APPIUM:

Webdriver+Mobile Feature

Appium is an open source, cross-platform test automation tool for native, hybrid and mobile web apps, tested on simulators (iOS), emulators (Android), and real devices (iOS, Android).

Supported Platforms

* iOS
* Android

Why Appium?

1. You don't have to recompile your app or modify it in any way, due to use of standard automation APIs on all platforms.
2. You can use any testing framework.
3. You can write the code in any language Supported by WebDriver
4. If you use Apple's UIAutomation library without Appium you can only write tests using JavaScript and you can only run tests through the Instruments application. Similarly, with Google's UiAutomator you can only write tests in Java. Appium opens up the possibility of true cross-platform native mobile automation.

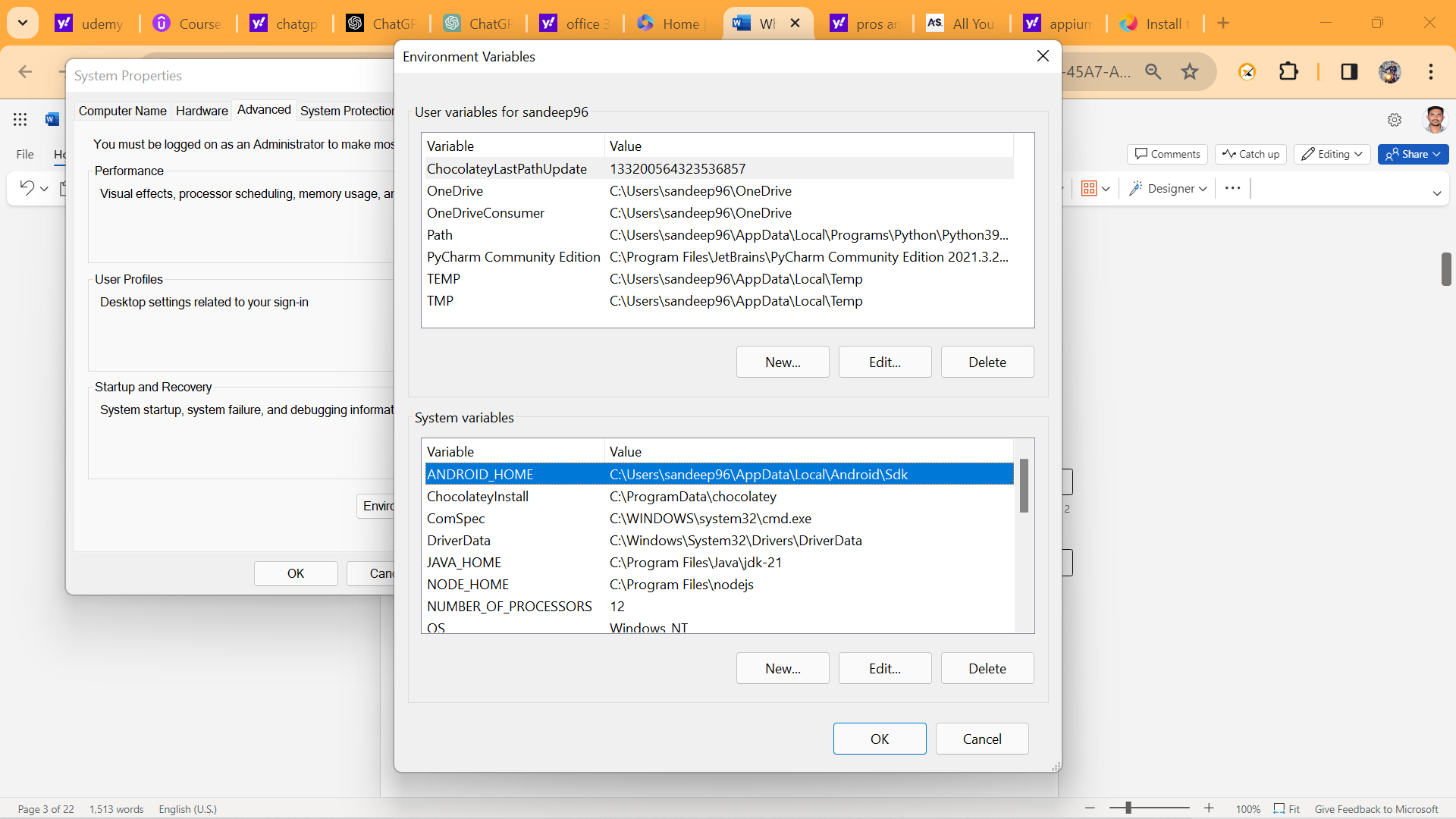
### Advantages

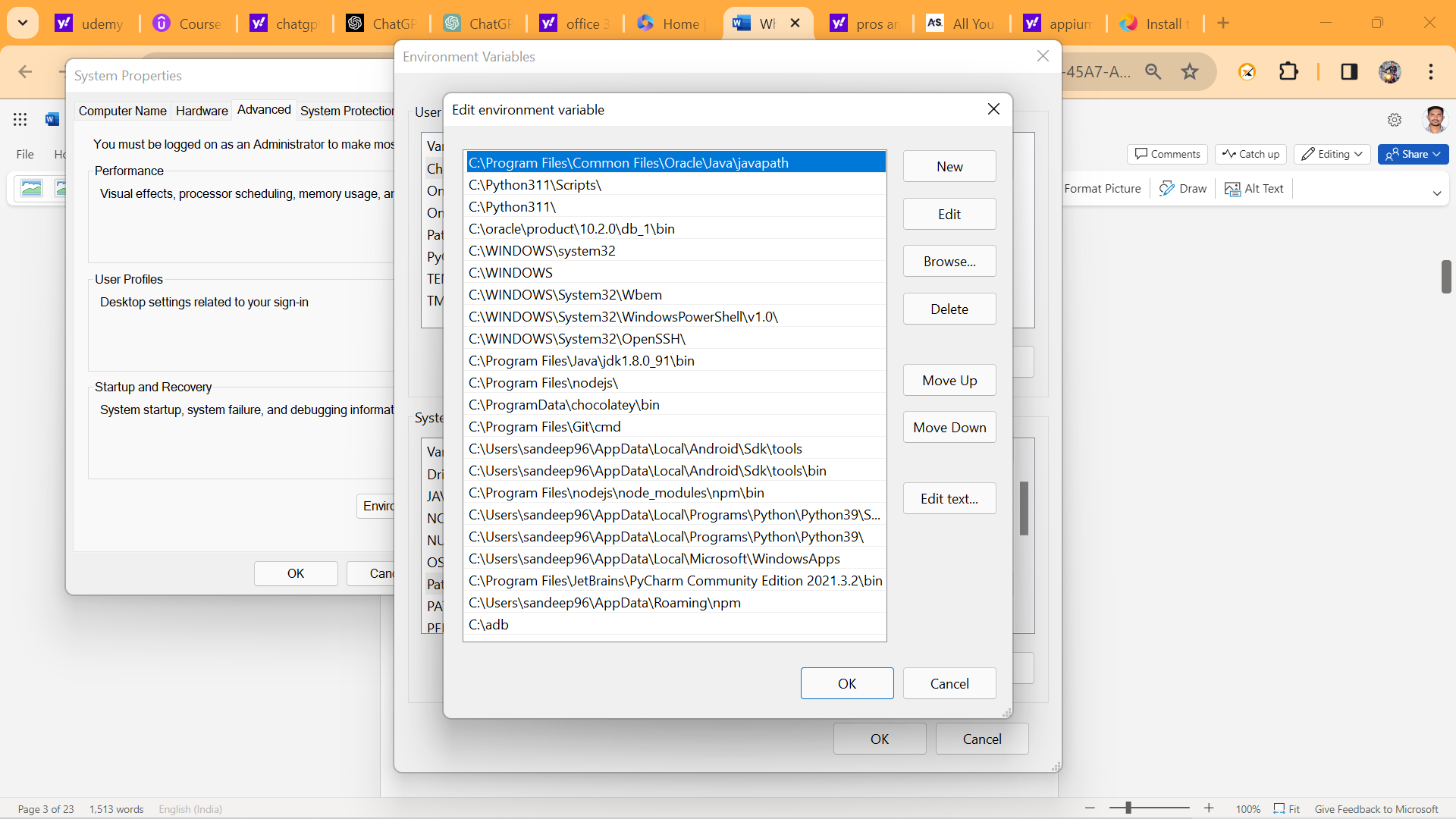
* Appium Framework is a free and open-source framework.
* It provides cross-platform solutions for native and hybrid apps.
* Programming languages like C#, Java, PHP, Python, Ruby are supported.
* App automation is made possible using Appium mobile testing.
* Support simulators, emulators, and real devices concurrently.
* JSON wire protocol supported.
* Independent of mobile device means could be used in any mobile device.
* Excellent support for Android versions above 4.1.

