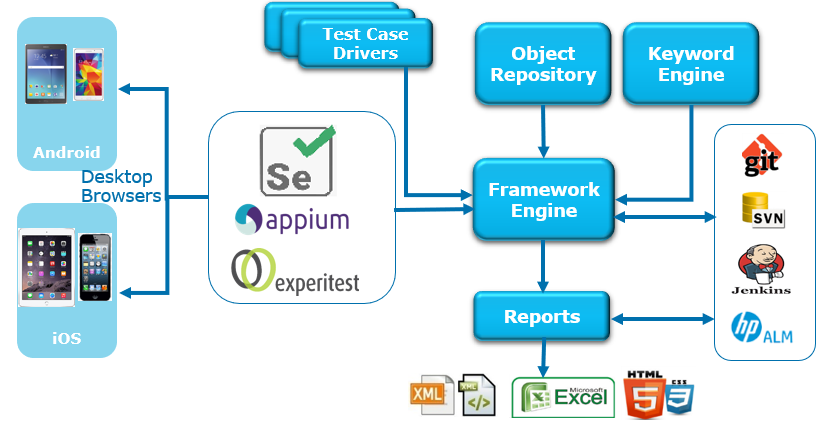
Framework Approach and Setup process:

**Technology stack:** Junit/TestNG, Java, extent reporting libs for rich HTML preview, Eclipse/Intelli J IDE, Selenium libs, browser drivers, Maven, Jenkins for Continuous execution(Compatible)

**Framework Design Diagram :**



The above diagram explains the capability and working model of Digital Hybrid framework.

On very high level my framework contains KeywordClass, PageObjectClass(Object Repository), ReportingClass, TestcaseDriverClass, and other utilities

**KeywordClass(Hybrid Driver)** – Contains all the keywords of Selenium functionality (For eg: click(), elementSendText(), waitForElementPresence(), dragAndDrop(), wait(), sleep() ..etc)

**PageObjectClass** – All the XPath and other elements are recorded in this classes, for better standard create separate class for separate pages in the application – this is equivalent to page object model.

**ReportingClass** – reporting class get logs for every test step execution, since every keyword is surrounded with try-catch exception handling, Pass/Fail log status will be recorded as XML tags vis reporting class.

**TestcaseDriverClass** – Test case driver class contain the test execution data, how many test cases to be executed, from which sheet to pickup for execution, which testing class to be executed, number of parallel executions..etc.

**Test Case Creation Approach:**

We write test cases using our own keywords in TestNG class For eg:

**Click**(HomePage.loginButton);

elementSendText(HomePage.userName, userName)

**Click - our custom keyword**

HomePage – Name of Page which Element is recorded

loginButton – elementName in HomePage

Test data can be directly provided from feature file with Examples via Scenario outline or we can store in any file formats like Excel, JSON or Database and write helper classes to supply test data.

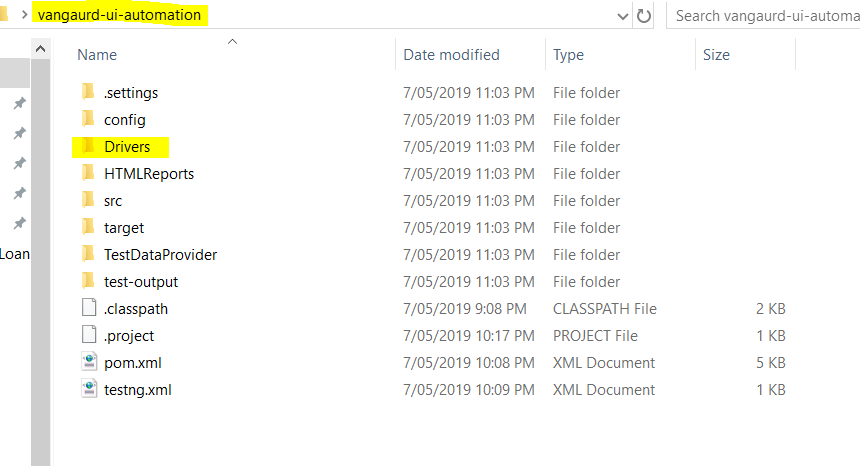
**For Execution:** TestNG class with test cases is used to execute and generate the results via TestNG runner or Command line execution or Maven or ANT or Jenkins ..etc. Rich HTML reports can be generated via our custom report storage file (XML/JSON which are generated while execution) or third-party libraries like Extent reports can serve the purpose.

