Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерный наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий.

**Отчёт по лабораторной работе №5**

**Дисциплина**: Проектирование мобильных приложений

**Тема**: UI Tests. Вариант 13

Выполнил студент гр. 3530901/80201: ­­­­­­­­­­­­­ ­ В.Д. Петров

(подпись)

Преподаватель: ­А.Н. Кузнецов

(подпись)

Санкт-Петербург

2020

**СОДЕРЖАНИЕ**

Введение…………………………………………………………………………….3

Задача №1. Простейший UI тест…………………………………………………..4

Задача №2. Тестирование навигации……………………………………………...7

Заключение……………………………………………………………………..…12

Список использованных источников……………………………………….…...13

**ВВЕДЕНИЕ**

Целями данной работы является ознакомление с принципами и получение практических навыков разработки UI тестов для Android приложений. Espresso Framework – фреймворк для написания тестов для Android приложений.

**ЗАДАЧА №1. ПРОСТЕЙШИЙ UI ТЕСТ**

Исходный код для данного задания представлен по ссылке:

<https://github.com/ADsty/labs-android_labs/tree/master/lab5>

Для написания простейшего теста, проверяющего изменение текста на кнопке при повороте экрана, было написано приложение (Листинг 1), в котором есть одна кнопка и одно текстовое поле, при нажатии на кнопку текст на ней меняется, текст в текстовом поле сохраняется.

Листинг 1 – исходный код приложения

class MainActivity : AppCompatActivity() {  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.activity\_main)  
 val button: Button = findViewById(R.id.button)  
 button.setOnClickListener**{** button.text = getString(R.string.button2)  
 **}** }  
}

После этого был создан espresso тест с 3 функциями – setUp для установки портретной ориентации устройства перед каждым тестом, afterCreating для проверки, что кнопка и текстовое поле появились на экране, их видно и на кнопку можно нажать, checkButtonTextAfterRotation для проверки текста на кнопке (Листинг 2).

Листинг 2 – espresso тест

class MainActivityTest {  
 @get:Rule  
 var activityTestRule: ActivityTestRule<MainActivity> =  
 ActivityTestRule(MainActivity::class.*java*)  
  
 private val buttonId = R.id.*button* private val textId = R.id.*editText* @Before  
 fun setUp() {  
 activityTestRule.*activity*.requestedOrientation = ActivityInfo.*SCREEN\_ORIENTATION\_PORTRAIT* }  
  
 @Test  
 fun afterCreating() {  
 onView(withId(buttonId)).check(matches(isEnabled())).check(matches(isClickable()))  
 onView(withId(textId)).check(matches(isEnabled())).check(matches(isFocusable()))  
 }  
  
 @Test  
 fun checkButtonTextAfterRotation() {  
 val buttonTextBefore = activityTestRule.*activity*.findViewById<Button>(buttonId).*text*.toString()  
 val editTextBefore = activityTestRule.*activity*.findViewById<EditText>(textId).*text*.toString()  
 onView(withId(buttonId)).perform(click())  
 val buttonTextChanged = activityTestRule.*activity*.findViewById<Button>(buttonId).*text*.toString()  
 activityTestRule.*activity*.requestedOrientation = ActivityInfo.*SCREEN\_ORIENTATION\_LANDSCAPE* Thread.sleep(200)  
 val buttonTextAfter = activityTestRule.*activity*.findViewById<Button>(buttonId).*text*.toString()  
 val editTextAfter = activityTestRule.*activity*.findViewById<EditText>(textId).*text*.toString()  
 Preconditions.checkState(buttonTextBefore == buttonTextAfter,"не совпадает")  
 Preconditions.checkState(buttonTextChanged == "Button Changed","не совпадает")  
 Preconditions.checkState(editTextBefore == editTextAfter,"не совпадает")  
 }  
}

