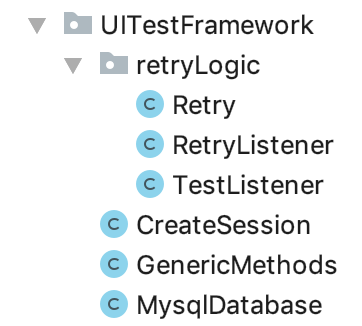
For any help/queries on this framework. Please feel free to drop an email @ [shankybnl@gmail.com](mailto:shankybnl@gmail.com) If you find it helpful, **share** with your network and **star** the repository :) Cheers!

**Mobile automation testing framework (Android and iOS) - supports both cucumber and testng tests**

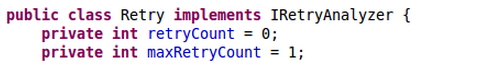
*Single code base framework to test android and iOS app using appium. It is a boilerplate code. Clone it and you are good to go!*

**Framework with testng tests setup and execution (**[**cucumber BDD tests here**](https://github.com/shankybnl/MobileAutomationFramework/blob/master/README.md#cucumber-bdd-tests)**)**

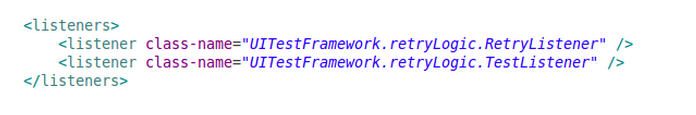
**Package : UITestFramework** : It includes the common classes (and methods) which are required by each test to perform actions. Below are classes in this package:

[](https://github.com/shankybnl/MobileAutomationFramework/blob/master/UIAutomation/images/1.png)

**retryLogic** : It has classes to implement retry in case of failure of a test. Retry count is set to 1 as of now. Test will be run once if it fails during the execution.

[](https://github.com/shankybnl/MobileAutomationFramework/blob/master/UIAutomation/images/2.png)

Add below listener to testng.xml file to include retry functionality.

[](https://github.com/shankybnl/MobileAutomationFramework/blob/master/UIAutomation/images/3.png)

**CreateSession.java** : All the methods to create a new session and destroy the session after the test(s) execution is over. Each test extends this class. Below are the methods in CreateSession class in their execution order.

1. invokeAppium() - method starts Appium server. Calls startAppiumServer method to start the session depending upon your OS.
2. createDriver(String os, Method methodName) - method creates the driver depending upon the passed parameter (android or iOS) and loads the properties files (config and test data properties files).
3. Tests execution (tests are present in IntegrationTests.tests package)
4. teardown() - method quit the driver after the execution of test(s)
5. stopAppium() - method to stop Appium server. Calls stopAppiumServer() method to stop session depending upon your OS.

**GenericMethods.java** : It is a common repository for all the webdriver and appium methods which are called in each coreLogic classes. Every new method which is being used in coreLogic classes should be added in this class as well. It is to reduce the duplicate code. Each screen class extends this class. Below are few methods defined in this class:

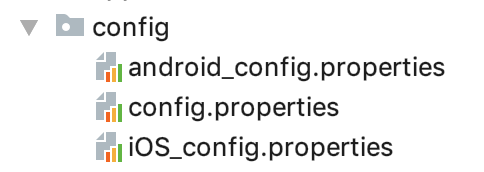
waitForVisibility(By targetElement) - method to wait for an element to be visible findElement(By locator) - method to find an element findElements(By locator) - method to find all the elements of specific locator

**MysqlDatabase.java** : This can be used if any DB values need to be verifiedIt has method to read DB and get data from required table. For more help, read on this link: <http://www.vogella.com/tutorials/MySQLJava/article.html>

**Package : app** : It contains the app build against which tests would be executed.

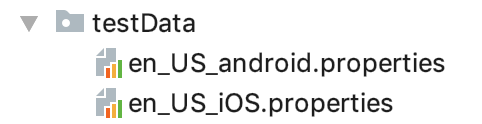
**Package : config** - It contains three files config.properties, android\_config.properties and iOS\_config.properties.

**config.properties** : Path to android and iOS config files are defined in this file. Other common required values like DB connection information etc. could be written in this file.

[](https://github.com/shankybnl/MobileAutomationFramework/blob/master/UIAutomation/images/4.png)

1. android\_config.properties - contains the location of the file where test data required by android tests is defined
2. iOS\_config.properties - contains the location of the file where test data required by iOS tests is defined

**Package: testData** : This package contains files having android and iOS test data. It contains two files: en\_US\_android.properties and en\_US\_iOS.properties.