**Justification :**

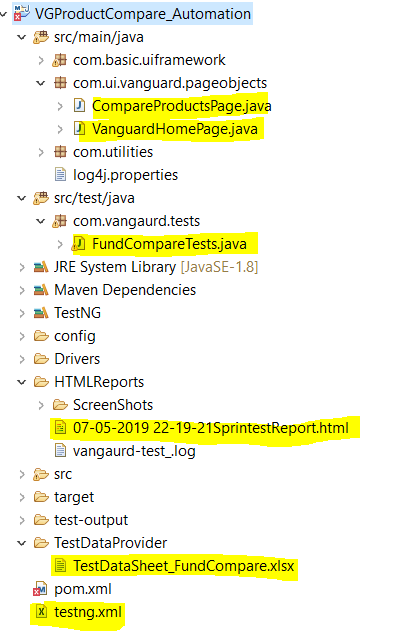
* This approach can be used for eliminating code duplications. If the test steps are repeated they can be created in Component class and reused across the application.
* Object Repository – All Xpath, Id, Names, cssselectors can be stored with name in Page Object classes
* Reports are created automatically – No code write is required from the tester to generate framework will handle.
* Test data organization is simple. Excel can be used to maintain test data.
* Execution is a cake walk. (can be execute from Maven life cycle, Jenkins jobs, direct from Eclipse, or From command prompt ..etc has many options to execute the test scenario’s)
* Can be easily integrated with CI/CD tools
* Can be integrate with other frameworks like Galen, Sikuli, Robot.etc.
* No limitation for number of browsers execution
* Grid support is awesome
* Defects can be directly logged to Defect management tool by extending utilities classes

**Setup and Usage Process:**

* Download project from Github repo
* Extract .zip folder and project structure should look like below;



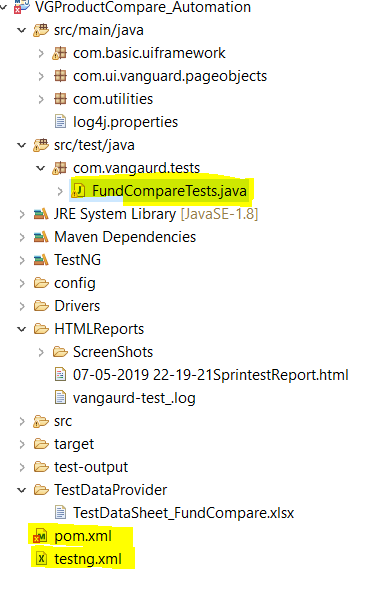
* Download **chromedriver.exe** and place in **Drivers** folder as highlighted in the above screenshot
* After importing the project to eclipse IDE/Intelli J it looks like below;



All the elements are recorded in **com.ui.vanguard.pageobjects**  with classes **CompareProductsPage & VanguardHomePage**

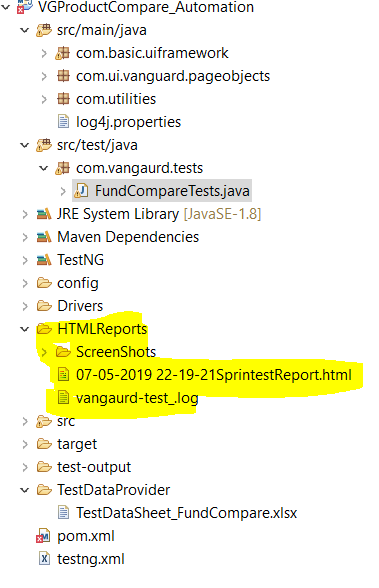
**Test cases** are written in **com.vangaurd.tests** with TestNG test class **FundCompareTests**

