Appium workshop

Syam Sasi
Martin Schneider

Technopark Trivandrum, 18.05.2019

Martin Schneider
(Carousell, Singapore)

Syam Sasi
(Carousell, Singapore)

ADAPT TECHNOLOGY ADHERE INNOVATION

PRATHIDHWANI TECHNICAL FORUM'S

18th Technical Workshop

Registration started http://techforum.prathidhwani.org

APPIUM WORKSHOP

ONE-DAY MOBILE AUTOMATION TESTING WORKSHOP

PRATHIDHWANI
Welfare organisation of IT employees

SATURDAY
09:30.am to 05:00.pm
TRAVANCORE HALL
Technopark, Trivandrum

18th MAY 2019

About us

Syam Sasi
Senior Software Engineer @ Carousell
https://about.me/syamsasi

Martin Schneider
Senior Software Engineer @ Carousell
mart.schneider@gmail.com

Code

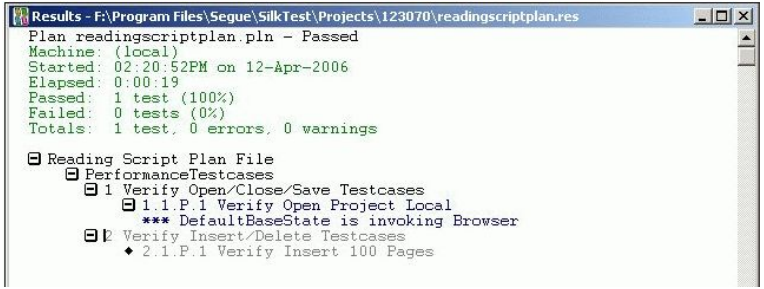
https://github.com/martinschneider/appium-workshop-2019-05-18

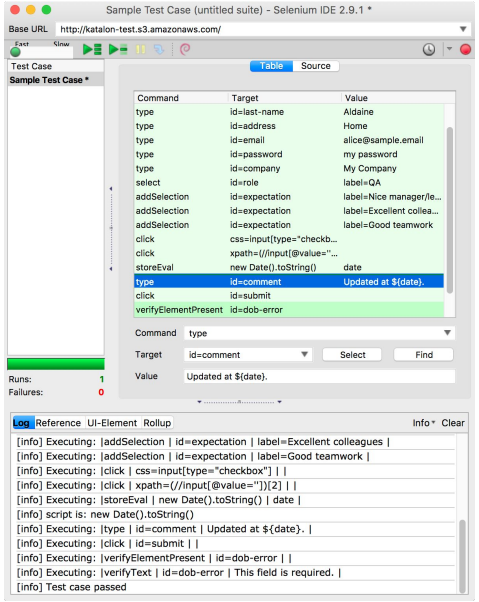
```
git clone git@github.com:martinschneider/appium-workshop-2019-05-18.git
mvn compile
```

Background and history of Appium

A brief (overly simplified) history of UI test automation

1993: QA Partner (Segue Software)
1996: QA Partner → Silk Test
2004: Selenium created at ThoughtWorks (Jason Huggins)
2005: Robot Framework (Pekka Klärck)
2006: Borland acquires Silk Test
2007: WebDriver (Simon Stewart)
2008: Selenium Grid
2008: Cucumber (Aslak Hellesøy)
2009: Selenium + WebDriver = Selenium 2.0
2009: MicroFocus acquires Borland
2009: Silk4J
2011: iOS Auto (Dan Cuellar)
2013: Espresso
2014: Appium 1.0 (Saucelabs)
2017: Silk Webdriver





appium

The history of Appium

2011: Dan Cuellar creates iOSAuto, a tool for iOS automation written in C#
2012: Dan presents Appium during a Lightning Talk at the 2012 Selenium Conference in London
2012: Dan rewrites appium in Python and open-sources it under the Apache 2 license
2012: Cooperation with Jason Huggins (Selenium co-founder) starts
2013: Saucelabs starts backing Appium
2013: Appium is rewritten using node.js
2013: Android support is added → first truly platform-independent mobile testing framework
2014: Appium 1.0 is released
2016: Drivers for Windows desktop and youi.tv automation are added
2016: Appium is donated to the JS Foundation

The journey continues!

http://appium.io/history.html
https://github.com/appium
http://appium.io/

Coffee break reading material ;-)

