Leaders in software testing

GUI Test Automation guideline

AN Direct

Table of content

1. INTRODUCTION

1.1. Goal

- Automate validation on multiple hardware configurations of the happy flow functionality
- Automation result is part of the sprint acceptance criteria

1.2. Scope

The scope of this project (GUI automation) is to set up a framework to start with GUI automation. The framework will contain at least one fully automated functionality based on an app with Mock hearing aids.

The baseline scenario for Android is:
An user which has already registered and verified his email address. Who has two hearing aids and configures both of the hearing aids with his Smartphone. The user skips the hearing profile measurement and goes to the dashboard.

The baseline scenario for iOS is:

1.3. Assumptions

- Every button and screen, that need to be automated, has an accessibility label
- Setting layout ID's (Android)
- Automation framework is based on app version with Mocks enabled in order to catch up issues with extern hardware.
  - This means that error flows and hardware related errors can't be tested.
  - There are build flavor (debug + binding mock) on HockeyApp (AN Direct account)
- Maven project with pom file for dependencies
- No automatic download new app from Bitrise
- No trigger after build (CI integration)
- Local device lab (physical and emulated devices)
- Automation setup is based on Macintosh platform / MacOS
- IntelliJ IDE is used
- Support two platforms (iOS and Android) native apps
- Support multiple versions per platform (iOS 9/10, Android 4.4 till 7)
- A test suite per platform
- No parallel execution per platform
- Tests are written in the TestNG test framework
- Keep automated test as small as possible, e.g. they fail on one assertion or one topic

1.4. Structure test suite

- iOS / Android
- Development (current sprint)
- Smoke test (ready to start regression test)
- Regression test (full set of devices)

2. Setting up Appium Desktop and Appium Server

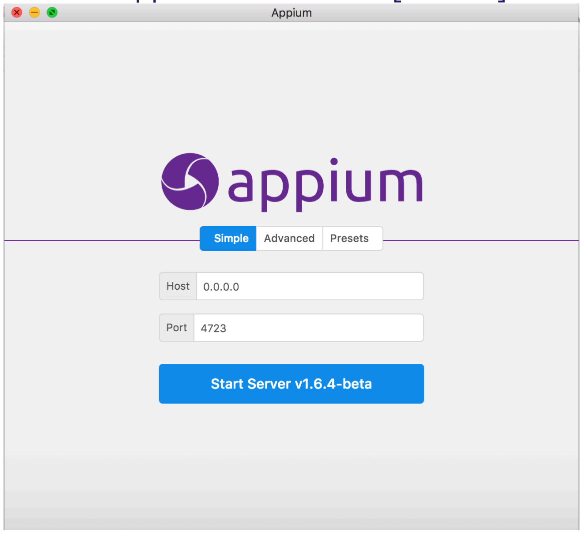
2.1. Appium Desktop

Appium desktop is used for building a test scenario, inspecting objects on a page and manually running automated test scenario’s or test suites.

2.2. Installing Appium Desktop

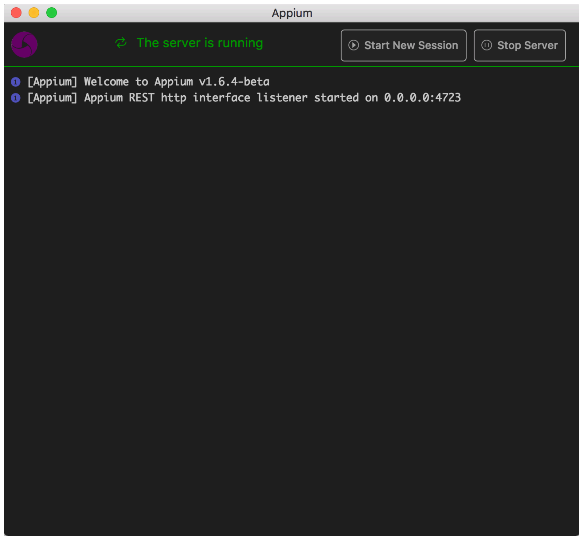
1. Download and install Appium Desktop from https://github.com/appium/appium-desktop/releases/tag/v1.0.0-beta.6