Disadvantages

* Doesn’t support image comparison.
* Appium does not support testing of Android Version lower than 4.2.

Installation process

1. 

We need

1. Android Studio
2. Java
3. Node js

DO NOT SKIP - Important Note on dependencies version compatibility

1.  Node version minimum should be 16. you can check your version with below command

node –v If it is less than 16, please upgrade

2.  This course is taught on latest Appium Server 2.0 version. If you use older Version 1.22, then you will run into issues.

You can check your Appium version with below command appium -v If you have less than 2 version, use below steps to install correctly

npm uninstall -g appium Delete  appium folder in node modules npm install -g appium@next

Starting the appium server we use appium command

1. Appium Java client 8.0.0 has few bugs. So please select the version Greater than 8.0.0 from mavenrepository.com. (Always select latest available)

So, as of now, below is Latest perfect working (Appium & TestNG) Combination which you can have in your POM.xml Project for smoother execution

<dependency>

<groupId>io.appium</groupId>

<artifactId>java-client</artifactId>

<version>9.0.0</version>

</dependency>

<! -- https://mvnrepository.com/artifact/org.testng/testng -->

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.14.3</version>

</dependency>

UIAUTOMATOR2

UiAutomator2 is a UI testing framework introduced by google to facilitate automation on a android device or emulators

In command prompt use the commands like

appium driver list

C:\Users\sv22036>appium driver list

Listing available drivers

appium driver install uiautomator2

It will install the uiautomator2 framework for android devices. We will check this one by using command like

For Appium server connection in eclipse we use

Uiautomator2Options op=new Uiautomator2Options ();

op.setDeviceName(“sandeepDevice”);

op.setApp("C:\\Users\\sandeep96\\Desktop\\GeneralStore\\General-Store.apk");

For android devices we use uiautomator2 framework.

AndroidDriver driver=new AndroidDriver(new URL(“http://127.0.0.1.4723”), op);

Here we will get compile time error and import that exception.

Installation of Appium Inspector

* Go to chrome
* Search Appium download
* Select GitHub releases appium/appium desktop
* Install [Appium-Server-GUI-windows-1.22.3-4.exe](https://github.com/appium/appium-desktop/releases/download/v1.22.3-4/Appium-Server-GUI-windows-1.22.3-4.exe)
* then setup the appium inspector

After successfully setup the appium inspector you must give the desired capabilities like

* App path
* DeviceName
* AutomationName UiAutomator2
* platformName android

Then click on the start session button.

From here onwards actual automation script will start and you must inspect all those elements

Mainly here we will use locators like

* AccessibilityId
* Xpath
* id

And also here we use appiumBy instead of By.

Then creating Base class with util methods and extend it to the child Appium tests.

Base Class:

Public class baseClass

{

@BeforeClass

Public void baseTest()

{

Uiautomator2Options op=new Uiautomator2Options ();

op.setDeviceName(“sandeepDevice”);

Op.setApp(“path”);

AndroidDriver driver=new AndroidDriver(new URL(“http://127.0.0.1.4723”), op);

}

}

Appium Test:

------------------

Public class AppiumTest extends baseClass

{

@Test

Public void appium()

{

Driver.findelement(appiumBy.accessibilityId(“preferences”). click ();

}}

LONG PRESS SCRIPT

-------------------------------

Base Class:

---------------

Public class baseClass

{

@BeforeClass

Public void baseTest()

{

Uiautomator2Options op=new Uiautomator2Options ();

op.setDeviceName(“sandeepDevice”);

Op.setApp(“path”);

AndroidDriver driver=new AndroidDriver(new URL(“http://127.0.0.1.4723”), op);

}

Public void longPress(WebElement ele)

{

((JavascriptExecutor)driver).executeScript(“mobile: LongClickgesture”,ImmutableMap.of(“ElementId”, (RemoteWebElemet)ele).getId(),” duration”,2000));

}

}

Long Press:

---------------

Public class LongPress extends BaseClass

{

@Test

Public void longpress()

{

Driver.findElement(appiumBy.accessibleId(“views”)).click();

Driver.findElement(appiumBy.xpath(“android.widget.TestView[@test=’Expandable Lists’]”)).click();

Driver.findElement(appiumBy.accessibleId(“1.custom Adapter”)). click ();

