A REAL TIME AUTOMATION TESTING IN WEB AND MOBILE APPLICATION USING SELENIUM AND APPIUM SERVER

Submitted in partial fulfillment of the

requirement for the award of the degree of

Bachelor of Computer Science

By

Sai Aniruthan K(RA2031005020089)
Gopinath R(RA2031005020071)
Saran R (RA2031005020090)

Under the guidance of

Mr. U. Udayakumar

Assistant professor

DEPARTMENT OF COMPUTER SCIENCE & APPLICATIONS -BSc (CS)
FACULTY OF SCIENCE AND HUMANITIES
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
Ramapuram, Chennai.
April 2023

SRM Institute of Science and Technology

Ramapuram, Campus.

Faculty of Science and Humanities

Department of Computer Science and Applications (B. Sc-CS)

BONAFIDE CERTIFICATE

Certified that this project report titled **"A REAL TIME AUTOMATION TESTING IN WEB AND MOBILE APPLICATION USING SELENIUM AND APPIUM SERVER"**is the bonafide work of **Sai Aniruthan K(RA2031005020089), Gopinath R(RA2031005020071), Saran R (RA2031005020090)** who carried out the project work done under my supervision during the academic year 2022-2023-Semester VI.

Course Code: UCS20D10L

Course Name: Project Work

Certified further, that to the best of my knowledge the work reported here in does not form part of any other project report on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Signature of Internal Guide **Signature of Head of the Department**

Signature of External Examiner

ACKNOWLEDGEMENT

I extend my sincere gratitude to the Chancellor **Dr. T.R. PACHAMUTHU** and to Chairman **Dr. R. SHIVAKUMAR** of SRM Institute of Science and Technology, Ramapuram, Chennai for providing me the opportunity to pursue the BSc (CS) degree at this University.

I express my sincere gratitude to **Maj. Dr. M. VENKATRAMANAN,** Dean (**S&H)**, SRM IST, Ramapuram for his support and encouragement for the successful completion of the project.

I record my sincere thanks to **Dr. J. DHILIPAN, Head (MCA) & Vice Principal (Admin, S&H)**, SRM IST, Ramapuram for his continuous support and keen interest to make this project a successful one.

I record my sincere thanks to **Dr. V. SARAVANAN, Vice Principal (Academic, S&H) & Head (B.Sc(CS))**, SRM IST, Ramapuram for his encouragement and keen interest to make this project a successful one.

I find no word to express profound gratitude to my guide **Mr.U. Udayakumar** Assistant Professor, Department of Computer Science and Applications-BSc (CS), SRM IST, Ramapuram for his kind co-operation and encouragement which helped us for the completion of this project.

I thank the almighty who has made this possible. Finally, I thank my beloved family member and friend for their motivation, encouragement and cooperation in all aspect which led me to the completion of this project.

Sai Aniruthan (RA2031005020089)
Gopinath R (RA2031005020071)
Saran R(RA2031005020090)

ABSTRACT

Automation testing is an emerging field that draws maximum benefits with minimum efforts. Automation tools helps in design and execution of test scripts saving time and cost involved in manual testing. This project mainly focuses on Smartphones are used by billions of people all over the world and are loaded with multiple mobile applications. Every application needs to be tested before it goes to the market. But mobile application testing is quite challenging, and there are so many aspects to keep in mind. Even the smallest functionality issue can trigger negative reviews and get mobile application deleted by the user. The main idea of this project is to detect bugs in early stages of development before it goes to market. To overcome the challenges of mobile application testing, we have designed a real time automation testing tool that uses the trending technologies such as eclipse, Appium, AndroidSDK, selenium and Java.

SNO	TITLE	PAGE NO
	ACKNOWLEDGEMENT	I
	ABSTRACT	II
	LIST OF FIGURES	III
	LIST OF ABBREVIATION	IV
CHAPTERS	**TITLE**	**PAGE NO**
1	**INTRODUCTION**	
	1.1 PROJECT INTRODUCTION	10
2	**WORKING ENVIRONMENT**	
	2.1 HARDWARE REQUIREMENTS	12
	2.2 SOFTWARE REQUIREMENTS	12
3	**SYSTEM ANALYSIS**	
	3.1 EXISTING SYSTEM	14
	3.2 DRAWBACKS OF THE EXISTING SYSTEM	14
	3.3 PROPOSED SYSTEM	15
	3.4 ADVANTAGES OF THE PROPOSED SYSTEM	15
	3.5 SELENIUM	16
	3.6 TEST AUTOMATION	17
	3.7 APPIUM	18
	3.8 UI AUTOMATOR	20

4	**SYSTEM DESIGN** 4.1 ARCHITECTURE DIAGRAMS 4.2 USE CASE DIAGRAM 4.3 ACTIVITY DIAGRAM 4.4 DATA FLOW DIAGRAM	 23 25 26 27
5	**PROJECT DESCRIPTION** 5.1 MODULES 5.2 MODULES DESCRIPTION	 29 30
6	**CODING AND TESTING** 6.1 MAIN CLASS 6.2 WEB APP AUTOMATION TESTING 6.3 MOBILE APP TESTING 6.4 REPORT AND EMAIL	 62 67 68 69
7	**CONCLUSION** 7.1 FUTURE ENHANCEMENT	 74
8	**APPENDIX** 8.1 SCREENSHOTS	 76
9	**BIBLIOGRAPHY AND REFERENCE**	77

LIST OF FIGURES

4.1 Architecture Diagram

4.2 Use Case Diagram

4.3 Activity Diagram

4.4 Data Flow Diagram

LIST OF ABBREVIATION

1. **JDK-** Java development kit.
2. **UI Automator-** User interface Automator.
3. **Android SDK-** Software development kit.
4. **API-** Application programming interface.
5. **GUI-** Graphical user interface.

CHAPTER 1

1.INTRODUCTION

PROJECT INTRODUCTION

Selenium is a set of different software tools each with a different approach to supporting test automation. Most Selenium QA Engineers focus on the one or two tools that most meet the needs of their project, however learning all the tools will give you many different options for approaching different test automation problems. The entire suite of tools results in a rich set of testing functions specifically geared to the needs of testing of web applications of all types. These operations are highly flexible, allowing many options for locating UI elements and comparing expected test results against actual application behavior. One of Selenium's key features is the support for executing one's tests on multiple browser platforms. Selenium is an open-source tool widely used for automating web browsers. It allows you to simulate user interaction with web pages and automate testing of web applications. Selenium is compatible with multiple programming languages, including Java, Python, C#, and JavaScript. This makes it easy for developers to integrate Selenium with their existing testing frameworks and tools. One of the key features of Selenium is its ability to run automated tests across multiple browsers and operating systems. This helps to ensure that web applications are consistent and functional across different platforms. Selenium is also highly customizable and extensible. It supports a wide range of plugins and extensions, which can be used to add additional functionality to your automated testing process. Overall Selenium is a powerful tool for automating web testing and is widely used in the software industry to ensure the quality and reliability of web application Selenium is an open-source tool widely used for automating web browsers. It allows you to simulate user interaction with web pages and automate testing of web applications.

CHAPTER 2

2. WORKING ENVIRONMENT

SYSTEM REQUIREMENTS

- Install Java JDK version 7 and above
- Install Android SDK version 22 and above its get inbuilt with android studio.
- Setup Appium command server setup/ download and install appium GUI exe latest
- Install Google USB Driver latest
- Install IDE -eclispse/Intelligent/android studio
- Set the environment variables for java jdk and android SDK.

2.1 HARDWARE REQUIREMENTS

- Main Processor : 2GHz
- RAM : 512 MB (min)
- Hard Disk : 80 GB
- Hardware Tools : Mobile, USB Cable.

2.2 SOFTWARE REQUIREMENTS

- Language : Java
- Software : Eclipse, Selenium, Appium, UI Automator
- Operating System : Windows 7 32bit

CHAPTER 3

3.SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

- In Existing System most of the legacy applications have zero or irrelevant and out of date documentation which leads to the inefficiency and high maintenance costs.
- Testing legacy applications involves a lot of complexities when they undergo enhancements or upgrades including reduced capability of testing all the functionalities.
- Lack of information regarding the improvements in business process.
- Lack of information regarding the upgrades or customizations in legacy systems that impact other existing functionalities

3.2 DRAWBACKS OF THE EXISTING SYSTEM

- The testing of those android that are lower than 4.2 is not allowed.
- Appium has limited support for hybrid app testing. You will not be able to test the action that allows switching of applications from native to web app and from web app to native.
- There is no support that will allow you to run Appium inspector on Microsoft windows.
- Doesn't support image comparison.
- Long time to configure appium for both android and iOS.

3.3 PROPOSED SYSTEM

Selenium is an automated, open-source test framework that can be employed to test mobile user interfaces that come with native, hybrid and mobile web applications. Even hybrid apps, in which a common code is used for both Android and iOS, are supported. These apps are created with frameworks like React Native, Flutter, and Ionic. Appium uses the UIAutomator framework , which is mean for testing the Android user interface to automate applications on Android device testing. The bootstrap.jar file works as a TCP server that sends the test command for acting on the Android device with the help of the UIAutomator

3.4 ADVANTAGES OF THE PROPOSED SYSTEM

- Allows running of parallel tests on multiple mobile devices simultaneously.
- Allows testing of apps or websites on cross-platform mobile devices (Android and iOS).
- Teams can broaden their device coverage by testing the latest iOS & Android devices with new OS's.
- The test automation tool for Native and Hybrid iOS is a cross-platform framework.
- This open-source tool supports multiple languages like Ruby, C#, Java, etc.

3.5 SELENIUM:

Selenium is one of the most widely used open-source Web UI (User Interface) automation testing suite. It was originally developed by Jason Huggins in 2004 as an internal tool at Thought Works. Selenium supports automation across different browsers, platforms and programming languages. This makes it easy for developers to integrate Selenium with their existing testing frameworks and tools. One of the key features of Selenium is its ability to run automated tests across multiple browsers and operating systems. This helps to ensure that web applications are consistent and functional across different platforms. Selenium is also highly customizable and extensible. It supports a wide range of plugins and extensions, which can be used to add additional functionality to your automated testing process. Overall Selenium is a powerful tool for automating web testing and is widely used in the software industry to ensure the quality and reliability of web applications. Many people get started with Selenium IDE. If you are not already experienced with a programming or scripting language you can use Selenium IDE to get familiar with Selenium commands. Using the IDE you can create simple tests quickly, sometimes within seconds. We don't, however, recommend you do all your test automation using Selenium IDE. To effectively use Selenium, you will need to build and run your tests using either Selenium 2 or Selenium 1 in conjunction with one of the supported programming languages. Which one you choose depends on you. At the time of writing the Selenium developers are planning on the Selenium-WebDriver API being the future direction for Selenium

3.6 TEST AUTOMATION:

First start by asking yourself whether or not you really need to use a browser. Odds are that, at some point, if you are working on a complex web application, you will need to open a browser and actually test it.

Functional end-user tests such as Selenium tests are expensive to run, however. Furthermore, they typically require substantial infrastructure to be in place to be run effectively. It is a good rule to always ask yourself if what you want to test can be done using more lightweight test approaches such as unit tests or with a lower-level approach.

Once you have made the determination that you are in the web browser testing business, and you have your Selenium environment ready to begin writing tests, you will generally perform some combination of three steps:

- Set up the data
- Perform a discrete set of actions
- Evaluate the results

Test automation has specific advantages for improving the long-term efficiency of a software team's testing processes test automation supports:

- Frequent regression testing
- Rapid feedback to developers
- Virtually unlimited iterations of test case execution
- Support for Agile and extreme development methodologies
- Disciplined documentation of test cases
- Customized defect reporting
- Finding defects missed by manual testing

3.7 APPIUM:

Appium is an open-source tool for automating native, mobile web, and hybrid applications on iOS mobile, Android mobile, and Windows desktop platforms. **Native apps** are those written using the iOS, Android, or Windows SDKs. **Mobile web apps** are web apps accessed using a mobile browser (Appium supports Safari on iOS and Chrome or the built-in 'Browser' app on Android). **Hybrid apps** have a wrapper around a "WebView" -- a native control that enables interaction with web content.

Importantly, Appium is "cross-platform": it allows you to write tests against multiple platforms (iOS, Android, Windows), using the same API. This enables code reuse between iOS, Android, and Windows test suites.

There are 3 kinds of mobile applications that we can automate using Appium:

Native apps are built for a particular device and operating system. Native apps are installed directly on a mobile device. They can be downloaded from a App Store or Google Play store and installed on the device.
A native app is written in the programming language specific for a platform.

Mobile web apps are web apps accessed using a mobile browser (mobile version' of a web site). They don't have to installed. Web apps are designed to look and behave like apps and purpose is simply to make content or functionality available on mobile

Hybrid apps have a wrapper around a "WebView" -- a native control that enables interaction with web content. A hybrid app is a web app that translates to native code on a platform like iPhone or Android. It uses a browser view and hooks to allow your web app to access features on your mobile device.

Appium Server:

Appium is a server written in Node.js. It can be built and installed from source or installed directly from NPM:

```
$ npm install -g appium

$ appium
```

Appium Clients:

There are client libraries (in Java, Ruby, Python, PHP, JavaScript, and C#) which support Appium's extensions to the WebDriver protocol. When using Appium, you want to use these client libraries instead of your regular WebDriver client.

Appium Desktop:

There is a GUI wrapper around the Appium server that can be downloaded for any platform. It comes bundled with everything required to run the Appium server, so you don't need to worry about Node. It also comes with an Inspector, which enables you to check out the hierarchy of your app. This can come in handy when writing tests.

3.8 UI AUTOMATOR:

UI Automator is a UI testing framework suitable for cross-app functional UI testing across system and installed apps. so, it allows you to perform operations such as opening the Settings menu or the app launcher in a test device. Your test can look up a UI component by using convenient descriptors such as the text displayed in that component or its content description.

The key features of the UI Automator testing framework include the following:

- An API to retrieve state information and perform operations on the target device. (Accessing device state)
- APIs that support cross-app UI testing. (UI Automator APIs).

Accessing device state:

The UI Automator testing framework provides a `UiDevice` class to access and perform operations on the device on which the target app is running. You can call its methods to access device properties such as current orientation or display size.
The `UiDevice` class also let you perform the following actions:

1. Change the device rotation.
2. Press a hardware key, such as "volume up".
3. Press the Back, Home, or Menu buttons.
4. Open the notification shade.
5. Take a screenshot of the current window.

UI Automator APIs:

The UI Automator APIs allow you to write robust tests without needing to know about the implementation details of the app that you are targeting. You can use these APIs to capture and manipulate UI components across multiple apps:

- UiCollection: Enumerates a container's UI elements for the purpose of counting, or targeting sub-elements by their visible text or content-description property.
- UiObject: Represents a UI element that is visible on the device.
- UiScrollable: Provides support for searching for items in a scrollable UI container.
- UiSelector: Represents a query for one or more target UI elements on a device.
- Configurator: Allows you to set key parameters for running UI Automator tests.

