What is Appium?:

Appium is an **open-source automation tool** for testing **mobile applications**. It uses the same API to **write tests for native, hybrid, and mobile web apps** on **iOS**, **Android**, and **Windows platforms**.

- **Native apps**: Written using iOS or Android SDKs (e.g., Swift, Kotlin)
- **Hybrid apps**: Web apps wrapped in a native shell (like Ionic, Cordova)
- **Mobile web apps**: Web apps accessed via mobile browsers

Key Concepts:

Feature	Description
Cross-platform	One API to test Android and iOS apps
Language Agnostic	Write tests in Java, Python, JavaScript, Ruby, C#, etc.
Uses WebDriver	Built on top of the **Selenium WebDriver** protocol
No App Recompile	Test apps without modifying or recompiling them

Architecture of Appium:

Client (Test Script) <---> Appium Server <---> Device/Emulator

- **Appium Client**: Your test code (Java, Python, etc.)
- **Appium Server**: Node.js server that receives commands via REST API and routes them to platform-specific drivers.
- **Automation Engines**:
  - Android: **UiAutomator2**, **Espresso**
  - iOS: **XCUITest**

Appium Setup:

Prerequisites:

1. **Node.js**
2. **Java JDK (for Android)**
3. **Android SDK** (with ADB)
4. **Xcode** (for iOS)
5. **Appium server**
6. **Device or emulator/simulator**

Installation Steps:

1. Install Appium CLI:

```
npm install -g appium
```

2. Install Appium Inspector (optional GUI):

   [Download Appium Inspector]

3. Install Appium Drivers:

```
appium driver install uiautomator2
appium driver install xcuitest
```

4. Start Appium Server:

```
Appium
```

Writing Your First Test:

Example in Java (Android):

```
DesiredCapabilities caps = new DesiredCapabilities();
caps.setCapability("platformName", "Android");
caps.setCapability("platformVersion", "13.0");
caps.setCapability("deviceName", "Android Emulator");
caps.setCapability("automationName", "UiAutomator2");
caps.setCapability("app", "/path/to/app.apk");

URL url = new URL("http://127.0.0.1:4723/");
AndroidDriver driver = new AndroidDriver(url, caps);
driver.findElement(By.id("com.example:id/button")).click();
```

capabilities.json

```
{
  "appium:deviceName": "emulator-5554",
  "platformName": "Android",
  "appium:platformVersion": "16",
  "appium:automationName": "uiAutomator2",
  "appium:app": "/path/to/app.apk"
}
```

Appium Inspector:

Used to locate elements via **XPath**, **Accessibility ID**, **Class Name**, etc.

- Connect to the Appium server
- Launch your app on the device/emulator
- Click through the app and inspect UI elements

Test Frameworks You Can Use:

- Java: TestNG, JUnit, Cucumber
- Python: pytest, unittest, Behave
- JavaScript: Mocha, Jasmine, WebDriverIO
- C#: NUnit, MSTest

Locator Strategies:

Strategy	Description
id	Unique element ID
accessibilityId	Used for cross-platform
xpath	XML path, flexible but slower
className	UI class name
androidUIAutomator	Android-specific
iOSPredicateString	iOS-specific

Useful Tools:

- **Appium Inspector** – to locate elements
- **ADB (Android Debug Bridge)** – for device control
- **UIAutomatorViewer** – for Android UI inspection
- **Xcode & Simulator** – for iOS testing

Appium Doctor

Appium Doctor is a command-line tool that checks whether all the necessary dependencies for running Appium are properly set up on your machine. It helps you diagnose common environment issues that could prevent Appium from working correctly.

What Appium Doctor Does:

- Check Node.js installation
- Verifies Android SDK & Java (for Android)
- Checks Xcode & Carthage (for iOS, macOS only)

- Validates environment variables like JAVA_HOME, ANDROID_HOME, etc.
- Confirms presence of necessary binaries and tools (e.g., adb, xcodebuild, ideviceinstaller)

How to Install Appium Doctor:

```
npm install -g appium-doctor
```

How to Run It:

```
appium-doctor
```

It will give you a breakdown of what's installed correctly and what's missing or misconfigured. You can also run it with a platform-specific focus:

```
appium-doctor --ios
appium-doctor --android
```

Output Example

```
✔ Node.js is installed

✔ ANDROID_HOME is set to /Users/you/Library/Android/sdk

✖ JAVA_HOME is NOT set

✖ adb could NOT be found
```

Errors will typically be in red, so you can quickly spot and fix them.

Appium Ecosystem:

Tool	Purpose
Appium Desktop	GUI tool for Inspector
Appium Server	Command line server
Appium Inspector	GUI to explore mobile app elements
Appium Doctor	Diagnose environmental issues
WebDriverAgent	iOS testing agent
UiAutomator2	Android automation engine

Advanced Topics:

- **Parallel Testing** (with Appium Grid or cloud services like BrowserStack/Sauce Labs)
- **Page Object Model (POM)** for cleaner test code
- **CI/CD Integration** (e.g., Jenkins, GitHub Actions)
- **Custom Appium Plugins** to extend server behavior
- **Cloud Device Testing** via Sauce Labs, BrowserStack, Bitrise, etc.