https://www.infoq.com/news/2018/04/cucumber-bdd-ten-years

https://www.seleniumhq.org/about/history.jsp

http://appium.io/history.html

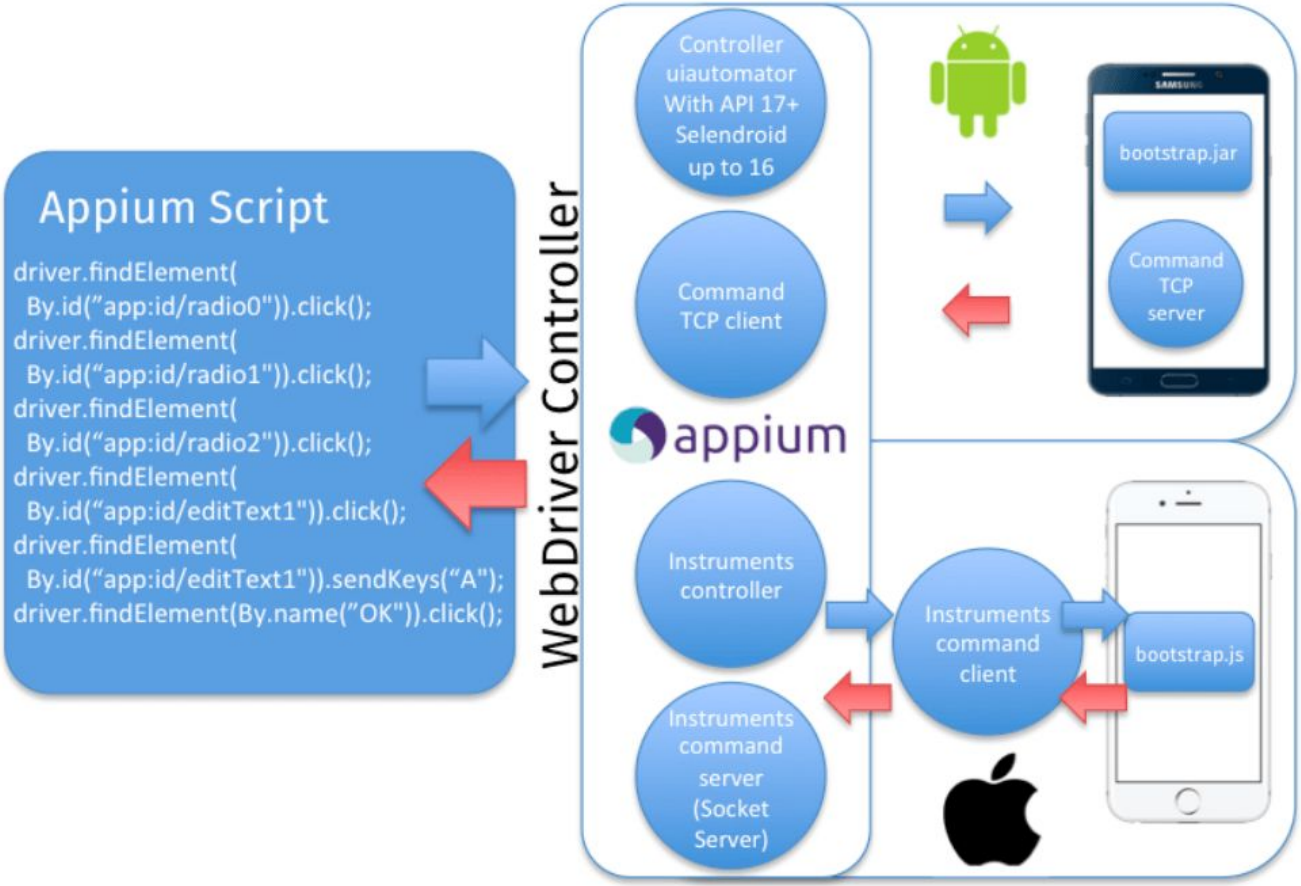
Appium philosophy

Appium philosophy

1. You shouldn't have to recompile your app or modify it in any way in order to automate it.
2. You shouldn't be locked into a specific language or framework to write and run your tests.
3. A mobile automation framework shouldn't reinvent the wheel when it comes to automation APIs.
4. A mobile automation framework should be open source, in spirit and practice as well as in name!

Appium design

Appium design

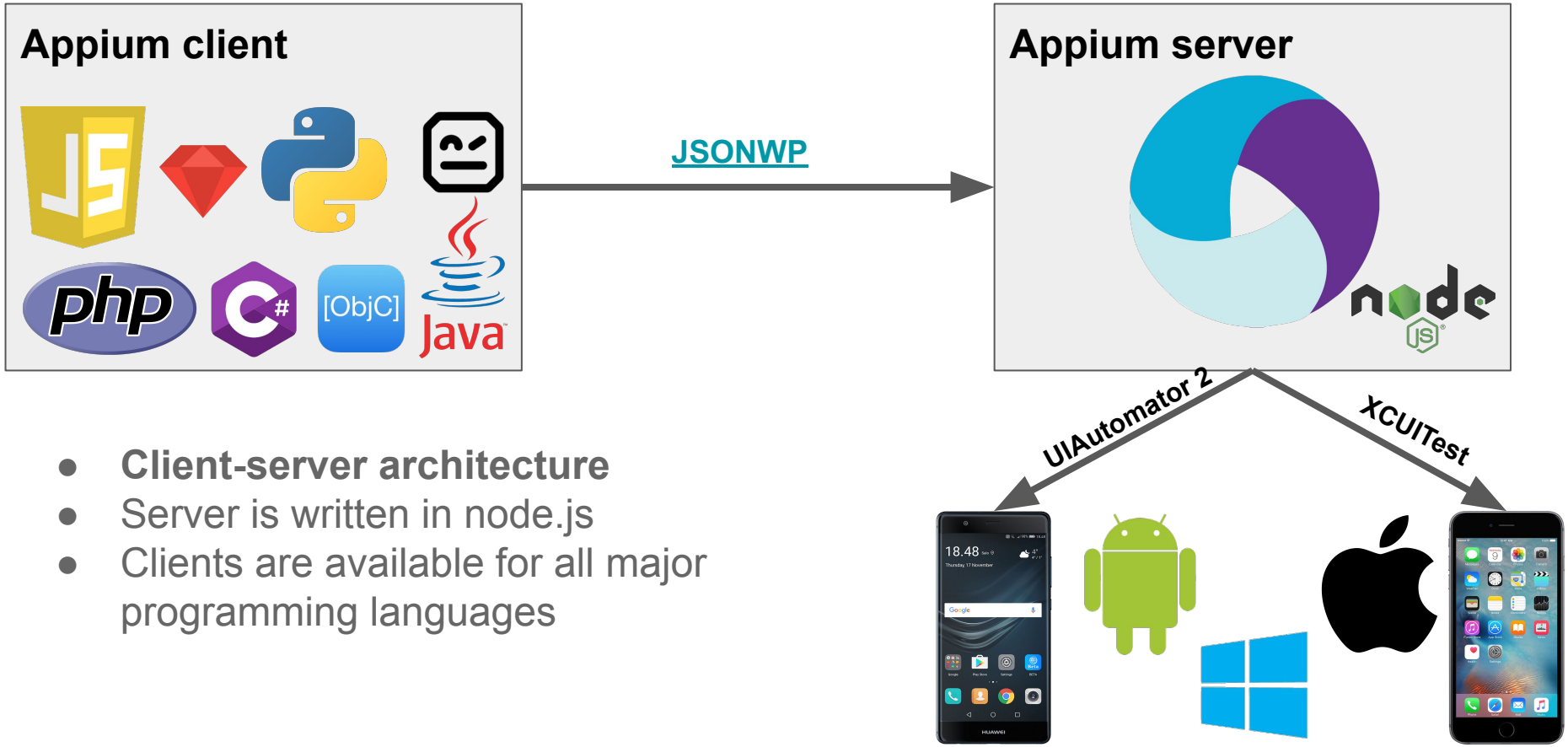


The vendor provided frameworks

- Android 4.2+: Google's UiAutomator/UiAutomator2
- Android 2.3+: Google's Instrumentation (Selendroid)
- iOS 9.3 and above: Apple's XCUITest
- iOS 9.3 and lower: Apple's UIAutomation
- Windows: Microsoft's WinAppDriver

Appium basics

Appium architecture



- **Client-server architecture**
- Server is written in node.js
- Clients are available for all major programming languages

