13 : Mobile Test Automation

IT6206 – Software Quality Assurance

Level III - Semester 6

Overview

- In this section, you will be introduced to the concepts of Mobile Test Automation.
- Mobile applications are an integral part of our daily lives, tailored to enhance user experiences and provide convenient solutions to a wide range of human requirements.
- It is important to identify optimal testing strategies to maintain the balance between efficiency, reliability, and precision of mobile applications.
- Throughout the years, many mobile application testing tools and frameworks such as Appium have been introduced to ease the process of testing native applications, browser-based applications as well as hybrid mobile applications.

Intended Learning Outcomes

At the end of this lesson, you will be able to;

- Understand the definition, examples, and architectures of mobile applications.
- Understand and differentiate between three main types of mobile applications, native, browser-based, and hybrid.
- Understand the importance of defining a test strategy and the importance of a testable architecture for a mobile application.
- Learn about the fundamental concepts of mobile application testing.
- Learn about tools and frameworks that are widely being used for mobile app testing.
- Learn about Appium architecture, Advantages of using Appium, as well as references to installing and working with Appium.

List of sub topics

13.1 Introduction to Mobile Applications and Architectures

- 13.1.1 What are Mobile Applications?
- 13.1.2 Examples of Mobile Applications
- 13.1.3 Mobile Application Architectures

13.2 Types of Mobile Applications

- 13.2.1 Native Applications
- 13.2.2 Browser-based Applications
- 13.2.3 Hybrid Applications

13.3 Test Strategy for Mobile Applications

- 13.3.1 Defining a Test Strategy
- 13.3.2 Testable Architecture
- 13.3.3 The decoupling approach

List of sub topics

13.4 Mobile Application Testing Fundamentals

13.4.1 Types of tests in Android

13.4.2 Types of tests for iOS applications

13.5 Mobile Testing Tools and Frameworks

13.5.1 Mobile Testing Tools and Frameworks - Examples

13.6 Appium Architecture

13.6.1 Appium Architecture - Introduction

13.6.2 Advantages of using Appium

13.7 Native Type Mobile Application Testing using Appium

List of sub topics

13.7 Native Type Mobile Application Testing using Appium

13.7.1 Appium in Android

13.7.2 Appium in iOS

13.7.3 Prerequisites to use Appium

13.7.4 Writing a test in Appium

13.1 Introduction to Mobile Application and Architectures

13.1.1 What are Mobile Applications?

- Mobile applications, which are also known as mobile apps, have become an integral part of our daily lives in this digital age.
- They are software applications, specifically designed for use on smartphones and tablets.
- They are tailored to enhance user experiences and provide convenient solutions to a wide range of needs while facilitating efficient communication, and easy access of information.

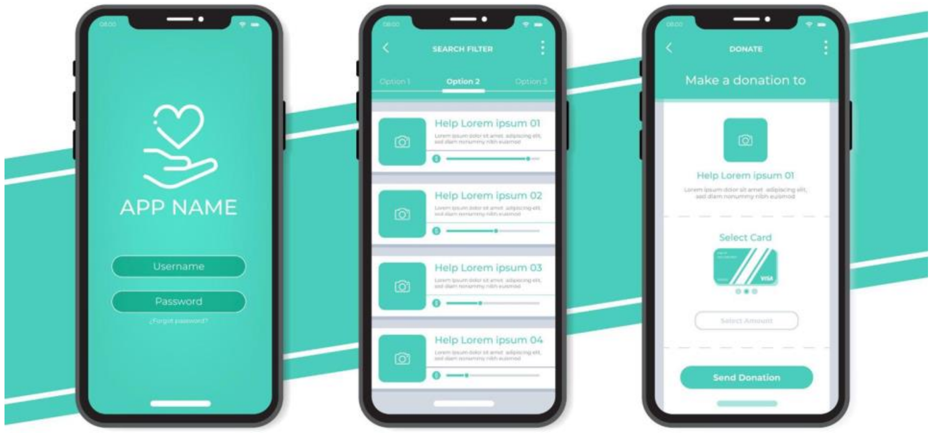


Image Source: "https://www.freepik.com"

13.1.2 Examples of Mobile Applications

Mobile Applications can be categorized into various types based on their purpose:

- Utility Apps: These apps serve practical purposes and aid users in their day-to-day activities. Examples: Weather forecasts apps, note-taking apps, financial management apps, language translation apps, calendar apps etc.