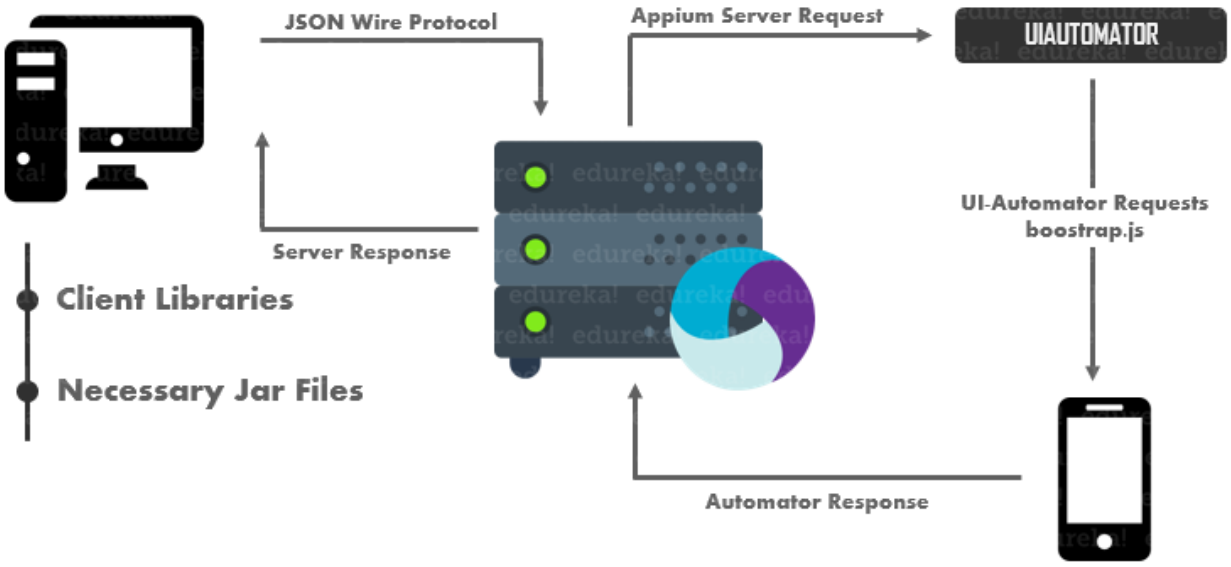
We need below list of items to be installed on your machine

1. JDK (Java Development Kit)
2. Andriod Studio
3. Appium
3. TestNG
4. Eclispe
5. And Download Selenium Jar

CHAPTER 4

4. SYSTEM DESIGN

4.1 ARCHITECTURE DIAGRAM:



Appium on Android uses the UIAutomator framework for automation. UIAutomator is a framework built by android for automation purposes. So, let's take a look at the exact way that Appium works on Android.

1. Appium client (c/Java/Python/etc) connects with Appium Server and communicates via JSON Wire Protocol.
2. Appium Server then creates an automation session for the client and also checks the desired capabilities of the client. It then connects with the respective vendor-provided frameworks like UIAutomator.
3. UIAutomator will then communicate with bootstrap.jar which is running in simulator/emulator/real device for performing client operations.
4. Here, **bootstrap.jar** plays the role of a TCP server, which we can use to send the test command in order to perform the action on the Android device using UIAutomator.

The above **Appium android architecture** diagram gives a visual representation by the below steps.

Client/Server Architecture

Appium is at its heart a webserver that exposes a REST API. It receives connections from a client, listens for commands, executes those commands on a mobile device, and responds with an HTTP response representing the result of the command execution. The fact that we have a client/server architecture opens up a lot of possibilities: we can write our test code in any language that has a http client API, but it is easier to use one of the Appium client libraries. We can put the server on a different machine than our tests are running on. We can write test code and rely on cloud services to receive and interpret the commands.

Session

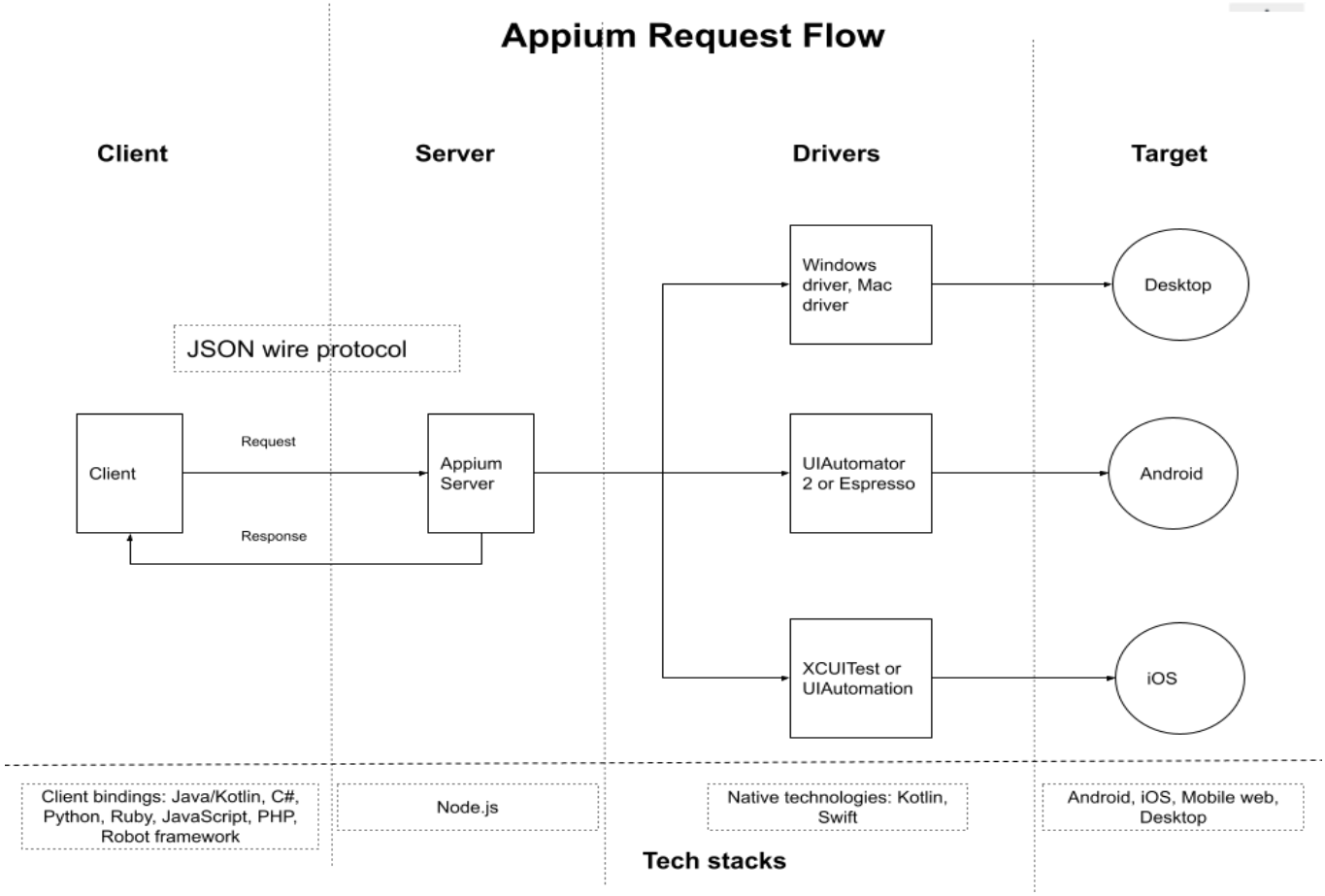
Automation is always performed in the context of a session. Clients initiate a session with a server in ways specific to each library, but they all end up sending a POST /session request to the server, with a JSON object called the 'desired capabilities' object. At this point the server will start up the automation session and respond with a session ID which is used for sending further commands.

Client/Server Architecture

Desired capabilities are a set of keys and values (i.e., a map or hash) sent to the Appium server to tell the server what kind of automation session we're interested in starting up. There are also various capabilities which can modify the behaviour of the server during automation.

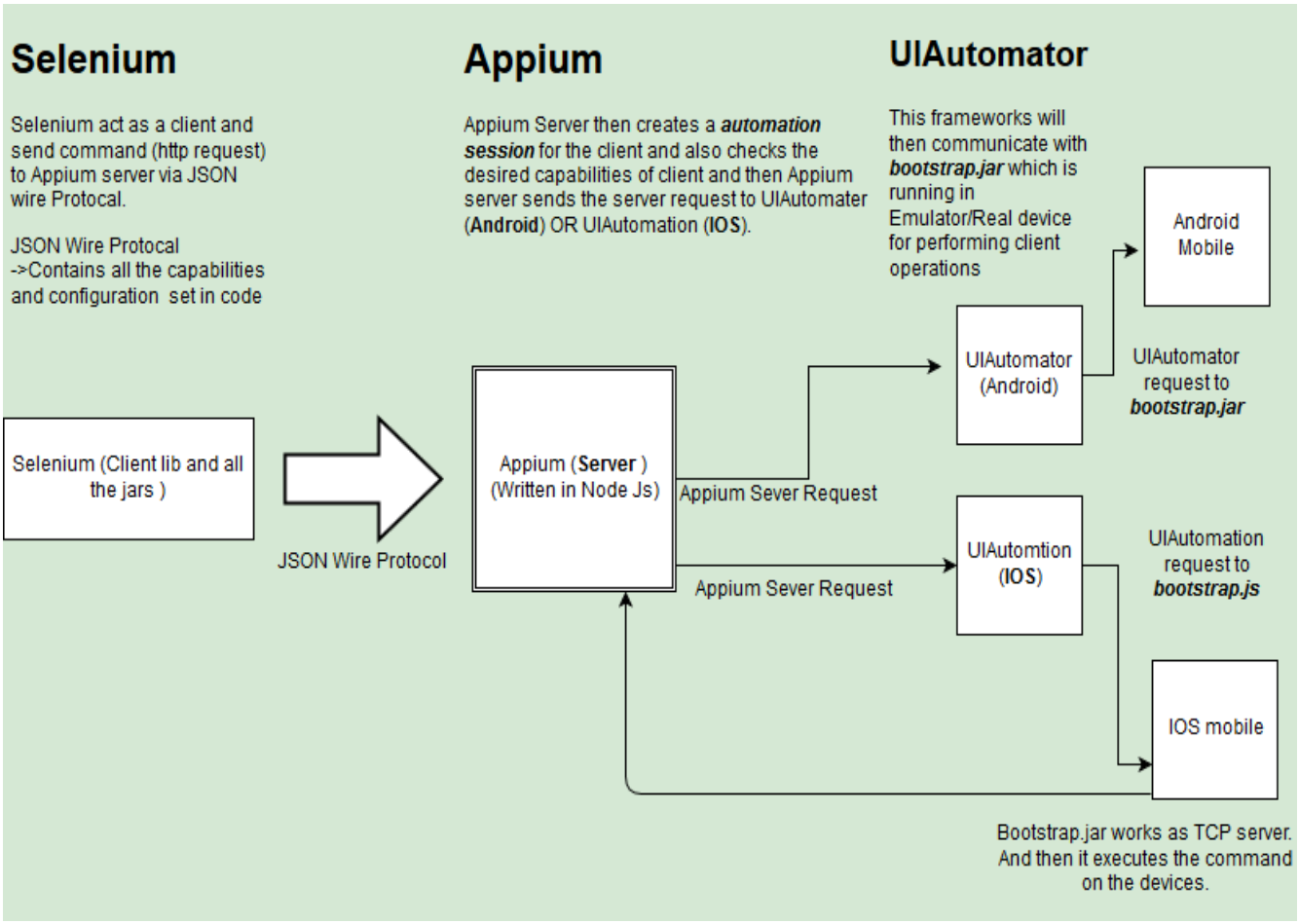
4.2 USE CASE DIAGRAM:

A use case diagram is a type of behavioral diagramdefined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which appium request flow.



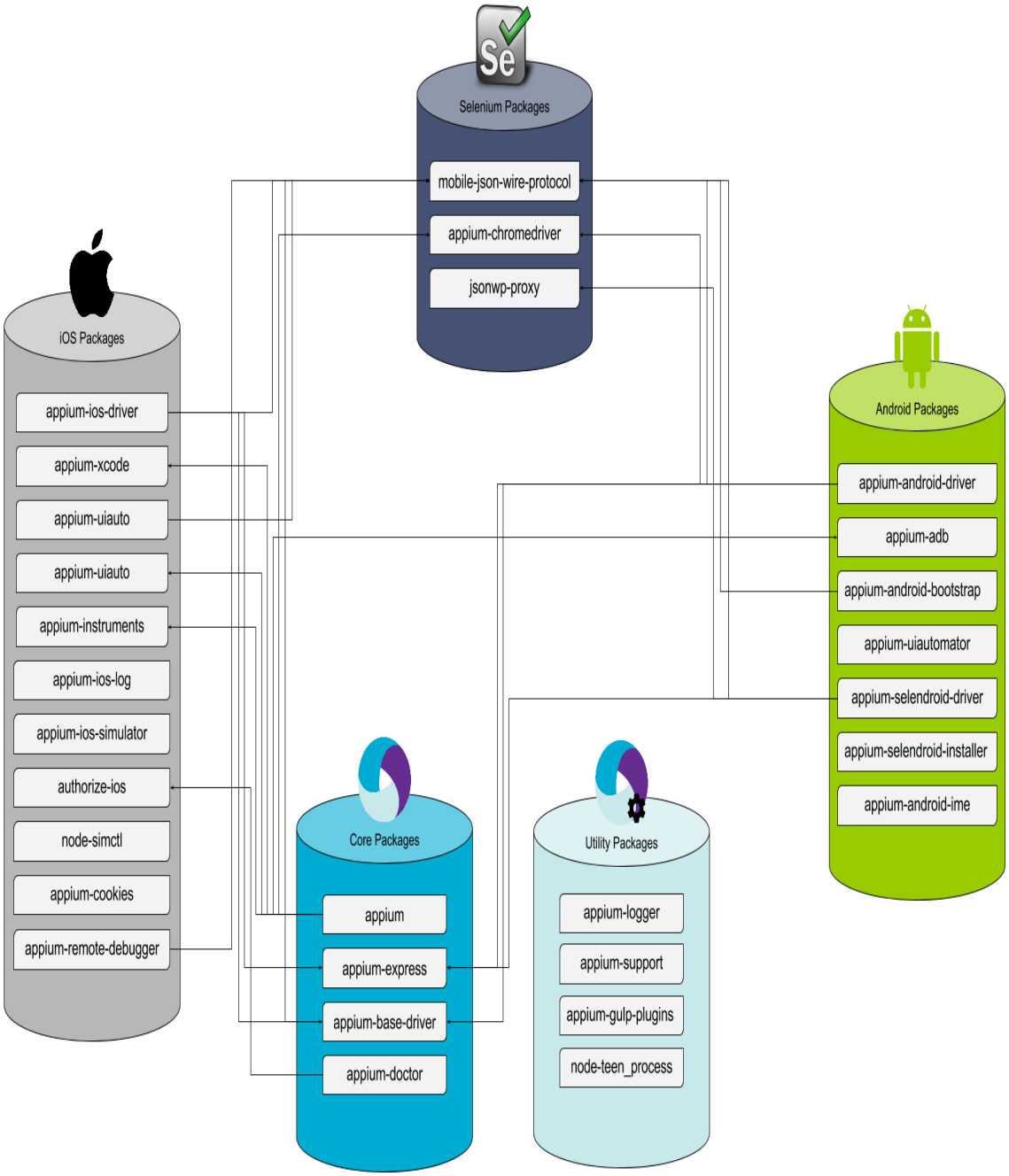
4.3 ACTIVITY DIAGRAM:

Activity diagrams are graphical representations of workflows of stepwise activitiesand actions with support for choice to explanation.



