

Comparison of Testing Frameworks for a React Native App

July 16, 2020

Contents

Acronyms	VII
1 Comparison of Detox and Appium	1
1.1 Comparison criteria definitions	1
1.2 Description of the comparison	4
1.2.1 Costs	4
1.2.2 Setup time	4
1.2.3 Time to implement	4
1.2.4 Learning Curve	4
1.2.5 Adaptability & Extendibility	5
1.2.6 Flakiness	5
1.2.7 Overhead	5
1.2.8 Speed of tests	5
1.2.9 Coverage	6
1.2.10 iOS and Android compatibility	7
1.3 Result of the Comparison	7
2 Developer guidelines	8
2.1 Integrate Detox into an RN app for iOS and Android	8
2.2 Additional integration steps for iOS	8
2.3 Additional integration steps for Android	9
2.4 Error fixing	11
2.4.1 Package errors and similar errors	11
2.4.2 error: cannot find symbol public ActivityTestRule<MainActivity>(…)	12
2.4.3 Delegate runner 'androidx.test.internal.runner.junit4.AndroidJUnit4ClassRunner' for AndroidJUnit4 could not be loaded	12
2.4.4 Possible error on Arch Linux	12
2.5 Writing and running tests	12

Appendices	15
.1 Use Cases	16
.1.1 User Registration	16
.1.2 User Login	16
.1.3 Forms	17
.1.4 User swipes	18
.1.5 Navigate	18
.1.6 Zoom	19
.2 Test scenarios	20
.2.1 User Registration Test	20
.2.2 User Login Test	21
.2.3 Form Test input and validate text	21
.2.4 Form Test using a switch	22
.2.5 Form Test using a Dropdown menu	23
.2.6 Form Test scrolling downwards	23
.2.7 User Swipe Test	24
.2.8 User Navigation Test	24
.2.9 User Zoom Test	25
.3 Detox tests code	26
.3.1 Detox login test	26
.3.2 Detox registration test	27
.3.3 Detox forms test	28
.3.4 Detox swipe test	29
.3.5 Detox navigation test	31
.3.6 Detox zoom test	31
.4 Appium tests code	32
.4.1 Appium Login test	32
.4.2 Appium Registration test	33
.4.3 Appium forms test	34
.4.4 Appium swipe test	35
.4.5 Appium navigation test	37
.4.6 Appium zoom test	38
.5 Cavy tests code	39

.5.1	Cavy Login test	39
.5.2	Cavy SignUp test	40
.5.3	Cavy forms test	41
.6	Appium developer guidelines	42
.6.1	Integrate Appium into RN app	42
.6.2	Integrate Appium for iOS	43
.6.3	Integrate Appium for Android	43
.6.4	Writing tests	44
.6.5	Running tests	46

List of Figures

2.1	Buildgradle file	11
2.2	Detox login test	13
3	Detox login test - e2e/login.spec.js	26
4	Detox registration test - e2e/register.spec.js	27
5	Detox forms test - e2e/form.spec.js	28
6	Detox swipe test part 1 - e2e/swipe.spec.js	29
7	Detox swipe test part 2 - e2e/swipe.spec.js	30
8	Detox navigation test - e2e/backnav.spec.js	31
9	Detox zoom test - e2e/zoom.spec.js	31
10	Appium login test - tests/specs/app.login.spec.js	32
11	Appium registration test - tests/specs/app.register.spec.js	33
12	Appium forms test - tests/specs/app.forms.spec.js	34
13	Appium swipe test part 1 - tests/specs/app.swipe.spec.js	35
14	Appium swipe test part 2 - tests/specs/app.swipe.spec.js	36
15	Appium navigation test - tests/specs/app.backnav.spec.js	37
16	Appium zoom test - tests/specs/app.zoom.spec.js	38
17	Cavy Login test	39
18	Cavy SignUp test	40
19	Cavy forms test	41
20	Login test	44
21	Webdriver.io method	45
22	Mobile commands	45

List of Tables

1.1	Cases and their points	3
1.2	Implementation times	4
1.3	Speed iOS tests	5
1.4	Speed Android tests	6
1.5	Coverage of the frameworks	6
1.6	Matrix Framework comparison	7
1	User registration	16
2	User Login	16
3	User fills out form	17
4	User swipes	18
5	User navigates backwards	18
6	User zooms in and out of the picture	19
7	User registration test	20
8	User Login test	21
9	Form Test input and validate text	21
10	Form Test using a switch	22
11	Form Test using a Dropdown menu	23
12	Form Test scrolling downwards	23
13	User swipe test	24
14	User navigation test	24
15	User zoom test	25

Listings

2.1	package.json test-runner	8
2.2	package.json configuration iOS	9
2.3	build.gradle repositories	9
2.4	app/build.gradle	10
2.5	app/build.gradle defaultConfig	10
2.6	build.gradle buildscript	10
2.7	package.json configuration Android	10
2.8	testID and text	12
2.9	Dropdown	12
10	command for Android build	43

Acronyms

RN React Native. 1

Chapter 1

Comparison of Detox and Appium

Following you find the comparison criteria definitions, the detailed description of the comparison and the results of the comparison.

1.1 Comparison criteria definitions

Following are the definitions of the comparison criteria:

1. Costs contain how much you have to pay to be able to use the framework.
2. Set up time means, how long it takes to integrate the testing framework into an app and get a first example test running.
3. Time to implement means, how long it takes to write runnable tests.
4. The learning curve implies, how long it takes to be able to use the framework for writing and changing tests, which depends on the quality of the documentation, necessary preknowledge, if there is additional research required and if there is support found, like e.g. in forums.
5. Adaptability defines how quickly tests can be changed by the programmer.
6. Extendibility defines how quickly tests can be expanded by the programmer.
7. Flakiness means that the same tests may sometimes pass and sometimes fail even if there has not been a change of the code. This can happen for example due to network requests being made which have different response times. The less flaky the test is, the higher the grade will be.
8. Overhead implies, how much code is needed to implement a certain test.
9. The speed of the test runs concerns the time for a test run from start to finish.
10. Coverage implies how many functionalities are included in the testing framework, like e.g. methods for finding UI elements and methods for changing a certain UI element. It also takes into account if it was possible to implement the tests completely or if parts of it couldn't be implemented.
11. The Android and iOS compatibility means, if the tests are able to run on Android and iOS.

The criteria have different weights, on which there has been made a decision together with some of the team members who are also testing RN apps. The weighting depends on the importance of the comparison criterion, 1 is low importance, 2 medium and 3 high importance. In the calculation the points given for the criterion will

then be calculated *1, *2 or *3. As an example, the first criterion for Detox (costs) got 5 points and a weight of 1, so the calculation will be **5(points)*1(weight)=5**.

The following table 1.1 shows how many points are given for the criteria in which case. The points in the matrix 1.6 at the end of this chapter range from 1-5, while 1 is the worst and 5 the best possible value.

Criterion	Case	Points
Costs	No costs	5
	<= 1000 Euro/year	3
	>= 1000 Euro/year	1
Setup time	<=2 hours	5
	2- <5 hours	4
	5- <8 hours	3
	8-16 hours	2
	>16 hours	1
Time to implement	<=10 hours	5
	>10hours<=20hours	4
	>20hours<=30hours	3
	>30hours<=40hours	2
	>40hours	1
Learning curve	easy to understand, detailed documentation	5
	difficult to understand,detailed documentation	4
	difficult to understand, mediocre documentation	3
	difficult to understand, bad documentation	2
	difficult to understand, no documentation	1
Adaptability	easy to adapt	5
	medium difficulty to adapt	3
	difficult to adapt	1
Extendibility	easy to extend	5
	medium difficulty to extend	3
	difficult to extend	1
Flakiness	no flaky test	5
	one flaky test	4
	two flaky tests	3
	three flaky tests	2
	more than three flaky tests	1
Overhead	little amount of code needed	5
	medium amount of code needed	3
	great amount of code needed	1
Speed running tests	<1 minute-2 minutes	5
	>2 minutes -4minutes	3
	>4 minutes	1
Coverage	all tests can be implemented	5
	1-2 tests can't be fully implemented	4
	3-4 tests can't be fully implemented	3
	5-6 tests can't be fully implemented	2
	more than 6 tests can't be fully implemented	1
Android and iOS compatible	all tests run on iOS and Android	5
	1-2 tests run only on iOS or Android	4
	>2 tests run only on iOS or Android	3
	all tests run only on iOS or Android	2
	tests run neither on iOS nor Android	1

Table 1.1: Cases and their points

1.2 Description of the comparison

Following the detailed comparison is described.