- Social Networking Apps: These apps allow users to create profiles, interact with friends, share content, and stay updated with the latest news and trends. Examples: Facebook, Instagram, Twitter, LinkedIn etc.

13.1.2 Examples of Mobile Applications cont.

- Entertainment Apps: With entertainment apps, we can access an abundance of multimedia content, including music, videos, movies, and games. Examples: Streaming platforms, gaming apps, music players, etc.

- E-commerce Apps: E-commerce apps allow users to browse products, make purchases, track orders, and receive personalized recommendations, by making shopping a convenient and immersive experience.

- Productivity Apps: These apps enable users to manage their time, tasks, and projects effectively. They encompass tools for document editing, project management, collaboration, and remote working, empowering individuals and teams to achieve their goals.

13.1.2 Examples of Mobile Applications cont.

- Health and Fitness Apps: Health-conscious individuals can benefit from mobile applications that focus on fitness tracking, workout routines, nutrition planning, and meditation guidance. These apps serve as virtual trainers, helping users maintain a healthy lifestyle and track their progress.

- Travel and Navigation Apps: When exploring new places or navigating through familiar ones, travel and navigation apps come to our rescue. They provide maps, real-time directions, transportation options, and information about nearby landmarks, ensuring smooth and hassle-free journeys.

13.1.3 Mobile Application Architectures

- Mobile architecture is a set of rules, techniques, and patterns on how to develop a mobile application.
- A mobile application without a proper architecture is harder to be implemented, tested, and maintained.
- Types of devices, types of applications, mobile platform, CPU load, screen time, etc. have to be considered when selecting an appropriate mobile app architecture.
- A good mobile app architecture adheres to the software development principles, with independent layers and easy scaling and troubleshooting capabilities.

13.1.3 Mobile Application Architectures cont.

- The most popular mobile app architecture is represented with 3 layers.
  - Presentation Layer
    - This layer is made up of two main components, the User Interface (UI) and the User Experience (UX). UI focuses on the design aspects of the application such as the colour, fonts, and alignments. UX manages the way a user is interacting with the application according to his/her requirements and preferences.
  - Business Layer
    - This layer represents the logic and the rules for the data exchange and the operation of the application. Within this layer, several tasks such as data validation, exception management, security etc. take place.

13.1.3 Mobile Application Architectures cont.

- Data Layer
  - The layer works on the data safety and maintenance of an application. It consists of data access components, service tools, network communication, routing, and utilities.

- Let’s discuss further about mobile application architectures in the upcoming sections.

13.2 Types of Mobile Applications

13.2.1 Introduction

- Mobile applications can be divided into three major categories as,
    - Native applications
    - Browser-based applications (Mobile web applications)
    - Hybrid applications

- In this section, we will be discussing about the key characteristics and differences of each of these categories.

13.2.2 Native Applications

- A native App is an app developed for a particular mobile device or a platform such as Android, iOS, or Windows.
- Android apps are usually written in Java while the iOS apps are written with tools such as Swift, and AppCode.
- Native apps are tied to the mobile operating systems that they are developed for. As a result, can not be run on any other operating system.
- They can run in both the online mode as well as the offline mode.
- Android mobile app architecture focuses on maintaining architectures with independent layers while the iOS mobile app architecture uses the Model-View-Controller (MVC) architecture more often.

13.2.3 Browser-based Applications

- These apps can be accessed over a mobile browser such as Safari or Chrome. They are often implemented using technologies such as HTML, CSS, and JavaScript.
- These apps can only be accessed with a valid network and served from a server without being stored offline in a mobile device.
- Responsive web designs enhances the user experience when using these kind of applications.
- Since browser-based applications are compatible with different platforms (Android, iOS, Windows), they are easier to be implemented and maintained when compared with native applications.

13.2.3 Browser-based Applications cont.

- However, these applications might lack in native device specific features such as GPS and cameras.

Image Source: https://www.creativefaze.com/mobile-web-apps-development-2018

13.2.4 Hybrid Applications

- Hybrid applications are mainly developed with web technologies such as HTML, CSS, javaScript, but run inside a native container, providing a feel of a native app.
- The web-to-native abstraction layers in such apps enable access to device specific capabilities (GPS, Camera), which are usually not accessible in web apps.
- Hybrid apps allows us to have one code base for all mobile operating systems.
- If there’s already an existing web page in existence for a specific company, it would be easier for then the build a hybrid app as a wrapper over the web age.