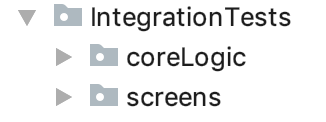
[](https://github.com/shankybnl/MobileAutomationFramework/blob/master/UIAutomation/images/5.png)

1. en\_US\_android.properties - All test data required by android tests is defined in this file.
2. en\_US\_iOS.properties - All test data required by iOS tests is defined in this file.

**Package: logger** : It contains Log.java class which contains methods to show the logs on console and save the logs in LogFile.txt of each run.

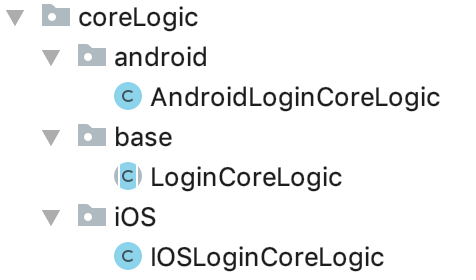
[](https://github.com/shankybnl/MobileAutomationFramework/blob/master/UIAutomation/images/6.png)

**Package: IntegrationsTests** : This package has sub-packages: screens, coreLogic, tests.

[](https://github.com/shankybnl/MobileAutomationFramework/blob/master/UIAutomation/images/7.png)

**Package: IntegrationTests.screens** : Classes in this package contains locators which are being used in coreLogic classes. Each page in mobile application is mapped to screen. E.g. for android login page, its AndroidLoginScreen. Segregated the locators on the bases of platform: android or iOS

**Package: IntegrationTests.screens.android** : Each screen on andriod app will be having as screen class under this package. It contains all the locators which are visible on that screen. E.g. - AndroidLoginScreen. Each android screen class extends GenericMethods.java.

[](https://github.com/shankybnl/MobileAutomationFramework/blob/master/UIAutomation/images/8.png)

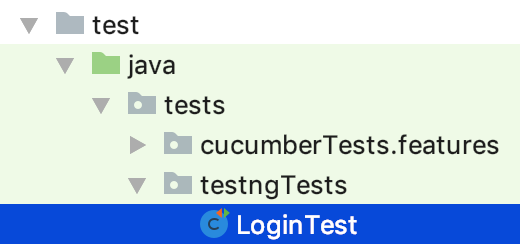
**Package: IntegrationTests.screens.ios** : Each screen on ios app will be having as screen class under this package. It contains all the locators which are visible on that screen. E.g. - IOSLoginScreen etc. Each iOS screen class extends GenericMethods.java.

**Package: IntegrationTests.coreLogic** : Classes in this package contains methods which performs intended actions and validations required by a test. Divided the coreLogic package depending on the platform : android, ios and base

**Package: IntegrationTests.coreLogic.base** : For each screen there would be corresponding coreLogic class. Classes under this package contains abstract methods which are defined in their respective classes in coreLogic.android and coreLogic.ios package. Eg: LoginCoreLogic.

**Package: IntegrationTests.coreLogic.android** : For each base coreLogic there would be corresponding android coreLogic (e.g. - AndroidLoginCoreLogic) where abstract method declared in base class are defined. Corresponding base class, coreLogic will be extended by android coreLogic class. E.g. for LoginCoreLogic base class, AndroidLoginCoreLogic will extend LoginCoreLogic.

**Package: IntegrationTests.coreLogic.ios** : For each base coreLogic class there would be corresponding ios coreLogic class (e.g. - IOSLoginCoreLogic) where abstract method declared in base class are defined.Corresponding base coreLogic class will be extended by ios coreLogic class. E.g. for LoginCoreLogic base class, IOSLoginCoreLogic will extend LoginCoreLogic.

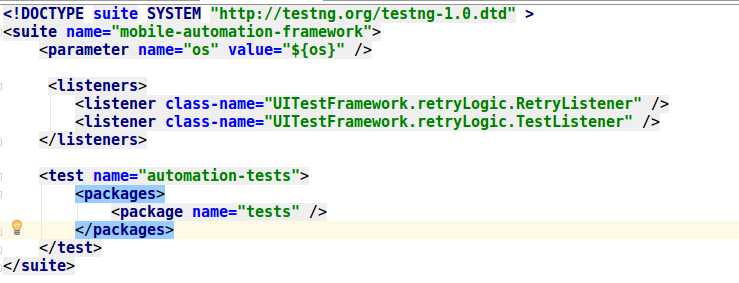
[](https://github.com/shankybnl/MobileAutomationFramework/blob/master/UIAutomation/images/9.png)

**Package: tests.testngTests** : This package contains all the tests. In each test there is instantiateHelpers(String invokeDriver) method which creates the object at runtime of the coreLogic classes required in the test. Object creation happens depending on the platform passed through invokeDriver parameter (android or ios). Then test calls methods defined in the coreLogic (of which object is created).