4.4 DATA FLOW DIAGRAM:

A data flow diagram (DFD) is a graphical representation of the “flow” of data through aninformation system.



CHAPTER 5

5. PROJECT DESCRIPTION

5.1 MODULES

5.2 MODULES DESCRIPTION

5.2.1 MODULE 1:

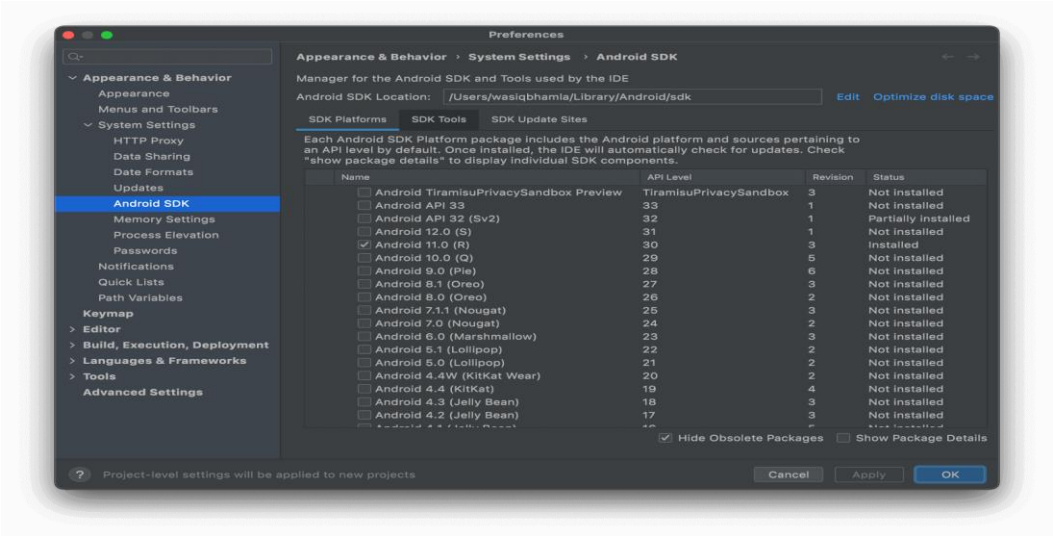
INTRO TO MOBILE TEST AUTOMATION

Automating mobile application testing is key to testing faster and extending test coverage, both on platforms and test scenarios. Mobile app automation is notoriously complex, but most test cases can successfully be automated. When done incorrectly, automation can be flaky and so time consuming it's not worth the effort to set up.

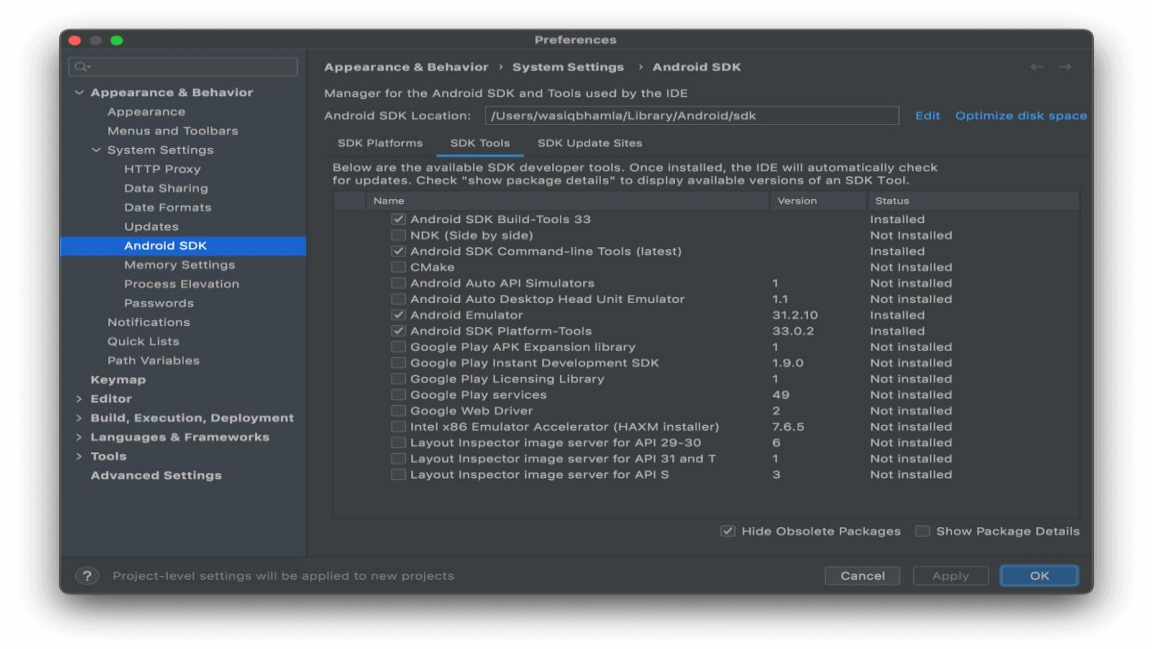
5.2.2 MODULE 2:

SETTING UP TOOL AND ENVIRONMENT

1. Download the Android installer and install it on your machine.
2. Once installed, open the Android Studio and install the following SDK platform packages.



3. Also, install the following SDK tools.



4. Once these steps are done, you need to set environment variables on your machine. Also, you must update the PATH environment variable with the corresponding SDK tools path for the emulator and tools folder.

5.2.3 MODULE 3:

REFRESHER FOR JAVA AND SELENIUM

- Page refresh operation ensures that all the web elements of a page are loaded completely before we proceed with our test cases automation Most commonly used method for page refresh in Selenium is the driver. Navigate ().
- refresh () method Get method and navigate methods include the recursive ways to refresh a page in Selenium.

5.2.4 MODULE 4:

GETTING STARTED WITH APPIUM

Installing Appium

Appium can be installed in one of two ways: via NPM or by downloading Appium Desktop, which is a graphical, desktop-based way to launch the Appium server.

Installation via NPM

If you want to run Appium via an npm install, hack with Appium, or contribute to Appium, you will need Node.js and NPM (use nvm, n, or brew install node to install Node.js. Make sure you have not installed Node or Appium with sudo, otherwise you'll run into problems). We recommend the latest stable version, though Appium supports Node.js 12+. (The minimal Node.js version follows EOL schedule)

The actual installation is as simple as:

```
npm install -g appium
```

Driver-Specific Setup

You probably want to use Appium to automate something specific, like an iOS or Android application. Support for the automation of a particular platform is provided by an Appium "driver". There are a number of such drivers that give you access to different kinds of automation technologies, and each come with their own particular setup requirements. Most of these requirements are the same requirements as for app development on a specific platform. For example, to automate Android applications using one of our Android drivers, you'll need the Android SDK configured on your system.

At some point, make sure you review the driver documentation for the platform you want to automate, so your system is set up correctly:

- The XCUITest Driver (for iOS and tv OS apps)
- The Espresso Driver (for Android apps)
- The UiAutomator2 Driver (for Android apps)
- The Windows Driver (for Windows Desktop apps)
- The Mac Driver (for Mac Desktop apps)

Verifying the Installation

To verify that all of Appium's dependencies are met you can use appium-doctor. Install it with `npm install -g appium-doctor`, then run the `appium-doctor` command, supplying the `--iOS` or `--android` flags to verify that all of the dependencies are set up correctly.

Appium Clients

Appium is just an HTTP server. It sits and waits for connections from a client, which then instructs Appium what kind of session to start and what kind of automation behaviours to enact once a session is started. This means that you never use Appium just by itself. You always have to use it with a client library of some kind (or, if you're adventurous, URL!).

Luckily, Appium speaks the same protocol as Selenium, called the WebDriver Protocol. You can do a lot of things with Appium just by using one of the standard Selenium clients. You may even have one of these on your system already. It's enough to get started, especially if you're using Appium for the purpose of testing web browsers on mobile platforms.

Appium can do things that Selenium can't, though, just like mobile devices can do things that web browsers can't. For that reason, we have a set of Appium clients

in a variety of programming languages, that extend the regular old Selenium clients with additional functionality. You can see the list of clients and links to download instructions at the Appium clients list.

Before moving forward, make sure you have a client downloaded in your favourite language and ready to go.

Starting Appium

Now we can kick up an Appium server, either by running it from the command line like so (assuming the NPM install was successful):

```
appium
```

Or by clicking the huge Start Server button inside of Appium Desktop.

Appium will now show you a little welcome message showing the version of Appium you're running and what port it's listening on (the default is 4723). This port information is vital since you will have to direct your test client to make sure to connect to Appium on this port. If you want to change the port, you can do so by using the -p flag when starting Appium (be sure to check out the full list of server parameters).

Prerequisites

- We'll assume you have an Android 8.0 emulator configured and running (the example will work on lower versions, just fix the version numbers accordingly)
- We'll assume you have this test APK downloaded and available on your local filesystem

Running Your First Test

In this section we'll run a basic "Hello World" Android test. We've chosen Android because it's available on all platforms. We'll be using the UiAutomator2 Driver so ensure you've read through that doc and gotten your system set up appropriately. We'll also be using JavaScript as the language so that we don't have to deal with additional dependencies.

(Chances are, you'll eventually want to automate something other than Android using something other than JavaScript. In that case, check out our sample-code, which has code samples for many languages and platforms.)

5.2.5 MODULE 5

UNDERSTANGING LOACTORS

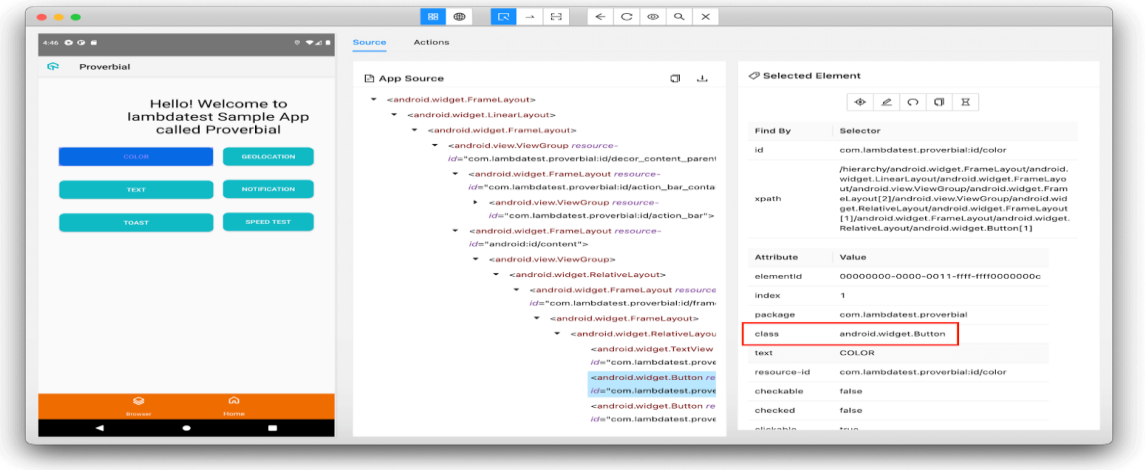
Class Name locator in Appium

Class name is another common strategy to identify the element in the application. The class is the full name of the XCUI element for iOS, which starts with XCUIElementType, and for Android is the fully qualified name of the element, which normally starts with android.widget.* when you select UIAutomator2 as the Automation name in the capability.

While using Java, you can use this locator in Appium as shown below:

```
import io.appium.java_client.MobileBy;
import org. openqa.selenium.By;
private final By colorByClassName = MobileBy.className
("android.widget.Button");
```

You can find exactly what class name there is for any element in Appium Inspector, as shown below:



Normally, you won't need the use of the class name unless the element is a dynamic one. There is only one element for that particular class name. Use cases would be many where you can use the class name locator, but it is highly suggested to use ID or accessibility id wherever possible.

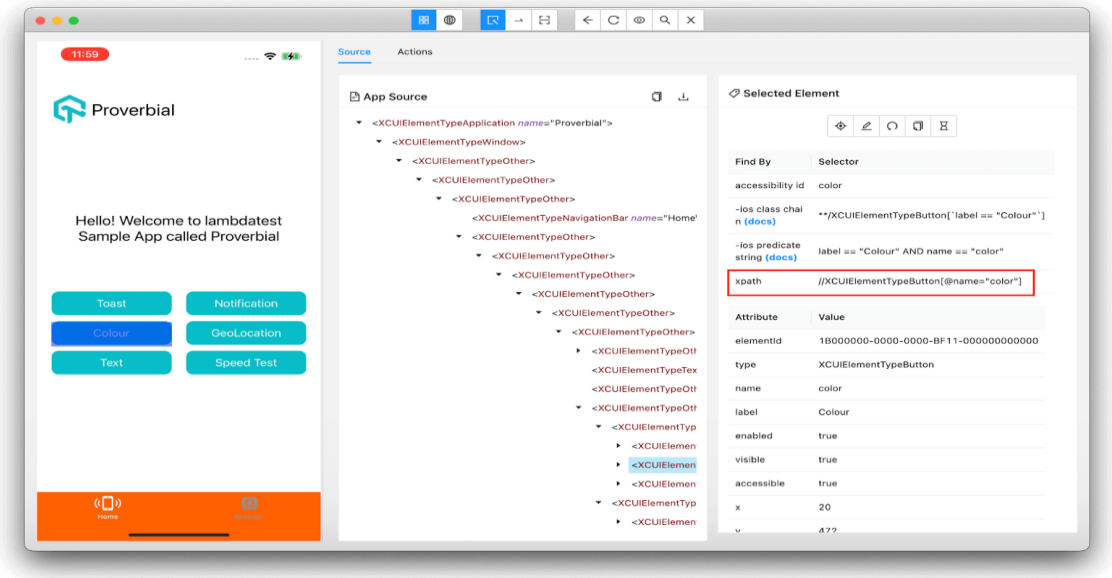
XPath locator in Appium

XPath scans the whole XML source tree of the application screen. It is the only locator in Appium that is not recommended by the Appium team out of all the locator strategies that Appium supports. The reason is that it has performance issues and is also the slowest performing locator strategy. Normally this locator is still supported when in the rare scenario when all the other locator strategies do not work, we can use XPath to find the element.