2. Start Appium Server with [default] settings



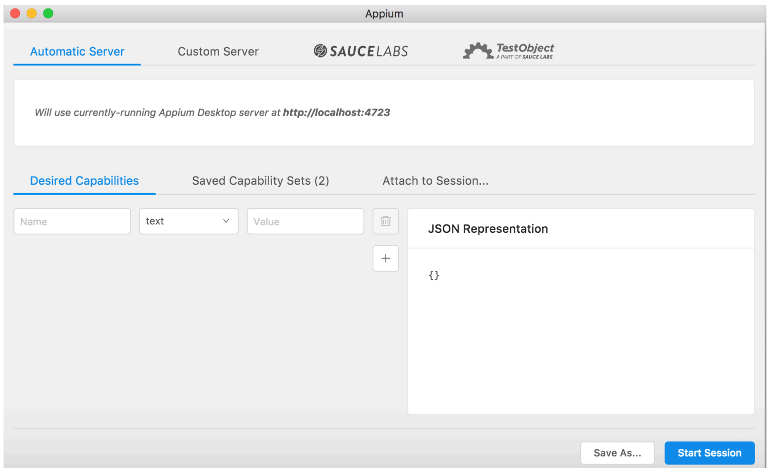
Host 0.0.0.0
Port 4723

3. Start new session



```
[Appium] Welcome to Appium v1.6.4-beta
[Appium] Appium REST http interface listener started on 0.0.0.0:4723
```

4. This is the general screen for setting capabilities to start using Appium for Android and iOS. After this section it is described how to set capabilities for Android and iOS in order to use Appium.


Appium

Automatic Server | Custom Server | SAUCELABS | TestObject

Will use currently-running Appium Desktop server at **http://localhost:4723**

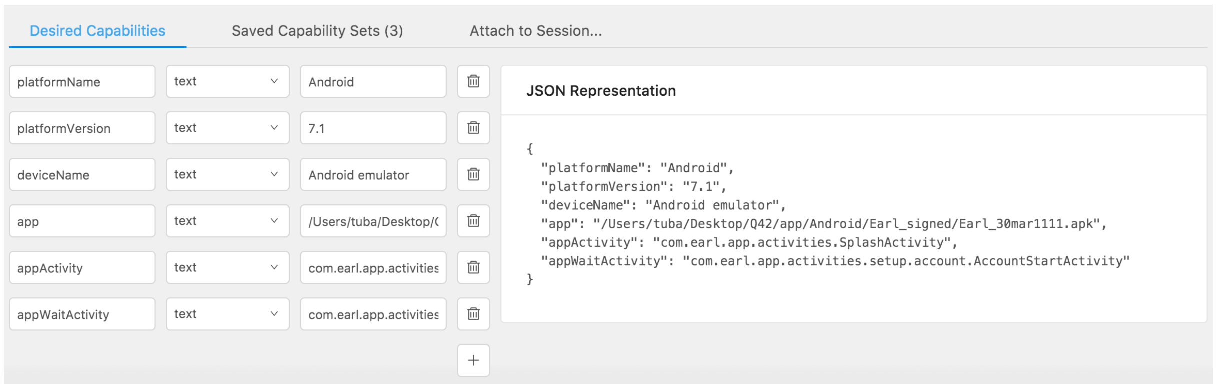
Desired Capabilities | Saved Capability Sets (2) | Attach to Session...

Name	text	Value

JSON Representation

```
{}
```

Android
Firs start an Android emulator through Terminal or Android Studio. Add the capabilities as done in the image below and start session. Ensure that the property platformVersion set in the pom.xml matches the running emulator.