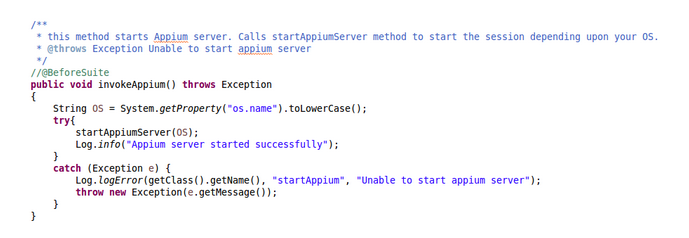
**Javadoc of the project can be found in doc folder. It contains information all classes and methods.**

**Execution flow of test test**

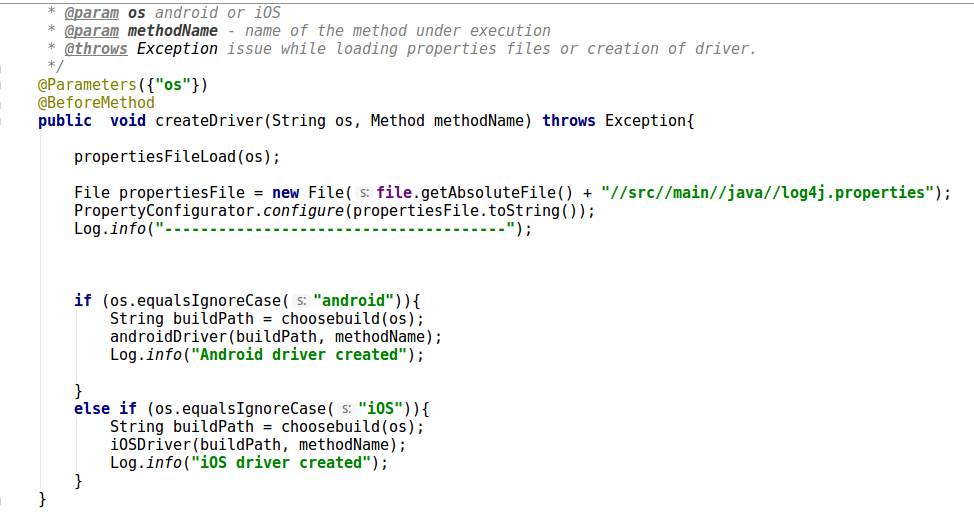
Below is execution flow with help of login test as example.

[](https://github.com/shankybnl/MobileAutomationFramework/blob/master/UIAutomation/images/10.png)

When testng\_android.xml file runs, it executes invokeAppium() method (part of CreateSession class in UITestFramework package) present under @BeforeSuite annotation. Right now, it is commented out. You can make changes in these method as per your platform to start appium server programmatically. As of now before running your test, start appium server on your machine

[](https://github.com/shankybnl/MobileAutomationFramework/blob/master/UIAutomation/images/11.png)

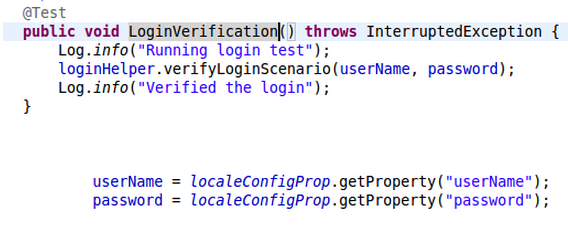
After this @BeforeMethod annotation is called, createDriver() method is present under this which loads log4.properties file (used by Log.java for logging purpose). It also load all the required property/config files having test data . First config.properties is loaded and then corresponding to the platform android or iOS, test data fies are loaded. (For eg: for android, android\_config.properties and en\_US\_android.properties files are loaded ) And creates android or iOS driver instance (depending on the platform name passed).