While using Java, you can use the XPath locator in Appium as shown below:

```
import io.appium.java_client.MobileBy;
import org.openqa.selenium.By;
. . .
private final By colorByXpath = MobileBy.xpath
(".//android.widget.Button[@text='COLOR']");
```

Appium Inspector also helps you with pre-built XPath expressions, which you can use directly. The same is shown below:



It is preferred not to use this locator strategy because even if there is no ID or accessibility id, you can still use other platform-specific locator strategies (which we will look at in some time) to find the element.

Strategies to use Android-specific locators in Appium

After commonly supported strategies to use locators in Appium, there are few platforms and automation type-specific locator strategies. Let's look into the Android-specific locators in Appium in detail.

UIAutomator2: UIAutomator Selector

The UIAutomator selector locator in Appium is one of the unique locator strategies where you have to create a Java statement and pass it as the locator text to the method. In this statement, you need to use the UiSelector class to build the Java statement.

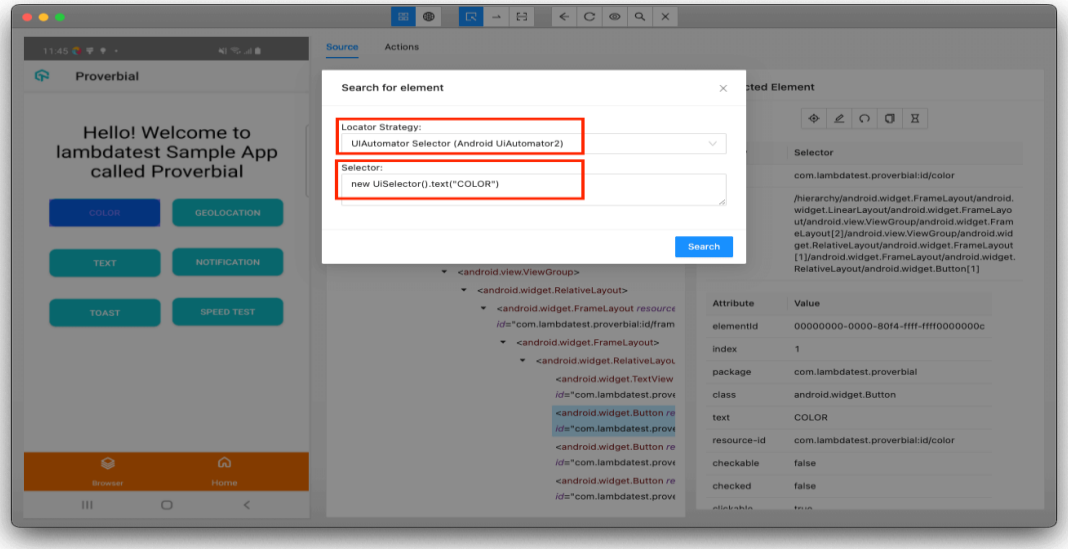
new UiSelector().<method_name>
Here, we are creating an instance for UiSelector and informing the server that we need to find an element that has the text COLOR.

Some of the frequently used methods in the UiSelector class are as follows:

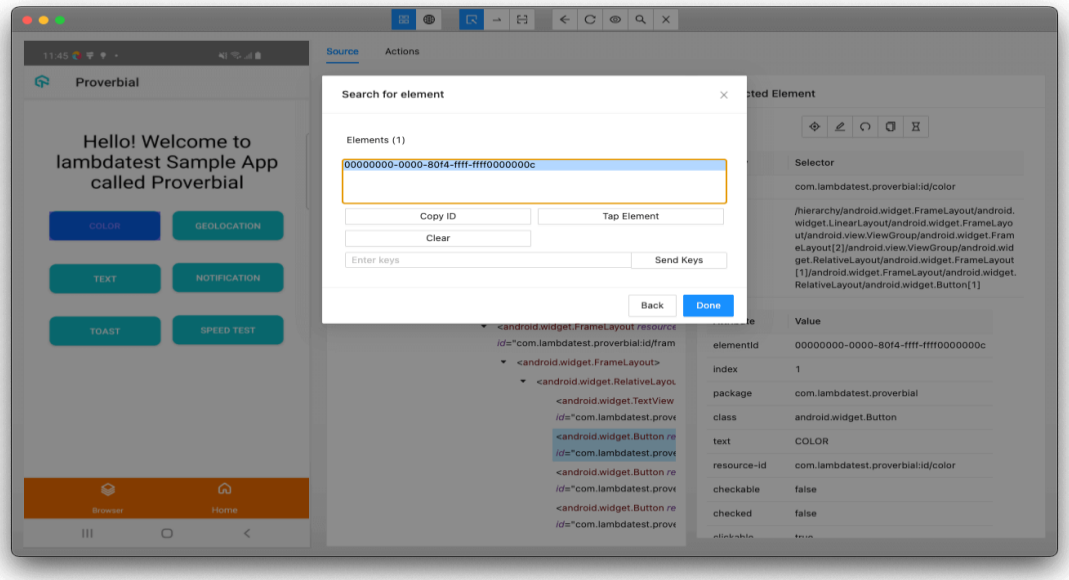
- **checked:** This method takes the expected value for the element to find an element that is checked. Mostly it is used with a checkbox element.
- **className:** This method takes a class name string, which you can find from Appium Inspector.
- **classNameMatches:** This method takes a regex expression to find an element based on its class name.
- **description:** This method takes in the content description attribute for the element, which can be found in Appium Inspector
- **descriptionContains:** This method takes in full or partial content description attributes for the element.
- **descriptionMatches:** This method uses a regex expression to find the element based on its content description.
- **descriptionStartsWith:** This method takes a starting part or full content description to find the element.
- **enabled:** This method takes a Boolean value to determine whether it's enabled.
- **index:** This method takes the element's index from the list of elements.
- **resourceId:** This method takes in the resource id equivalent to the ID locator.
- **resourceIdMatches:** This method takes a regex expression for resource id to find the element based on its resource id.
- **selected:** This method takes a Boolean value to find the selected or unselected element.
- **text:** This method takes in a text value to find an element by its text value.
- **textContains:** This method takes a partial text to find an element by its partial text value.
- **textMatches:** This method takes a regex value to find an element.

- **textStartsWith:** This method takes in a text value for an element to find it based on the text starting value.

You can verify if your selector is working or not by executing the locator expression in the Search for element window in the Appium Inspector.



In Appium Inspector, click on the magnifying glass icon and select locator strategy as UIAutomator Selector, create your selector, and click on the Search button. You will see the element once it's found on the next screen, as shown below:



You can also use the UiScrollable class to make the element move into view along with the UiSelector class.

5.2.6 MODULE 6

HANDLING TOUCH GESTURES

Automating Mobile Gestures with UiAutomator2 Backend

Touch actions are the most advanced and the most complicated way to implement any Android gesture. Although, there is a couple of basic gestures, like swipe, fling or pinch, which are commonly used in Android applications and for which it makes sense to have shortcuts, where only high-level options are configurable.

Mobile: Longclickgesture

This gesture performs long click action on the given element/coordinates. Available since Appium v1.19

Supported arguments

- elementId*:* The id of the element to be clicked. If the element is missing then both click offset coordinates must be provided. If both the element id and offset are provided then the coordinates are parsed as relative offsets from the top left corner of the element.
- *x*: The x-offset coordinate
- *y*: The y-offset coordinate
- *duration*: Click duration in milliseconds. 500 by default. The value must not be negative

Mobile: DoubleClick Gesture

This gesture performs double click action on the given element/coordinates. Available since Appium v1.21

Supported arguments

- *elementId*: The id of the element to be clicked. If the element is missing then both click offset coordinates must be provided. If both the element id and offset are provided then the coordinates are parsed as relative offsets from the top left corner of the element.
- *x*: The x-offset coordinate
- *y*: The y-offset coordinate

5.2.7 MODULE 7

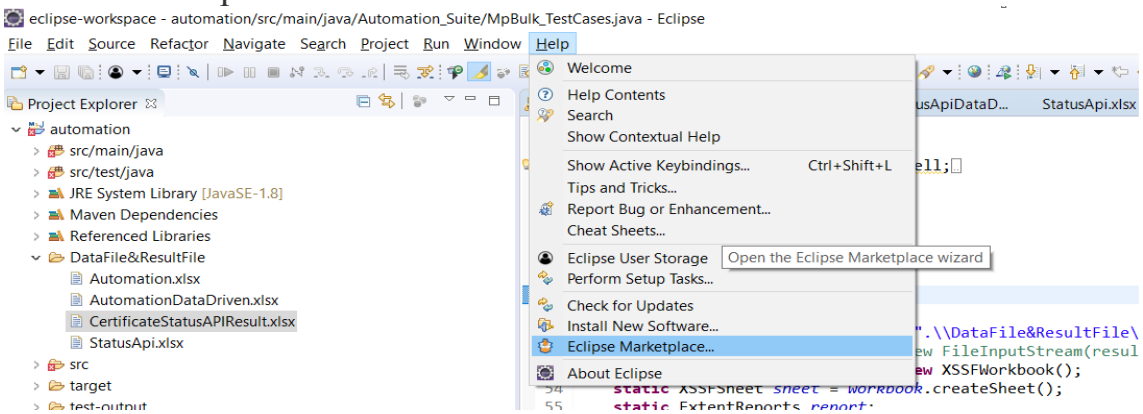
TestNG Test Framework

Installing and Setting up TestNG

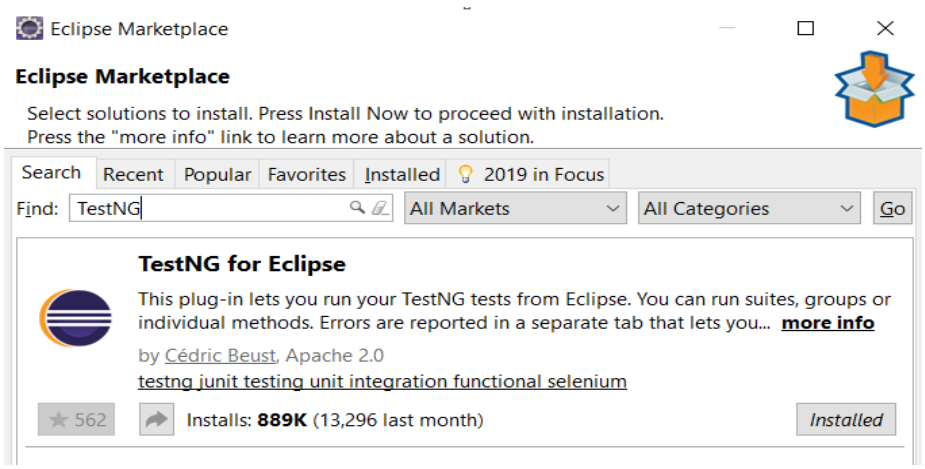
It's pretty easy to install TestNG. If you are using Eclipse IDE, it comes as a plugin to it. Below are the steps to install TestNG:

1. Install Eclipse IDE from the Eclipse website. It serves as a platform for you to code on and write your test cases

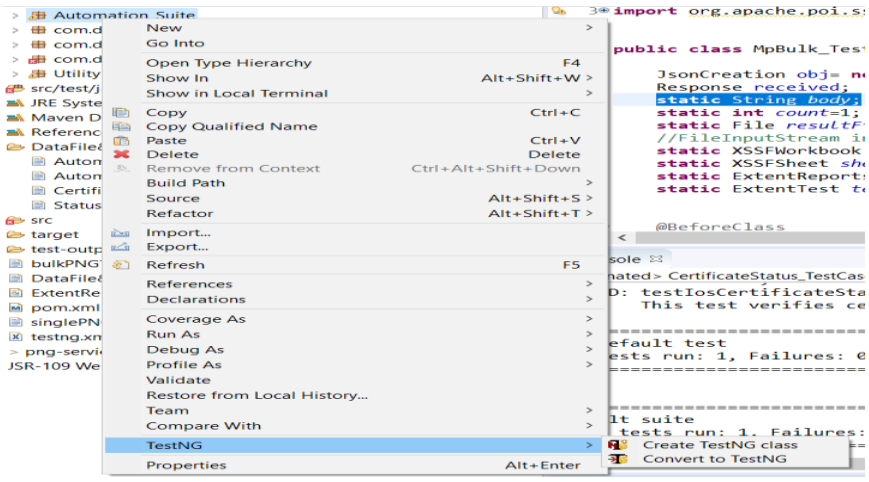
2. Once installed, go to help and navigate to the 'Eclipse Marketplace'. The referenced snapshot is below:



3. Click on 'Eclipse Marketplace'. You will be directed to the marketplace modal. Type TestNG in the search keyword and hit 'Go'. The referenced snapshot is below:



4. If TestNG is not installed in your Eclipse, rather than the ‘Installed’ button you would see ‘install’. Click on install and your TestNG framework will be installed in your Eclipse. As a good practice, Eclipse would recommend you to restart to use the features of the installed plugin
5. Post-restarting your Eclipse, re-verify whether you can see options for creating a TestNG class or not as below:



TestNG Annotations

An annotation is a tag that provides information about the method, class, and suite. It helps to define the execution approach of your test cases and the different features associated with it. Below are the major annotations used:

- ***@Test***– This is the root of TestNG test cases. In order to use TestNG, all methods should be annotated with this annotation. Below is an example:

```
@Test

public void setupTestNG ()

{

System.out.println(“this is a test annotation method”)

}
```

A few attributes associated with the TestNG annotation are:

1. **Description**: You can describe your test case under the description, stating what it does

```
@Test (description=" This test validates login functionality")
```

2. **Priority**: You can prioritize the order of your test methods in TestNG by defining a priority. Based on the defined priority, the test shall execute in that order.

```
@Test(priority=1)
```

3. **DependsOnMethod**: This attribute works miracles if one test case is dependent on the other test case. For example, to view your profile details, you need to login to the application. So, your profile test case is dependent on the login test case

```
@Test (dependsOnMethod=" Login")
```

4. **Enabled**: Using this attribute, you can choose to execute or skip the execution of this test case. Setting it to true execute it and setting it to false will skip the test from the execution cycle

```
@Test(enabled='true')
```

5. **Groups**: Using this attribute, you can club your test cases into a single group and specify the specified group you wish to execute in your TestNG XML file. The test cases clubbed to that group will only be executed and the rest will be skipped

```
@Test (groups=" Smoke Suite")
```

While the above ones should help you get started, other major annotations are:

- **@BeforeMethod and @AfterMethod** – These annotations run before and after each test method
- **@BeforeClass and @AfterClass** – These annotations run once before and after the first *@test* method in a class
- **@BeforeTest and @AfterTest** – The Before Test annotation runs before the *@BeforeClass* annotation and the After Test annotation runs after the *@AfterClass* annotation

- **@BeforeSuite and @AfterSuite**– These annotations run before and after any test annotated method in a class respectively. These annotations start the beginning of a test and the end of it, for all the classes in a suite

5.2.8 MODULE 8

CONFIG, LOGGING & REPORTING

TestNG is an open-source test automation framework created by Cedric Beust, where 'NG' stands for Next Generation. TestNG is similar to JUnit, however, it is preferred over JUnit, particularly while testing coordinated classes. It is because TestNG offers ease of using **multiple annotations, grouping, dependence, prioritization,** and **parametrization.**

TestNG allows QAs to categorize or prioritize the test cases, induce HTML reports, log dispatches, run tests in parallel, and perform many more actions. These features help QA leverage TestNG with Selenium in automation testing.