1.2.1 Costs

Detox and Appium don't cost anything and are Open Source, which leads to a rating of 5 for both frameworks according to the table 1.1.

1.2.2 Setup time

On Detox setting up the app for iOS tests until it was possible to run the first included example test, was 30 minutes. For Android it took about one hour to get the first test running. This implementation time leads to a rating of 5.

To integrate Appium into the app took 12 hours, which leads to a rating of 2.

1.2.3 Time to implement

The following table 1.4 compares the implementation times of the tests.

Test	Detox	Appium
SignUp	30 minutes	1 hour
Login	30 minutes	30 minutes
Forms	8 hours	12 hours
Swipe	4 hours	4 hours
Navigation	3 hours	3 hours
Zoom	1 hour	2 hours
Total	17 hours	22,5 hours

Table 1.2: Implementation times

The sum of the implementation for all the Detox tests is 17 hours, so it is rated with 4. The tests with Appium took a few hours longer with 22,5 hours so it receives a 3.

1.2.4 Learning Curve

The documentation of Detox is detailed and easy to understand, which speeds up the learning process and leads to a rating of 5.

The documentation for Appium is not as well as the Detox documentation and all in all mediocre, because in general there was more additional research necessary. Also it was more difficult to get an overview and understanding of it, because you are using three different frameworks at once instead of one: Appium, Webdriver.io and Jasmine versus only Detox. So we have the case "difficult to understand and mediocre documentation" of the table 1.1, which leads to a rating of 3.

1.2.5 Adaptability & Extendibility

The tests can be easily extended by simply adding more test cases with the `it()`-clause and adding more `await element()`-statements. Also the tests are easy to adapt by using different ids or texts of the different UI elements and by replacing the various Detox methods like `replaceText()` by a different method.

Like with Detox, each test is contained in an `it()`-clause, so you can easily add more tests or extend tests by adding more `$()`-statements. Also the code can be quickly adapted to other UI elements by changing the label inside the `$()`-clauses and replacing the `webdriver.io` methods like for example `setValue()` by another method.

Both frameworks are easy to adapt and extend, which leads to a 5.

1.2.6 Flakiness

In Detox a lot of runs of the same tests always delivered the same result, so flakiness was not an issue, leading to a 5.

In Appium the swipe test had flakiness issues, sometimes the same code failed. Concerning the other tests there were no flakiness issues. Because there was only one flaky test, Appium gets rated with a 4.

1.2.7 Overhead

Detox only needs a little amount of code, which leads to a 5. Appium needs a lot more code in some cases, like for example for implementing the `touchPerform()` methods. In Appium for identifying UI-elements `Webdriver.io` needs `testIDs` for Android and `accessibilityLabels` for iOS. So compared to Detox, Appium needs more code to identify a UI element, because Detox needed only one identifier for both iOS and Android. Also compared to Detox Appium needs a lot more code to implement the scrolling and the swipe function for Android. In the test scenario UC-3-T.4 (Scrolling down), the navigation test and the swipe test there was different code needed for Android and iOS. All in all Appium needs a medium amount of code, leading to a rating of 3.

1.2.8 Speed of tests

Following the speed of the tests is shown. The first table contains the speed of the tests with iOS.

Test	Detox	Appium
SignUp&Login	26 seconds	27 seconds
Forms	20 seconds	24 seconds
Swipe	34 seconds	16 seconds
Navigation	20 seconds	10 seconds
Zoom	20 seconds	9 seconds
Total	2 minutes	1 minute 26 seconds

Table 1.3: Speed iOS tests

Now the speed of the tests with Android is shown.

Test	Detox	Appium
SignUp&Login	28 seconds	23 seconds
Forms	22 seconds	11 seconds
Swipe	not working	36 seconds
Navigation	48 seconds	8 seconds
Zoom	15 seconds	22 seconds
Total	1 minute 53 seconds	1 minute 40 seconds

Table 1.4: Speed Android tests

With Detox the navigation test finished on iOS in 20 seconds, but back navigation was not possible, so this shortens the test time. Also with Detox the zoom test on Android finished after 15 seconds, but zooming was not possible, so again this shortens the test time. The total speed of tests for Appium is 3 minutes and 6 seconds, leading to a 3. Detox needs 3 minutes and 53 seconds, which is also rated with a 3.

1.2.9 Coverage

In this section is described which functionalities are included in the frameworks. The methods of the frameworks used in the tests are shown in the table 1.5.

Functionality	Method Detox	Method Webdriver.io/Jasmine
Check visibility of UI element	toBeVisible()/toBeNotVisible()	isDisplayed()).toEqual(true/false)
Check text of an element	toHaveText()	text().toEqual(booleann)
Fetch UI element by text	by.text()	-
Fetch UI element by id	by.id()	\$('~testID')(iOS) \$('~accessibilityLabel')(Android)
Fetch text of UI element	-	getText()
Navigate backwards	device.pressBack(Android)	driver.touchPerform()
Paste text into text field	replaceText()	setValue()
Pause test execution	-	driver.pause(milliseconds)
Scrolling	scrollTo()	driver.touchPerform()
Swipe	swipe()	driver.touchPerform()
Tap element	tap()	click()
Wait for UI element to be shown	waitFor(element(by.id('SMS'))). toBeVisible().withTimeout (2000)	\$('~Home').waitForDisplayed (20000)
Zooming	pinchWithAngle()(iOS)	driver.multiTouchPerform()

Table 1.5: Coverage of the frameworks

As can be seen in the table, Detox and Appium, which is in our case combined with the Webdriver.io and Jasmine framework, cover a very similar amount of functionalities with their methods. Nevertheless, for Detox as well as for Appium there couldn't be found anything to implement the test scenarios UC-3-T.2 and UC-3-T.3, concerning the switch and the Dropdown menu.

Also with Appium, concerning the Alert you couldn't check the text of the Alert box because you couldn't

fetch the Alert element, only with an own, framework independent implementation. This was not an issue with Detox.

With Detox and Appium it was not possible to check a certain image size before and after zooming. So to conclude, because with Detox 3 tests had issues, this will result in 3 points. Appium has problems concerning 4 tests, which also results in 3 points.

1.2.10 iOS and Android compatibility

With Detox the swipe() method only works on iOS. Zooming in and out of a picture was only possible with iOS. For iOS back navigation from the external SMS app was not possible.

With Appium all the tests were able to run on iOS as well as on Android, so 5 points are given. Detox sometimes had problems with parts of three tests, so 3 points are given.

1.3 Result of the Comparison

The following table 1.6 shows the result of the comparison of the frameworks. As can be seen, Detox has 13 points more than Appium, meaning that according to the in section ?? determined definition of done, this will be the framework that the author recommends to codecentric.

Comparison criteria	Detox	Appium	Weight
1. Costs	5	5	1
2. Setup time	5	2	2
3. Time to implement	4	3	2
4. Learning curve	5	3	2
5. Adaptability	5	5	3
6. Extendibility	5	5	3
7. Flakiness	5	4	3
8. Overhead	5	3	2
9. Speed running tests	3	3	2
10. Coverage	3	3	3
11. Android and iOS compatible	3	5	3
Total sum	112	99	

Table 1.6: Matrix Framework comparison

Chapter 2

Developer guidelines

This chapter contains the developer guidelines for using the Detox framework. It will be described how you can integrate Detox into a working RN app and how you can write tests. First of all you need to have Android Studio installed. For the iOS tests you need macOS (at least macOS High Sierra 10.13.6) and Xcode 10.2+ with the Xcode command line tools. As the IDE the author used Visual Studio Code.

