Abbreviations

[**APPIUM (BDD Framework)** 2](#_Toc118290827)

[Cucumber : 2](#_Toc118290828)

[Given – When – Then Approach 2](#_Toc118290829)

[Sample Feature File: 2](#_Toc118290830)

[**Prerequisites to execute automation testing on mobile application: (Andriod and iOS devices)** 3](#_Toc118290831)

[Android: 3](#_Toc118290832)

[iOS : 3](#_Toc118290833)

[**Appium Integration with TestNG in eclipse** 4](#_Toc118290834)

[**Appium Project Folder Structure** 5](#_Toc118290835)

[**BDD framework integration with AWS Device farm** 11](#_Toc118290836)

[**BDD framework integration with CI/CD** 12](#_Toc118290837)



# **APPIUM (BDD Framework)**

Appium is an open-source test automation framework for use with native, hybrid and mobile web apps.

BDD framework i.e., Behaviour Driven Development is a software development approach that allows the tester/business analyst to create test cases in simple text language (English) and BDD allows the users to work with multiple test data with minimum intervention in the software code and thereby helps to increase the reusability of the code, which is a time-saving mechanism in software development/ test automation.

# **Cucumber :**

Cucumber is a testing approach which supports Behaviour Driven Development (BDD). It explains the behaviour of the application in a simple English text using Gherkin language.

## Given – When – Then Approach

Given: Some given context (Preconditions).

When: Some Action is performed (Actions).

Then: Particular outcome/consequence after the above step (Results).

## Sample Feature File:

**Feature:** BDD implementation using Cucumber

Scenario: Check login functionality with valid credentials

Given user is on the login page

When user enters credentials

When user clicks on Login

Then user lands on homepage

# 

# **Prerequisites to execute automation testing on mobile application: (Android and iOS devices)**

## Android:

* Operating system - Windows
* Java SE Development Kit (JDK) 1.8 or greater.
* The latest Eclipse IDE for Java Developers.
* Appium Server
* Android Studio

## iOS:

* Operating system - iOS
* Java SE Development Kit (JDK) 1.8 or greater.
* The latest Eclipse IDE for Java Developers.
* Homebrew
* Nodejs and Npm
* Appium Server
* XCode
* XCode command line Tools
* Apple ID

# **Appium Integration with TestNG in eclipse**

1. Need to install Appium in local system either through Appium desktop or command prompt.
2. Launch Appium server with the Local IP address and Port number as below.

Graphical user interface, application

Description automatically generated

1. Launch Appium driver in eclipse with same details using the below code

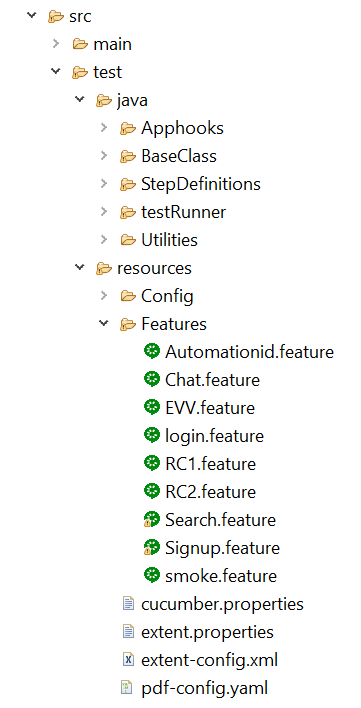


URL url = new URL("http://0.0.0.0:4723/wd/hub");

driver = new Appium Driver<MobileElement>(URL, capabilities);

# **Appium Project Folder Structure**

**#1) src –** The folder contains all the test scripts, generics, readers and utilities. All these resources are nothing but the simple java classes. Under the source (src) folder, we have created a hierarchy of folders.