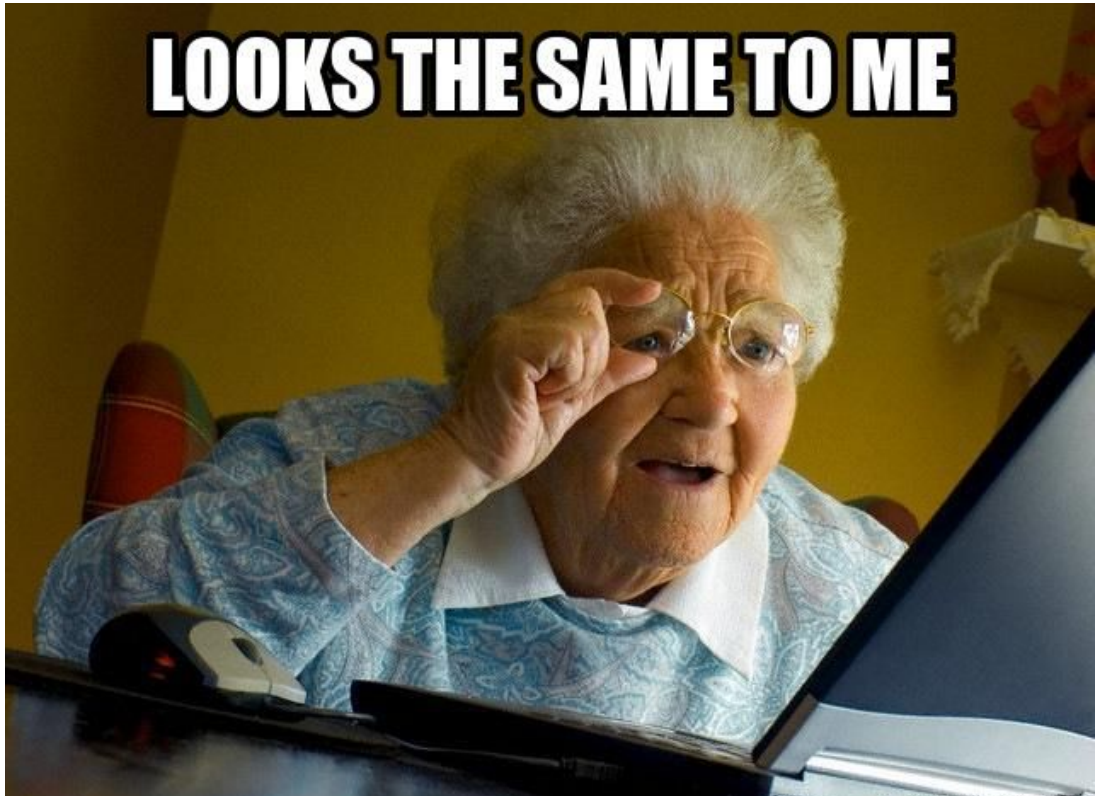
Selenium

```
WebDriver driver = new FirefoxDriver();
driver.get("http://www.google.com");

WebElement element = driver.findElement(By.name("q"));
element.sendKeys("Appium Meet-up");
element.submit();
...
```

Appium

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


IOSElement element = driver.findElement(By.name("search"));
element.sendKeys("Appium Meet-up");
element.submit();
...
```

Selenium and Appium



Appium vs. native frameworks

- Appium is highly portable.
- We can use the same tool to automate Android, iOS, Web (with Selenium) and Windows Desktop applications → multi-platform test frameworks.

- Appium is slower than native automation.
- There is an overhead that comes with the client/server architecture.

Appium basics

Our first Appium test:

- /appium-demo/src/test/java/io/github/martinschneider/appium/android/**CounterTest01.java**
- /appium-demo/src/test/java/io/github/martinschneider/appium/android/**CounterTest02.java**

Appium capabilities

Appium Capabilities

General Capabilities

automationName - XCUITest or UiAutomator2

platformName - iOS or Android

platformVersion - 7.0, 11.0

deviceName - iPhone6, Samsung Galaxy S8

app - /path/to/apk, app or ipa

fullReset - true or false

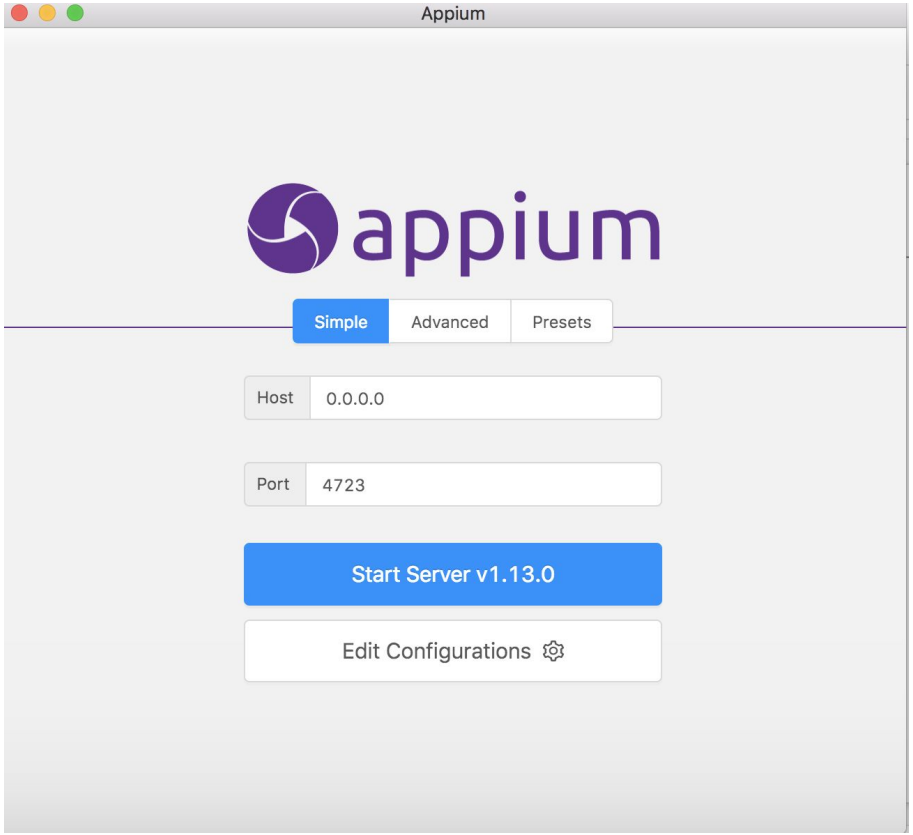
newCommandTimeout - in seconds

Appium Desktop

Why Appium Desktop?

- Inspect Android and iOS App UI
- Search for a mobile element
- Record the actions

Appium Desktop Demo



Inspecting Android App

1. Run appium command in command prompt/terminal
2. Open Appium Desktop
3. Click on Appium > New Session Window
4. Enter the following capabilities and save

*Note: **Pixel** is the name of the emulator*