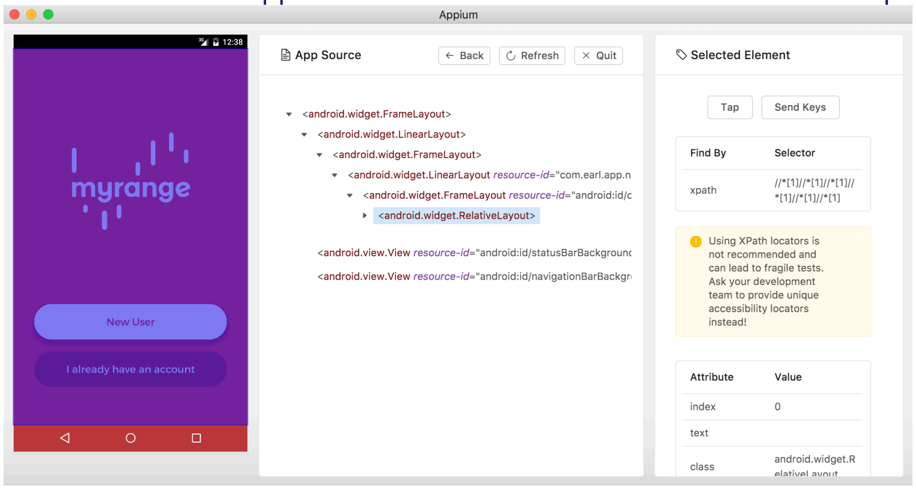

Desired Capabilities | Saved Capability Sets (3) | Attach to Session...

platformName	text	Android
platformVersion	text	7.1
deviceName	text	Android emulator
app	text	/Users/tuba/Desktop/(
appActivity	text	com.earl.app.activities
appWaitActivity	text	com.earl.app.activities

JSON Representation

```
{
  "platformName": "Android",
  "platformVersion": "7.1",
  "deviceName": "Android emulator",
  "app": "/Users/tuba/Desktop/Q42/app/Android/Earl_signed/Earl_30mar1111.apk",
  "appActivity": "com.earl.app.activities.SplashActivity",
  "appWaitActivity": "com.earl.app.activities.setup.account.AccountStartActivity"
}
```

This will start the app in the Android emulator and launch the Inspector.



App Source | Back | Refresh | Quit

```
▾ <android.widget.FrameLayout>
  ▾ <android.widget.LinearLayout>
    ▾ <android.widget.FrameLayout>
      ▾ <android.widget.LinearLayout resource-id="com.earl.app.n
        ▾ <android.widget.FrameLayout resource-id="android:id/c
          ▸ <android.widget.RelativeLayout>

  <android.view.View resource-id="android:id/statusBarBackgrounc
  <android.view.View resource-id="android:id/navigationBarBackgr
```

Selected Element

Tap | Send Keys

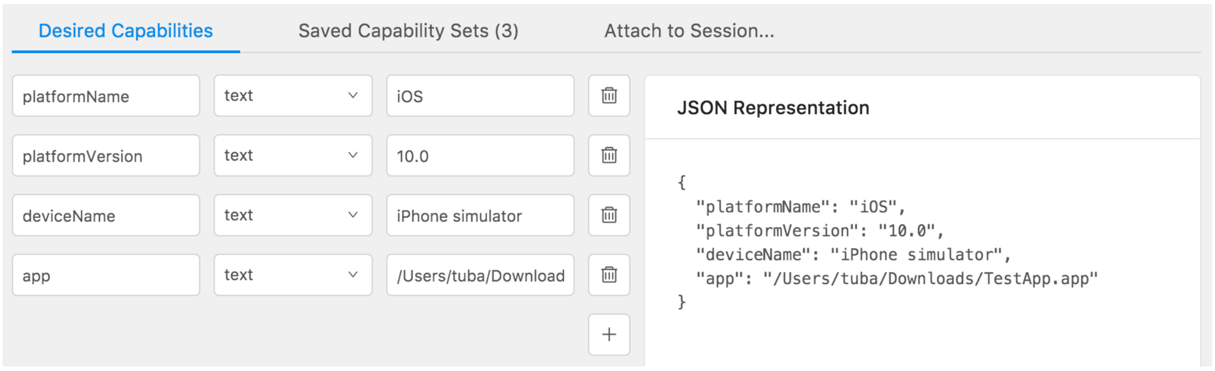
Find By	Selector
xpath	//*[1]//*[1]//*[1]//*[1]//*[1]//*[1]

Using XPath locators is not recommended and can lead to fragile tests. Ask your development team to provide unique accessibility locators instead!