WebElement ele=((JavascriptExecutor)driver).executeScript(“mobile: LongClickgesture”,ImmutableMap.of(“ElementId”, Driver.findElement(appiumBy.xpath(“android.widget.TestView[@test=People Names’]”));

LongPress(ele);

/\* (RemoteWebElemet)ele).getId(),” duration”,2000)); \*/

}

}

SCROLLING GESTURE:

Base Class:

Public class baseClass

{

@BeforeClass

Public void baseTest()

{

Uiautomator2Options op=new Uiautomator2Options ();

op.setDeviceName(“sandeepDevice”);

Op.setApp(“path”);

AndroidDriver driver=new AndroidDriver(new URL(“http://127.0.0.1.4723”), op);

}

Public void longPress(WebElement ele)

{

((JavascriptExecutor)driver).executeScript(“mobile: LongClickgesture”,ImmutableMap.of(“ElementId”, (RemoteWebElemet)ele).getId(),” duration”,2000));

}

Public void scrollabe()

{

Driver.findElement(AppiumBy.androidUIAutomator(“new UIScrollable(new UISelector()).scrollIntoView(test (\ “webView\”))”);

}

ScrollingTo Element:

----------------------------

Public class Scrolling extends BaseClass

{

@Test

Public void scroll ()

{

Driver.findElement(appiumBy.accessibleId(“views”)). click();

Scrollable ();

/\*Driver.findElement(AppiumBy.androidUIAutomator(“new UIScrollable(new UISelector()).scrollIntoView(test (\ “webView\”))”); \*/

}

}

SWIPE FUNCTIONALITY:

Base Class

Public class baseClass

{

@BeforeClass

Public void baseTest()

{

Uiautomator2Options op=new Uiautomator2Options ();

op.setDeviceName(“sandeepDevice”);

Op.setApp(“path”);

AndroidDriver driver=new AndroidDriver(new URL(“http://127.0.0.1.4723”), op);

}

Public void longPress(WebElement ele)

{

((JavascriptExecutor)driver).executeScript(“mobile: LongClickgesture”,ImmutableMap.of(“ElementId”, (RemoteWebElemet)ele).getId(),” duration”,2000));

}

Public void scrollabe()

{

Driver.findElement(AppiumBy.androidUIAutomator(“new UIScrollable(new UISelector()).scrollIntoView(test (\ “webView\”))”);

}

Public void swipe (WebElement Ele, String direction)

{

((JavascriptExecutor) driver).executeScript("mobile: swipeGesture", ImmutableMap.of(

“ElementId”, (RemoteWebElemet)ele).getId(),  
   
 "direction", "direction",  
 "percent", 0.75  
));

SWIPE FUNCTIONALITY:

-----------------------------------

Public class Swipe extends BaseClass

{

@Test

Public void swipe ()

{

Driver.findElement(appiumBy.accessibleId(“views”)). click();

Driver.findElement(appiumBy.accessibleId(“Gallery”)). click ();

Driver.findElement(appiumBy.xpath(“android.widget.TestView[@test=’1. photos’]”)). click ();

String firstImg=Driver.findElement(appiumBy.xpath(“(android.widget.TestView)[1])”);

Assert.assertEquals(firstImg,” true”);

//swipe

/\*((JavascriptExecutor) driver).executeScript("mobile: swipeGesture", ImmutableMap.of(

“ElementId”, (RemoteWebElemet)ele).getId(),  
   
 "direction", "left",  
 "percent", 0.75  
));\*/

Swipe (firstImg,” left”);

Assert.assertEquals(firstImg,” false”);

DRAG AND DROP:

Base Class

Public class baseClass

{

@BeforeClass

Public void baseTest()

{

Uiautomator2Options op=new Uiautomator2Options ();

op.setDeviceName(“sandeepDevice”);

Op.setApp(“path”);

AndroidDriver driver=new AndroidDriver(new URL(“http://127.0.0.1.4723”), op);

}

Public void longPress(WebElement ele)

{

((JavascriptExecutor)driver).executeScript(“mobile: LongClickgesture”,ImmutableMap.of(“ElementId”, (RemoteWebElemet)ele).getId(),” duration”,2000));

}

Public void scrollabe()

{

Driver.findElement(AppiumBy.androidUIAutomator(“new UIScrollable(new UISelector()).scrollIntoView(test (\ “webView\”))”);

}

Public void swipe (WebElement Ele, String direction)