```
{
  "platformName":"android",
  "deviceName":"Pixel",
  "automationName":"UiAutomator2",
  "app":"/path/to/apk/file",
  "avd","Pixel"
}
```

carousell

Android locator types

Android locator types

- accessibility
- id
- xpath
- class
- uiAutomator
- image

http://appium.io/docs/en/commands/element/find-elements/
https://saucelabs.com/blog/advanced-locator-strategies

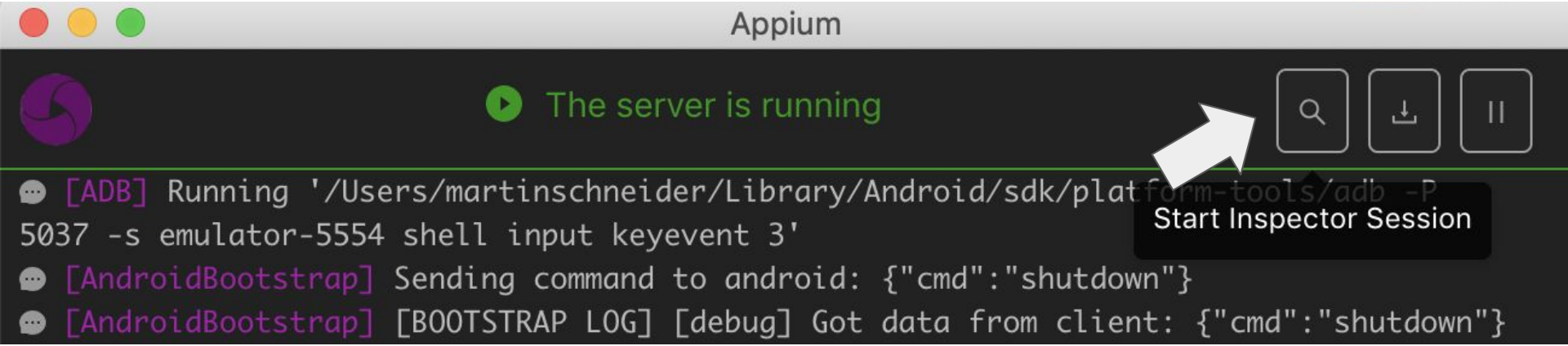
Android locator types

- /appium-demo/src/test/java/io/github/martinschneider/appium/android/**CounterTest03.java**
- /appium-demo/src/test/java/io/github/martinschneider/appium/android/**CounterTest04.java**

carousell

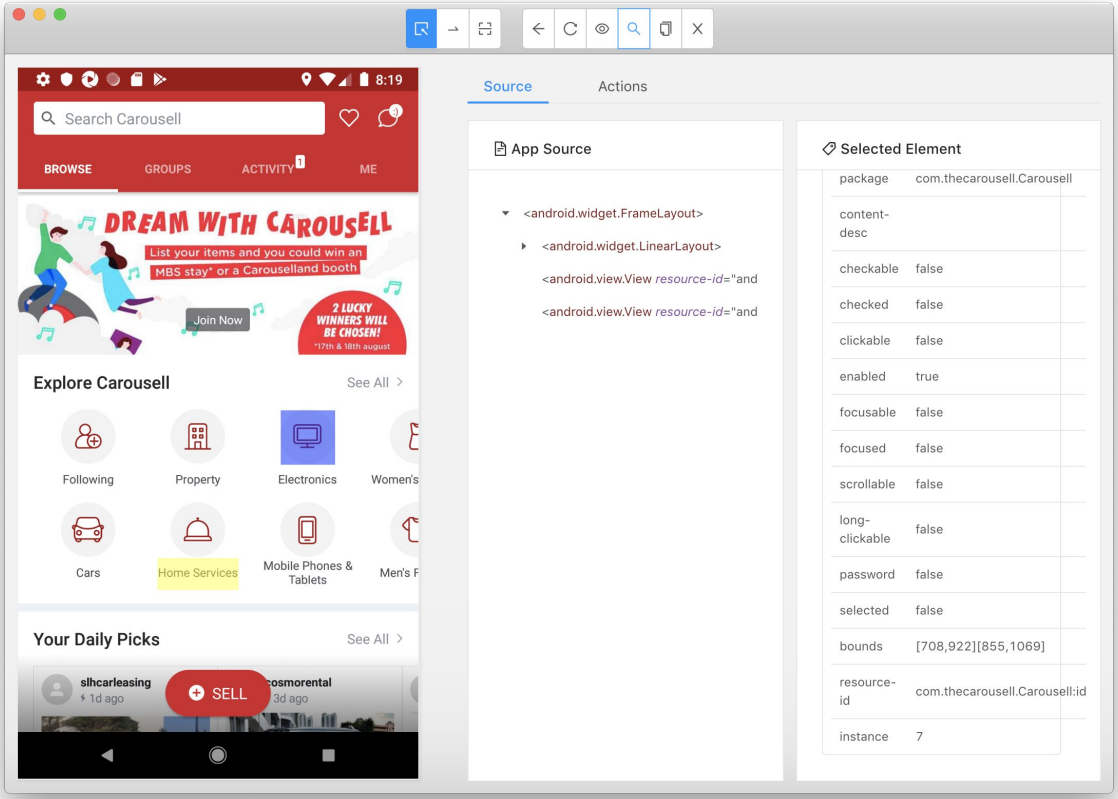
Inspector tools

Appium inspector




```
{
  "platformName": "android",
  "appPackage": "com.thecarousell.Carousell",
  "appActivity": "com.thecarousell.Carousell.activities.EntryActivity",
  "appPath": "/Users/martinschneider/carousell.apk",
  "deviceName": "test",
}
```

Appium inspector

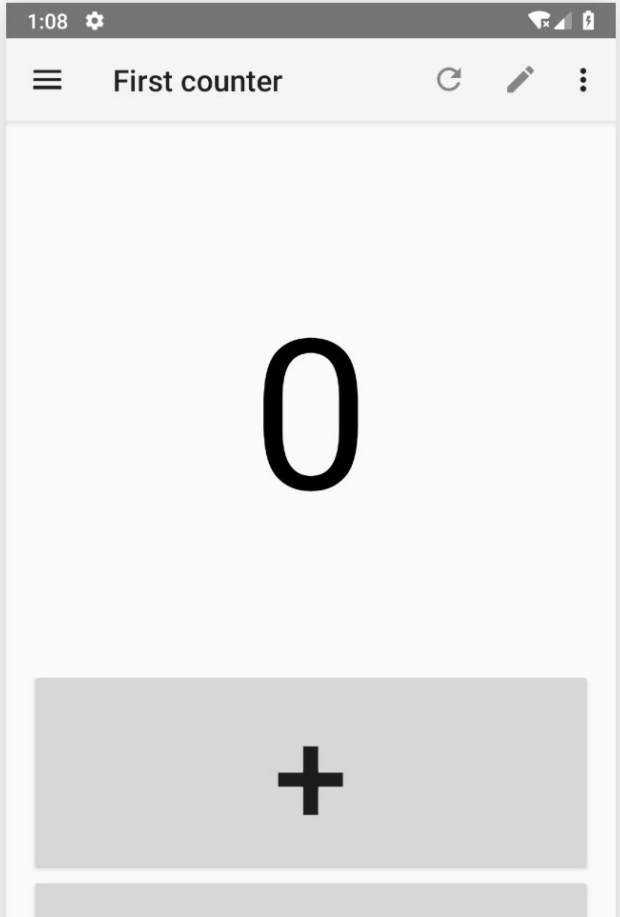


Macaca inspector

- https://macacajs.github.io/app-inspector/
- `npm i app-inspector -g`
- `adb devices`
- `app-inspector -u YOUR-DEVICE-ID`
- http://10.0.1.117:5678
- Does not require Appium or an Appium session

Macaca App Inspector

Star 256



View Source