2.1 Integrate Detox into an RN app for iOS and Android

First you need to install Node 8.3.0 or above with `npm install node`. Install the Detox command line tools with `sudo npm install -g detox-cli`. Then at the root folder of your project install Detox with

```
npm install detox --save-dev.
```

As test-runners you can choose between Jest and Mocha. Depending on what you want either run `npm install jest --save-dev` or `npm install mocha --save-dev`. Depending on your choice you now need to run `detox init -r jest` or `detox init -r mocha`.

Add the following two lines to the package.json file between "detox" and "configurations":

```
"detox": {  
  "test-runner": "jest",  
  "runner-config": "e2e/config.json",  
  
  "configurations": {
```

Listing 2.1: package.json test-runner

If you run into trouble, the tutorial of Detox can be found on the Website

<https://github.com/wix/Detox/blob/b35ca64d07f3004ba8ed7ab6d0c8e5f1ba7b2edd/docs/Introduction.GettingStarted.md>

The link is containing the version that the author used, so if instead the newest version is needed, it can be found here:

<https://github.com/wix/Detox/blob/master/docs/Introduction.GettingStarted.md>

2.2 Additional integration steps for iOS

- Install Homebrew (<https://brew.sh/>)

- To ensure everything needed for Homebrew tool installation is installed, run
`xcode-select --install`
- Install applesimutils with

```
brew tap wix/brew
brew install applesimutils
```

In the in listing 2.2 shown package.json file, under the key "binaryPath", example.app should be <yourProjectName>.app. Under the key "build" change example.xcodeproj to <yourProjectName>.xcodeproj and -scheme example should be -scheme <yourProjectName>.

```
"detox": {
  "configurations": {
    "ios.sim.debug": {
      "binaryPath": "ios/build/Build/Products/Debug-iphonesimulator/
                    example.app",
      "build": "xcodebuild -project ios/example.xcodeproj -scheme
                example -configuration Debug -sdk iphonesimulator
                -derivedDataPath ios/build",
      "type": "ios.simulator",
      "device": {
        "type": "iPhone 11 Pro"
      }
    }
  }
}
```

Listing 2.2: package.json configuration iOS

"For React Native 0.60 or above, or any other iOS apps in a workspace (eg: CocoaPods) use -workspace ios/example.xcworkspace instead of -project.

Also make sure the simulator model specified under the key device.type (e.g. iPhone 11 Pro above) is actually available on your machine (it was installed by Xcode). Check this by typing `applesimutils -list` in the terminal to display all available simulators." (Detox 2020)

2.3 Additional integration steps for Android

In the android root buildscript (i.e. build.gradle), register google() and detox:

```
// Note: add the 'allproject' section if it doesn't exist
allprojects {
  repositories {
    // ...
    google()
    maven {
      // All of Detox' artifacts are provided via the npm module
      url "$rootDir/../../node_modules/detox/Detox-android"
    }
  }
}
```

Listing 2.3: build.gradle repositories

In the app's buildscript (i.e. app/build.gradle) add this in the dependencies section:

```
dependencies {
    // ...
    androidTestImplementation('com.wix:detox:+')
}
```

Listing 2.4: app/build.gradle

In the app's buildscript (i.e. app/build.gradle) add this to the defaultConfig section:

```
android {
    defaultConfig {
        // ...
        testBuildType System.getProperty('testBuildType', 'debug')
        testInstrumentationRunner 'androidx.test.runner.AndroidJUnitRunner'
    }
}
```

Listing 2.5: app/build.gradle defaultConfig

Add the latest Kotlin version (the author used 1.3.41) to your classpath in the root build-script (i.e. android/build.gradle):

```
buildscript {
    // ...
    ext.kotlinVersion = '1.3.41'

    dependencies {
        // ...
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:
$kotlinVersion"
    }
}
```

Listing 2.6: build.gradle buildscript

Create the following directory. add the file android/app/src/androidTest/java/com/[your.package]/DetoxTest.java and fill in the code from here:

<https://github.com/wix/Detox/blob/master/examples/demo-react-native/android/app/src/androidTest/java/com/example/DetoxTest.java>

In this file change the package name to your project's name, which can be found in your AndroidManifest.xml file at the top (package="nameOfyourapp").

Add this to your package.json:

```
"detox" : {
  "configurations": {
    "android.emu.debug": {
      "binaryPath": "android/app/build/outputs/apk/debug/app-debug.apk",
      "build": "cd android && ./gradlew assembleDebug assembleAndroidTest -DtestBuildType=debug && cd ..",
      "type": "android.emulator",
      "device": {
```

```

        "avdName": "Pixel_API_29"
    },
    "android.emu.release": {
        "binaryPath": "android/app/build/outputs/apk/release/app-release.apk",
        "build": "cd android && ./gradlew assembleRelease assembleAndroidTest -DtestBuildType=release && cd ..",
        "type": "android.emulator",
        "device": {
            "avdName": "Pixel_API_29"
        }
    }
}
}
}

```

Listing 2.7: package.json configuration Android

As shown in figure 2.1 the minSdkVersion needs to be set to 18 in the /android/build.gradle file.

```

android > build.gradle
1  // Top-level build file where you can add
2
3  buildscript {
4      ext.kotlinVersion = '1.3.41'
5      ext {
6          buildToolsVersion = "28.0.3"
7          minSdkVersion = 18

```

Figure 2.1: Buildgradle file

If you run into problems, the tutorial for integrating Detox for Android can be found on the Website <https://github.com/wix/Detox/blob/b35ca64d07f3004ba8ed7ab6d0c8e5f1ba7b2edd/docs/Introduction.Android.md>

Again, the link is containing the version that the author used, so if instead the newest version is needed, it can be found here:

<https://github.com/wix/Detox/blob/master/docs/Introduction.GettingStarted.md>

2.4 Error fixing

Following are some possible errors and how to fix them.

2.4.1 Package errors and similar errors

In general it can be said that the first action that fixed package errors and similar errors, was to delete the node-modules folder and run "yarn install".

2.4.2 error: cannot find symbol public ActivityTestRule<MainActivity>(...)

For this error make sure that the paths to the DetoxTest.java file and the MainActivity.java file both are the same in the last three subdirectories so like e.g. this:

```
(...)/java/com/yourapp/MainActivity.java and
(...)/java/com/yourapp/DetoxTest.java and
```

Also the package name at the top of these files needs to be the same.

2.4.3 Delegate runner 'androidx.test.internal.runner.junit4.AndroidJUnit4ClassRunner' for AndroidJUnit4 could not be loaded

Here it can help to delete the line `@RunWith (AndroidJUnit4.class)` and to add an empty constructor in the file DetoxTest.java like this: `public DetoxTest () {}`

2.4.4 Possible error on Arch Linux

When setting up Detox on Arch Linux the author encountered the error "Could not find any matches for com.wix:detox:+ as no versions of com.wix:detox are available.". The error could be solved by deleting the `node_modules` folder and the `package-lock.json` file and running `yarn install`.

2.5 Writing and running tests

You can specify which device or simulator you want in the file `package.json`. If you want to create a new test file, do this inside of the folder `<Path to your project>/e2e` and name it like this: `"nameofTest.spec.js"`. In general, Detox uses Matchers to find UI elements, Actions to emulate user interactions with those elements and Expectations to verify values on those elements (Detox 2020).

