Testing Android apps using Espresso in Java or Kotlin

Overview

In this tutorial, we will "create" some UI tests for an Android application using Espresso testing library.

Tests can be written either in Java or Kotlin.

Requirements

- gradle 5.6
- Android SDK (e.g. v28)
- JDK
- (optional) Android Studio

Description

For this tutorial, we'll use a basic Espresso example project provided by Google, with minor updates as tracked in this fork.

The Android application is quite simple: it contains two activities, where the main one as an input to change some text along with two buttons that will give the possibility to change the text in the current activity of in a new one.



The code related to the previous activities can be found here.

The project contains 2 tests, implemented both in Java and in Kotlin (thus, overall 4 tests although 2 "duplicated").

Tests use the Espresso testing library for UI-based tests. Espresso comes as part of Android SDK; a brief overview can be found here.

Espresso provides ways of interacting with the UI and assertions that can be used to perform checks in your tests.

app/src/androidTest/java/com/example/android/testing/espresso/BasicSample/ChangeTextBehaviorTest.java

```
* Copyright 2018, The Android Open Source Project
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
        http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package com.example.android.testing.espresso.BasicSample
import androidx.test.ext.junit.rules.activityScenarioRule
import android.app.Activity
import androidx.test.core.app.ActivityScenario
import androidx.test.core.app.launchActivity
import androidx.test.espresso.Espresso.onView
import androidx.test.espresso.action.ViewActions
import androidx.test.espresso.action.ViewActions.*
import androidx.test.espresso.assertion.ViewAssertions.matches
import androidx.test.espresso.matcher.ViewMatchers
import androidx.test.espresso.matcher.ViewMatchers.withId
import androidx.test.espresso.matcher.ViewMatchers.withText
import androidx.test.ext.junit.runners.AndroidJUnit4
import androidx.test.filters.LargeTest
import org.junit.Before
import org.junit.Rule
import org.junit.Test
import org.junit.runner.RunWith
/**
* The kotlin equivalent to the ChangeTextBehaviorTest, that
 * showcases simple view matchers and actions like [ViewMatchers.withId],
 * [ViewActions.click] and [ViewActions.typeText], and ActivityScenarioRule
 * Note that there is no need to tell Espresso that a view is in a different [Activity].
*/
@RunWith(AndroidJUnit4::class)
@LargeTest
class ChangeTextBehaviorKtTest {
    /**
     * Use [ActivityScenarioRule] to create and launch the activity under test before each test,
     \ast and close it after each test. This is a replacement for
     * [androidx.test.rule.ActivityTestRule].
     * /
    @get:Rule var activityScenarioRule = activityScenarioRule<MainActivity>()
    @Test
    fun changeText_sameActivity() {
        \ensuremath{{//}} Type text and then press the button.
        onView(withId(R.id.editTextUserInput))
                .perform(typeText(STRING_TO_BE_TYPED), closeSoftKeyboard())
        onView(withId(R.id.changeTextBt)).perform(click())
        // Check that the text was changed.
        onView(withId(R.id.textToBeChanged)).check(matches(withText(STRING_TO_BE_TYPED)))
    }
    @Test
    fun changeText_newActivity() {
```

```
// Type text and then press the button.
```

app/src/androidTest/java/com/example/android/testing/espresso/BasicSample/ChangeTextBehaviorKtTest.java /* * Copyright 2018, The Android Open Source Project * Licensed under the Apache License, Version 2.0 (the "License"); * you may not use this file except in compliance with the License. * You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License. */ package com.example.android.testing.espresso.BasicSample import androidx.test.ext.junit.rules.activityScenarioRule import android.app.Activity import androidx.test.core.app.ActivityScenario import androidx.test.core.app.launchActivity import androidx.test.espresso.Espresso.onView import androidx.test.espresso.action.ViewActions import androidx.test.espresso.action.ViewActions.* import androidx.test.espresso.assertion.ViewAssertions.matches import androidx.test.espresso.matcher.ViewMatchers import androidx.test.espresso.matcher.ViewMatchers.withId import androidx.test.espresso.matcher.ViewMatchers.withText import androidx.test.ext.junit.runners.AndroidJUnit4 import androidx.test.filters.LargeTest import org.junit.Before import org.junit.Rule import org.junit.Test import org.junit.runner.RunWith /** * The kotlin equivalent to the ChangeTextBehaviorTest, that * showcases simple view matchers and actions like [ViewMatchers.withId], * [ViewActions.click] and [ViewActions.typeText], and ActivityScenarioRule * Note that there is no need to tell Espresso that a view is in a different [Activity]. * / @RunWith(AndroidJUnit4::class) @LargeTest class ChangeTextBehaviorKtTest { /** * Use [ActivityScenarioRule] to create and launch the activity under test before each test, * and close it after each test. This is a replacement for