```
<FrameLayout>
  <LinearLayout>
    <FrameLayout>
      <android.view.ViewGroup>
        <FrameLayout>
          <android.view.ViewGroup>
            <ImageButton></ImageButton>
            <TextView></TextView>
            <android.support.v7.widget.Linear
              <TextView></TextView>
              <TextView></TextView>
              <ImageView></ImageView>
            </android.support.v7.widget.Linea
          </android.view.ViewGroup>
        </FrameLayout>
        <FrameLayout>
          <android.support.v4.widget.DrawerLay
            <FrameLayout>
              <RelativeLayout>
                <FrameLayout>
                  <TextView></TextView>
                </FrameLayout>
                <Button></Button>
                <Button></Button>
              </RelativeLayout>
```

text	+
resource-id	me.tsukanov.counter:id/incrementButton
class	android.widget.Button
package	me.tsukanov.counter
content-desc	null
checkable	false
checked	false
clickable	true
enabled	true
focusable	true
focused	false
scrollable	false
long-clickable	false
password	false
bounds	42,1181,996,364
xpath_lite	//*[@resource-id="me.tsukanov.counter:id/incrementButton"]

Waiting strategies

Waiting strategies

- /appium-demo/src/test/java/io/github/martinschneider/appium/android/**LoginTest01.java**
- /appium-demo/src/test/java/io/github/martinschneider/appium/android/**LoginTest02.java**
- /appium-demo/src/test/java/io/github/martinschneider/appium/android/**LoginTest03.java**
- /appium-demo/src/test/java/io/github/martinschneider/appium/android/**LoginTest04.java**

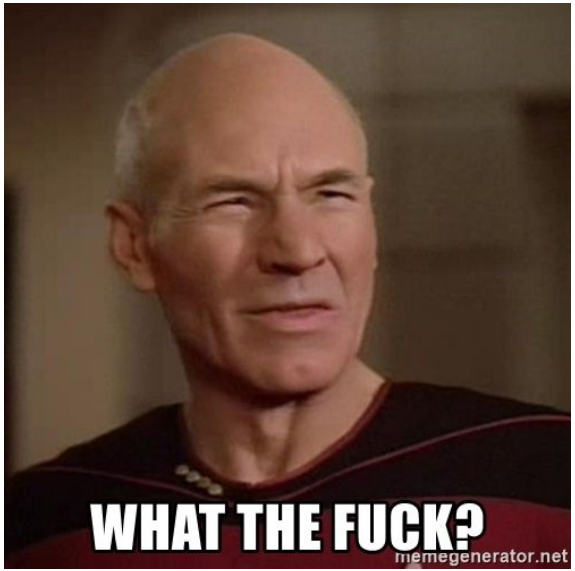
Static wait

```
Thread.sleep(3000);
```

- Fixed duration for every execution
  - either too long (waste of execution time)
  - or too short (test will still fail)

- Remember: Test behave differently
  - on different environments
  - under different network conditions
  - etc.

- Just ugly and wrong! Avoid at all costs!



Implicit wait

```
driver.manage().timeouts().implicitlyWait(10,
TimeUnit.SECONDS);
```

- Will wait **up to** the specified time **for every element**.
- Speeds up test execution locally but also slows down overall test execution because it is a global setting (no ability to fail fast)!
- Better than static waits but still not quite convenient enough….

Explicit wait

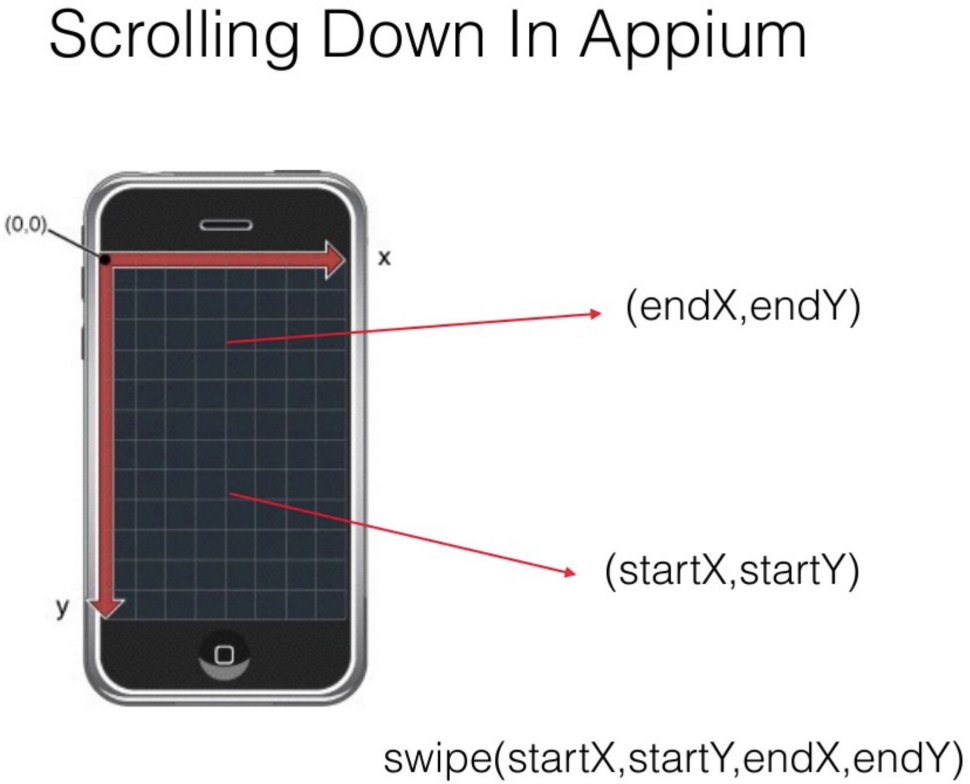
```
WebDriverWait wait = new WebDriverWait(driver, 10);
    wait.until(presenceOfElementLocated(goToLogin))
        .click();
    wait.until(presenceOfElementLocated(username))
        .sendKeys(username);
```

- Waits until a condition is met.
- This isn’t limited to visibility.
- We can implement custom conditions.
- **We can use different timeouts for different use-cases!**

carousell

Swipe and scroll

The X Y Coordinates



The Touch Action Class

1. Touch Action

   *press*release*moveTo*tap*wait*longPress.cancel.perform

```
TouchAction().press(el1).moveTo(el2).release();
```

2. Multi Touch

```
action1 = TouchAction().tap(el1);
action2 = TouchAction().tap(el2);