Following will be shown how the Login test was implemented as an example. This is the code:

In line 1 you can put a name as a parameter inside of the `describe()`-method to describe this test file. Line 2 and 3 mean that before each test the app will be reloaded, which is recommended to do like this. In line 6 and 7 you can define one test with a name as a parameter of the `it()`-method. Detox recommends to always identify elements by an id so we can grab the element with the matcher `by.id()`. Only if it doesn't work we should use other methods, like `by.text()` which grabs an element by its text property, e.g.

```
<Button testID={ 'Login ' }/> vs. <Button text={ 'Login ' }/>
```

Listing 2.8: testID and text

Sometimes if Detox can't find the UI element, this can be solved by wrapping the element inside of a view tag and adding the testID to the View tag like so:

```
<View testID='Dropdown'>
  <SelectInput label="Dropdown"/>
</View>
```

Listing 2.9: Dropdown

```

1  describe('Login test', () => {
2    beforeEach(async () => {
3      await device.reloadReactNative();
4    });
5
6    it('should be able to fill in forms, show an alert and login successfully',
7    async () => {
8
9      await element(by.text('Login')).tap();
10     await element(by.id('button-login-container')).tap();
11
12     await element(by.id('input-email')).replaceText('hi@test.de');
13     await element(by.id('input-password')).replaceText('Test1234!');
14
15     await element(by.text('LOGIN')).tap();
16
17     await expect(element(by.text('Success', 'You are logged in!'))).toBeVisible();
18     await element(by.text('OK')).tap();
19
20   });
21 }

```

Figure 2.2: Detox login test

In the lines 9 and 10 we tap on the chosen elements with the `tap()`-method. In the lines 12 and 13 we replace the text of the chosen UI elements with the `replaceText()`-method. In line 17 we check if the chosen UI element is visible with the `toBeVisible()`-method.

Start Android studio and run the Android device simulator. To build your app run

`detox build -c android.emu.debug` for Android and `detox build -c ios.sim.debug` for iOS. Before you can run the tests, you need to run `react-native start` at the root folder of your RN project. In a second terminal you can start the tests with the command `detox test -c ios.sim.debug` for iOS and with `detox test -c android.emu.debug` for Android. If you only want to run a certain test, specify the path to this test like e.g. so:

```
detox test -c ios.sim.debug Path_to_your_app/e2e/firstTest.spec.js
```

Keep in mind that if you changed other code of your RN app than the code of your test files, you need to build your app again before running the tests, so the tests are running on the changed code. You can do this with the command `react-native run-android` for Android and with `react-native run-ios` for iOS.

The whole Detox documentation can be found here:

<https://github.com/wix/Detox/tree/master/docs>

The Detox example app from the author can be found at these two repositories ("wdioDemoApp"):

<https://gitlab.codecentric.de/anja.bender/react-native-app/-/tree/master>

https://github.com/Lumary2/Detox_and_Appium/tree/master

Bibliography

Detox (2020), 'Detox documentation'. Accessed: 12/04/2020.
URL: *<https://github.com/wix/Detox/tree/master/docs>*

Appendices

.1 Use Cases

.1.1 User Registration

Code	UC-1
Actor	The user
Description	The user registers for a new account.
Preconditions	The Sign up screen is shown.
Scenario	<ol style="list-style-type: none"> 1. System shows the input fields for registering(Email, password, repeat password). 2. User fills in the input fields and taps on the button 'Sign up'. 3. System shows a Pop-up saying "you successfully signed up!". 4. The user clicks on 'ok'. 5. The Pop-up disappears.
Exceptions	<ol style="list-style-type: none"> 2. The system shows a pop-up saying 'Some fields are not valid'. <ol style="list-style-type: none"> 2.1 The user taps on 'try again'. 2.2 Return to step 1.
Result	The user has been registered.

Table 1: User registration

.1.2 User Login

Code	UC-2
Actor	The user
Description	The user logs in into the app.
Preconditions	The Login screen is shown.
Scenario	<ol style="list-style-type: none"> 1. System shows the input fields for login(Email, password). 2. User fills in the input fields and taps on the button 'Login'.
Exceptions	<ol style="list-style-type: none"> 2. The system shows an error message about an invalid input. <ol style="list-style-type: none"> 2.1 Return to step 1.
Result	The user has logged in.

Table 2: User Login

.1.3 Forms

Code	UC-3
Actor	The user
Description	The user fills out a form.
Preconditions	The Form screen is shown.
Scenario	<ol style="list-style-type: none"> 1. System shows the From screen. 2. User fills in the input fields. 3. System shows the input in the input field and another input field. 4. User taps on the Switch. 5. System shows that the Switch is activated. 6. User taps on the Dropdown menu. 7. System shows the selection of the Dropdown menu. 8. User selects an option from the Dropdown menu. 9. System shows the selected option in the Dropdown field. 10. User scrolls downwards. 11. System scrolls down the screen, if possible and shows at least one Button(depending on screen size).
Result	The user filled out the form.

Table 3: User fills out form

.1.4 User swipes

Code	UC-4
Actor	The user
Description	The user swipes left and right.
Preconditions	The Swipe screen is shown.
Scenario	<ol style="list-style-type: none"> 1. System shows a picture with text. 2. User swipes left. 3. Step 1 and 2 are repeated five times. 4. System shows a picture with text. 5. User swipes right. 6. Step 4 and 5 are repeated five times. 7. System shows a picture with text.
Result	The user has swiped left and right.

Table 4: User swipes

.1.5 Navigate

Code	UC-5
Actor	The user
Description	The user navigates backwards.
Preconditions	None
Scenario	<ol style="list-style-type: none"> 1. System shows the Home screen. 2. User taps on 'SMS'. 3. System shows the screen where you can write SMS. 4. User taps on the area enabling backnavigation. 5. System shows the Home screen.
Result	The user navigated backwards.

Table 5: User navigates backwards

.1.6 Zoom

Code	UC-6
Actor	The user
Description	The user zoomed in and out of a picture.
Preconditions	None
Scenario	<ol style="list-style-type: none"> 1. System shows the Home screen. 2. User taps on the WebView button. 3. System shows the WebView screen with a picture. 4. User zooms in on the picture. 5. System shows the enlarged picture. 6. User zooms out of the picture. 7. System shows the picture with its original size.
Result	The user zoomed in and out of the picture.

Table 6: User zooms in and out of the picture

.2 Test scenarios

.2.1 User Registration Test

Code	UC-1-T.1
Test Type	E2E test
Description	Tests if user can register.
Preconditions	None
E2E test case	<ol style="list-style-type: none"> 1. User taps on Login icon. 2. System show Login page. 3. User taps on Sign up text field. 4. System shows the input fields for registering(Email, password, repeat password). 5. User fills in "hi@test.de" as email and "Test1234" as password and as confirmed password. 6. System shows a Pop-up saying "you successfully signed up!". 7. The user clicks on 'ok'. 8. The Pop-up disappears.

Table 7: User registration test

.2.2 User Login Test

Code	UC-2-T.1
Test Type	E2E test
Description	Tests if user can login.
Preconditions	User is registered
E2E test case	<ol style="list-style-type: none"> 1. User taps on Login icon. 2. System show Login page. 3. User fills in "hi@test.de" as email and "Test1234" as password. 4. System shows a Pop-up saying "you are logged in!". 5. The user clicks on 'ok'. 6. The Pop-up disappears.

Table 8: User Login test

.2.3 Form Test input and validate text

Code	UC-3-T.1
Test Type	E2E test
Description	Tests if user can input text.
Preconditions	None
E2E test case	<ol style="list-style-type: none"> 1. System shows the From screen. 2. User fills 'Hello, this is a demo app' in the input field. 3. System shows the input in the input field and in the other input field.

Table 9: Form Test input and validate text

.2.4 Form Test using a switch

Code	UC-3-T.2
Test Type	E2E test
Description	Tests if user can use a switch.
Preconditions	None
E2E test case	<ol style="list-style-type: none">1. System shows the form screen.2. User taps on the Switch.3. System shows that the Switch is activated.