Attribute	Value
index	0
text	
class	android.widget.RelativeLayout

On this screen you can select an element and find the identifier which is needed for GUI automation. When identifier is not defined the xpath is shown below the section "Find By".

iOS
For iOS you don't have to start a simulator manually. Appium starts the simulator on starting the session, but the defined simulator in Appium must be installed on the computer. The capabilities that are needed for iOS are. In order to make it work for iOS 9.3 you need to add an additional capability "automationName" : "XCUITest". iOS versions lower than 9.3 aren't supported.



platformName	text	iOS
platformVersion	text	10.0
deviceName	text	iPhone simulator
app	text	/Users/tuba/Download

```
{
  "platformName": "iOS",
  "platformVersion": "10.0",
  "deviceName": "iPhone simulator",
  "app": "/Users/tuba/Downloads/TestApp.app"
}
```

The inspector part is the same as on Android.

2.3. Appium Server
The appium server is used to communicate with a client (app) on a device. The test suite send all the commands to the server and receives the responses back.

2.4. Installing Appium Server
The appium server or CLI can be installed from the commandline:

- npm install -g appium

2.5. Starting the appium server
The appium server is started from the commandline in a terminal. The command to start Appium for iOS is, e.g.:

```
appium --address 127.0.0.1 --port 4723 --default-capabilities '{"platformName":"iOS",
"noReset":"true", "automationName":"XCUITest"}'
```

Or:

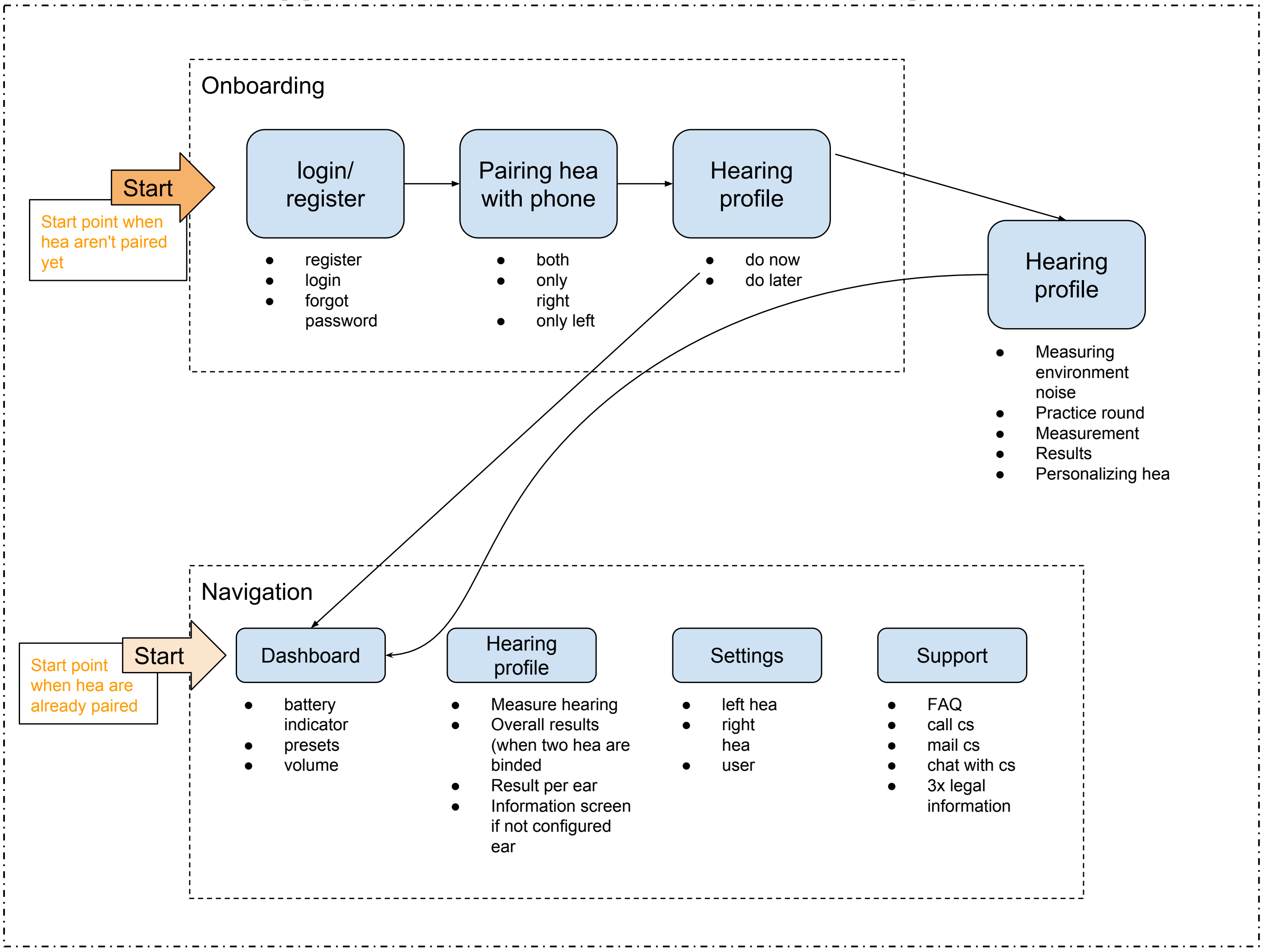
```
appium --address 127.0.0.1 --port 4723 --default-capabilities '{"platformName":"Android",
"noReset":"true"}'
```