* [androidx.test.rule.ActivityTestRule].

```
*/
@get:Rule var activityScenarioRule = activityScenarioRule<MainActivity>()
@Test
fun changeText_sameActivity() {
    // Type text and then press the button.
    onView(withId(R.id.editTextUserInput))
            .perform(typeText(STRING_TO_BE_TYPED), closeSoftKeyboard())
    onView(withId(R.id.changeTextBt)).perform(click())
    \ensuremath{{//}} Check that the text was changed.
    onView(withId(R.id.textToBeChanged)).check(matches(withText(STRING_TO_BE_TYPED)))
}
@Test
fun changeText_newActivity() {
    \ensuremath{{//}} Type text and then press the button.
    onView(withId(R.id.editTextUserInput)).perform(typeText(STRING_TO_BE_TYPED),
            closeSoftKeyboard())
    onView(withId(R.id.activityChangeTextBtn)).perform(click())
    // This view is in a different Activity, no need to tell Espresso.
    onView(withId(R.id.show_text_view)).check(matches(withText(STRING_TO_BE_TYPED)))
}
companion object {
    val STRING_TO_BE_TYPED = "Espresso"
}
```

There are different types of tests: local unit tests (i.e. unit tests that don't need a device/emulator) and instrumentation tests (which run on a device /emulator).

Depending on which tests, checks and tasks you want to run, the gradle syntax needs to be adapted accordingly; you may use "gradle tasks" (or "gr adlew tasks") to find out all available tasks.

Thus, you'll find multiple ways of running the tests using the command line.

For local unit tests, you may use:

}

- ./gradlew check
- ./gradlew test
- gradle clean cleanTest test

In this case, test results will be stored as multiple JUnit XML files in build/test-results.

For instrumentation tests, you may use:

- ./gradlew connectedAndroidTest
- ./gradlew app:connectedCheck

In this case, test results will be stored as one JUnit XML file in a directory similar to build/outputs/<xxxx>-results/connected.

If you're using Android Studio, then you can also use it to write and run your tests.

However, if you decide to manually export the results to an XML file, that format is not a JUnit XML compatible one in is thus not supported.

- # run the tests in the installed app
- ./gradlew app:connectedCheck
- # ...or... install app and run the tests
- #./gradlew connectedAndroidTest

()	Please note You may start an emulator from within Android Studio (Tools => AVD Manager)										
	dio File Edit View Navigate Code Analyze Refactor Build Run app [~/exps/testing-samples/ui/espresso/BasicSample/app]/src/androi C S S C S S C S S C S S C S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S					Tools VCS Window Help	🎙 🏠 🔿 🏶 🕑 🖇 💷 〈ふ 奈 🕪)) 46% 🗈 Sat 16:11.				
						📮 AVD Manager	p esso/Bas	p esso/BasicSample/ChangeTextBehaviorTest.java [app]			
						🔍 SDK Manager					
	ie) 📑 app	> src > test > Java	t ⟩ • com ⟩ •	example > android > a	ngeTextBel	🖳 Layout Inspector	rTest.java × C MainActivity.java × activity_main.xml ×				
					And	Connection Assistant					
¹ Your Vir						App Links Assistant					
		Your Virtual	Device	S		Tasks & Contexts					
		Android Studio		<pre>m within Android Studio (Tools => AVD Manager). Code Analyze Refactor Build Ruf Tools VCS Window Help</pre>							
	C Type	Name	Play Store	Resolution	API	IDE Scripting Console	BI	Size on Disk	Actions		
	G	Pixel XL API 29		1440 × 2560: 560dpi	29	Create Command-line Launcher		3.3 GB	▶ 🗸 🔻		
	re 6					JShell Console ◦ Groovy Console ▲ Kotlin ►					
	You may	/ also use the comr	nand line,	for example:							
	emulator -avd Pixel_XL_API_29										
	If you need to find out the name of the AVD, you may use:										
	emulat	or -list-avds									