Table 10: Form Test using a switch

.2.5 Form Test using a Dropdown menu

Code	UC-3-T.3
Test Type	E2E test
Description	Tests if user can use a dropdown menu.
Preconditions	None
E2E test case	<ol style="list-style-type: none"> 1. System shows the form screen. 2. User taps on the Dropdown menu. 3. System shows the selection of the Dropdown menu. 4. User selects the option 'This app is awesome' from the Dropdown menu. 5. System shows the selected option in the Dropdown field.

Table 11: Form Test using a Dropdown menu

.2.6 Form Test scrolling downwards

Code	UC-3-T.4
Test Type	E2E test
Description	Tests if user can scroll downwards.
Preconditions	None
E2E test case	<ol style="list-style-type: none"> 1. System shows the Form screen. 2. User scrolls downwards. 3. System scrolls down the screen, if possible and show at least one Button at the bottom (depending on screen size).

Table 12: Form Test scrolling downwards

.2.7 User Swipe Test

Code	UC-4-T.1
Test Type	E2E test
Description	Tests if user can swipe left and right.
Preconditions	None
E2E test case	<ol style="list-style-type: none"> 1. User taps on Swipe icon. 2. System shows picture with text 'FULLY OPEN SOURCE'. 3. User swipes left. 4. The steps 2 and 3 are repeated five times and the system shows picture with these text in this order: 'CREATE COMMUNITY', 'JS.FOUNDATION', 'SUPPORT VIDEOS', 'EXTENDABLE', 'COMPATIBLE'. 5. System shows picture with text 'COMPATIBLE'. 6. User swipes right. 7. The steps 5 and 6 are repeated five times and the system shows picture with these text in this order: 'COMPATIBLE', 'EXTENDABLE', 'SUPPORT VIDEOS', 'JS.FOUNDATION', 'CREATE COMMUNITY'.

Table 13: User swipe test

.2.8 User Navigation Test

Code	UC-5-T.1
Test Type	E2E test
Description	Tests if user can navigate backwards.
Preconditions	None
E2E test case	<ol style="list-style-type: none"> 1. System shows the Home screen. 2. User taps on 'SMS'. 3. System shows the screen where you can write SMS. 4. User taps on the area enabling backnavigation. 5. System shows the Home screen.

Table 14: User navigation test

.2.9 User Zoom Test

Code	UC-6-T.1
Test Type	E2E test
Description	Tests if user can zoom in and out of a picture.
Preconditions	None
E2E test case	<ol style="list-style-type: none">1. System shows the Home screen.2. User taps on the WebView button.3. System shows the WebView screen with a picture.4. User zooms in on the picture.5. System shows the enlarged picture.6. User zooms out of the picture.7. System shows the picture with its original size.

Table 15: User zoom test

.3 Detox tests code

Following you find all the code written in the Detox framework. The whole project can be found at these two repositories ("wdioDemoApp"):

<https://gitlab.codecentric.de/anja.bender/react-native-app/-/tree/master>

https://github.com/Lumary2/Detox_and_Appium/tree/master

In each caption is the path to the file.

.3.1 Detox login test

```
1 describe('Login test', () => {
2   beforeEach(async () => {
3     await device.reloadReactNative();
4   });
5
6   it('should be able to fill in forms, show an alert and login successfully',
7     async () => {
8
9     await element(by.text('Login')).tap();
10    await element(by.id('button-login-container')).tap();
11
12    await element(by.id('input-email')).replaceText('hi@test.de');
13    await element(by.id('input-password')).replaceText('Test1234!');
14
15    await element(by.text('LOGIN')).tap();
16
17    await expect(element(by.text('Success', 'You are logged in!'))).toBeVisible();
18    await element(by.text('OK')).tap();
19  });
20 }
21
```

Figure 3: Detox login test - e2e/login.spec.js

.3.2 Detox registration test

```
1 describe('Register test', () => {
2   beforeEach(async () => {
3     await device.reloadReactNative();
4   });
5
6   it('should be able to fill in forms, show an alert and sign up successfully',
7     async () => {
8
9     await element(by.text('Login')).tap();
10    await element(by.id('button-sign-up-container')).tap();
11
12    await element(by.id('input-email')).replaceText('hi@test.de');
13    await element(by.id('input-password')).replaceText('Test1234!');
14    await element(by.id('input-repeat-password')).replaceText('Test1234!');
15
16    await element(by.text('SIGN UP')).tap();
17
18    await expect(element(by.text('Signed Up!',
19      'You successfully signed up!'))).toBeVisible();
20    await element(by.text('OK')).tap();
21  });
22 }
```

Figure 4: Detox registration test - e2e/register.spec.js

.3.3 Detox forms test

```

2 | describe('Forms test', () => {
3 |   beforeEach(async () => {
4 |     await device.reloadReactNative();
5 |     await element(by.text('Forms')).tap();
6 |
7 |   });
8 |
9 |   it('should be able to fill in the input and validate the text', async () => {
10 |
11 |     const text = 'Hello, this is a demo app';
12 |     await element(by.id('text-input')).replaceText(text);
13 |     await expect(element(by.id('input-text-result'))).toHaveText(text);
14 |
15 |   });
16 |
17 |   it('should be able to turn on the switch', async () => {
18 |
19 |     await expect(element(by.id('switch'))).toHaveValue('0');
20 |     element(by.id('switch')).tap();
21 |     expect(element(by.id('switch'))).toHaveValue('1');
22 |
23 |   });
24 |
25 |   it('should be able to select a value from the select element', async () => {
26 |     const valueOne = 'This app is awesome';
27 |     await expect(element(by.text(valueOne))).toBeNotVisible();
28 |     await element(by.id('Dropdown')).tap();
29 |     await element(by.text(valueOne)).tap();
30 |
31 |     /* next line finds multiple matching elements, so it throws error.
32 |     proves that text is there though.*/
33 |     //await expect(element(by.text(valueOne))).toBeVisible();
34 |
35 |   });
36 |
37 |   it('should be able to scroll to bottom', async () => {
38 |
39 |     await element(by.id('Forms-screen')).scrollTo('bottom');
40 |     await expect(element(by.id('activeButton'))).toBeVisible();
41 |
42 |   });
43 |

```

Figure 5: Detox forms test - e2e/form.spec.js

3.4 Detox swipe test

The test continues on the next page.

```
1 describe('Swipe test', () => {
2   beforeEach(async () => {
3     await device.reloadReactNative();
4     await element(by.text('Swipe')).tap();
5   });
6
7   /*Swiping works only on iOS, on Android it jumps back to the previous card.*/
9   it('should be able to swipe from left to right and back and contain correct text',
10    async () => {
11      text1 = 'FULLY OPEN SOURCE';
12      text2 = 'CREATE COMMUNITY';
13      text3 = 'JS.FOUNDATION';
14      text4 = 'SUPPORT VIDEOS';
15      text5 = 'EXTENDABLE';
16      text6 = 'COMPATIBLE';
17
18
19      await expect(element(by.text(text1))).toBeVisible();
20      await element(by.text(text1)).swipe('left');
21
22      await expect(element(by.text(text2))).toBeVisible();
23      await element(by.text(text2)).swipe('left');
24
25      await expect(element(by.text(text3))).toBeVisible();
26      await element(by.text(text3)).swipe('left');
27    });
28 }
```

Figure 6: Detox swipe test part 1 - e2e/swipe.spec.js

```
28     await expect(element(by.text(text4))).toBeVisible();
29     await element(by.text(text4)).swipe('left');
30
31     await expect(element(by.text(text5))).toBeVisible();
32     await element(by.text(text5)).swipe('left');
33
34     await expect(element(by.text(text6))).toBeVisible();
35
36     //swipe back and check text
37     await element(by.text(text6)).swipe('right');
38
39     await expect(element(by.text(text5))).toBeVisible();
40     await element(by.text(text5)).swipe('right');
41
42     await expect(element(by.text(text4))).toBeVisible();
43     await element(by.text(text4)).swipe('right');
44
45     await expect(element(by.text(text3))).toBeVisible();
46     await element(by.text(text3)).swipe('right');
47
48     await expect(element(by.text(text2))).toBeVisible();
49     await element(by.text(text2)).swipe('right');
50
51     await expect(element(by.text(text1))).toBeVisible();
```