In order to use TestNG Reporter Log, it is necessary to understand the significance of reporting in a test.

Why is Reporting important

Reports can be insightful and help in deciding the future course of action as it a one-stop source of all the accessible information. With automated report generation frameworks, it enhances the visualization of test results for the betterment of overall analysis.

However, despite sharing all the major information regarding the tests, TestNG reports might at times fail to convey the exact root of failure due to inadequate logging. As a result, you might need to run the whole class to find the problem.

To overcome this problem TestNG provides an inbuilt class called the Reporter Class that is used for logging. This tutorial explores how to use the TestNG Reporter Log in Selenium to identify the cause of a failed test which would improve debugging.

Reporter Class is an inbuilt class in TestNG, which is available under the **org.testng** package. This class provides test styles to log dispatches that will be included in the HTML reports generated by TestNG. Reporter Class is one of the simplest ways of generating log information, where the logs in the reports can be either stoner-generated or system-generated reports.
TestNG Reporting Class is quite useful as it helps analyze failed tests based on the detailed information from the logs. This avoids the need to rerun the entire test case. By specifying different logs at each step, QAs can use this classification when debugging the failures, making debugging easier.

How to use TestNG Reporter Log in Selenium

Selenium WebDriver is widely used for test automation of websites. However, it lacks reporting of the test's logs, hence TestNG is used to create report logs that would help identify and debug failed tests. TestNG induces dereliction of HTML reports once the test cases are executed.
Following TestNG Reports can be created:

- emailable-report.html
- index.html
- Reporter Class

To understand how TestNG report works using an example, here is a sample Selenium test with TestNG Report.

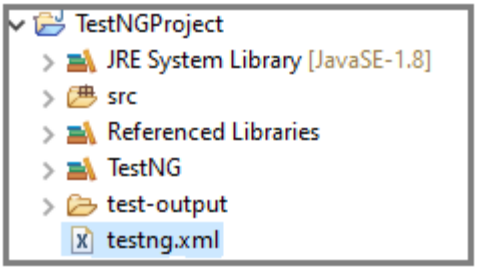
Step 1: Create a Selenium Test Class named as **EmailReport** and **TestNG.xml** file using the following code

```
package testngpackage;

import org.testng.Assert;

import org.testng. SkipException;

import org.testng.annotations.Test;

public class EmailReport {

//To make it pass

@Test

public void pass Test(){
```

```
Assert.assertTrue(true);

}

//To make it fail

@Test

public void failTest(){

Assert.assertTrue(false);

}

//To make it skip

@Test

public void skipTest(){

throw new SkipException("Skipping -This method is skipped testing ");

}

}
```

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="testngpackage" parallel="methods">

<test name="testngTest">

<classes>

<class name="testngpackage.EmailReport" />

</classes>

</test>

</suite>
```

Step 2: Execute the testng.xml file and refresh the project. You can see the testng.xml file within the project folder **TestNGProject** as seen below.



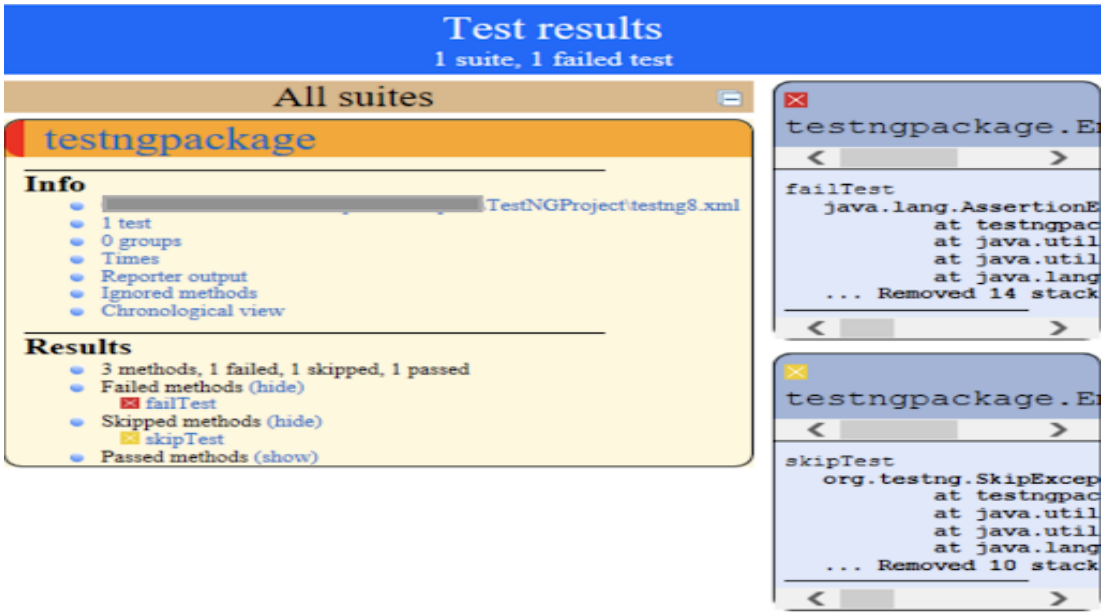
Step 3: Expand **test-output** folder and you should find a default generated TestNG report **emailable-report.html** and **index.html**. Open the report in the browser. Reports will be seen as below:

Emailable Report in TestNG

Test	# Passed	# Skipped	# Failed	Time (ms)	Included Groups	Excluded Groups
testngpackage						
testngTest	1	1	1	74		

Class	Method	Start	Time (ms)
testngpackage			
testngTest — failed			
testngpackage.EmailReport	failTest	1589713167690	19
testngTest — skipped			
testngpackage.EmailReport	skipTest	1589713167690	17
testngTest — passed			
testngpackage.EmailReport	passTest	1589713167690	27

HTML Index Report in TestNG



5.2.9 MODULE 9

APPIUM FOR WEB APPLICATION

Android Mobile Web Automation

Appium supports automating the Chrome browser both real and emulated Android devices.

Pre-requisites:

- Make sure Chrome is installed on your device or emulator.
- Chromedriver needs to be installed (a default version comes with Appium) and configured for automating the specific version of Chrome available on the device. See here for more information and details about compatibility.

```
DesiredCapabilities capabilities = new DesiredCapabilities ();

capabilities.setCapability(MobileCapabilityType.PLATFORM_NAME, "Android");

capabilities.setCapability(MobileCapabilityType.PLATFORM_VERSION, "9.0");

capabilities. setCapability (MobileCapabilityType.DEVICE_NAME, "Android Emulator");

capabilities. setCapability (MobileCapabilityType.AUTOMATION_NAME, "UIAutomator2");

capabilities.setCapability(MobileCapabilityType.BROWSER_NAME, "Chrome");
```

Troubleshooting Chromedriver

If your test target requires newer Chromedriver version, Chromedriver feature will help. It has been available since Appium 1.15.0 with the security option. Read the linked documentation to learn how to use it. Chromedriver Executable Dir capability also helps when you need a specific Chromedriver version. As of Chrome version 33, a rooted device is no longer required. If running tests on older versions of Chrome, devices needed to be rooted as Chromedriver required write access to the /data/local directory to set Chrome's command line arguments.

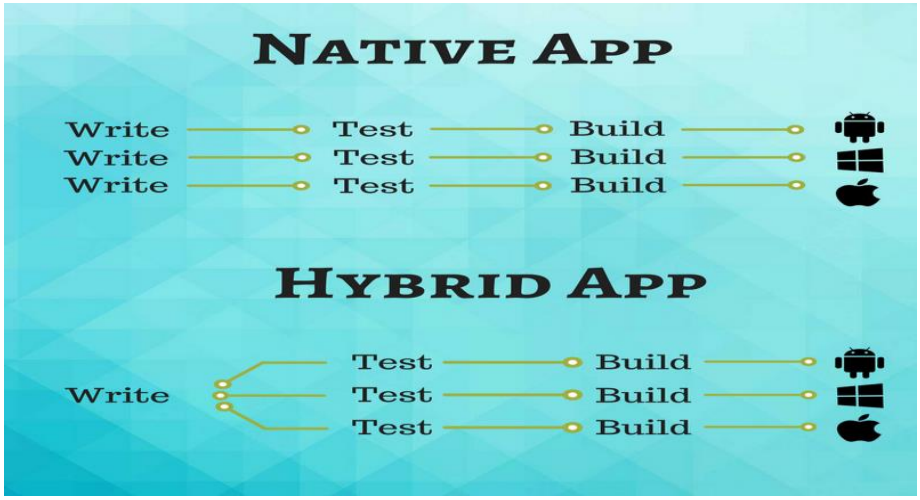
If testing on Chrome app prior to version 33, ensure adb shell has read/write access to /data/local directory on the device:

```
$ adb shell su -c chmod 777 /data/local
```

There is a desired capability showChromedriverLog which, when set to true, writes the Chromedriver logs inline with the Appium logs. This can be helpful for debugging.

5.2.10 MODULE 10

APPIUM FOR NATIVE & HYBRID APP



Appium is an open-source automation tool that can be used to automate native and hybrid mobile applications for both iOS and Android platforms. With Appium, cross-platform testing is simple since the same test scripts can be used for multiple platforms.

NATIVE APP:

To automate a native application on Android device, please follow the steps listed below:

- Connect the mobile device or Create an Emulator
- Get details of the mobile device
- Get details of the app which is to be launched
- Start Appium Server
- Write Appium test script
- Interact with elements using the UIAutomator Viewer
- Run the script and automate the app

HYBRID APP:

Automating hybrid Android apps

Appium comes with built-in hybrid support via Chromedriver, which allow the automation of any Chrome-backed Android web views.

There is an additional step necessary within your app build, unfortunately. As described in the Android remote debugging docs it is necessary to set to true the setWebContentsDebuggingEnabled property on the android. Webkit. WebView element.

Once you have set your desired capabilities and started an Appium session, follow the generalized instructions above.

Automating hybrid iOS apps

To interact with a web view Appium establishes a connection using a custom remote debugger. When executing against a simulator this connection is established directly as the simulator and the Appium server are on the same machine. Appium can automate WkWebView and UIWebView elements. Unfortunately, it is not currently able to handle SafariViewController elements.

Once you've set your desired capabilities and started an Appium session, follow the generalized instructions above.

Execution against an iOS real device

When executing against an iOS real device, Appium is unable to access the web view directly. Therefore, the connection has to be established through the USB cable. Appium can establish the connection natively since version 1.15 via appium-ios-device. ios-webkit-debugger-proxy is only necessary for Appium below version 1.15.

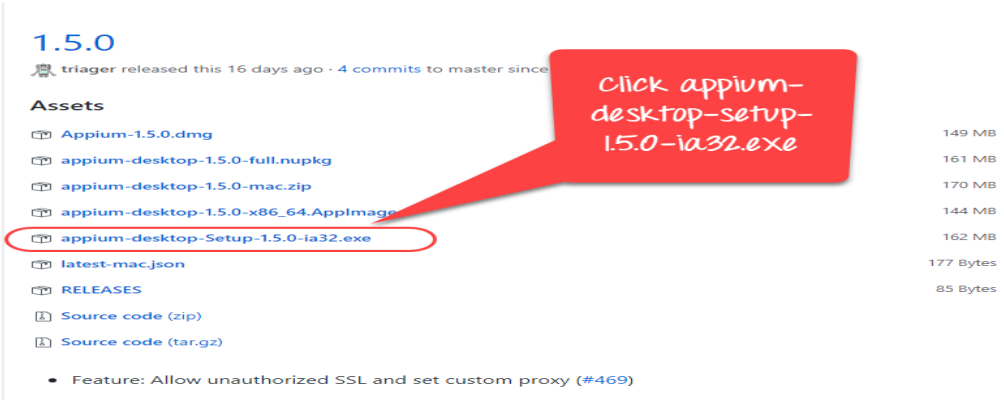
5.2.11 MODULE 11

PROJECT WORK WITH APPIUM

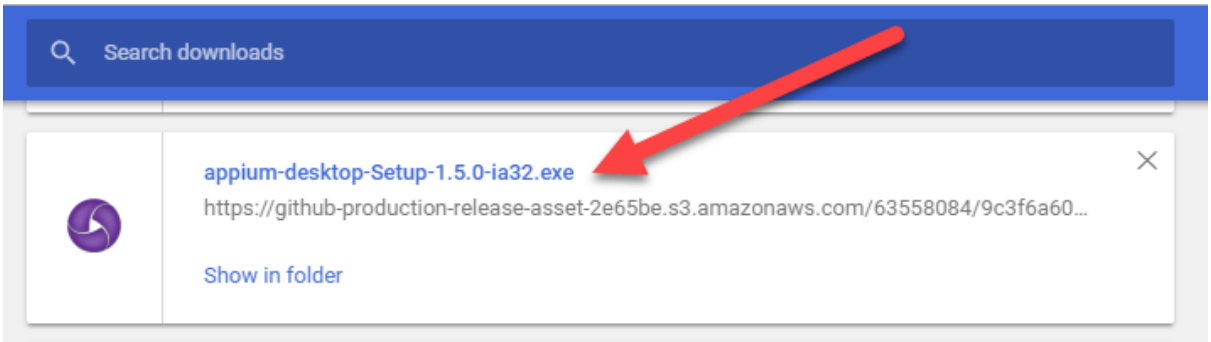
Appium Studio is an Open-source GUI app to install Appium Server. It comes bundled with all the pre-requisites to install and use Appium Server. It also has an Inspector to get basic information on your Apps. It comes with a Recorder to create boilerplate code to automate your mobile apps.

Step 1) Go to http://appium.io/ and click on Download Appium.

Step 2) For Windows, select the exe file and download. The file is around 162MB will take time to download based on your internet speed.

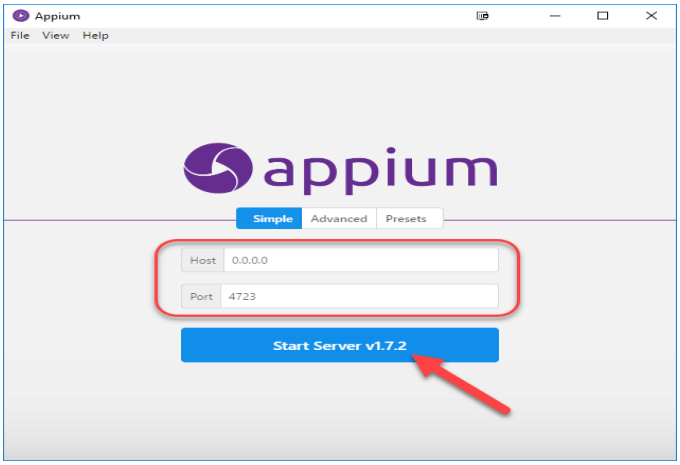


Step 3) Click on the downloaded exe.

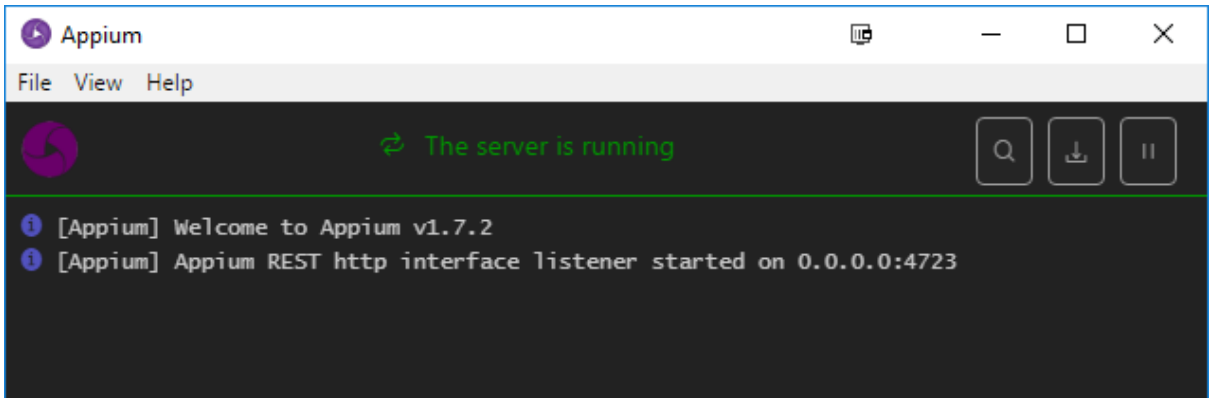


Step 4) On a Windows machine, there is no need to install Appium. It runs directly from the exe. Once you click the exe you will see the following image for few minutes.

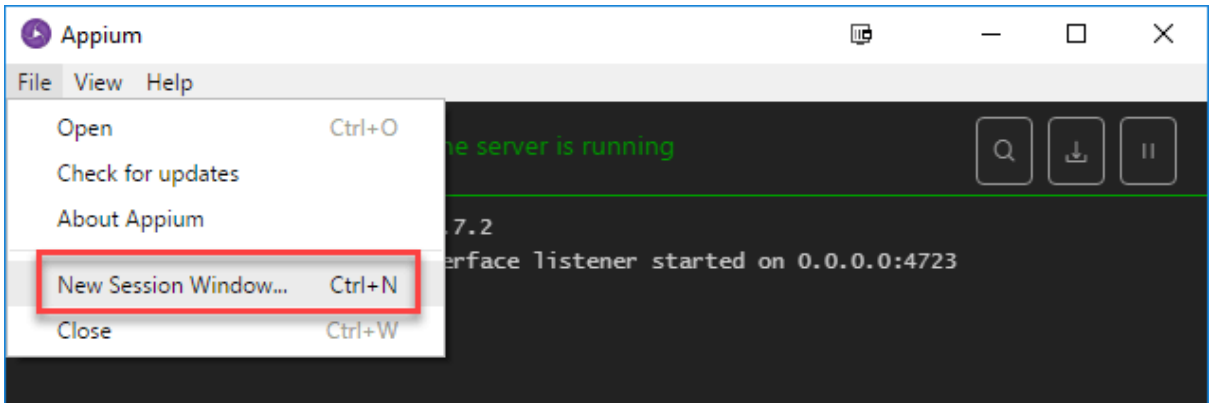
Step 5) Next you will see the Server Start Window. It populates the default host and port option which you can change. It also mentions the version of Appium being used.



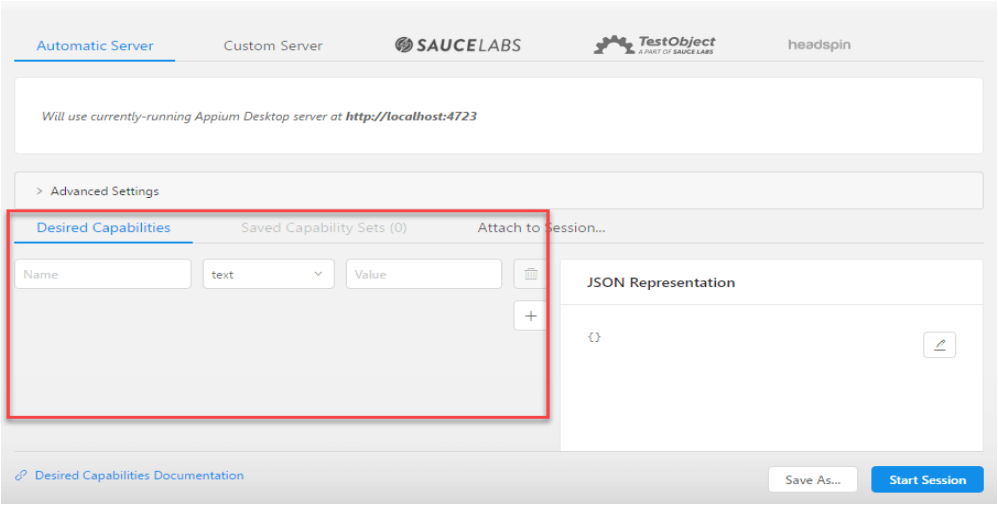
Step 6) On clicking the Start Server Button, a new server is launched on the specified host and port. Server log output is shown.



Step 7) Click New Session Window**.**



Step 8) You can enter the Desired Capabilities and start a session.



APPIUM Inspector:

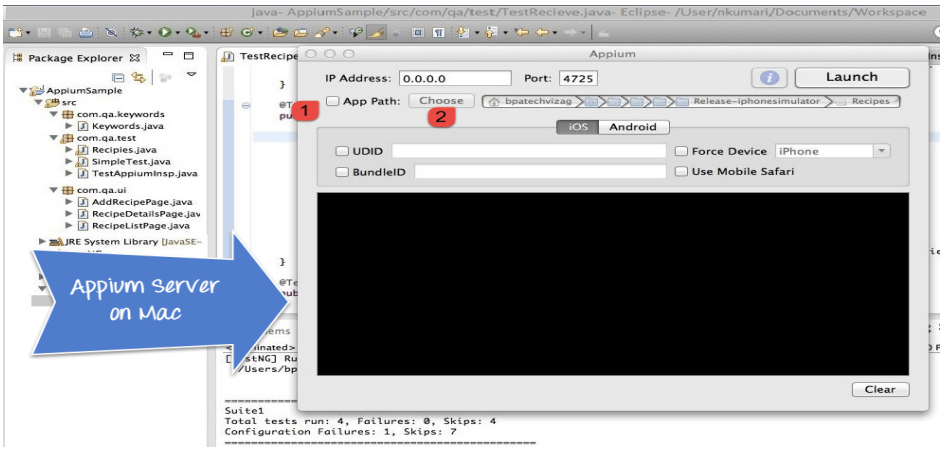
Similar to Selenium IDE record and playback tool, Appium has an 'Inspector' to record and Playback. It records and plays native application behaviour by inspecting DOM and generates the test scripts in any desired language. However, currently, there is no support for Appium Inspector for Microsoft Windows. In Windows, it launches the Appium Server but fails to inspect elements. However, UIAutomator viewer can be used as an option for Inspecting elements.

Steps to start with Appium Inspector on Mac machine: -

Step 1) Download and start your Appium server with the default IP Address 0.0.0.0 and the port 4725.

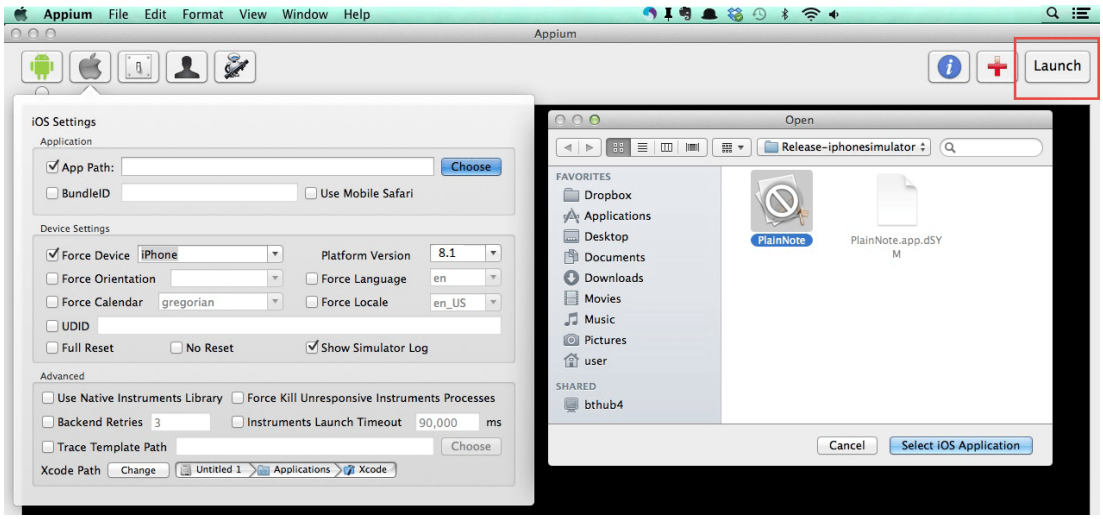
1. Select the source file or .app files from local to test.
2. Check the ‘App Path’ Checkbox to enable ‘Choose’ button.

Step 2) Now, click on ‘Choose’ button will give the option to browse and select test file from the local drive.

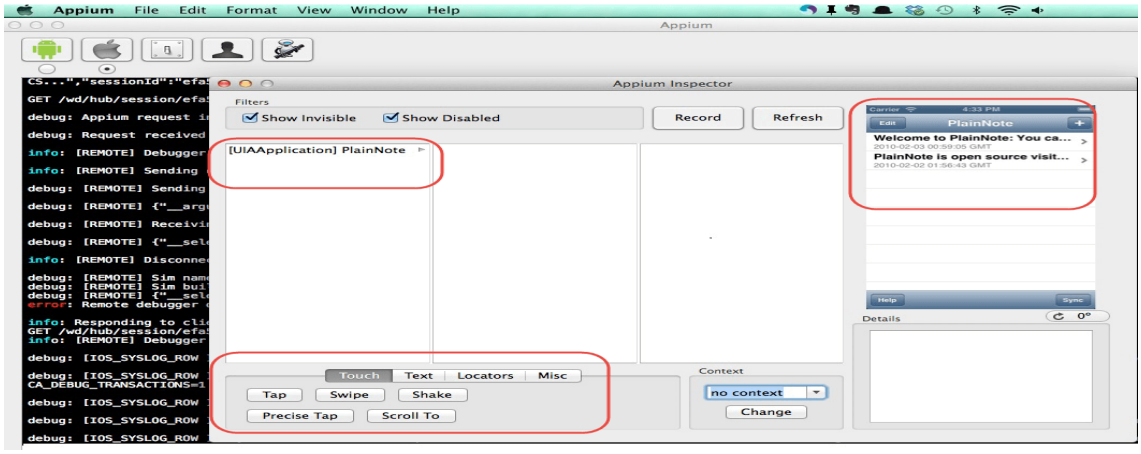


Step 3) Start Simulator on Mac machine.

Step 4) Click ‘Launch’ button from a top right corner, which enables a blue colour icon. Again, click on this blue colour icon, it will open the Appium inspector and Simulator with a pre-selected application.



Step 5)– Launching your Appium Inspector will show the element hierarchy in column-wise structure. Also, a user can apply actions using buttons like Tap, Swipe, etc.

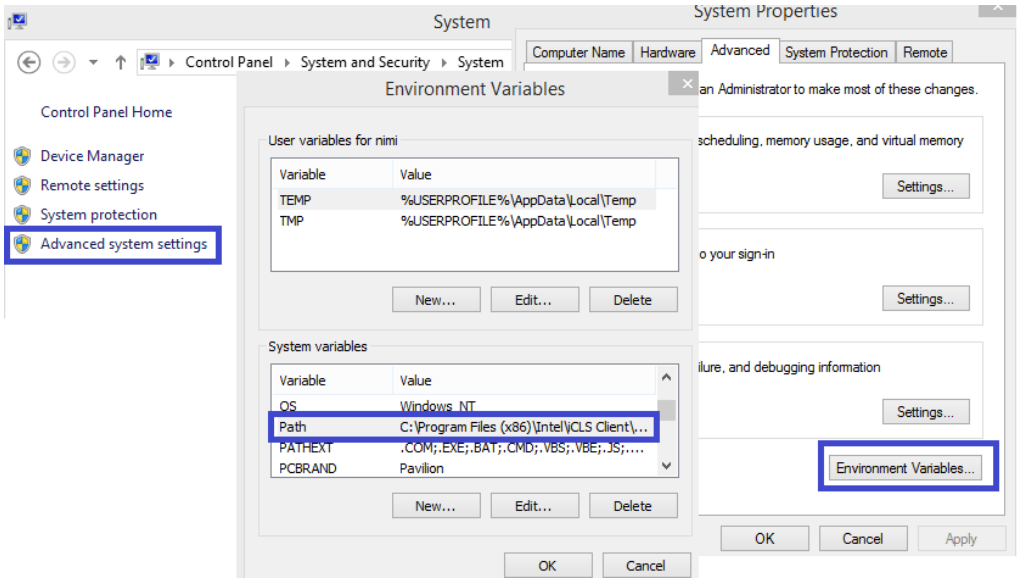


Step 6) Click on ‘Stop’ button to stop recording.

Attach Android Emulator to Appium

Step 1) Install Android SDK in your system.

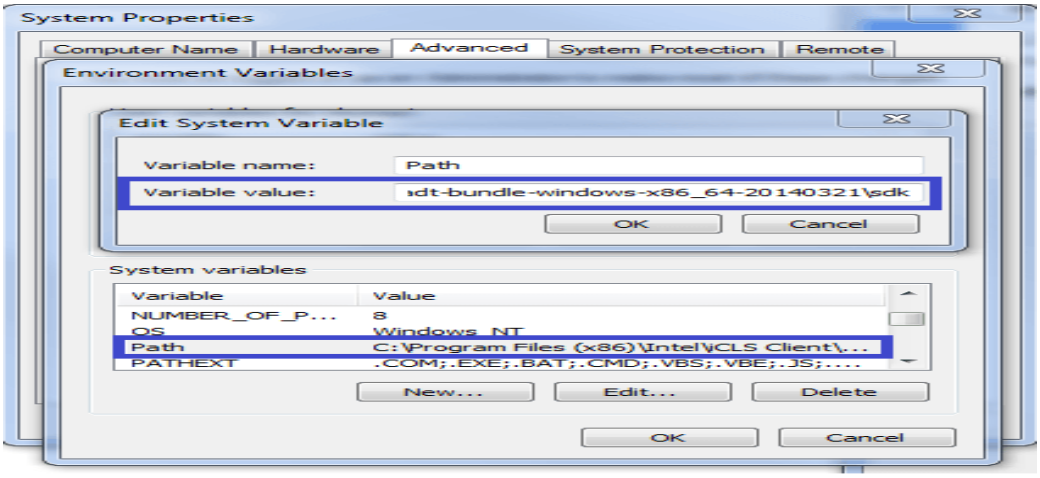
Go to Control panel >> System and Security >> System and from the left panel click on ‘Advanced System Settings’. From ‘System Properties’ pop up, click on ‘Advanced’ tab and then click on “Environment Variables” button.



Step 2) Now, from ‘Environment variables’ pop up, ‘double click on ‘Path’ and set ANDROID_HOME variable that point to your SDK directory. In the path append the whole SDK folder path.

e.g. –

C:\User\ABC\Desktop\adt-bundled-windows-x86_64-20140321\sdk

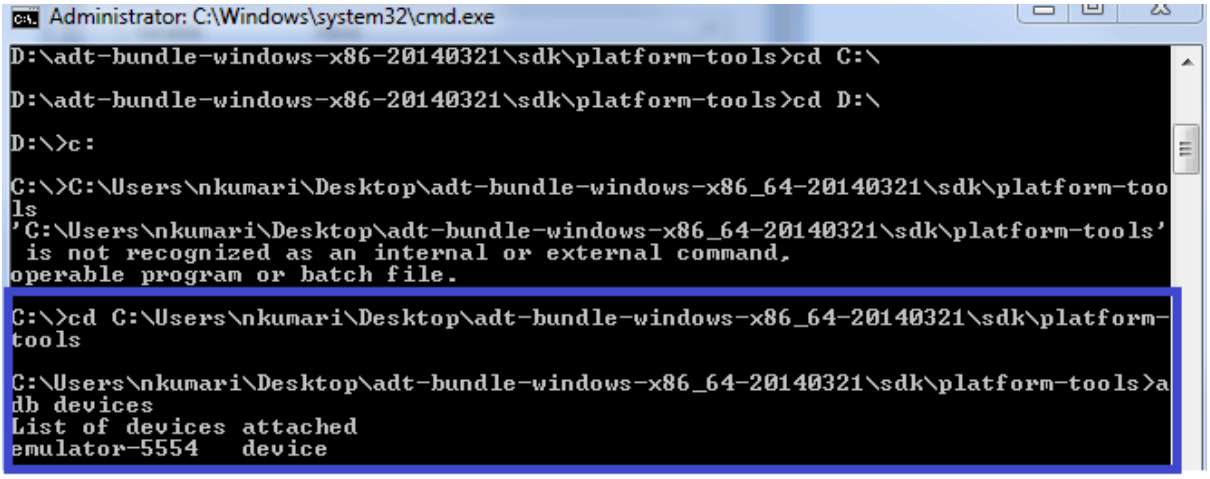


Double Click on 'Path' and add SDK path

Step 3) Start your Android emulator or any attach any Android device to your system (Make sure you have Android Debugging option enabled in your Android device. To check Debugging Option. Go to Device Settings >> Developer Options >> Check "Debugging Option").

Step 4) Open Command Prompt and navigate to your Android SDK's \platform-tools\ directory (E.g., D:\adt-bundle-windows-x86_64-20130514\sdk\platform-tools).

Step 5)– Run 'adb devices' command. You can see your connected device listed in Command Prompt window. (In CMD write '>adb devices'- This command will list the connected emulator instances. E.g.: adb –s emulator-5554 install <Location of .apk file>)



Start AVD Manager and SDK manager and Android emulator and verfiy from ">adb device" command.

Step 6)– Run 'adb start-server' command. It will start ADB server that will be used by Appium to send commands to your Android device.

Step 7) Now, navigate to Appium directory in your system and start Appium by clicking an Appium.exe file.

Step 8) Do not alter the IP address or port number and click 'Launch' button. Your Appium console starts at 127.0.0.1:4723 as shown in below.

APPIUM Server

Step 9) Click on 'Start' button, Appium server started running on your system.

5.2.12 MODULE 12

YOUR NEXT STEPS

While the specific mobile testing requirements may vary depending upon the application you want to test, here's a general mobile testing automation checklist to ensure your mobile app turns out to be high quality:

1. Test Environment Preparation

You first need to prepare the test environment for mobile testing automation. It is essential because when you run mobile app tests in a stable and consistent environment, there are minimum chances of false positives and negatives. By simulating real-world conditions, you can test the functionality of your mobile app in scenarios that users will face in real life.

Additionally, a well-prepared test environment can improve the efficiency and effectiveness of the testing process by reducing the time and effort required to diagnose and resolve issues that may arise during testing.
Here's how you can prepare the test environment for mobile test automation:

- Define the device, operating system, and network configuration for testing.
- Set up the automation test environment, including hardware and software requirements.
- Install required tools and software, such as the testing framework, application binary, and mobile device emulator.

2. Test Case Preparation

Once you have established a stable and consistent environment for mobile testing automation, prepare the test cases. It is essential because well-prepared test cases serve as a roadmap for the testing process and offer a clear understanding of the testing scope.Test case preparation can also help you identify potential problems early in the development cycle and save time and resources, as developers can fix issues before they become complex or difficult to resolve.
Here's what we do during test case preparation:

- Identify the critical functionalities and use cases of the application.
- Write test cases that cover all scenarios, including positive, negative, and edge cases.
- Consider the test data requirements and prepare test data accordingly.

3. Test Script Development

A test script is a set of instructions written in a programming language that implements the steps of a test case, allowing the tests to run automatically

without manual intervention. It ensures that the tests are repeatable, reliable, and accurate, which can save time and resources by reducing the need for manual testing.

Test script development is critical in mobile testing automation because it improves the testing process's efficiency, reliability, and accuracy. It also ensures that the mobile application meets the quality standards for its successful deployment.

Here's what you should do during the test script development stage:

- Develop test scripts using a scripting language compatible with the testing framework.
- Ensure that the scripts are reusable and maintainable.
- Make use of reusable functions and modules.

4. Test Execution

In test execution, we run tests and evaluate the results to determine if the mobile application meets the specified requirements and quality standards. It is a critical step in the mobile testing process that provides the actual validation of the mobile application's functionality and performance.

Here's what you should do in the test execution phase:

- Execute the test scripts on the specified device, operating system, and network configuration.
- Record the test results and report any failures.
- Verify the application's performance and behavior on different devices and operating systems.

5. Test Result Analysis

In the test analysis phase of mobile testing automation, we evaluate and interpret the results of the tests to determine if the mobile application meets the specified requirements and quality standards. The idea is to provide a clear and comprehensive understanding of the mobile application's quality and identify any areas that need improvement.

With test result analysis, you can prioritize and track the resolution of issues and decide if additional testing is necessary to ensure that the application meets the required quality standards.

Here's what you should do in the test result analysis phase:

- Analyze the test results to identify the causes of any failures.
- Report and track defects and issues found during testing.
- Evaluate the overall quality of the application and recommend improvements.

6. Test Maintenance

Executing tests and evaluating the results is only half the job done. You also must ensure that the automated tests are still properly running after you have made changes to the application or its environment. The test maintenance phase aims to keep the test automation infrastructure reliable and effective so that it can continue to provide valuable feedback to the development team.

Here's what you should do during the test maintenance phase:

- Regularly review and update the test cases, scripts, and data as required.
- Maintain the automation testing environment and tools to ensure they are up to date.
- Re-run the test suite after each release or change to maintain the application's quality

CHAPTER 6

6. CODING AND TESTING

6.1 MAIN CLASS:

```
package testpack;
import java.io.File;
import java.net.URL;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Properties;
import java.util.concurrent.TimeUnit;
import java.time.Duration;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import org.apache.commons.io.FileUtils;
import org.openqa.selenium.By;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
import com.aventstack.extentreports.Status;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;
import org.testng.Assert;
import com.aventstack.extentreports.ExtentReports;
import com.aventstack.extentreports.ExtentTest;
import com.aventstack.extentreports.Status;
import com.aventstack.extentreports.reporter.ExtentHtmlReporter;
import com.aventstack.extentreports.reporter.configuration.ChartLocation;
import com.aventstack.extentreports.reporter.configuration.Theme;
```

```
import io.appium.java_client.android.AndroidDriver;
public class FacebookAutomation{

   //public static WebDriver driver;
         //public static AndroidDriver driver;
         public static ExtentHtmlReporter htmlReporter;
   public static ExtentReports extent;
   public static ExtentTest test;
   public static String folderPath;
   public static String filename;
   public static String foldername;
   public static void main(String[] args) throws Exception {

         test = extent.createTest("Automation");
          System.setProperty("webdriver.chrome.driver","C:\\Users\\Gopinath\\eclipse-
workspace\\test11111\\chrome driver\\chromedriver.exe");

          ChromeOptions options = new ChromeOptions();
          options.addArguments("--remote-allow-origins=*");
          driver = new ChromeDriver(options);

              driver.manage().window().maximize();

              driver.get("https://www.geeksforgeeks.org/");
              test.log(Status.PASS, "Navigated to geeks");
              try
              {
              driver.findElement(By.xpath("//input[@class='ant-input ant-input-
lgaaa']")).sendKeys("testing");
              test.log(Status.PASS, "Successfully entered the search text");
              }
              catch(Exception e)
              {
                 test.log(Status.FAIL, "search text is not entered");
                 takeScreenshot("xpath issue");
              }
                        extent.flush();
                        driver.quit();
                        setUp();
                        sendmail();
   }

   public static void sendmail()
   {
         final String user="manokaran.sr@gmail.com";//change accordingly
         final String password="srivaishu";//change accordingly
         String to="saianiruthan@gmail.com";//change accordingly

         //Get the session object
         Properties props = new Properties();
```

```
            props.setProperty("mail.transport.protocol", "smtp");
            props.setProperty("mail.host", "smtp.gmail.com");
            props.put("mail.smtp.auth", "true");
            props.put("mail.smtp.port", "465");
            props.put("mail.debug", "true");
            props.put("mail.smtp.socketFactory.port", "465");
            props.put("mail.smtp.socketFactory.class","javax.net.ssl.SSLSocketFactory");
            props.put("mail.smtp.socketFactory.fallback", "false");
            Session session = Session.getDefaultInstance(props,
            new javax.mail.Authenticator() {
              protected PasswordAuthentication getPasswordAuthentication() {
              return new PasswordAuthentication(user,password);
               }
             });

             //Compose the message
              try {

                    Transport transport = session.getTransport();
                     InternetAddress addressFrom = new InternetAddress(user);
               MimeMessage message = new MimeMessage(session);
               message.setFrom(new InternetAddress(user));
               message.addRecipient(Message.RecipientType.TO,new InternetAddress(to));
               message.setSender(addressFrom);
               message.setSubject("javatpoint");
               message.setText("This is simple program of sending email using JavaMail API");

              //send the message
               transport.connect();
               Transport.send(message);
               transport.close();


               System.out.println("message sent successfully...");

               } catch (MessagingException e) {e.printStackTrace();}
           }


public static void setUp()
{

        String dateName = new SimpleDateFormat("yyyy"+"-"+"MM"+"-"+"dd"+"-
"+"hh"+"-"+"mm"+"-"+"ss").format(new Date());
        foldername = "Naruto_"+dateName;
        File dir1 = new File(System.getProperty("user.dir")+"\\TestReport\\"+foldername);
//Specify the Folder name here
        dir1.mkdir(); //Creates the folder with the above specified name
        folderPath = dir1.getAbsolutePath();
        filename = "Naruto_"+dateName+".html";
```

```
    htmlReporter = new ExtentHtmlReporter(folderPath+"/"+filename);
    extent = new ExtentReports();
    extent.attachReporter(htmlReporter);
    extent.setSystemInfo("Project", "Mobile testing");
    extent.setSystemInfo("QA Tester", "Naruto");
    extent.setSystemInfo("Environment", "Test FaceBook");
    extent.setSystemInfo("Framework", "NULL");
    extent.setSystemInfo("Organization", "Group31");

    htmlReporter.config().setChartVisibilityOnOpen(true);
    htmlReporter.config().setDocumentTitle("Test Execution Summary Report");
    htmlReporter.config().setReportName("FB Reports");
    htmlReporter.config().setTestViewChartLocation(ChartLocation.TOP);
    htmlReporter.config().setTheme(Theme.STANDARD);
}

//Capture Screenshot and save it in saved location
public static String getScreenshot(WebDriver driver, String screenshotName) throws
Exception {
        String dateName = new SimpleDateFormat("yyyyMMddhhmmss").format(new
Date());
        TakesScreenshot ts = (TakesScreenshot) driver;
        File source = ts.getScreenshotAs(OutputType.FILE);
        //pass the relative path in the html report
        String htmlrelativepath = "../"+foldername +
"/"+screenshotName+"_"+dateName+".png";
        //to move the screenshot taken to the report folder
        String destination = folderPath+ "/"+screenshotName+"_"+dateName+".png";
        File finalDestination = new File(destination);
        FileUtils.copyFile(source, finalDestination);
        return htmlrelativepath;
}

//take screenshot
public static void takeScreenshot(String stepname) throws Exception
{
        String screenshotpath = getScreenshot(driver,stepname);
        test.addScreenCaptureFromPath(screenshotpath);
}

                public static WebDriver driver;
                public static int count;
                public static void launch(String url){

        System.setProperty("webdriver.chrome.driver","./driver/chromedriver.exe");
                        driver = new ChromeDriver();
                        driver.manage().timeouts().implicitlyWait(25,TimeUnit.SECONDS);
                        driver.manage().window().maximize();
                        driver.get(url);}
          public static void Wait(int Time) throws InterruptedException{
```

```
                Thread.sleep(Time);}
        public static void clickBtn(WebElement Element){
                Element.click();}
        public static void sendkeys(WebElement element, String value){
                element.clear();
                element.sendKeys(value);}
        public static WebElement waitTillElementlickable(WebDriver driver,WebElement
webElement,int seconds, String Command)
                {
                        try
                          {
                                WebDriverWait wait = new WebDriverWait(driver,
Duration.ofSeconds(seconds));
                                WebElement a
=wait.until(ExpectedConditions.elementToBeClickable(webElement));
                                System.out.println(Command);
                                return a;
                               }
                        catch (Exception e)
                          {
                                System.out.println("unable to wait till elemn=ent is clicked");
                                //status = false;
                                }
                        finally
                          {

driver.manage().timeouts().implicitlyWait(20,TimeUnit.SECONDS);
                          }
                        return webElement;
                          }
        public static WebElement waitTillElementvisible(WebDriver driver,WebElement
webElement1,int seconds, String Command)
        {
                try
                {
                        WebDriverWait wait = new WebDriverWait(driver,
Duration.ofSeconds(seconds));
                        WebElement b =
wait.until(ExpectedConditions.visibilityOf(webElement1));
                        System.out.println(Command);
                        return b;
                        }
                catch (Exception e)
                {
                        System.out.println("unable to wait till elemn=ent is clicked");//status =
false;
                        }
                finally
                {
                        driver.manage().timeouts().implicitlyWait(20,TimeUnit.SECONDS);
```

```
                }
                return webElement1;
        }
        public static void getText(WebElement element)
        {
                List<WebElement> elements = driver.findElements(By.xpath("//div[@class =
'hmenu-item hmenu-title ']"));
                System.out.println(elements.size());
                for (int i = 0; i < elements.size(); i++)
                        //Formatter formatter = new Formatter();
                {
                        WebElement elements1 = elements.get(i);
                        String text = elements1.getText();
                        count=count+1;