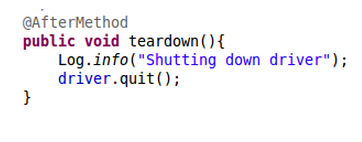
[](https://github.com/shankybnl/MobileAutomationFramework/blob/master/UIAutomation/images/12.png)

Next comes “instantiateHelpers” group. It would be present in every test under @BeforeMethod annotation. It creates object of all the coreLogic classes (depends on platform ios/android) which is required by test to run that test. Object is used to access all the methods present in coreLogic classes. In this case, we are creating object of AndroidLoginCoreLogic class which has methods to login.

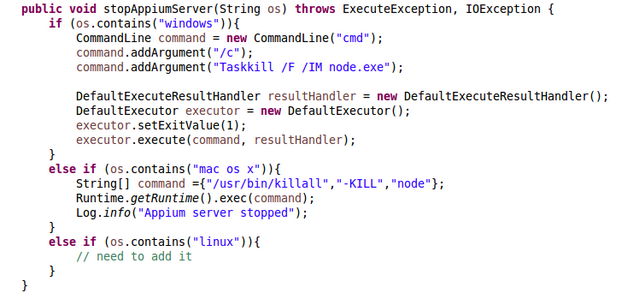
[](https://github.com/shankybnl/MobileAutomationFramework/blob/master/UIAutomation/images/13.png)

Next @Test Method is executed. Here, we are calling the methods which is required to verify login flow present in AndroidLoginCoreLogic class using LoginCoreLogic object. And loading required test data from en\_US\_android\_config.properties/en\_US\_iOS\_config.properties.

[](https://github.com/shankybnl/MobileAutomationFramework/blob/master/UIAutomation/images/14.png)

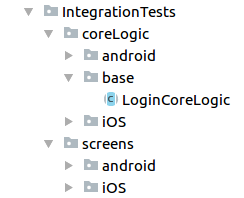
[](https://github.com/shankybnl/MobileAutomationFramework/blob/master/UIAutomation/images/15.png)

Once tests execution is completed. @AfterMethod annotation (present in CreateSession class) is executed to quit the driver.

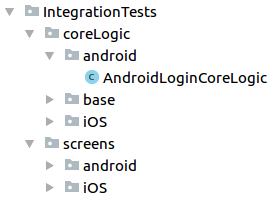
[](https://github.com/shankybnl/MobileAutomationFramework/blob/master/UIAutomation/images/16.png)

@AfterSuite will be called if it is not commented. It is to stop appium server.

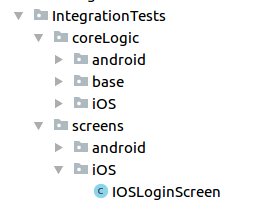
Corresponding coreLogic classes and screens files should be added in same hierarchy. Example for LoginTest. LoginCoreLogic would be under base->LoginCoreLogic.

[](https://github.com/shankybnl/MobileAutomationFramework/blob/master/UIAutomation/images/17.png)

AndroidLoginCoreLogic would be under android->AndroidLoginCoreLogic.

[](https://github.com/shankybnl/MobileAutomationFramework/blob/master/UIAutomation/images/18.png)

IOSLoginCoreLogic would be under iOS -> IOSLoginCoreLogic.

[](https://github.com/shankybnl/MobileAutomationFramework/blob/master/UIAutomation/images/19.png)

Similar thing should be done for element locators. There is no base folder in case of screens.For android locators, it would be under screens->android->AndroidLoginScreen. For iOS locators, it would be under screens->ios->IOSLoginScreen.

**How to execute a test**

Maven is used as build tool (can be downloaded from [here](https://maven.apache.org/download.cgi)). pom.xml file is present in base directory which has all the required dependencies and code to invoke testng.xml file when executed from command line.

Connect your device to your machine or start the emulator.

*Note: start appium server on your machine if not included programatically*

**Run below commands to execute android testng test:**

$ cd mobileautomationframework/ $ mvn test -Dos=android -Dsurefire.suiteXmlFiles=testng.xml

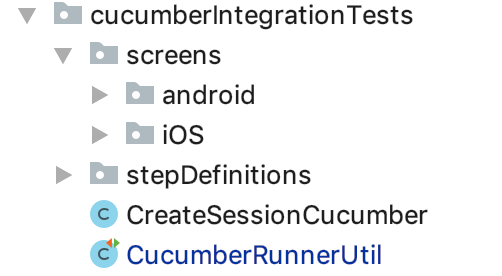
*Include iOS app on which you want to run test. Provide its path in config.xml file (iOSAppPath=src/app/path-to-your-iOSfile). And write screen locators in IOSLoginScreen and methods in IOSLoginCorelogic. Now you are ready to run below commands.*