Figure 7: Detox swipe test part 2 - e2e/swipe.spec.js

.3.5 Detox navigation test

```

 9      it('should be able to open app inside of app and navigate backwards', async () => {
10          await element(by.id('SMS')).tap();
11
12          // Following statement can only be used with iOS, Android throws error.
13          //await waitFor(element(by.id('SMS'))).toBeNotVisible().withTimeout(2000);
14
15          try {
16              await waitFor(element(by.id('SMS'))).toBeVisible().withTimeout(2000);
17          }
18          catch (e) {
19              console.log("SMS button is now hidden by another window.")
20          }
21          /*method is only for Android, has to be pressed twice because first keyboard
22          gets closed*/
23          await device.pressBack();
24          await device.pressBack();
25
26          //next line can only be used for Android because iOS cant backnavigate
27          //automatized
28          await expect(element(by.id('SMS'))).toBeVisible();
29      }

```

Figure 8: Detox navigation test - e2e/backnav.spec.js

.3.6 Detox zoom test

```

1  describe('Picture and zooming in and out test', () => {
2      beforeEach(async () => {
3          await device.reloadReactNative();
4      });
5
6      it('should be able to show picture and zoom in and zoom out of it', async () => {
7
8          await element(by.text('PictureView')).tap();
9
10         //pinchWithAngle()-method is only for iOS, (direction,speed,angle)
11         await element(by.type('RCTImageView')).pinchWithAngle('outward', 'slow', 0);
12
13         await element(by.type('RCTImageView')).pinchWithAngle('inward', 'slow', 0);

```

Figure 9: Detox zoom test - e2e/zoom.spec.js

.4 Appium tests code

Following you find all the code written in the Appium framework. The whole project can be found at these two repositories ("native-demo-app-master_RN_60"):

<https://gitlab.codecentric.de/anja.bender/react-native-app/-/tree/master>

https://github.com/Lumary2/Detox_and_Appium/tree/master

In each caption is the path to the file.

.4.1 Appium Login test

```
5 describe('WebdriverIO and Appium, when interacting with a login form,', () => {
6   beforeEach(() => {
7     $('~Home').waitForDisplayed(20000);
8     $('~Login').click();
9   });
10
11   it('should be able to fill in forms, show an alert and login successfully', () => {
12
13     $('~button-login-container').click();
14     $('~input-email').setValue('hi@test.de');
15     $('~input-password').setValue('Test1234!');
16
17     if (driver.isKeyboardShown()) {
18       driver.hideKeyboard();
19     }
20     $('~button-LOGIN').click();
21     NativeAlert.waitForIsShown();
22     expect(NativeAlert.text()).toEqual('Success\nYou are logged in!');
23
24     NativeAlert.pressButton('OK');
25     //for Android you could also use this command:
26     //driver.execute('mobile: acceptAlert');
27
28     //and for iOS:
29     //driver.execute('mobile: alert', {'action': 'accept'});
30
31     NativeAlert.waitForIsShown(false);
```

Figure 10: Appium login test - tests/specs/app.login.spec.js

.4.2 Appium Registration test

```
5 describe('WebdriverIO and Appium, when interacting with a login form,', () => {
6   beforeEach(() => {
7     $('~Home').waitForDisplayed(20000);
8     $('~Login').click();
9   });
10
11   it('should be able to fill in forms, show an alert and sign up successfully', () => {
12     $('~button-sign-up-container').click();
13     $('~input-email').setValue('hi@test.de');
14     $('~input-password').setValue('Test1234!');
15     $('~input-repeat-password').setValue('Test1234!');
16
17     if (driver.isKeyboardShown()) {
18       driver.hideKeyboard();
19     }
20     $('~button-SIGN UP').click();
21     NativeAlert.waitForIsShown();
22     expect(NativeAlert.text()).toEqual('Signed Up!\nYou successfully signed up!');
23
24     NativeAlert.pressButton('OK');
25     //for Android you could also use this command:
26     //driver.execute('mobile: acceptAlert');
27
28     //and for iOS:
29     //driver.execute('mobile: alert', {'action': 'accept'});
30     NativeAlert.waitForIsShown(false);
```

Figure 11: Appium registration test - tests/specs/app.register.spec.js

.4.3 Appium forms test

```

11 describe('WebdriverIO and Appium, interacting with form elements', () => {
12   beforeEach(() => {
13     $('~Home').waitForDisplayed(20000);
14     $('~Forms').click();
15   });
16
17   it('should be able to fill in the input and validate the text', () => {
18     const text = 'Hello, this is a demo app';
19     $('~text-input').setValue(text);
20     expect($('~input-text-result').getText()).toEqual(text);
21
22     if (driver.isKeyboardShown()) {
23       driver.hideKeyboard();
24     }
25   });
26
27   it('should be able to turn on the switch', () => {
28     expect(FormScreen.isSwitchActive()).toEqual(false);
29     FormScreen.switch.click();
30     expect(FormScreen.isSwitchActive()).toEqual(true);
31   });
32
33   it('should be able to select a value from the selected element', () => {
34     const valueOne = 'This app is awesome';
35
36     $('~select-Dropdown').click();
37     FormScreen.picker.selectValue(valueOne);
38     expect(FormScreen.getDropDownText()).toContain(valueOne);
39   });
40
41   it('should be able to scroll down', () => {
42     //coordinates for Android, when testing with iOS comment this block out.
43     driver.touchPerform([
44       { action: 'press', options: { x: 459, y: 1296 } },
45       { action: 'wait', options: { ms: 1000 } },
46       { action: 'moveTo', options: { x: 459, y: 229 } },
47       { action: 'release' }
48     ]);
49
50     //following line works only on iOS.
51     //driver.execute('mobile: scroll', { direction: 'down' });
52     expect($('~button-Active').isDisplayed()).toEqual(true);
53   });
54
55
56
57
58

```

Figure 12: Appium forms test - tests/specs/app.forms.spec.js

.4.4 Appium swipe test

The test continues on the next page.

```

4 describe('WebdriverIO and Appium', () => {
5   beforeEach(() => {
6     $('~Home').waitForDisplayed(20000);
7     $('~Swipe').click();
8   });
9
10
11   it('should be able to swipe from left to right and back and contain correct text',
12     () => {
13
14     text1 = 'FULLY OPEN SOURCE';
15     text2 = 'CREAT COMMUNITY';
16     text3 = 'JS.FOUNDATION';
17     text4 = 'SUPPORT VIDEOS';
18     text5 = 'EXTENDABLE';
19     text6 = 'COMPATIBLE';
20
21     expect($('~title').getText()).toEqual(text1);
22
23     driver.touchPerform([
24       { action: 'press', options: { x: 864, y: 766 } },
25       { action: 'wait', options: { ms: 1000 } },
26       { action: 'moveTo', options: { x: 216, y: 766 } },
27       { action: 'release' },
28     ]);
29     //also possible for iOS:
30     //driver.execute('mobile: swipe', { direction: 'left' });
31     driver.pause(1000);
32
33     expect($('~title').getText()).toEqual(text2);
34
35     SwipeScreen.carousel.swipeLeft();
36     expect($('~title').getText()).toEqual(text3);
37
38     SwipeScreen.carousel.swipeLeft();
39     expect($('~title').getText()).toEqual(text4);
40