13.3 Test Strategy for Mobile Applications

13.3.1 Defining a test strategy

- An ideal way of testing would be to consider every line of code in the application that is being implemented.

- But it is clear that this method is not realistic due to very low efficiency and higher costs.

- Therefore, it is important to identify an optimal testing strategy to maintain the balance between the characteristics such as efficiency, reliability, and precision.

13.3.1 Defining a test strategy cont.

- The more a test environment is similar to a real device where the application is installed, we can expect better performance through the testing process.

- Tests with high-fidelity run on emulated devices or the physical device itself. Tests with low-fidelity might run on a local workstation's JVM.

- Even if some tests are correctly designed and implemented, there could be some unexpected errors which cause the tests to fail. The tests that do not pass 100% of the time are called flaky tests.

13.3.2 Testable Architecture

- Having a proper testable architecture is important to apply testing to different parts of the application in isolation. It improves the readability, maintainability, reusability, and the scalability of the code.

- An architecture which is not testable leads to disadvantages such as,
  - Bigger, slower. more flaky tests - Some classes that can not be unit-tested might have to be covered by bigger UI or Integration tests.
  - As the bigger tests are slower, it becomes impossible to test all the scenarios and possible states of an app.

13.3.3 The decoupling approach

- Decoupling is the practice of extracting a part of a function, class, or a module from the rest of the app for more efficient and easy testing.
- Common decoupling techniques include,
    - Splitting the app into layers such as presentation, domain, and data (or even splitting into modules, as one module for each feature.
    - Make dependencies easy to replace.
    - Avoid direct framework dependencies in classes with business logic.
    - For more details, you can refer the following article. https://developer.android.com/training/testing/fundamentals

13.4 Mobile Application Testing Fundamentals

13.4.1 Types of tests in Android

There are many classifications for android based tests according to characteristics such as subject, scope etc.

- Tests depending on the subject
    - Functional Testing
    - Performance Testing
    - Accessibility Testing
    - Compatibility testing

- Tests depending on the scope (degree of isolation)
    - Unit testing
    - End-to-end testing
    - Medium Testing (to check the integration between two or more units)

13.4.1 Types of tests in Android cont.

Tests in Android can be classified on whether they are instrumented tests or local tests as well.

- Instrumented Tests- These tests are executed on a physical Android device or an emulator. The app is installed along with a test app that injects commands and reads the state.

- Local Tests- These tests executes on the machine or a server where the app is being implemented. They are also called host-side tests.

13.4.2 Types of tests for iOS applications

iOS application testing can also be classified into different types. Following is an example.

- Manual Testing using the device
  - System testing - To check whether the different components of the application works together.
  - iOS UI testing - To test the UI/UX components of the system such as Inputs, Hard keys, Soft Keys, Screen, etc.
  - Security testing - To discover the vulnerabilities, threats, and risks of the system.
  - Field testing - To verify the behavior of the application on the mobile device’s data network.

13.4.2 Types of tests for iOS applications cont.

- Manual Testing using the emulator
  - Unit Testing - To check whether a given module (a single component) is functioning as expected.
  - Integration testing - To check whether there are any issues related to various integration points of the system.
  - UI Testing

- iOS Automation Testing
  - Regression Testing - To ensure that an application performs in the expected way even after some changes are implemented.
  - Built Verification Test (BVT) - Usually executed as an automated suite on the new build release to ensure that the build is testable and can be sent to further testing.

13.4.2 Types of tests for iOS applications cont.

- iOS Automation Testing cont.
  - Compatibility Testing - To check whether the app being implemented supports different versions and models of iOS devices on the market.
  - Performance Testing - To check the behaviour of the application under different conditions such as heavy data transfers, idle status, communicating states etc.

13.5 Mobile Testing Tools and Frameworks

13.5.1 Mobile Testing Frameworks - Examples

There are many frameworks and tools used for mobile application testing. Some of such frameworks will be discussed in this section.

1. Appium
2. SmartBear TestComplete
3. Katalon
4. Ranorex

[images - with icons of these frameworks]

13.5.2 Mobile Testing Frameworks - Examples cont.

Appium

- Appium is an open-source project which is widely used in test automation in many app platforms including mobile (iOS, Android, Tizen).
- It allows developers and testers to write UI automation code for a platform according to single, unified API.
- Similar to Selenium, Appium allows the testers to write test scripts in multiple programming languages such as Java, JavaScript, PHP, Ruby, Python, and C#.
- Appium allows its users to reuse their source codes for Android and iOS, reducing the time and effort of building tests.