                        ArrayList<String> list=new
ArrayList<String>();//(Arrays.asList(input.split(";"))) ;
                        list.add(text);
                        list.remove("Shop By Department");
//              list.forEach(System.out::println);
                }
        }
}
```

6.2 WEB APP AUTOMATION TESTING

```
package testpack;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import com.aventstack.extentreports.Status;

public class Automation extends FacebookAutomation
{
        public static WebDriver driver;
        public static void main(String[] args)throws Exception
        {
         test = extent.createTest("Automation");
         System.setProperty("webdriver.chrome.driver","C:\\Users\\Gopinath\\eclipse-
workspace\\test11111\\chrome driver\\chromedriver.exe");
    ChromeOptions options = new ChromeOptions();
    options.addArguments("--remote-allow-origins=*");
    driver = new ChromeDriver(options);
    driver.manage().window().maximize();
    driver.get("https://sprightly-cascaron-28998e.netlify.app/");
    test.log(Status.PASS, "Navigated to geeks");
    try
```

```
    {
        Thread.sleep(5000);
      driver.findElement(By.xpath("//select//option[text()='INR']")).click();
      test.log(Status.PASS, "Successfully entered the search text");
    }
    catch(Exception e)
    {
        test.log(Status.FAIL, "search text is not entered");
        takeScreenshot("xpath issue");
    }
     driver.close();
        }
}
```

6.3 MOBILE APP TESTING

```
package testpack;

import java.net.URL;

import org.openqa.selenium.By;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.remote.DesiredCapabilities;

import org.openqa.selenium.remote.RemoteWebDriver;

import org.testng.Assert;

import io.appium.java_client.android.AndroidDriver;

public class MobileAppTest {

public static AndroidDriver driver1;

public static void main(String[] args) throws Exception {

//set up desired capabilities

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setCapability("deviceName", "Naruto");

capabilities.setCapability("platformName", "Android");

capabilities.setCapability("platformVersion", "12.0");

capabilities.setCapability("udid", "d6301c81");
```

```
capabilities.setCapability("appPackage", "com.facebook.katana");

capabilities.setCapability("appActivity", "com.facebook.katana.LoginActivity");

//Create a new RemoteWebDriver

RemoteWebDriver driver = new RemoteWebDriver(new
URL("http://127.0.0.1:4723/wd/hub"), capabilities);

//Enter the username and password

WebElement userName =
driver1.findElement(By.id("com.facebook.katana:id/login_username"));

userName.sendKeys("username");

WebElement password =
driver1.findElement(By.id("com.facebook.katana:idlogin_password"));

password.sendKeys("password");

//Click on the Login button

WebElement loginButton =
driver1.findElement(By.id("com.facebook.katana:id/login_login"));

loginButton.click();

//Verify the login was successful

WebElement homeButton =
driver1.findElement(By.id("com.facebook.katana:id/home_fragment_title"));

Assert.assertTrue(homeButton.isDisplayed());

//Close the driver

driver.quit();

}

}
```

6.4 REPORT AND EMAIL

```
package testEmail;

import java.util.Properties;

import javax.mail.*;

import javax.mail.internet.*;

public class testEm {

public static void main(String[] args) {

final String user="manokaran.sr@gmail.com";//change accordingly

final String password="srivaishu";//change accordingly

String to="saianiruthan@gmail.com";//change accordingly

//Get the session object

Properties props = new Properties();

props.setProperty("mail.transport.protocol", "smtp");

props.setProperty("mail.host", "smtp.gmail.com");

props.put("mail.smtp.auth", "true");

props.put("mail.smtp.port", "465");

props.put("mail.debug", "true");

props.put("mail.smtp.socketFactory.port", "465");

props.put("mail.smtp.socketFactory.class","javax.net.ssl.SSLSocketFactory");

props.put("mail.smtp.socketFactory.fallback", "false");

Session session = Session.getDefaultInstance(props,

new javax.mail.Authenticator() {

protected PasswordAuthentication getPasswordAuthentication() {

return new PasswordAuthentication(user,password); }});

//Compose the message
```

```
try {
Transport transport = session.getTransport();
InternetAddress addressFrom = new InternetAddress(user);
MimeMessage message = new MimeMessage(session);
message.setFrom(new InternetAddress(user));
message.addRecipient(Message.RecipientType.TO,new InternetAddress(to));
message.setSender(addressFrom);
message.setSubject("javatpoint");
message.setText("This is simple program of sending email using JavaMail API");
//send the message
transport.connect();
Transport.send(message);
transport.close();
System.out.println("message sent successfully...");
} catch (MessagingException e) {e.printStackTrace();
}
}
}
```

CHAPTER 7

7. CONCLUSION

Selenium and Appium are powerful tools for automating mobile application testing. They provide a range of features and functionalities that can help in testing mobile applications effectively. Mobile application automation testing with Selenium and Appium requires expertise in programming languages such as Java, Python, and JavaScript. Automation testing can help in reducing the time and effort required for manual testing, improving the accuracy and reliability of the testing process, and providing faster feedback on the quality of the mobile application. The success of automation testing depends on careful planning, designing, and execution of the test cases. The testing process should be integrated with the software development life cycle to ensure that defects are identified and addressed early in the development process. It is important to maintain and update the test scripts and frameworks regularly to keep up with changes in the mobile application and its environment. In conclusion, mobile application automation testing using Selenium and Appium is an essential part of the mobile application development process. It helps in ensuring the quality and reliability of the application and can save time and effort in the long run. However, it requires expertise and careful planning to be successful.

7.1 FUTURE ENHANCEMENT

There are several potential future enhancements that can be considered to further improve the testing process. Some of these include:

- Integration with continuous integration and continuous delivery (CI/CD) pipelines: By integrating the automated testing process with CI/CD pipelines, the testing process can be automated and run more frequently, leading to faster feedback and improved quality.

- Use of AI and machine learning techniques: AI and machine learning can be used to enhance the test cases and scripts by identifying patterns in the data and optimizing the testing process.

- Mobile device cloud testing: Mobile device cloud testing allows testing on a variety of devices and platforms simultaneously, reducing the time and cost of testing.

- Load testing: Load testing can be added to the testing process to ensure the mobile application can handle large volumes of users and transactions.

- Cross-platform testing: Testing on multiple platforms such as iOS, Android, and Windows can be added to the testing process to ensure the application works seamlessly across all platforms.

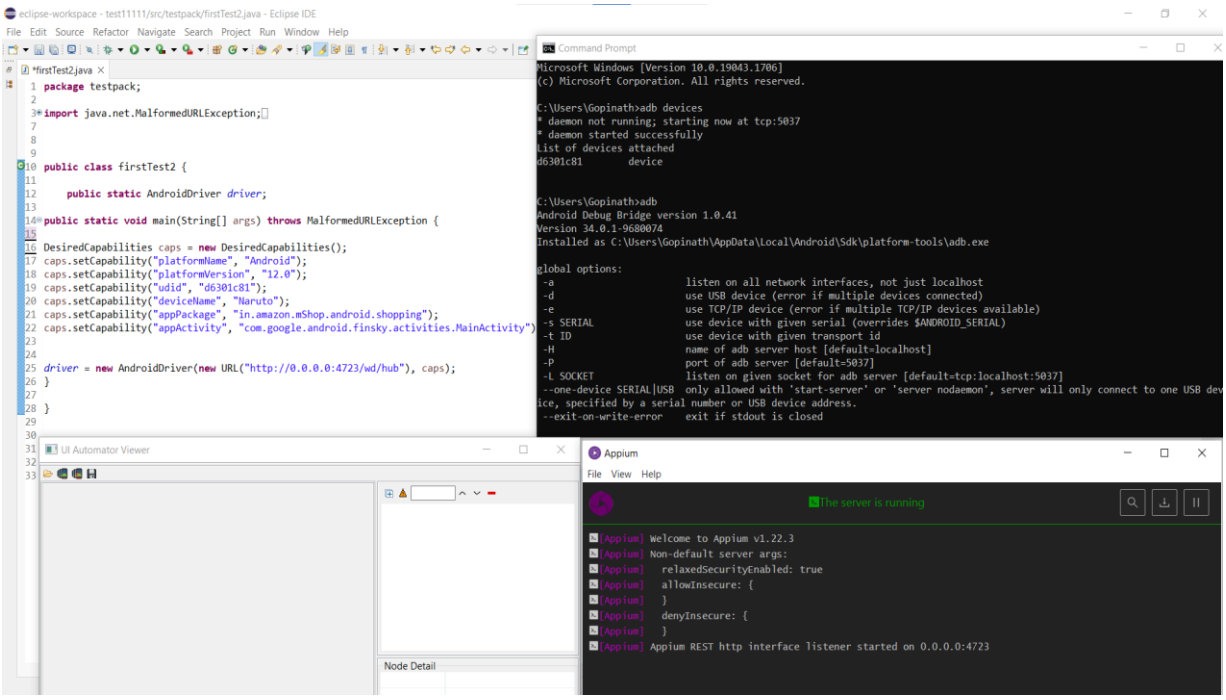
- Accessibility testing: Accessibility testing can be added to the testing process to ensure the mobile application is accessible to users with disabilities.

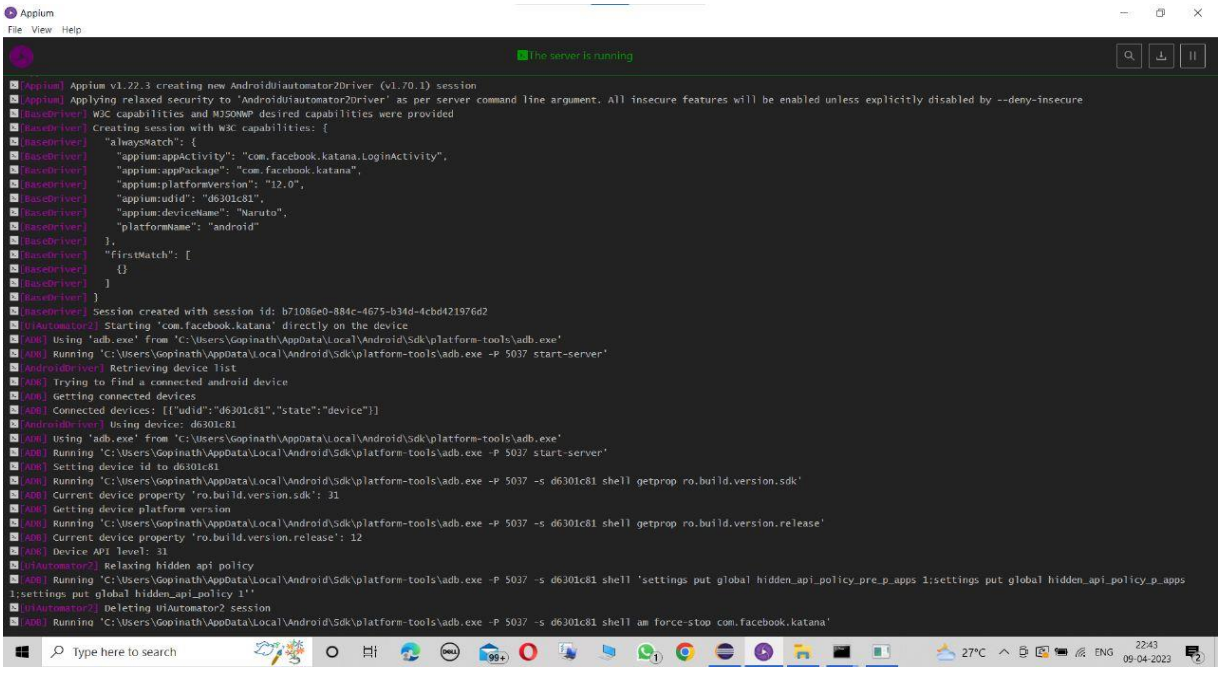
In conclusion, web application and mobile application automation testing using Selenium and Appium is a continuous process that can be improved by integrating it with other technologies such as CI/CD pipelines, machine learning, mobile device cloud testing, load testing, cross-platform testing, and accessibility testing.

CHAPTER 8

8.APPENDIX

8.1 SCREENSHOTS





BIBLIOGRAPHY AND REFERENCE

1. A SURVEY ON SELENIUM AUTOMATION TESTING TOOL
   Publishers Volume No: 9
   Issue No: 2320-2882
   Year of Publication: 4 April 2021

2. SELENIUM FRAMEWORK FOR WEB AUTOMATION TESTING
   Publishers Volume No: 15
   Issue No: 2406-8535
   Year of Publication:3 March 2020

3. SELENIUM AUTOMATION TESTING TOOL
   Publishers Volume No: 11
   Issue No: 2278-0181
   Year of Publication:6 Dec 2017

4. Mobile Application Testing, Smartbear.com, accessed on 23 May 2014, URLhttp://en.m.wikipedia.org/wiki/Mobile_application_testing.

5. Appium – Automation for Apps, accessed on 23 Sept2015 URL http://appium.io.

6. Appium writing test with Appium, accessed on 23 Sept2015, URL http://docs.saucelabs.com/tutorials/appium/

7. Android-UI testing tutorial, accessed on 25 Sept 2015, URL http://www.tutorialspoint.com/android/android_ui_testing.html

8. Testing Support Library, accessed on 2 Oct 2015, URLhttps://developer.android.com/tools/testing-support-library/index.html

9. Test Automation, accessed on 5 Oct 2015, URLhttps://en.wikipedia.org/wiki/Test_automation

10. Testing, A.U., UI Testing, accessed on 2014, URLhttp:/developer.android.com/tools/testing/testing_ui.htm

11. Monkey Talk, accessed on 15 Aug 2014, URLhttps://prativas.wordpress.com/using-monkeytalk-in-androidstudio/

12. Ranorex, accessed on 2104, URLhttp://www.ranorex.com/support/user-guide20/instrumentation wizard/android.html

13. Robotium- the world's leading android test automation framework, accessed on 2 Oct 2015, URLhttp://code.google.com/p/robotium.

14. Appium Essentials, accessed on 7 Oct 2105, URLhttp://www.slideshare.net/Products123/appium-ssentials-sample-chapter

15. Tushar Pradhan," Mobile Application Testing", Mobility-Whitepaper-Mobile Application Testing-1012-1.pdf, 28 Sept 2015.

16. Robotium Tech, accessed on 7 Sept 2015, URLhttp://robotium.com.

17. Robotium- the world's leading android test automation framework, accessed on 2 Oct 2015, URLhttp://code.google.com/p/robotium.

18. Appium server blogs http://appium.io/docs/en/advanced-concepts/grid/

19. Paul C. Jorgensen, "Software Testing A Craftsman's Approach", 4 the edition, Taylor & Francis group, 2014.

20. Guilherme de ClevaFarto, Andre Takeshi Endo, "Evaluating the Model-Based Testing Approach in the context of Mobile Applications", Electronic notes in Theoretical computer science 314, pp. 3-21, 2015.

21. Kolawa, Adam; Huizinga, Dorota, "Automated DefectPrevention: Best Practices in Software Management",Wiley-IEEE Computer Society Press,ISBN 0-470-04212-5, pp. 74, 2007.