MultiAction().add(action1).add(action2).perform();
```

- /appium-demo/src/test/java/io/github/martinschneider/appium/android/swipe/**SwipeTest.java**

Page object model

Page objects

- De-facto standard design pattern in UI test automation.
- A page object is an object-oriented class that serves as an interface to a page/screen of your AUT.
- Goals: enhance maintenance, avoid code duplication
- Ideally one page object per page/screen/dialog/component.
- Tests don’t care about the layout and locators of your app. They just interact with page objects.
- We can use mechanisms like Dependency Injection to simplify the use of page objects.

Theory is boring...

- /appium-demo/src/test/java/io/github/martinschneider/appium/android/**LoginTest05.java**
- /appium-demo/src/test/java/io/github/martinschneider/appium/android/**LoginTest06.java**
- /appium-demo/src/test/java/io/github/martinschneider/appium/android/**LoginTest07.java**

How to design a test framework?

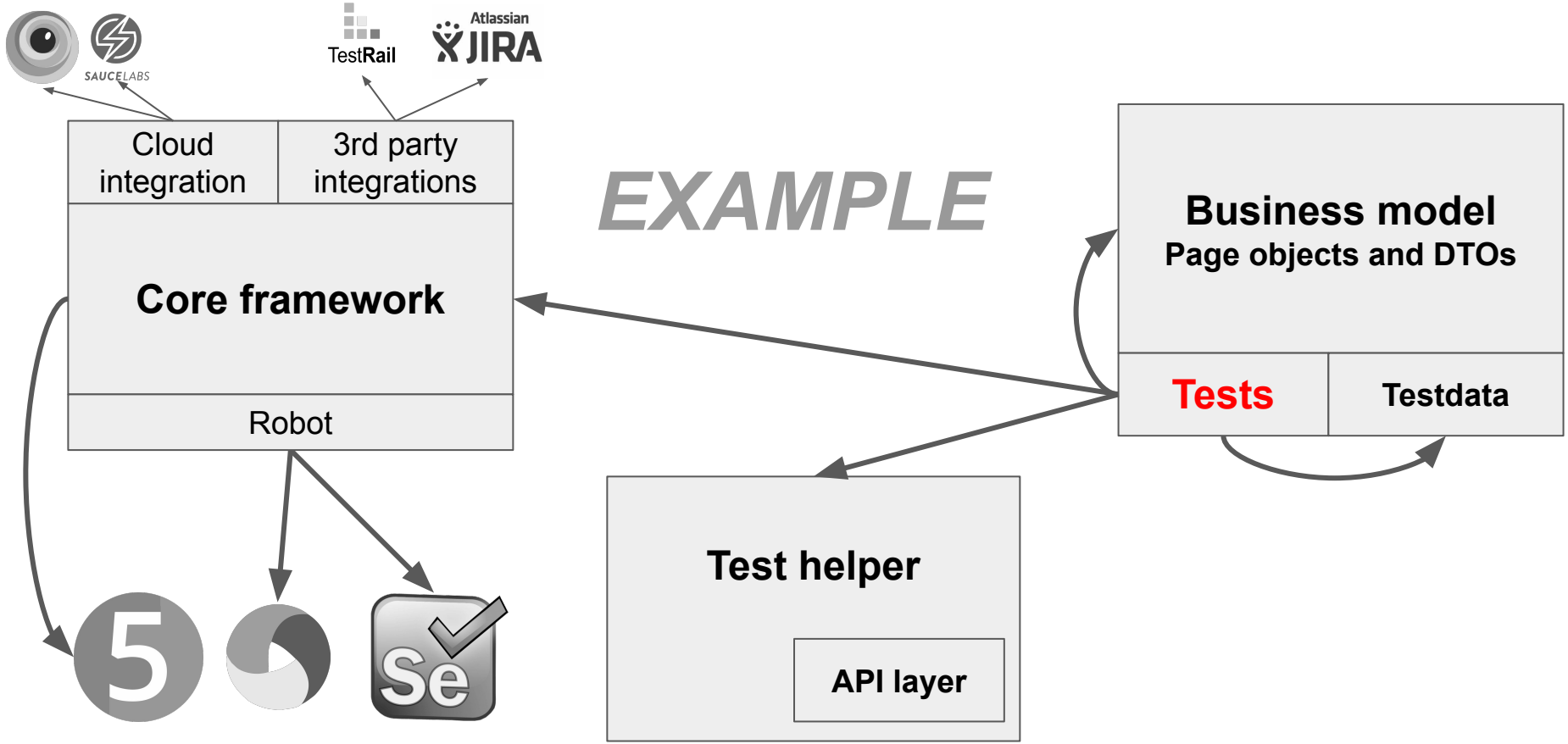
Some thoughts

How to design a test framework?

- Don’t re-invent the wheel but combine existing tools in a way that best fit your use-case.
- Always keep re-usability, adaptability, extensibility and scalability in mind.
- A quick (and “dirty”) PoC is a start but what’s your use-case in 6 months / 1 year / 5 years… Is your design ready for the change?
- → Have a vision and lay the foundation accordingly.
- Good design now = less refactoring later.
- Test automation requires software engineers (test engineer = **engineer**).
- Apply the same (or better) practices you have for building the solutions you sell to your customers, always lead by example (quality!).
- Powerful under the hood but super-easy to use.
- A framework is cool but don’t forget the culture in which it is used.

https://www.linkedin.com/in/martin-schneider

How to design a test framework?



IOS automation

Locator Strategy for iOS

Accessibility ID

```
driver.findElement(MobileBy.AccessibilityID("foo"));
```

iOS NS Predicates

```
driver.findElementByIosNsPredicate(tableViews()[1].cells().firstWithPredicate("label == 'Olivia' "));
```

iOS Class Chain

```
driver.findElement(MobileBy.iOSClassChain("**/XCUIElementTypeCell[`name BEGINSWITH "C"`]/XCUIElementTypeButton[10]"));
```

- /appium-demo/src/test/java/io/github/martinschneider/appium/ios/**IOSTest.java**

Real device vs. simulators/emulators

Appium On Real Devices - Android

1. Enable USB Debugging
2. Connect the device using usb
3. Note down the udid
4. Device is detected when run the `adb devices` command

https://developer.android.com/studio/debug/dev-options

Appium On Real Devices - iOS

1. cd /usr/local/lib/node_modules/appium/node_modules/appium-xcuitest-driver/WebDriverAgent
2. mkdir -p Resources/WebDriverAgent.bundle
3. /bin/bash ./Scripts/bootstrap.sh -d
4. brew install carthage
5. brew install libimobiledevice --HEAD
6. npm install -g ios-deploy
7. UI Automation enabled on iOS device
8. Xcode has proper code sign enabled

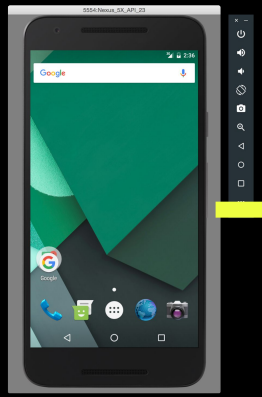
http://appium.io/docs/en/drivers/ios-xcuitest-real-devices/

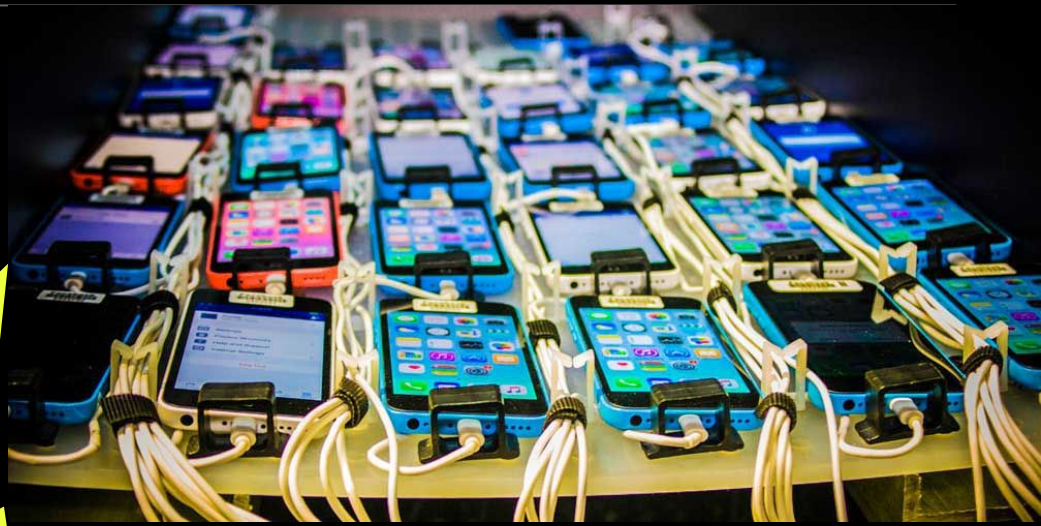
Which one I need?

Cloud automation

Next question: How can we scale this?

Next question: How can we scale this?





AWS

SAUCELABS

pCloudy.com

On premise vs. cloud

• Full control ○ Number of devices ○ Types of devices ○ Software libraries ○ ...	• No maintenance required • Fixed costs • Support
• Maintenance ○ replace devices ○ fix issues ○ infrastructure ○ ...	• Less flexibility ○ Available devices ○ Software update cycle? (Appium, Java)

Two approaches towards cloud testing

#1: Client-side execution

Step 1: Upload your APP to the cloud

Step 2: Create a WebDriver instance