**a) test**– The “test” folder is constituted of majorly two ingredients – test suite and the folders representing the various modules of the application under test. Thus, each of these folders contains the test scripts specific to the module to which it is associated. Test suite is a logical combination of more than one test scripts. Thus, the user can mark an entry of any of the test scripts within the test suite that he/she desires to execute in the subsequent runs.

Graphical user interface

Description automatically generated with low confidence

**b) utilities** – The “utilities” folder is constituted of various generics, constants, Readers and classes for implementing user-defined exceptions. Each of the folders under utilities has got its own significance.

The folder is constituted of the classes which contain functions and methods that can be shared and used amongst the multiple classes. Thus, it is always recommended to create a separate class for such activities instead of coding them repeatedly in each of the test scripts.

* + **Preconditional Methods**
  + **Post Conditional Methods**

Graphical user interface, text, application, chat or text message

Description automatically generated

**Common Methods**

Like Pre and postconditions, there may be methods and functions those can be used by more than one test script. Thus, these methods are grouped together in a class. The test script can access these methods using the object of the common class.

**#2) Test data from JSON files**

We can store test data for your application in a JSON file and have a single test script which can run tests for all the test data in the file. For example, if you want to test the account registration functionality of your web application, you can save multiple sets of credentials in a JSON file and retrieve the data from the file to your script for execution.

**Test Data Format in JSON**

Graphical user interface, text, application

Description automatically generated

Mark that the test data formats are solely user defined. Thus, based on your requirements, you can customize the test data files.

**#3) library –** The folder acts as a repository/artifactory for all the required jar files, libraries, drivers etc to successfully build the test environment and to execute the test scripts. Refer the following figure to check out the libraries we would be employed within our project.