Последний тест делает следующее: после установки портретной ориентации в функции setUp текст на кнопке и текстовом поле сохраняется в переменные buttonTextBefore и editTextBefore, после этого с помощью метода onView.perform(click()) происходит нажатие на кнопку, текст на кнопке сохраняется в другую переменную buttonTextChanged для проверки, что после нажатия текст был изменен, после этого меняем ориентацию устройства на альбомную, после чего запускаем Thread.sleep для ожидания, пока ориентация устройства изменится (~100 миллисекунд по информации из интернета, однако на моем устройства ровно 100 миллисекунд не хватает, 130 миллисекунд иногда тоже недостаточно, поэтому берется немного больше), затем текст на кнопке после изменения ориентации устройства и текст в текстовом поле сохраняются в новые переменные buttonTextAfter и editTextAfter и методом checkState проверяется содержимое переменных – проверяется что до поворота экрана/нажатия кнопки и после поворота/нажатия кнопки текст на кнопке остался одинаковым, после тех же изменений текст в текстовом поле остался прежним, а после нажатия кнопки, но до поворота экрана текст на кнопке был равен “Button Changed”. Все тесты были успешно пройдены (Рис. 1).

**Изображение выглядит как текст

Автоматически созданное описание**

Рис. 1 Результаты прохождения тестов

Состояние EditText сохраняется в методе onSaveInstanceState, в данном методе без каких-либо переопределений уже есть возможность перенести значение текста в EditText во вновь созданную activity с помощью метода onRestoreInstanceState.

**ЗАДАЧА №2. ТЕСТИРОВАНИЕ НАВИГАЦИИ**

Исходный код для данного задания представлен по ссылке:

<https://github.com/ADsty/labs-android_labs/tree/master/lab5.2>

Для проверки навигации был выбран вариант решения с помощью флагов Intent. Был написан класс EspressoTest, в котором и происходит тестирование навигации. Для начала рассмотрим вспомогательные функции:

isActivityLaunched – метод, который проверяет, запущена ли данная activity, а точнее, отображается ли на экране кнопка, которая есть только в данной activity. Если мы ожидаем, что activity нет на экране, то проверяется отсутствие кнопки. Если же желаемый результат не достигнут, то тест, который использует данный метод, завершится с ошибкой.

Листинг 3 – isActivityLaunched

private fun isActivityLaunched(isCurrent: Boolean, index: Int){  
 val button = when(index){  
 1 -> R.id.*from\_1\_to\_2* 2 -> R.id.*from\_2\_to\_1* 3 -> R.id.*from\_3\_to\_1* else -> -1  
 }  
 if(isCurrent)  
 onView(withId(button)).check(matches(ViewMatchers.isDisplayed()))  
 else  
 onView(withId(button)).check(ViewAssertions.doesNotExist())  
}

checkActivities – метод, который проверяет что на экране отображается только нужная activity. Вызывает внутри себя метод isActivityLaunched с соответствующими флагами, проверяя что все activity кроме нужной нам не отображаются на экране, а нужная нам activity отображается. Поскольку использует isActivityLaunched, то несоответствие ожидаемому результате приведет к ошибке при исполнении теста.

Листинг 4 – checkActivities

private fun checkActivities(indexOfActivity: Int) {  
 when(indexOfActivity) {  
 1 -> {  
 isActivityLaunched(true, 1)  
 isActivityLaunched(false, 2)  
 isActivityLaunched(false, 3)  
 onView(withId(R.id.*text\_about*)).check(ViewAssertions.doesNotExist())  
 }  
  
 2 -> {  
 isActivityLaunched(false, 1)  
 isActivityLaunched(true, 2)  
 isActivityLaunched(false, 3)  
 onView(withId(R.id.*text\_about*)).check(ViewAssertions.doesNotExist())  
 }  
  
 3 -> {  
 isActivityLaunched(false, 1)  
 isActivityLaunched(false, 2)  
 isActivityLaunched(true, 3)  
 onView(withId(R.id.*text\_about*)).check(ViewAssertions.doesNotExist())  
 }  
  
 4 -> {  
 isActivityLaunched(false, 1)  
 isActivityLaunched(false, 2)  
 isActivityLaunched(false, 3)  
 onView(withId(R.id.*text\_about*)).check(matches(ViewMatchers.isDisplayed()))  
 }  
 }  
}

fromActivityToAboutAndBack – метод, в котором проверяется, что мы находимся в нужной activity методом checkActivities, после чего открывается шторка NavigationDrawer, в меню выбирается кнопка about и идет проверка, что мы теперь находимся в activity about, после этого нажимается кнопка back, закрывается шторка Navigation Drawer и идет проверка что мы вернулись в предыдущую activity.