```
capabilities.setCapability("app", appUrl);
new AppiumDriver("http://cloud.com/wd/hub", capabilities);
```

Step 3: Execute your tests as before

For example, Browserstack, Saucelabs, pCloudy...

Two approaches towards cloud testing

#2: Server-side execution

Step 1: Upload your APP to the cloud

Step 2: Package and upload your tests to the cloud

Step 3: Tests get executed on the cloud

For example, AWS Devicefarm (public cloud)

Client- vs. server-side execution

- Easier migration
  - Execution flow remains the same
  - Reporting and CI remain the same
- More control
  - Software updates on the client side
  - easier to use a mixed/hybrid mode (=use different cloud providers or local + cloud)

- Performance (maybe)

- Network latency

- Trust and compliance
- Integration overhead
  - Test results and reporting
  - Network challenges

Factors to consider

What's your use-case?

- **Devices**
  - specific models, manufacturers, OS versions
- **Parallelisation**
  - How to distribute n tests across m devices efficiently (each test runs once)?
  - How to run n tests on m devices in parallel (each test runs m times)?
- **Time constraints, execution speed, waiting times**
  - run tests once a day ("nightly tests") → not time-critical
  - run tests multiple times a day to verify changes → highly time-critical
- **Client- vs. server execution mode**
- **API support**
- **Cost and pricing models**
- **Future use-cases**

Device models

- In the US Apple and Samsung cover 81% of the market (UK 87%, Singapore 82%, Australia 89% etc.)

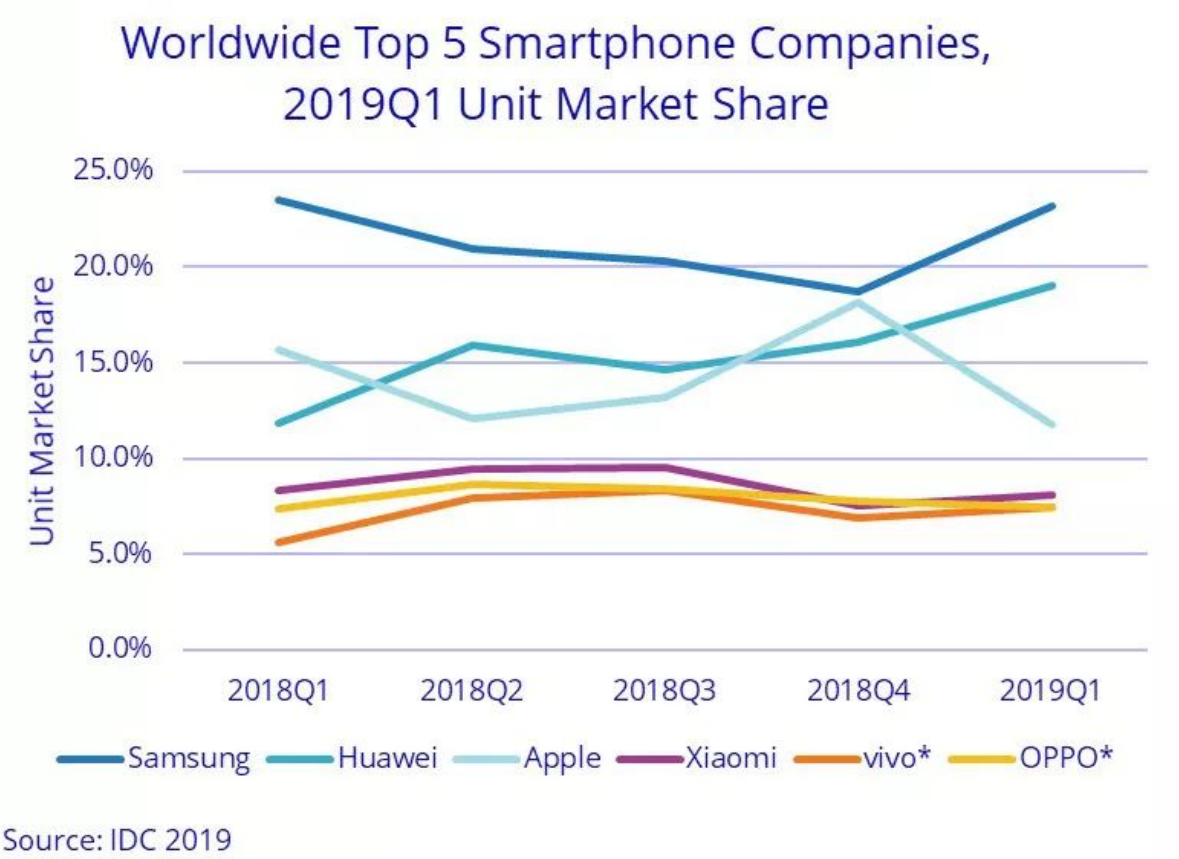
- What about Xiaomi, Oppo, Vivo, Lyf...?

Device market share in India

Device Vendor	Rank	Traffic %
Samsung	1	27.94
Xiaomi	2	19.89
Apple	3	9.56
Oppo	4	7.13
Vivo	5	5.93
Lyf	6	5.2
Motorola	7	3.37
Lenovo	8	2.93
Micromax	9	2.32
OnePlus	10	1.96

Source: https://deviceatlas.com/

Device models



Source: IDC 2019

- Huawei (13,3%) surpassed Apple (11,9%) in market share in Q2/2018
- Q1/2019: 19% global market share

Market specifics

- India: 20% Xiaomi (#3) = 2/3 are neither Apple nor Samsung
- Myanmar (2018): 19% Oppo (#2), 18% Vivo (#3), 13% Xiaomi (#4), 11% Huawei (#5) = 70% not Apple and Samsung
- Pakistan: 11% QMobile (#2)
- Bangladesh: 14% Symphony (#2)

Sources:
https://www.gartner.com/en/newsroom/press-releases/2018-08-28-gartner-says-huawei-secured-no-2-worldwide-smartphone-vendor-spot-surpassing-apple-in-second-quarter, https://deviceatlas.com/

Device models

- AWS Device Farm doesn't have any Huawei, Xiaomi, Oppo or Vivo models (in their public cloud)
- Browserstack only supports Samsung, Google and Apple phones (and one OnePlus model)

Sources:
https://www.gartner.com/en/newsroom/press-releases/2018-08-28-gartner-says-huawei-secured-no-2-worldwide-smartphone-vendor-spot-surpassing-apple-in-second-quarter, https://deviceatlas.com, https://www.browserstack.com/list-of-browsers-and-platforms/app_automate, http://awsdevicefarm.info

Demos

We will run the same login test on AWS Devicefarm, Browserstack and locally.

AWS Devicefarm

- package the test configuration into a test spec and upload to AWS
- upload the APK file to AWS
- package our tests (+framework) and upload to AWS
- select an available device (using device filters)
- execute the tests and process the results

Browserstack

- upload the APK file to Browserstack
- configure the execution (this includes the device selection) using `Capabilities`
- execute the tests and process the results

www.justtestlah.qa
https://medium.com/@mart.schneider/mobile-test-automation-using-aws-device-farm-6bcf825fa27d

Appium tips and tricks

#1 Network Conditions

- **Toggle Airplane** - `driver.toggleAirplaneMode();`
- **Toggle Data** - `driver.toggleData();`
- **Toggle Wifi** - `driver.toggleWifi();`

#2 Android Emulator commands

- **Send SMS** -
```
driver.sendSMS("555-123-4567", "Hey?");
```
- **GSM Call** -
```
driver.makeGsmCall("5551234567", GsmCallActions.CALL);
```
- **GSM Signal** -
```
driver.setGsmSignalStrength(GsmSignalStrength.GOOD);
```
- **Geo Location** -
```
driver.setLocation(new Location(49, 123, 10));
```

#3 Dealing With An Element When Visibility Is False