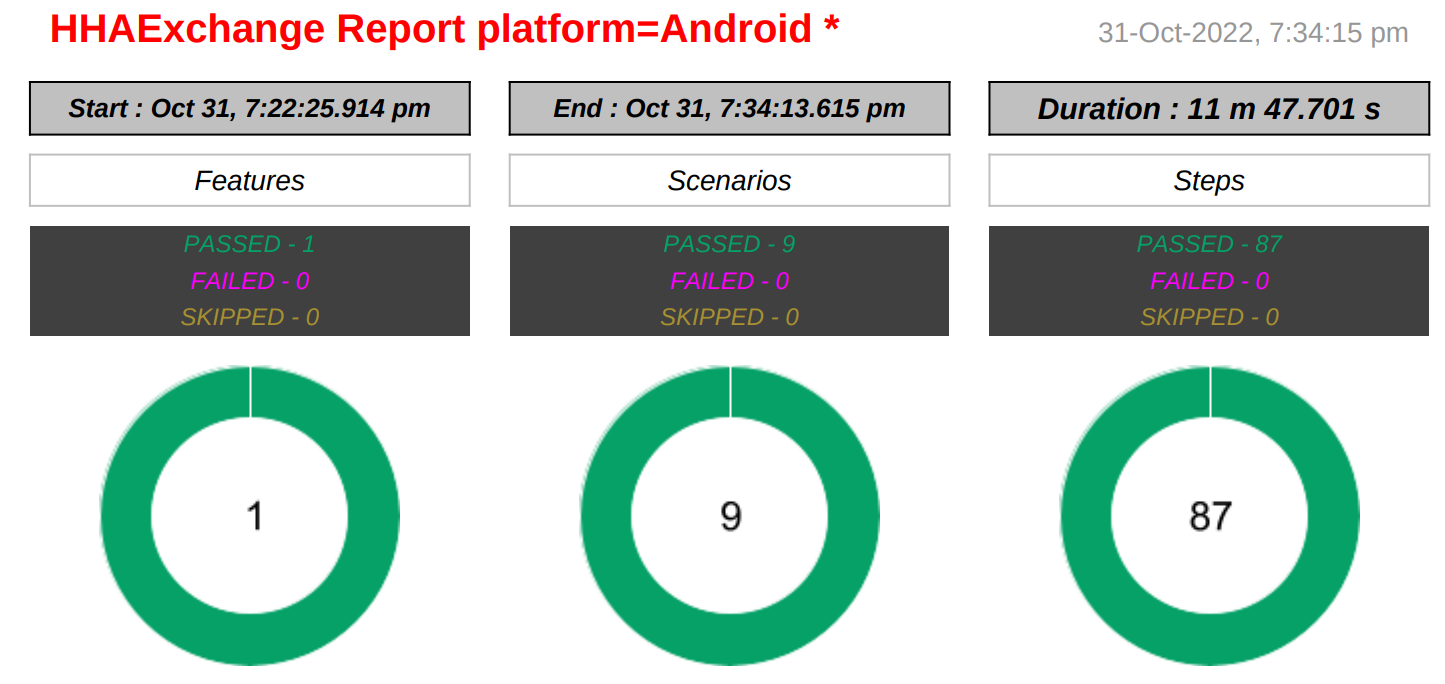
Graphical user interface, text

Description automatically generated

**#4) Reports –** The Reports folder contains a cucumber extent-report folder along with the timestamp. In that folder we will get a pdf generated file that stores the report upon each execution.

Graphical user interface, text, application, chat or text message

Description automatically generated



**#5) test material –** The folder contains the actual test data that needs to be uploaded if any. This folder would come into picture when we come across test scenarios where the user is required to upload files, documents, pictures, reports etc.

Graphical user interface, application

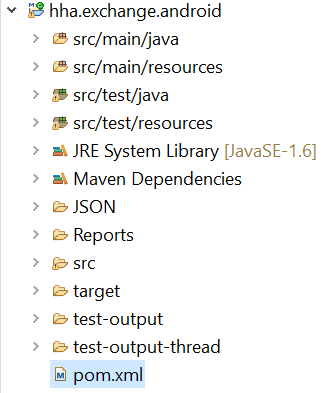
Description automatically generated

**#6) pom.xml –** The first step in establishing our Java project is the creation of a Maven Project Object Model (POM) file. This is an XML document that defines how our code will be built, what additional dependencies it has access to, and how tests are run.

The POM file includes:

* Project Element- At the top level is the <project> element.
* Group, Artifact and Version - The <groupId>, <artifactId> and <version> elements define the identity of the Maven artifact produced by this project.
* Packaging - The <packaging> element defines the type of artifact that Maven will produce.
* Properties - The <properties> element defines values for properties that are either recognized directly by Maven or that can be referenced later on in the POM file as a way of sharing common values.
* Plugins - Each phase is executed by a plugin, and each plugin can be configured within a <plugin> element, nested under the <build><plugins> elements.
* Dependencies - Finally we have the <dependencies> element. This element defines the additional libraries that our application relies on.

The details of the dependencies are usually found through one of the many Maven repository search engines, such as <https://search.maven.org/>.



**#7) testng.xml –** Testng.xml file is a configuration file that helps in organizing our tests. It allows testers to create and handle multiple test classes, define test suites and tests.

Graphical user interface, text, application

Description automatically generated

# **BDD framework integration with AWS Device farm**

Device Farm is an app testing service that you can use to test and interact with your Android, iOS, and web apps on real, physical phones and tablets that are hosted by Amazon Web Services (AWS).

AWS device farm Support for various frameworks and programming languages depends on the language used and also Device Farm supports all Appium server versions 1.6.5 and above.