Common Issues:

Issue	Solution
"Could not connect to server"	Ensure the Appium server is running
App not installing	Check device compatibility & permissions
Element not found	Use Appium Inspector for accurate locators
Appium crashes	Check logs, verify versions of SDK, Node, and Appium

Resources:

- Official Site
- Appium GitHub
- Appium Documentation
- WebDriver Protocol
- Appium Pro Blog

Waits in Appium

How to Handle Drop-Downs in an Android App

Handling TextBox, CheckBox, and Radio Buttons in an Android App

Vertical Scrolling in an Android App

How to Handle Switches/Toggles in an Android App

How to Handle Progress Bar | Horizontal Scrolling in Android App

How to Handle Alerts and Popups in an Android App

How to Handle Drag and Drop in an Android App

How to Handle an Expandable List

How to Handle Date Picker

Common Android Hardware Keys handling:

AndroidDriver driver = new AndroidDriver(); //AndroidDriver Instance

Function	Code
Back	driver.pressKey(new KeyEvent(AndroidKey.BACK));
Home	driver.pressKey(new KeyEvent(AndroidKey.HOME));
App Switch (Recent Apps)	driver.pressKey(new KeyEvent(AndroidKey.APP_SWITCH));
Enter	driver.pressKey(new KeyEvent(AndroidKey.ENTER));
Volume Up	driver.pressKey(new KeyEvent(AndroidKey.VOLUME_UP));
Volume Down	driver.pressKey(new KeyEvent(AndroidKey.VOLUME_DOWN));
Camera	driver.pressKey(new KeyEvent(AndroidKey.CAMERA));
Power	driver.pressKey(new KeyEvent(AndroidKey.POWER));
Delete (Backspace)	driver.pressKey(new KeyEvent(AndroidKey.DEL));
Menu	driver.pressKey(new KeyEvent(AndroidKey.MENU));
Search	driver.pressKey(new KeyEvent(AndroidKey.SEARCH));

WebDriverIO:

WebDriverIO is a test automation framework based on Node.js. It's used for testing web and mobile applications using the WebDriver protocol or Chrome DevTools.

Prerequisites:

Before installing WebDriverIO, make sure you have the following installed:

- Node.js (v18 or newer recommended): Install from: https://nodejs.org/
    - Verify installation:
      ```
      node -v
      npm -v
      ```
- Code Editor (e.g., VS Code): https://code.visualstudio.com/
- Git (optional, for version control): https://git-scm.com/

Setting Up WebDriverIO From Scratch

1. Create a Project Folder:
    ```
    mkdir wdio-project
    cd wdio-project
    ```

2. Initialize NPM:
    ```
    npm init -y
    ```

3. Install WebDriverIO CLI:
    ```
    npm init wdio@latest .
    ```

Or

```
npm init wdio@latest ./path/to/new/project
```

The wizard will prompt a set questions that guides you through the setup. You can pass a --yes parameter to pick a default set up which will use Mocha with Chrome using the Page Object pattern.

```
npm init wdio@latest . -- --yes
```

Or

Install CLI Manually:

```
npm install @wdio/cli --save-dev
```

Run WDIO Configuration Wizard

```
npx wdio config
```

- The wizard will ask:
    - Where should your tests be launched? → local or cloud (choose "local" for learning)
    - Which framework do you want to use? → Mocha, Jasmine, or Cucumber (e.g., select Mocha)
    - Do you want to use a compiler? → Babel or TypeScript (choose as per your preference; JS doesn't need any)
    - Where are your test specs? → default is fine (./test/specs/**/*.js)
    - Which reporter to use? → spec is fine to start
    - Do you want to add a service? → yes, typically use chromedriver or selenium-standalone
    - Base URL → e.g., http://localhost or your target test URL
    - This will create a wdio.conf.js configuration file.

Write a Simple Test:

- Create a test file:

```
mkdir -p test/specs
```

- In test/specs/example.e2e.js, write:

```
describe('Google Search', () => {
   it('should open Google and check the title', async () => {
      await browser.url('https://www.google.com');
      const title = await browser.getTitle();
      console.log('Title is:', title);
      expect(title).toContain('Google');
   });
});
```

Run the Test:

- npx wdio run wdio.conf.js

If you like to run specific test files you can add a --spec parameter:

- npx wdio run ./wdio.conf.js --spec example.e2e.js

Define suites in your config file and run just the test files defined by in a suite:

- npx wdio run ./wdio.conf.js --suite exampleSuiteName

Async/Await:

- **async** makes a function return a Promise.
- **await** pauses the function execution until the Promise is resolved, then returns the result.

- Example:
```
async function loadData() {
  const data = await getData();
  console.log(data); // "Data loaded"
}
```

Waits in WebDriverIO:
- **waitForClickable()**
- **waitForDisplayed()**
- **waitForEnable()**
- **waitForExit()**
- **waitUntil()**
- **pause()**

Update the Node and packages:
- https://www.npmjs.com/package/npm-check-updates