```

Figure 13: Appium swipe test part 1 - tests/specs/app.swipe.spec.js

```
41 SwipeScreen.carousel.swipeLeft();
42 expect($('~title').getText()).toEqual(text5);
43
44 SwipeScreen.carousel.swipeLeft();
45 expect($('~title').getText()).toEqual(text6);
46
47 /*You can replace the statement SwipeScreen.carousel.swipeRight()
48 with the following
49 4 lines.*/
50 driver.touchPerform([
51   { action: 'press', options: { x: 216, y: 766 } },
52   { action: 'wait', options: { ms: 1000 } },
53   { action: 'moveTo', options: { x: 864, y: 766 } },
54   { action: 'release' }
55 ]);
56
57 expect($('~title').getText()).toEqual(text5);
58 SwipeScreen.carousel.swipeRight();
59
60 expect($('~title').getText()).toEqual(text4);
61 SwipeScreen.carousel.swipeRight();
62
63 expect($('~title').getText()).toEqual(text3);
64 SwipeScreen.carousel.swipeRight();
65
66 expect($('~title').getText()).toEqual(text2);
67 SwipeScreen.carousel.swipeRight();
68
69 expect($('~title').getText()).toEqual(text1);
```

Figure 14: Appium swipe test part 2 - tests/specs/app.swipe.spec.js

.4.5 Appium navigation test

```
5 describe('Backnavigation and app in app test', () => {
6   beforeEach(() => {
7     $('~Home').waitForDisplayed(20000);
8   });
9
10  it('should be able to open app inside of app and navigate backwards', () => {
11
12    $('~sms').click();
13
14    driver.pause(2000);
15
16    //checks if sms button is now hidden
17    expect($('~sms').isDisplayed()).toEqual(false);
18
19    //only for Android. press back two times because first keyboard gets closed.
20    driver.pressKeyCode(4);
21    driver.pressKeyCode(4);
22
23    //driver.touchPerform for iOS (iPhone X coordinates, relative to top left
24    //corner)
25    /*driver.touchPerform([
26      { action: 'press', options: { x: 54, y: 20 } },
27      { action: 'wait', options: { ms: 1000 } },
28      { action: 'release' },
29    ]);*/
30    driver.pause(2000);
```

Figure 15: Appium navigation test - tests/specs/app.backnav.spec.js

.4.6 Appium zoom test

```
5 describe('Picture and zooming in and out test', () => {
6   beforeEach(() => {
7     $('~Home').waitForDisplayed(20000);
8   });
9
10
11   it('should be able to show picture and zoom in and zoom out of it', () => {
12     $('~PictureView').click();
13     driver.pause(2000);
14
15     driver.multiTouchPerform([
16       [{ action: 'press', options: { x: 150, y: 342 } }],
17       { action: 'wait', options: { ms: 1000 } },
18       { action: 'moveTo', options: { x: 100, y: 342 } },
19       { action: 'wait', options: { ms: 1000 } },
20       { action: 'release' },
21     ],
22     [{ action: 'press', options: { x: 150, y: 342 } },
23       { action: 'wait', options: { ms: 1000 } },
24       { action: 'moveTo', options: { x: 200, y: 342 } },
25       { action: 'wait', options: { ms: 1000 } },
26       { action: 'release' },
27     ],
28   );
29
30   driver.multiTouchPerform([
31     [{ action: 'press', options: { x: 100, y: 342 } }],
32     { action: 'wait', options: { ms: 1000 } },
33     { action: 'moveTo', options: { x: 150, y: 342 } },
34     { action: 'wait', options: { ms: 1000 } },
35     { action: 'release' },
36   ],
37   [{ action: 'press', options: { x: 200, y: 342 } },
38     { action: 'wait', options: { ms: 1000 } },
39     { action: 'moveTo', options: { x: 150, y: 342 } },
40     { action: 'wait', options: { ms: 1000 } },
41     { action: 'release' },
```

Figure 16: Appium zoom test - tests/specs/app.zoom.spec.js

.5 Cavy tests code

Following you find all the code written in the Cavy framework.

.5.1 Cavy Login test

```
1  export default function (spec) {  
2  
3    spec.describe('Login test', function () {  
4  
5      spec.it('should be able to fill in forms, show an alert and login successfully',  
6        async function () {  
7          /*tap on Login button in Nav bar couldnt be accomplished because UI  
8             element couldnt be fetched, so the code needs to be changed so that it starts  
9             directly from Login Screen.*/  
10         spec.press('loginContainer');  
11  
12         spec.fillIn('inputEmail', 'hi@test.de');  
13         await spec.fillIn('inputPassword', 'Test1234!');  
14         await spec.press('submitButton');  
15  
16         /* The alert element couldnt be fetched with Cavy, so no interaction with it  
17            was possible.*/  
18  
19  
20       });  
21     });
```

Figure 17: Cavy Login test

.5.2 Cavy SignUp test

```
1  export default function (spec) {
2
3      spec.describe('Register test', function () {
4
5          spec.it('should be able to fill in forms, show an alert and sign up successfully',
6              async function () {
7                  /*tap on Login button in Nav bar couldnt be accomplished because UI
8                     element couldnt be fetched, so the code needs to be changed so that it starts
9                     directly from Login Screen.*/
10                 spec.press('regContainer');
11
12                 spec.fillIn('inputEmail', 'hi@test.de');
13                 await spec.fillIn('inputPassword', 'Test1234!');
14                 await spec.fillIn('signinPW', 'Test1234!');
15
16                 await spec.press('submitButton');
17
18                 /* The alert element couldnt be fetched with Cavy, so no interaction with it
19                    was possible.*/
20
21             }
12
```

Figure 18: Cavy SignUp test

5.3 Cavy forms test

```
1  export default function (spec) {
2
3      spec.describe('Form test', function () {
4
5          spec.it('should be able to fill in the input and validate the text',
6              async function () {
7                  spec.fillIn('text-input', 'Hello, this is a demo app');
8                  /*there is no method in Cavy to check if the text has been input
9                     right.*/
10             });
11
12             /*Cavy doesnt have a method for checking the value of a switch*/
13             spec.it('should be able to turn on the switch', async function () {
14                 //method not working for switch
15                 //spec.press('switch');
16             });
17
18             spec.it('should be able to select a value from the select element',
19                 async function () {
20
21                     //method not working for dropdown
22                     //spec.press('dropdown');
23                 });
24             spec.it('should be able to scroll to bottom', async function () {
25
26                 //Cavy is not able to scroll
27                 spec.exists('activeButton');
```

Figure 19: Cavy forms test

.6 Appium developer guidelines

This guide shows how you can include Appium into your working React Native(RN) app, using Webdriverio, the Jasmine Testing framework, UIAutomator2 and XCUITest. An important information is, that you can use different drivers and Testing frameworks with Appium. Webdriver.io, the Jasmine Testing framework, UIAutomator2 and XCUITest are only the choice of the author. Details about which drivers and platforms are supported can be found here:

<http://appium.io/docs/en/about-appium/platform-support/index.html>

A list of all the supported clients is here:

<http://appium.io/docs/en/about-appium/appium-clients/index.html>

First of all you need to have Android Studio and Xcode (the author used version 11.5) installed. As the IDE the author used Visual Studio Code.

.6.1 Integrate Appium into RN app

There is a getting started guide on the Appium website, but the author used another approach for the integration, using boilerplate code. If you instead don't want to use boilerplate code, use the tutorial from Appium which you find here:

<http://appium.io/docs/en/about-appium/getting-started/>

First you will need to install Node.js and NPM. Its recommended to use the latest stable version. To install Appium run the command `npm install -g appium`. When you installed Node or Appium with sudo there might occur an error when building the app, because e.g. React-Native might not be able to access the `node_module` folders it needs for the building due to missing access rights. To fix this you need to adjust the access rights of the affected folders accordingly.