13.5.2 Mobile Testing Frameworks - Examples cont.

<u>SmartBear TestComplete</u>

- TestComplete is a testing tool which has the ability to handle a wide range of test automation requirements specially for the mobile applications.
- It has the scriptless Record and Replay option as well as the keyword-driven tests to facilitate the building of automated UI tests.
- It has more sophisticated features such as object recognition engine with artificial intelligence, automated test reporting and analysis, and data-driven testing (to separate data from test commands for the ease of maintenance).

13.5.2 Mobile Testing Frameworks - Examples cont.

Katalon

- Katalon is a comprehensive quality management platform which provides easy and efficient testing capabilities.
- In mobile testing automation, Katalon supports native apps, mobile web apps, as well as hybrid apps of both Android and iOS platforms.
- It works on real devices as well as emulators with special testing features such as Record and Playback, cloud-based mobile platform integration, cross-environmental execution, etc.

13.5.2 Mobile Testing Frameworks - Examples cont.

Ranorex

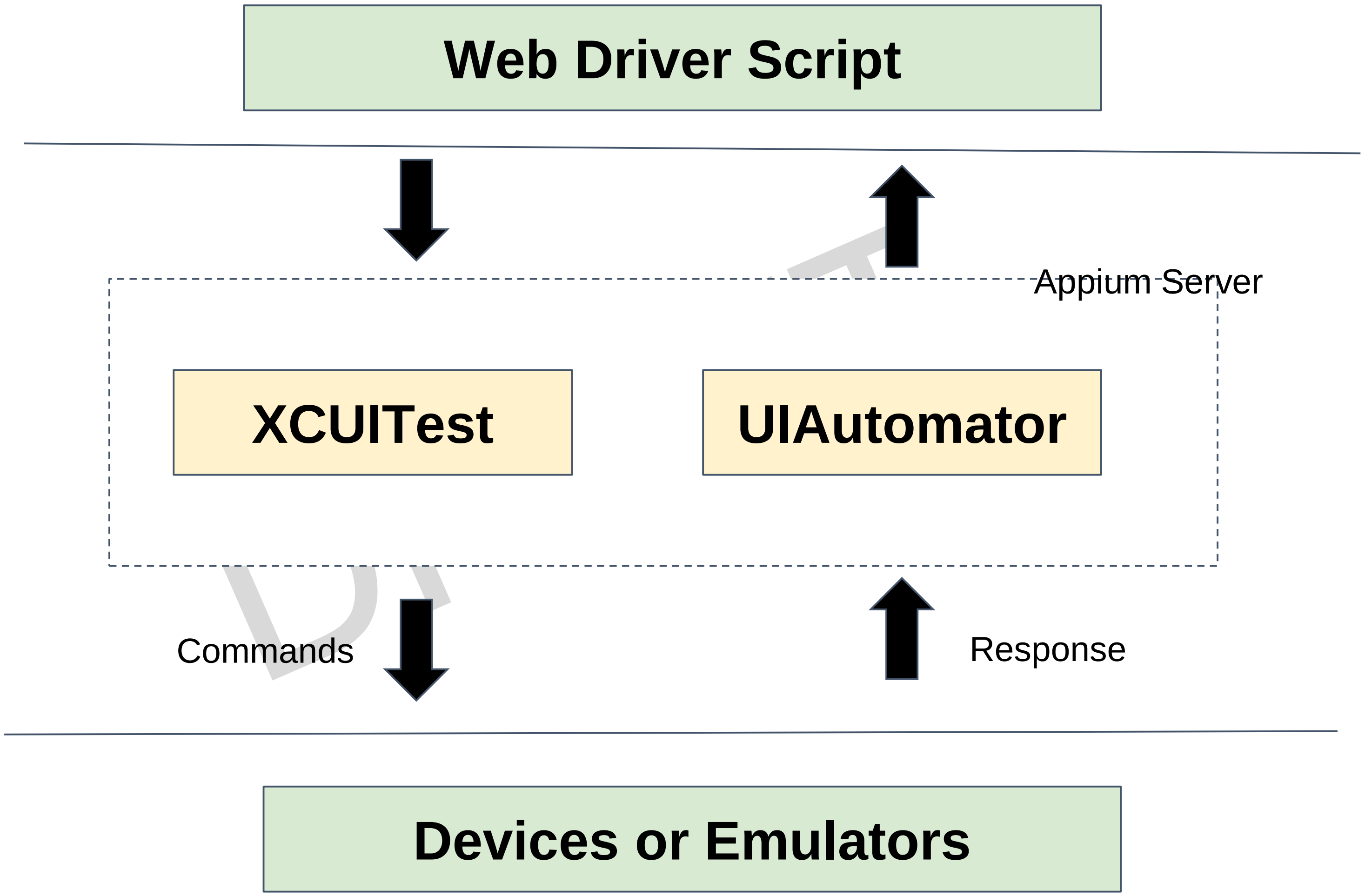
- Ranorex Studio's tools can be used to automate mobile app and mobile web app testing on both real devices and emulators.
- It supports both Android and iOS platforms.
- Ranorex facilitates testing capabilities such as
    - Record and Playback functionality
    - Integrating tests into existing CI environment
    - Ability to check device-specific parameters such as battery, memory, CPU state etc.
    - Ability to simulate real user interactions such as "touch", "swipe", "change orientation" etc.