**Test Execution** can be done from anyone of the below highlighted;



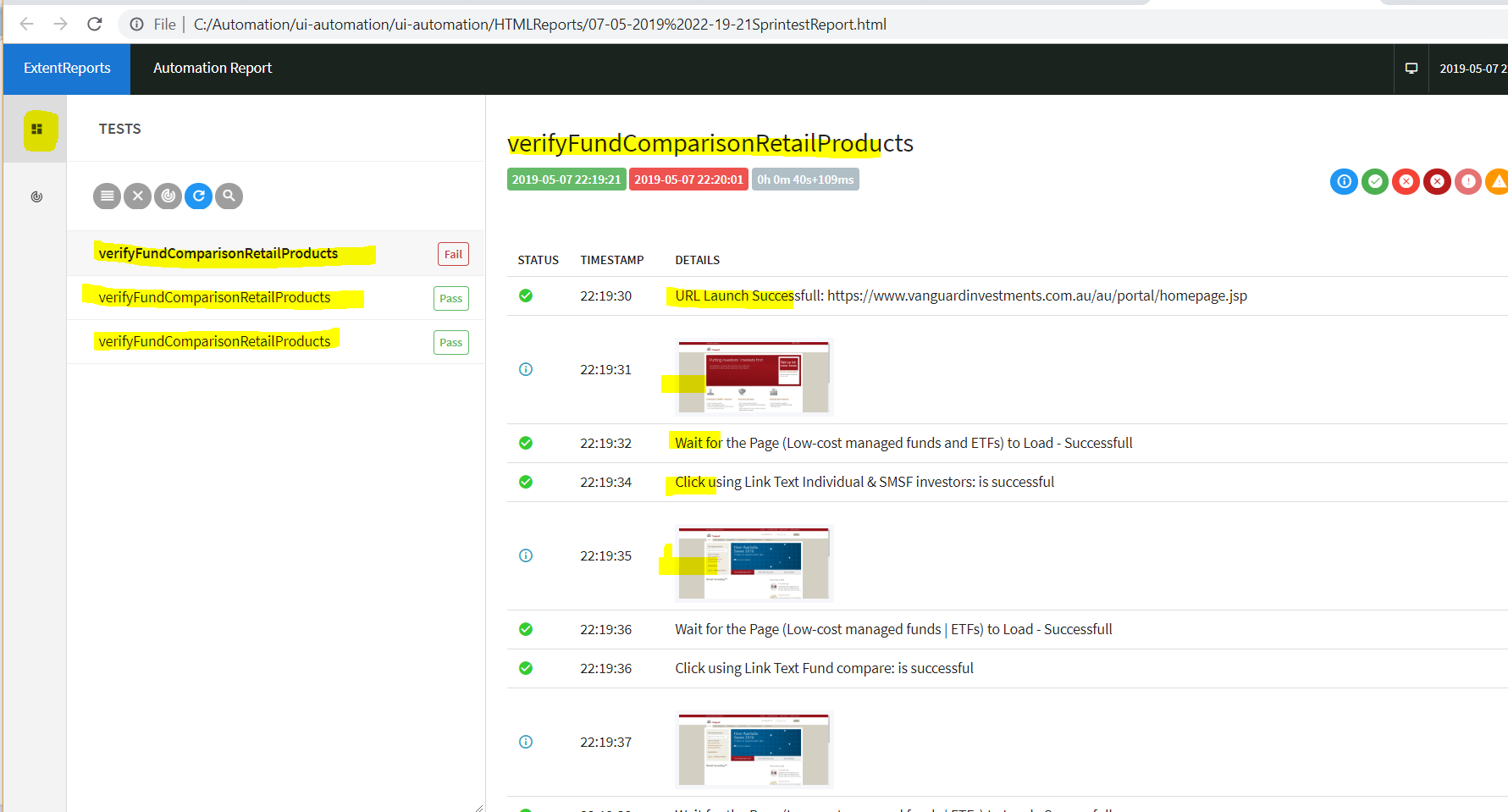
From **FundCompareTests.java** class or **pom.xml** or **testNG.xml**

Results are generated in **HTMLReports** folder as shown below;



**HTML File, log files** are generated as part of result generation.

NOTE: If time permits can be able to create Excel, XML and other formats as desired.



Above is the sample of HTML reports. Screenshots are taken as part of report generation for every test step in every test case with status of PASS,FAIL, ERROR,SKIP..etc.

There are 3 test cases designed as part of this assignment.

1. Failed Scenario (Assertion failure for value compared with Product Table)
2. Valid Scenario compared with Product table
3. Valid Scenario compared with Product table

Below is snapshot of excel sheet used for assertions.

Since all the values are table driven, all the column are dynamically pulled and created a ArrayList to store values for

**Entity**

**Product1**

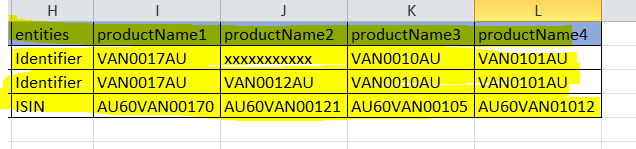
**Product2**

**Product3**

**Product4**

These ArrayList contains all the values w.r.t columns displayed in the comparison table.

In Excel Sheet we are validating the data with single values as shown below;



NOTE:IF time permits other assertion models can be implemented which includes more product data comparison