In some cases it might be necessary (for the author it was not necessary, so if you use the boilerplate code from below, maybe skip this and come back to it in case of problems) to go through the driver-specific setup tutorial for iOS and Android which can be found in the second section "Driver-specific Setup" here:

<http://appium.io/docs/en/about-appium/getting-started/>

Next you need to download the boilerplate code from here:

<https://gitlab.codecentric.de/anja.bender/react-native-app/-/tree/master/appium-boilerplate-master>

Thats the project the author used, if you need the latest version from the boilerplate code, take it from here:

<https://github.com/webdriverio/appium-boilerplate>

Create a folder in your RN app and name it config. Copy the files and the folder from the config folder of the boilerplate project into it.

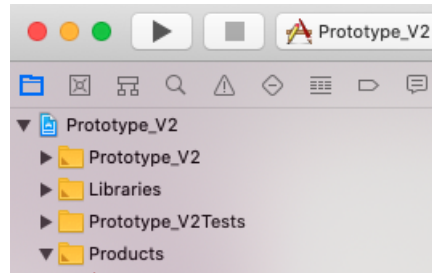
It has to be kept in mind, that the author didn't check in detail which files and dependencies may be excluded from the boilerplate project, meaning that maybe there is more code included from the boilerplate code than necessary. Copy all the devDependencies from the package.json file of the boilerplate project into your devDependencies in your package.json file. If your babel/core dependency appears twice, the version that worked for the author was ^7.7.7.

Next copy all the scripts from the package.json file of the boilerplate project package.json file into your scripts in your package.json file. If "lint" appears twice, "eslint config tests" was the one working for the author.

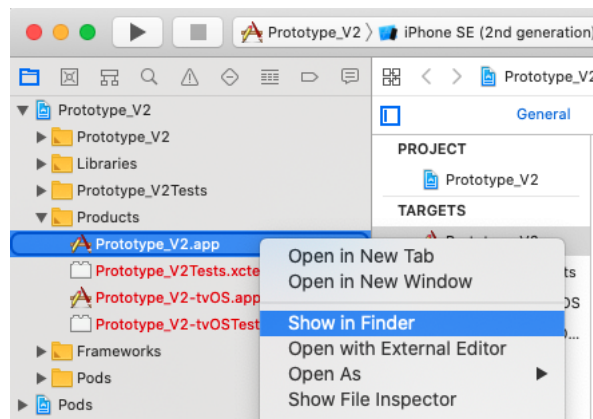
After installing yarn with `npm install -g yarn` run the command `yarn install` in the root directory of your project. In the root directory create the folder "apps".

.6.2 Integrate Appium for iOS

We need to create a zip file of the iOS app, for this open the ios folder of the RN app with Xcode and click on the triangle at the top left to build and run the app.



Now still inside of Xcode open the Products folder inside of the iOS Project folder and right-click on the file 'YourProjectName.app' and then on "show in Finder".



Right click on the file and compress it. Next you need to copy this zip file into the /apps/ folder of your RN app. Make sure the name of the zip file is the same as the name specified after 'appium:app' in the wdio.conf.js file.

.6.3 Integrate Appium for Android

We need to create an apk file which is a build of Android. For this first create an assets folder at the directory android/app/src/main/. Then in the project root directory run this command:

```
react-native bundle --platform android --dev false --entry-file
index.js --bundle-output
android/app/src/main/assets/index.android.bundle --assets-dest
android/app/src/main/res
```

Listing 10: command for Android build

Then go to the android directory with 'cd android/' and run the command

```
./gradlew assembleDebug
```

Then you can find your apk in this folder: cd app/build/outputs/apk/ Copy the file app-debug.apk from this folder into the apps folder. Make sure the name of the apk file is the same as the name specified after 'appium:app' in the wdio.android.app.conf.js file.

In some cases, like e.g. on Arch Linux you might have to set the JAVA_HOME variable in your .bashrc file. In Arch Linux the path should usually look like this:

```
export JAVA_HOME="/usr/lib/jvm/default"
```

.6.4 Writing tests

You can specify which device or simulator you want in the files wdio.ios.app.conf.js and wdio.android.app.conf.js.

At the project root directory create a folder for the tests named tests and inside of it create a folder specs, in which we will save all the tests. To create a new test, create a file named "app.TheNameYoulike.spec.js". Following parts of the code of a Login test are shown:

```

5  describe('WebdriverIO and Appium, when interacting with a login form,', () => {
6      beforeEach(() => {
7          $('~Home').waitForDisplayed(20000);
8          $('~Login').click();
9      });
10
11     it('should be able to fill in forms, show an alert and login successfully',
12         () => {
13
14             $('~button-login-container').click();
15             $('~input-email').setValue('hi@test.de');
16             $('~input-password').setValue('Test1234!');
17
18             if (driver.isKeyboardShown()) {
19                 driver.hideKeyboard();
20             }
21             $('~button-LOGIN').click();
22         }

```

Figure 20: Login test

The beforeEach()-method runs before each test. Each test is contained in an it()-method. Everything you see in that code snippet belongs to the Webdriver.io framework, the matching API documentation can be found here:

<https://webdriver.io/docs/api.html>

But on this Website not all methods are shown, more Webdriver.io methods can be found on the Appium Website:

<http://appium.io/docs/en/about-appium/intro/?lang=en>

The version that the author used can be found in the web archive: <https://web.archive.org/web/20190908183938/http://appium.io/docs/en/about-appium/intro/?lang=en>

And on the website have a look at the category commands in the top navigation bar. Then if you clicked on a certain command there you will find the following window, you need to click on JavaScript to find the Webdriver.io command:

Example Usage

Java	Python	JavaScript	Ruby	C#	PHP
------	--------	------------	------	----	-----

```
// webdriver.io example
driver.hideKeyboard();

// wd example
await driver.hideDeviceKeyboard();
```


Figure 21: Webdriver.io method

In line 7 we fetch the UI element called Home and wait until it is displayed for a maximum of 20 seconds. To identify UI elements for Webdriver.io you need to add a testID and accessibilityLabel to the UI element, in case of an Input element like this:

```
<Input testID={'input-email'} accessibilityLabel={'input-email'} />
```

The testID is needed for iOS, and the accessibilityLabel for Android. With the click()-method in line 8 we click on the chosen element. In line 15 the String hi@test.de is being filled in into the chosen text field. In line 18 we check if the keyboard is shown, and if this is the case the keyboard will be hidden with driver.hideKeyboard(). It is also possible to execute native mobile commands with the method driver.execute(), in the picture it says browser.execute(), but for the author driver.execute() was working:

Execute Mobile Command

 Edit this Doc

Execute a native mobile command

Example Usage

Java	Python	JavaScript	Ruby	C#	PHP
------	--------	------------	------	----	-----

```
// webdriver.io example
var result = browser.execute('mobile: scroll', {direction: 'down'})

// wd example
await driver.execute('mobile: scroll', {direction: 'down'});
```

Figure 22: Mobile commands

The list of all the mobile commands are on the already mentioned Appium website, in the category "com-

mands" at the top navigation bar. Because Appium doesn't monitor the internal state of the running app during test execution, sometimes manual waiting is necessary by e.g. using the `driver.pause(milliseconds)`-method. With the Jasmine framework you can use `expect()`-statements to check if certain values are like you expect. For example if you want to check if the UI element labeled as "sms" is not displayed anymore you write:

```
expect($('~sms').isDisplayed()).toEqual(false);
```

You can find all the methods of the Jasmine testing framework on this website: <https://jasmine.github.io/api/3.5/global>

A complete project with different tests from the author can be found at: https://gitlab.codecentric.de/anja.bender/react-native-app/-/tree/master/native-demo-app-master_RN_60

.6.5 Running tests

Start Android studio and run the Android device simulator. At the root folder, run the command `react-native start`, in a new terminal window the command `"appium"` and in a third terminal `react-native run-ios` or `react-native run-android`. With the command `npm run ios.app` you can start your tests for iOS and with `npm run android.app` for Android. The Appium example app from the author can be found at these two repositories ("native-demo-app-master_RN_60"):

<https://gitlab.codecentric.de/anja.bender/react-native-app/-/tree/master>

https://github.com/Lumary2/Detox_and_Appium/tree/master