noReset = false, means the app will be launced immediately without resetting the installation
fastReset = clears cache and settings without reinstall app
fullReset = uninstall and install app

- You can add or remove settings as you please, see http://appium.io/slate/en/master/?java#the-default-capabilities-flag
- http://appium.io/slate/en/master/?java#appium-server-arguments

3. AUTOMATION FRAMEWORK

3.1. Overview application structure and functionality



Timestamp: Thursday, April 20, 2017

3.2. Happy flow test scenario's

- Onboarding
  - New user
    - Register new user
      - Email verification with register code
    - Both hearing aids
      - With hearing test
    - One hearing aid
  - Existing user
    - Both hearing aids
      - Do later > Without hearing test
  - User login

    - Forgot password

- Change user settings
  - Forgot hearing aid
    - Pair a hearing aid
  - User logout

- Dashboard
  - Change program
  - Change volume
  - Validate battery state is shown

- Hearing test
  - Perform a practice round

- Support
  - Ask a question

3.3. Earl automation framework structure

Main - Java
All Java classes that contain

- Config – to load the xml config files
- Enumerations – e.g. which platforms that are available
- Logging – add logging to the test cases
- Pages – Page objects representations
- Utility – helper classes
  - Testbase – every test case extends this base
- Webdriverfactory – creation of webdrivers for running the test cases

Main – Resources - config
All config xml files for which devices, users and testlab the test suite should run on. In the Recources folder the xml files define with the config and in the config folder the config the resources files are loaded into the java classes.

Config
- DeviceProperties
- PropertyConfiguration
- UserProperties
- WebdriverProperties

Resources
- Devices
  - Android_emulated.xml
  - Android_physical.xml
  - iOS_emulated.xml
  - iOS_physical.xml
- Users
  - <every user has it's own account settings>
- Webdriver
  - RemoteWebDriver
  - AppiumDriver
  - AndroidDriver

- iOSDriver

App

Android
Have the latest apk file saved in the local file system:
../ src/app/android/earl.apk
The same apk file can be used for the emulator as on a real device.

iOS
Have the latest ipa and app file saved in the local file system:
../src/app/ios/earl.ipa
or
../src/app/ios/earl.app

For testing on a real device the ipa file is needed and for testing on a simulator the app file is needed.

Test
All test cases for Android and iOS platforms

POM.xml
The pom.xml is the project dependency to run the test suite.

```
<properties> => to define the default properties to run a test suite. The defaults can be overwritten by the commandline options from 3.5 (Running a test suite).

<profiles> => to define test groups of test cases. These tags can then be assigned to the individual test cases.
```

Terminal
Commandline to execute the TestNG suites.