**Run below commands to execute iOS testng test:**

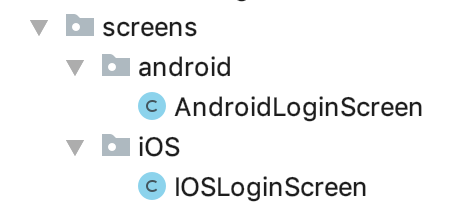
$ cd mobileautomationframework/ $ mvn test -Dos=iOS -Dsurefire.suiteXmlFiles=testng.xml

xxxxxxxxxxxxxxxxxxxxxxxxxxxx

**Cucumber BDD tests**

[](https://github.com/shankybnl/MobileAutomationFramework/blob/master/UIAutomation/images/20.png)

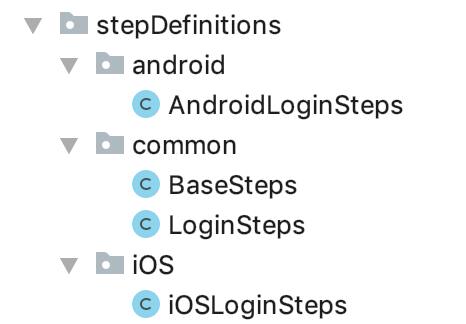
**Package : cucumberIntegrationTests** :This package has sub-packages: screens and stepDefinitions. It also includes CucumberRunnerUtil class and CreateSessionCucumber class which are required to keep configurations and creating appium instance respectively.

[](https://github.com/shankybnl/MobileAutomationFramework/blob/master/UIAutomation/images/21.png)

**Package: IntegrationTests.screens** : Classes in this package contains locators which are being used in coreLogic classes. Each page in mobile application is mapped to screen. E.g. for android login page, its AndroidLoginScreen. Segregated the locators on the bases of platform: android or iOS

**Package: IntegrationTests.screens.android** : Each screen on andriod app will be having as screen class under this package. It contains all the locators which are visible on that screen. E.g. - AndroidLoginScreen. Each android screen class extends GenericMethods.java.

**Package: IntegrationTests.screens.ios** : Each screen on ios app will be having as screen class under this package. It contains all the locators which are visible on that screen. E.g. - IOSLoginScreen etc. Each iOS screen class extends GenericMethods.java.

[](https://github.com/shankybnl/MobileAutomationFramework/blob/master/UIAutomation/images/22.png)

**Package: stepDefinitions.common** : This package has all classes with common methods which can be used across android and iOS platform. It also has BaseSteps class which has initiation and appium driver creation for android or iOS platform.

**Package: stepDefinitions.android** : It should contain all the stepDefinitions for android features. e.g. AndroidLoginSteps

**Package: stepDefinitions.iOS** : It should contain all the stepDefinitions for iOS features. e.g. iOSLoginSteps

**CucumberRunnerUtil.java**: cucumberTestng.xml invokes this class. It has all the configuration for cucumber to execute tests and testng annotations to leverage their benefits.

[](https://github.com/shankybnl/MobileAutomationFramework/blob/master/UIAutomation/images/23.png)

**CreateSessionCucumber.java** : Similar to [Create session class](https://github.com/shankybnl/MobileAutomationFramework/blob/master/**CreateSession.java) to create driver object and loading test data etc.

**How to execute a test**

Maven is used as build tool (can be downloaded from [here](https://maven.apache.org/download.cgi)). pom.xml file is present in base directory which has all the required dependencies and code to invoke testng.xml file when executed from command line.

Connect your device to your machine or start the emulator.

*Note: start appium server on your machine if not included programatically*

**Run below commands to execute android testng test:**

$ cd mobileautomationframework/ $ mvn test -Dos=android -Dsurefire.suiteXmlFiles=cucumberTestng.xml

*Include iOS app on which you want to run test. Provide its path in config.xml file (iOSAppPath=src/app/path-to-your-iOSfile). And write screen locators in IOSLoginScreen and methods in IOSLoginCorelogic. Now you are ready to run below commands.*

**Run below commands to execute iOS testng test:**

$ cd mobileautomationframework/ $ mvn test -Dos=iOS -Dsurefire.suiteXmlFiles=cucumberTestng.xml