```
// first, find our  element
WebElement ref = wait
.until(ExpectedConditions.presenceOfElementLocated(element));

// get the location and dimensions of the reference element, and find its
center point
Rectangle rect = ref.getRect();
int refElMidX = rect.getX() + rect.getWidth() / 2;
int refElMidY = rect.getY() + rect.getHeight() / 2;

// set the center point of our desired element; w
int desiredElMidX = refElMidX;
int desiredElMidY = refElMidY - rect.getHeight();

// perform the TouchAction that will tap the desired point
TouchAction action = new TouchAction<>(driver);
action.press(PointOption.point(desiredElMidX, desiredElMidY));
action.waitAction(WaitOptions.waitOptions(Duration.ofMillis(500)));
action.release();
action.perform();
```

#4 Speeding Up Your Tests

- **Deeplinks**

```
driver.get("theapp://login/" + AUTH_USER + "/" + AUTH_PASS);
```

- **OptionalIntentArguments**

```
caps.setCapability("optionalInentArguments", String.format("-e \"username\" \"%s\" -e \"password\" \"%s\"", AUTH_USER, AUTH_PASS));
```

#5 Fine Tuning The Capabilities - Cross Platform

- **Cross Platform**
  - noReset
  - fullReset
  - isHeadless

#6 Fine Tuning The Capabilities - Android

- **Android**
  - disableAndroidWatchers
  - autoGrantPermissions
  - skipUnlock
  - ignoreUnimportantViews

#7 Fine Tuning The Capabilities - iOS

- **iOS**
  - usePrebuildWDA and derivedDataPath
  - useJSONSource
  - iosInstallPause
  - maxTypingFrequency
  - realDeviceScreenShotter
  - simpleIsVisibleCheck

#8 Testing App Upgrades

Android

```
driver.installApp("/path/to/apk");
driver.startActivity(activity);
```

iOS

```
HashMap<String, String> bundleArgs = new HashMap<>();
bundleArgs.put("bundleId", BUNDLE_ID);
driver.executeScript("mobile: terminateApp", bundleArgs);
HashMap<String, String> installArgs = new HashMap<>();
installArgs.put("app", appUpgradeVersion);
driver.executeScript("mobile: installApp", installArgs);
driver.executeScript("mobile: launchApp", bundleArgs);
```

#9 Switching Between Apps

- **Android**

```
driver.startActivity(settingsAppPackageName, settingsAppActivityName);
```

- **iOS**

```
// launch the photos app (with the special bundle id seen below)
HashMap<String, Object> args = new HashMap<>();
args.put("bundleId", "com.apple.mobileslideshow");
driver.executeScript("mobile: launchApp", args);

// re-activate that AUT (in this case The App)
args.put("bundleId", "io.cloudgrey.the-app");
driver.executeScript("mobile: activateApp", args);
```

#10 Testing The Push Notifications

1. Kill The App `driver.terminateApp(bundleId or pkg);`

2. Open The Notification Pannel
    a. **Android** `driver.openNotifications();`
    b. **iOS** Swipe From Top Down to show the notification centre

3. Wait For Push Notification Content

4. Close The Panel and Relaunch The App
    `driver.activateApp(bundleId or pkg);`

#11 Automating Custom iOS Alerts

```
wait.until(ExpectedConditions.alertIsPresent());
HashMap<String, String> args = new HashMap<>();
args.put("action", "getButtons");
List<String> buttons = (List<String>)driver.executeScript("mobile:
alert", args);
// find the text of the button which isn't 'OK' or 'Cancel'
String buttonLabel = null;
for (String button : buttons) {
if (button.equals("OK") || button.equals("Cancel")) {
continue;}
buttonLabel = button;}
if (buttonLabel == null) {
throw new Error("Did not get a third alert button as we were
expecting");
}
args.put("action", "accept");
args.put("buttonLabel", buttonLabel);
driver.executeScript("mobile: alert", args);
```

#12 Automating A Picker Wheel On iOS

1. **Using SendKeys**

```
pickerWheelElement.sendKeys("March");
```

2. **Picker Wheel Value**

```
HashMap<String, Object> params = new HashMap<>();
params.put("order", "next");
params.put("offset", 0.15);
params.put("element", ((RemoteWebElement)
pickerWheelElement).getId());
driver.executeScript("mobile:
selectPickerWheelValue", params);
```

#13 Video Recording The Test Execution

```
IOSStartScreenRecordingOptions
iOSStartScreenRecordingOptions =new
IOSStartScreenRecordingOptions().withVideoQuality(IOSStartSc
reenRecordingOptions.VideoQuality.HIGH).withTimeLimit(Durati
on.ofSeconds(SCREEN_RECORD_DURATION));

driver.startRecordingScreen(iOSStartScreenRecordingOptions);
record = ((IOSDriver) driver).stopRecordingScreen();
byte[] decode = Base64.getDecoder().decode(record);
File videoFile =new File(new
StringBuilder().append("MyVideo.Mp4").toString());
 FileUtils.writeByteArrayToFile(videoFile, decode);
```

#14 Parallel And Distributed Testing

- Running multiple Appium servers, and sending one session to each server
- Running one Appium server, and sending multiple sessions to it
- Running one or more Appium servers behind the Selenium Grid Hub, and sending all sessions to the Grid Hub
- Leveraging a cloud provider (which itself is running many Appium servers, most likely behind some single gateway)

#15 Desktop App Automation

- **Appium for Mac** - https://github.com/appium/appium-for-mac
- **WinAppDriver** - https://github.com/Microsoft/WinAppDriver

Quiz Time

Go to https://kahoot.it

References

- http://appium.io
- https://appiumpro.com/editions
- https://github.com/SrinivasanTarget/awesome-appium
- https://github.com/appium/appium/tree/master/docs/en/advanced-concepts
- https://github.com/singapore-appium-meetup/meetups
- https://www.justtestlah.qa