Results and reporting
TestNG reports in HTML file with suite and results (past/failed).

3.4. Building new test cases

- Select a platform
- Select OS version
- Start appium (desktop or appium server on commandline/CLI)
- Download latest version app
- Install app
- Set breakpoints in your script
- Run your script on your emulator
- Inspect objects
- Write your script with assertions and unit tests
- Update method libraries
- Run your script on a physical device
- Debug your script
- Run your script on the set of cloud/local devices
- Debug your script
- Add your script to the regression set

3.5. Running test suite

The test groups are added to the test case, like @Test(groups = {**"smoke"**, **"android"**})
The groups that now are defined:

- Android
- iOS
- Smoke
- Development
- regression

The tests can be run from the IDE but also from the commandline with Maven. To run a single testcase the following command can be used:

```
mvn test -Dtest=Android_BindingTest
```

To run a group of test cases the following command can be used:

```
mvn test -Dgroups=smoke,android
```

or

```
mvn test –Dgroups=smoke
```

or

```
mvn test –Dgroups=android
```

To run a test suite that are groups inside a profile can run with the following command:

```
mvn test -Pandroid-smoke
```

Other parameters are:

```
-DwebDriverConfigType=local
-DuseSystemProps=false
-Dpermissions=true
-Dfile=app/android/Earl_v1.0.0_170630_debug.apk
-DdeviceConfigFile=emulator_Android_7.1.1
-DuserSettingsFile=marc
```

To select a specific testcase:

```
mvn test -Dtest=Android_SmokeTest
mvn test –Dtest=iOS_SmokeTest
```

To run a specific test method in a class:

```
mvn test -Dtest=Android_SmokeTest#earlShouldStart -DdeviceConfigFile=emulator_Android_7.1.1
mvn test –Dtest=iOS_SmokeTest#earlShouldStart –DdeviceConfigFile=emulator_iOS_10.3
```

3.6. Run reports

After running the maven reports in the target folder a surefire-report is created (index.html). With this report the test suite result can be found. There is also a report variant that can be shared (emailable-report.html).

3.7. Maintenance existing test cases

Within the framework there are fixme and todo comments. Beside these the following steps can be taken to update the testcases:

- Add a testcase in the Android or iOS test folder
- Add to the group <development>
- Validate the pages
- Run the inspector from Appium desktop (page names and buttons)
- Debug the new flows on a physical and emulated device
- If maintenance is done then the group can be changed to new test group like smoke test or regression
- Run the group with the new or updated test case

3.8. ToDo

- Android
  - logout flow after binding
  - add a soft assert, see https://rameshbaskar.wordpress.com/2013/09/11/soft-assertions-using-testng/
  -
- iOS
  - Create an mocking build in the CI build street and add it to HockeyApp
  - Validate mocking of app, many times the hearing aid can't be found
  - After binding, logout and login again you land on logout page instead of dashboard page
  - Add accessibility label on menu (tapbar)

Not included in the framework:

- Parallel execution of test cases or multiple devices
- Remote cloud device execution
- Create API to confirm new user email address or implement gmail API to receive the confirmation URL from the email
- Password field can't be filled automated on the login page
- With the mocking iOS app the permissions popup are not triggered and not tests