13.6 Appium Architecture

13.6 Appium Architecture - Introduction

- Appium is a web server written in Node.js.
- Appium **client-server architecture** works in the following manner.
  - Received connection from client and initiates a session
  - Listens for commands issued
  - Executes those commands
  - Returns the command execution status
- Appium server receives a connection from client in the form of a JSON object over HTTP. Once the server receives the details, it creates a session, as specified in JSON and return a session ID.
- This session ID will be maintained until the Appium server is running and all the testings will be performed in the context of the nelly created session.

13.6 Appium Architecture - Introduction cont.



Appium Architecture

13.6 Advantages of using Appium

- Some of the advantages of using Appium are given below.
  - Appium is an open-source tool which is backed by a very active community.
  - Appium supports multiple languages such as Java, JavaScript, Objective C, C#, PHP, Python, Perl, etc.
  - Appium doesn’t force to recompile an app or modify it. Therefore, the same version which a developer plans to submit to the play store or apple store can be tested using Appium.
  - Appium allows the testers to write cross-platform tests as well.
  - Appium uses frameworks such as XCUITest (for iOS apps) and UIAutomator (for Android apps) to communicate command to the devices.

13.7 Native Type Mobile Application Testing with Appium

13.7.1 Appium in Android

- On Android, Appium proxies commands to a **UIAutomator** script running on the device.
- **UIAutomator** is a native UI automation framework of Android that allows the testers to run **Junit** test cases directly into the device using the command line.
- **bootstrap.jar** plays the role of a TCP server and send the test commands to perform the actions on Android device using UIAutomator.