{

((JavascriptExecutor) driver).executeScript("mobile: swipeGesture", ImmutableMap.of(

“ElementId”, (RemoteWebElemet)ele).getId(),  
   
 "direction", "direction",  
 "percent", 0.75  
));

Drag Functionality:

----------------------

Public class DragAndDrop extends BaseClass

{

@Test

Public void drag ()

{

Driver.findElement(appiumBy.accessibleId(“views”)). click ();

Driver.findElement(appiumBy.accessibleId(“Drag and Drop”)). click ();

String source=Driver.findElement(appiumBy.id(“io.appium.android.apis:id/drag\_dot\_”));

// Java

((JavascriptExecutor) driver).executeScript("mobile: dragGesture", ImmutableMap.of(

"elementId", ((RemoteWebElement) element).getId(),

"endX", 619,

"endY", 560

));Here we need to click on any option in the above options i,e taps co-ordinates it will gives the end X and end Y values.

### mobile: flingGesture

This gesture performs fling gesture on the given element/area. Available since Appium v1.19

#### Supported arguments

* *elementId*: The id of the element to be flinged. If the element id is missing then fling bounding area must be provided. If both the element id and the fling bounding area are provided then this area is effectively ignored.
* *left*: The left coordinate of the fling bounding area
* *top*: The top coordinate of the fling bounding area
* *width*: The width of the fling bounding area
* *height*: The height of the fling bounding area
* *direction*: Direction of the fling. Mandatory value. Acceptable values are: up, down, left and right (case insensitive)
* *speed*: The speed at which to perform this gesture in pixels per second. The value must be greater than the minimum fling velocity for the given view (50 by default). The default value is 7500 \* displayDensity

#### Returned value

The returned value is a boolean one and equals to true if the object can still scroll in the given direction

#### Usage examples

// Java  
boolean canScrollMore = (Boolean) ((JavascriptExecutor) driver).executeScript("mobile: flingGesture", ImmutableMap.of(  
 "elementId", ((RemoteWebElement) element).getId(),  
 "direction", "down",  
 "speed", 500  
));

### mobile: pinchOpenGesture

This gesture performs pinch-open gesture on the given element/area. Available since Appium v1.19

#### Supported arguments

* *elementId*: The id of the element to be pinched. If the element id is missing then pinch bounding area must be provided. If both the element id and the pinch bounding area are provided then the area is effectively ignored.
* *left*: The left coordinate of the pinch bounding area
* *top*: The top coordinate of the pinch bounding area
* *width*: The width of the pinch bounding area
* *height*: The height of the pinch bounding area
* *percent*: The size of the pinch as a percentage of the pinch area size. Valid values must be float numbers in range 0..1, where 1.0 is 100%. Mandatory value.
* *speed*: The speed at which to perform this gesture in pixels per second. The value must not be negative. The default value is 2500 \* displayDensity

#### Usage examples

// Java  
((JavascriptExecutor) driver).executeScript("mobile: pinchOpenGesture", ImmutableMap.of(  
 "elementId", ((RemoteWebElement) element).getId(),  
 "percent", 0.75  
));

### mobile: pinchCloseGesture

This gesture performs pinch-close gesture on the given element/area. Available since Appium v1.19

#### Supported arguments

* *elementId*: The id of the element to be pinched. If the element id is missing then pinch bounding area must be provided. If both the element id and the pinch bounding area are provided then the area is effectively ignored.
* *left*: The left coordinate of the pinch bounding area
* *top*: The top coordinate of the pinch bounding area
* *width*: The width of the pinch bounding area
* *height*: The height of the pinch bounding area
* *percent*: The size of the pinch as a percentage of the pinch area size. Valid values must be float numbers in range 0..1, where 1.0 is 100%. Mandatory value.
* *speed*: The speed at which to perform this gesture in pixels per second. The value must not be negative. The default value is 2500 \* displayDensity

#### Usage examples

// Java  
((JavascriptExecutor) driver).executeScript("mobile: pinchCloseGesture", ImmutableMap.of(  
 "elementId", ((RemoteWebElement) element).getId(),  
 "percent", 0.75  
));

Real Time Project:

------------------------

C:\Users\sandeep96\AppData\Local\Android\Sdk\platform-tools>adb install C:\Users\sandeep96\Desktop\GeneralStore\General-Store.apk

Performing Streamed Install

Success

By using above command we will install the General-Store app into our virtual device.