4. Configuration

- Accounts / licences
    - Apple developer
        - certificate to build apps
        - download Xcode + simulators + XCUITest
    - Testobject
        - Could device lab
- Tools
    - Xcode
        - 8.3.3
        - Commandline tools
        - Hardware IO Tools
    - Debug build iOS
        - https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/ConfiguringYourApp/ConfiguringYourApp.html
    - Brew (https://brew.sh/)
        - 1.1.11
    - Python ()
        - 3.0
    - node (https://nodejs.org/en/)
        - 7.5.0
    - npm
        - 4.2.1
    - Appium client
        - 4.2.1
    - Android studio (https://developer.android.com/studio/index.html)
        - ADB / emulator images
    - Selenium standalone server (http://docs.seleniumhq.org/download/)
        - 3.4.0
        - incl. selenium grid / Hub
    - Appium (http://appium.io/ and https://github.com/appium/appium-desktop)
        - Appium Desktop
    - Appium server / CLI (https://www.npmjs.com/package/appium)
        - 1.6.4
        - npm install -g appium
    - WebDriverAgent (https://github.com/facebook/WebDriverAgent)
        - Npm install appium-xcuitest-driver
        - brew install ideviceinstaller
        - brew install carthage
        - npm install -g ios-deploy
        - npm install -g deviceconsole
        - gem install xcpretty
        - brew install libimobiledevice --HEAD (iOS 10)
        - mkdir -p Resources/WebDriverAgent.bundle
        - sh ./Scripts/bootstrap.sh -d
        - WebDriverAgent.xcodeproj
        - Signing WebDriverAgent
            - https://github.com/appium/appium-xcuitest-driver/blob/master/docs/real-device-config.md
    - iOS and Android object inspector
        - Appium desktop app
    - Eclipse or IntelliJ

        - https://eclipse.org/
        - https://www.jetbrains.com/idea/download/
    - Java / JDK
        - 1.8.0_60
        - Set JAVA_HOME in .bash_profile
        -
    - TestNG (https://blog.jetbrains.com/idea/2006/06/testng-plugin/)
        - 6.10
    - Maven project config (http://maven.apache.org/install.html)
        - 3.3.9
    - TestObject (cloud account)
    - Android emulators (Android studio of Genymotion)
        - 5.1
        - 6.0.1
        - 7.1
    - iOS simulators
        - 10.3.5
        - 9.3.0
- Hardware
    - Mac laptop (MacOS 10.12.5)
    - USB hub (for at least 10 devices)
    - Power charger
    - Android/ios emulators
    - Multiple Android devices (4.4 till 7.1.2)
    - Multiple iOS devices (9.3 till 10.3)
- Devices
    - Prepare Android devices (USB debugging enabling)
    - Prepare iOs devices (developer inspection enabling)

5. Attachments

5.1. Selenium API

http://seleniumhq.github.io/selenium/docs/api/java/index.html

5.2. Appium API

https://github.com/appium/appium/blob/master/docs/en/writing-running-appium/caps.md

5.3. Setting JAVA_HOME

Open a terminal and run the following command
- vi ~/.bash_profile

- Make the following changes
  - If there is a line related to JAVA_HOME delete it.

  - Insert following line:

export JAVA_HOME=$(/usr/libexec/java_home)
export PATH=$JAVA_HOME:$PATH
  - Press esc and Use :wq to quit vi
- On the terminal type *source ~/.bash_profile* to refresh the profile.

- To test that JAVA_Home has been set correctly type the following commands and observe that the output

echo $JAVA_HOME
(In this case the output should be something like /Library/Java/JavaVirtualMachines/jdk1.8.0_65.jdk/Contents/Home)

5.4. Installing HAXM => to run emulators

https://software.intel.com/en-us/android/articles/installation-instructions-for-intel-hardware-accelerated-execution-manager-mac-os-x

5.5. Alternative iOS inspector

https://github.com/mykola-mokhnach/Appium-iOS-Inspector

5.6. Current implemented ID's

In the Git repository is an instruction folder with an iOS and an Android folder with an overview of the current implemented ID's. With these ID's Appium can find pages, buttons and perform actions.

5.7. Appium doctor

To validate that your system is setup properly for appium then run the appium doctor. To install the appium doctor:
- npm install -g appium-doctor

Starting the appium doctor from the commandline:
- appium-doctor

If there is an error like "Error: Could not detect Mac OS X Version from sw_vers output: '10.12.4" than the config of the appium doctor need to be updated.

Updating appium desktop version for newer Mac versions:

- run appium doctor
- Open the appium doctor config file:
- manual add new mac version to config file

See also the following discussion:https://discuss.appium.io/t/appium-doctor-returns-could-not-detect-mac-os-x-version-using-osx-10-10/1264/2