After running the tests and generating the JUnit XML report(s) (e.g., TEST-Pixel_XL_API_29(AVD) - 10-app-.xml), it can be imported to Xray (either by the REST API or by using one of the CI plugins or through **Import Execution Results** action within the Test Execution).

curl -H "Content-Type: multipart/form-data" -u admin:admin -F "file=@app/build/outputs/androidTest-results
/connected/TEST-Pixel_XL_API_29\(AVD\)\ -\ 10-app-.xml" http://jiraserver.example.com/rest/raven/1.0/import
/execution/junit?projectKey=CALC

A Test Execution will be created containing information about the executed scenarios.

	alculator / CA xecution	results - T	TEST-Pixel_XL_AP	_29(AVD)) - 10-app	.xml -	[1572119925	079]		
/ Edit	Commer	t Assign M	More - Close Issue Reope	en Issue Adr	nin 👻					
Details										
Type: Affects Ve Compone Labels: Test Envir	rsion/s: nt/s: onments:	Test Executio None None None None	n		Status: Resolution: Fix Version/s:		RESOLVED (View Wo Fixed None	rkflow)		
Test Plan:		None								
Description	1									
Overall E	xecution State	us							+ Add -	
TOTAL TES	STS: 4 Filter(s)									
	Apply Rank							Show 100 \$ entries	Columns 🗸	
	Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Status		
	1	CALC-5145	changeText_sameActivity	Generic	0	0	Administrator	PASS	►]
	2	CALC-5146	changeText_newActivity	Generic	0	0	Administrator	PASS	►]
	3	CALC-5147	changeText_newActivity	Generic	0	0	Administrator	PASS	►]
	4	CALC-5148	changeText_sameActivity	Generic	0	0	Administrator	PASS	►	•••

Each test is mapped to a Generic Test in Jira, and the Generic Test Definition field contains the name of the class concatenated with the method name of the corresponding automated test.

The Execution Details of the Generic Test contains information about the "Test Suite" (as per JUnit format), which in this case corresponds to the first test full classpath.

Iculator / Test Execution: CALC-5144 , nangeText_sameActivity	/ Test: CALC-5148		Export Test as Text Return to Test	Execution 4 Previous
Execution Status PASS +	Finished On: 26/Oct/19 8:58 PM		Assignee: Administrato Executed By: Administrator Tests environments: -	r Versions: Revision:
Comment	Preview Comment	Execution Defects (0) Create Defect Create Sub-Task Add Defects	Execution Evidence (0)	Add Evidence
Test Description None Test Details				
Test Type: Generic Definition: com.example.a	ndroid.testing.espresso.BasicSample.ChangeT	extBehaviorTest.changeText_sameActivity		
Results				/
Context		Error Message	Duration	Status
TestSuite com.example.android.testing.es	spresso.BasicSample.ChangeTextBehaviorKtTe	- st	1 sec	PASS

You can also import local unit tests. Note that you'll have a JUnit XML file per test class. You may use junit-merge utility in case you want to merge all these reports to a single JUnit XML report.

References

- https://developer.android.com/training/testing/espresso
 https://developer.android.com/training/testing/espresso/additional-resources
 https://developer.android.com/training/testing
 https://developer.android.com/training/testing/ui-testing/index.html
 https://developer.android.com/studio/test
 https://developer.android.com/kotlin