Configuration (JAVA+TestNG):

<https://docs.aws.amazon.com/devicefarm/latest/developerguide/test-types-appium.html>

# **BDD framework integration with CI/CD(AWS Code build/Code pipeline)**

**BDD Framework**

Icon

Description automatically generated

Icon

Description automatically generated

**AWS Device Farms**



**Test Report**

Icon

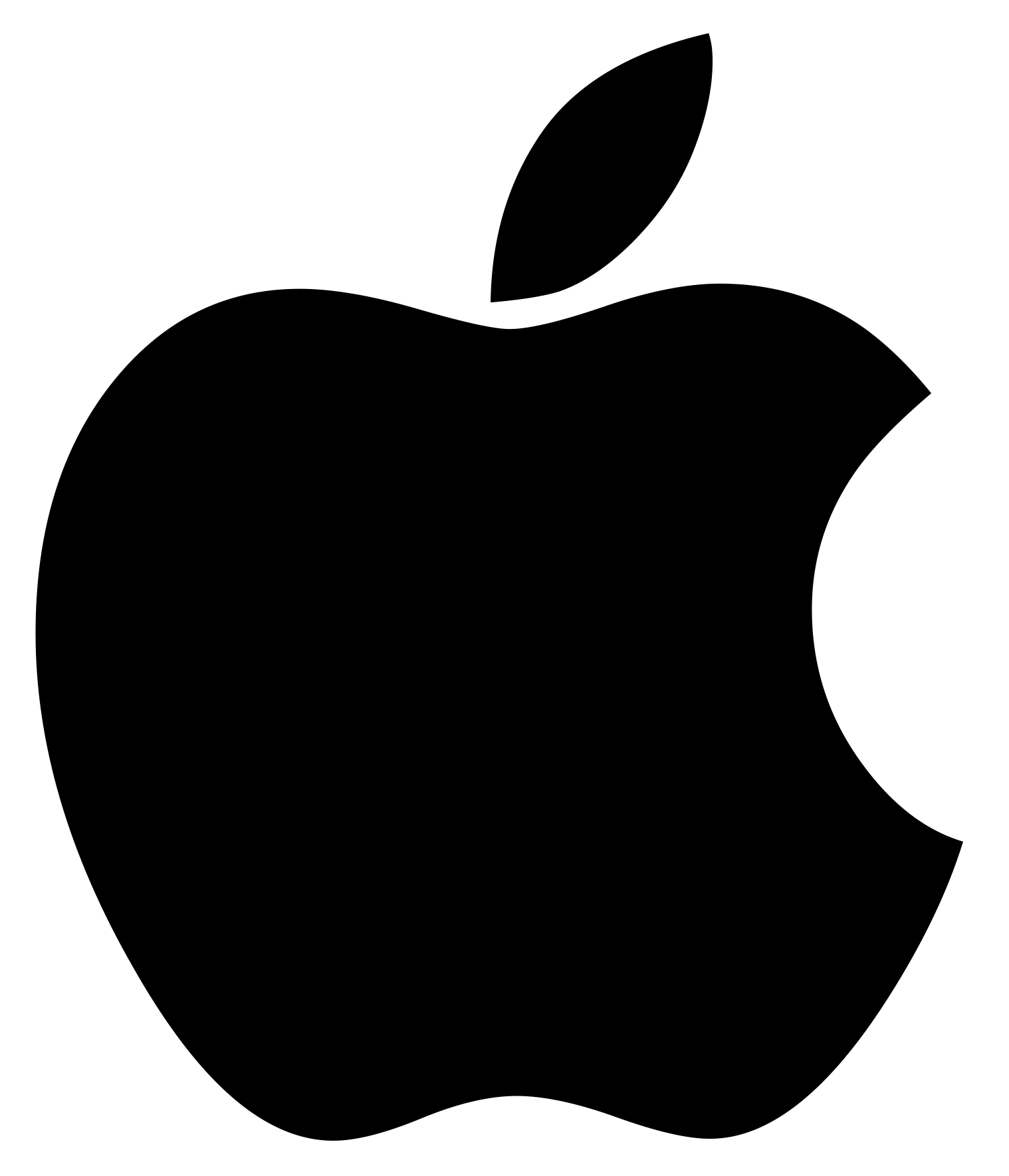
Description automatically generated

**.apk/.ipa**

Icon

Description automatically generated

**IOS Device**



**Android Device**



**Notification**

Icon

Description automatically generated