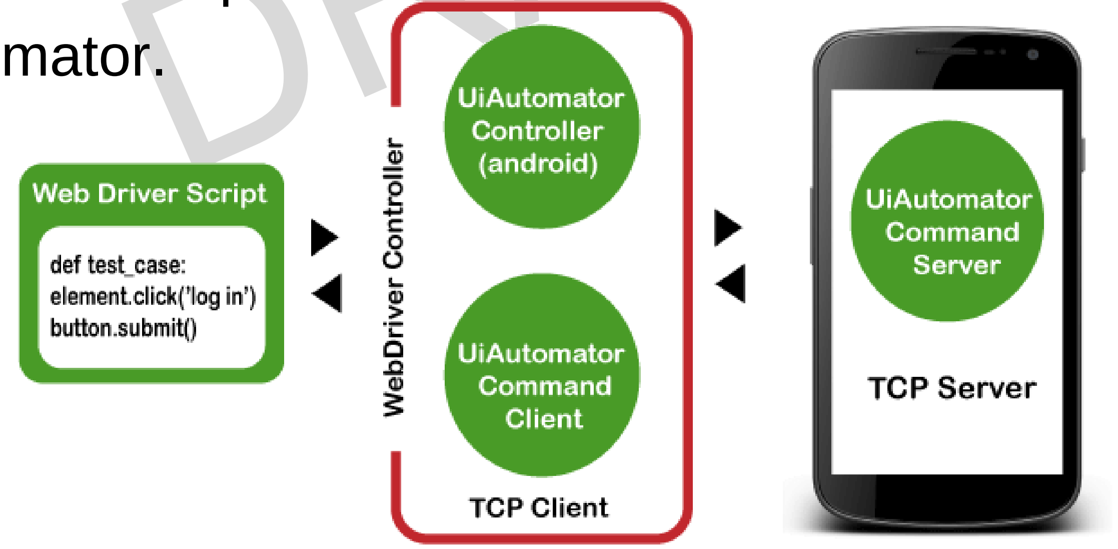


Image Source: https://www.javatpoint.com/appium

13.7.2 Appium in iOS

- On iOS devices, Appium uses Apple's **UIAutomation** API to interact with the UI elements. **UIAutomation** is a JavaScript library provided by Apple which is used by Appium to automate the iOS apps.
- Once a test script is executed, it goes to the Appium server in the form of JSON through an HTTP request. Then the Appium server sends the command to the instruments and the instruments look for the **bootstrap.js** file. Then, the commands are executed in the bootstrap.js file within the iOS instruments' environment.

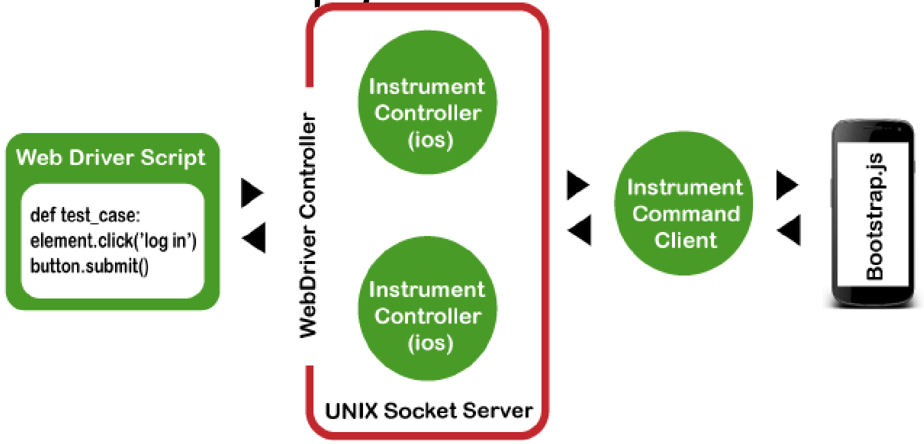


Image Source: https://www.javatpoint.com/appium

13.7.3 Prerequisites to use Appium

- Install **Java** (JDK)
- Install **Android Studio**
- Install additional Android **SDK tools**
- Install **Appium jar** file
- Install **Appium Desktop Client**
- Install **Eclipse IDE** for Java

To learn more about the installation of Appium and setting up Appium environment with necessary dependencies, you can refer to the following web articles.

https://www.javatpoint.com/appium

https://appium.io/docs/en/2.0/quickstart/install/

You can also install **Appium Doctor**, which is an application tool to verify Appium installation.

13.7.4 Writing an Appium Test

- The official Appium Documentation will guide you step-by-step in writing your first Appium test using,
    - JavaScript https://appium.io/docs/en/2.0/quickstart/test-js/
    - Python https://appium.io/docs/en/2.0/quickstart/test-py/
    - Java https://appium.io/docs/en/2.0/quickstart/test-java/ or
    - Ruby https://appium.io/docs/en/2.0/quickstart/test-rb/

- Appium Documentation provides detailed descriptions regarding the Appium Drivers, Plugins, server security, and parameters used to start an Appium session (Capabilities). If you are interested in exploring more on these topics, you can refer to the **"Guides"** section of the documentation.

  https://appium.io/docs/en/2.0/guides/migrating-1-to-2/

References

[1] Mobile Test Automation with Appium - comprehensive guide to build mobile test automation solutions using Appium (2017), Nishant Verma, Packt Publishing Ltd, ISBN 978-1-78728-016-8
[2] Fundamentals of testing Android apps, "https://developer.android.com/training/testing/fundamentals"
[3] Appium Documentation, "http://appium.io/docs/en/2.0/"
[4] SmartBear TestComplete, "https://smartbear.com/product/testcomplete/"
[5] About Katalon platform, "https://docs.katalon.com/docs"

[6] Ranorex, Software Testing Resources, "https://www.ranorex.com/resources/"

[7] Appium Tutorial-Javatpoint, "https://www.javatpoint.com/appium"

[8] Introduction to Appium, "https://www.guru99.com/introduction-to-appium.html"