Листинг 5 – fromActivityToAboutAndBack

private fun fromActivityToAboutAndBack(index : Int){  
 checkActivities(index)  
 val drawer = when(index){  
 1 -> R.id.*drawer1* 2 -> R.id.*drawer2* 3 -> R.id.*drawer3* else -> -1  
 }  
 onView(withId(drawer)).check(matches(isClosed(Gravity.*LEFT*))).perform(DrawerActions.open())  
 onView(withId(R.id.*about*)).perform(click())  
 checkActivities(4)  
 inst.sendKeyDownUpSync(KeyEvent.*KEYCODE\_BACK*)  
 onView(withId(drawer)).check(matches(isOpen(Gravity.*LEFT*))).perform(DrawerActions.close())  
 checkActivities(index)  
}

testRotation – метод, который проверяет, что при повороте экрана состояние текущего activity не изменяется. Использует внутри себя checkActivities

Листинг 6 – testRotation

private fun testRotation(index: Int){  
 rule.*activity*.*requestedOrientation* = ActivityInfo.*SCREEN\_ORIENTATION\_PORTRAIT* checkActivities(index)  
 rule.*activity*.*requestedOrientation* = ActivityInfo.*SCREEN\_ORIENTATION\_LANDSCAPE* checkActivities(index)  
 rule.*activity*.*requestedOrientation* = ActivityInfo.*SCREEN\_ORIENTATION\_PORTRAIT*}

Теперь рассмотри тестовые функции:

testRotation – метод, использующий в себе вспомогательную функцию testRotation для каждого из activity, после чего переходит в следующую activity с помощью нажатия на кнопку. В результате все приложение проверяется на то, что при повороте экрана не ломается текущее activity.

Листинг 7 – testRotation

@Test  
fun testRotation(){  
 testRotation(1)  
 onView(withId(R.id.*from\_1\_to\_2*)).perform(click())  
 testRotation(2)  
 onView(withId(R.id.*from\_2\_to\_3*)).perform(click())  
 testRotation(3)  
}

checkBackStack – метод проходит вглубь по приложению и нажимает кнопку back, после чего смотрит, что activity расположены в правильном порядке в backstack. Проверяются следующие порядки:

1->2->back->проверка что перешли в 1;

1->2->3->back-> проверка что перешли в 2;

1->2->3->1->проверка что перешли в 1;

1->2->3->back->проверка что перешли в 2 ->back-> проверка что перешли в 1

1->back->проверка что activity было уничтожено

Листинг 8 – checkBackStack

@Test  
fun checkBackStack(){  
 checkActivities(1)  
 onView(withId(R.id.*from\_1\_to\_2*)).perform(click())  
 checkActivities(2)  
 Espresso.pressBack()  
 checkActivities(1)  
  
 onView(withId(R.id.*from\_1\_to\_2*)).perform(click())  
 checkActivities(2)  
 onView(withId(R.id.*from\_2\_to\_3*)).perform(click())  
 checkActivities(3)  
 Espresso.pressBack()  
 checkActivities(2)  
  
 onView(withId(R.id.*from\_2\_to\_3*)).perform(click())  
 checkActivities(3)  
  
 onView(withId(R.id.*from\_3\_to\_1*)).perform(click())  
 checkActivities(1)  
  
 onView(withId(R.id.*from\_1\_to\_2*)).perform(click())  
 onView(withId(R.id.*from\_2\_to\_3*)).perform(click())  
 Espresso.pressBack()  
 checkActivities(2)  
 Espresso.pressBack()  
 checkActivities(1)  
 Espresso.pressBackUnconditionally()  
 Assert.assertTrue(rule.*activity*.*isDestroyed*)  
}

testNavigation – метод, в котором проверяется, что нажимая на соответствующие кнопки, мы переходим в соответствующие activity, включая activity about. В результате теста все кнопки в приложении будут проверены.

Листинг 9 – testNavigation

@Test  
fun testNavigation() {  
 checkActivities(1)  
 fromActivityToAboutAndBack(1)  
 onView(withId(R.id.*from\_1\_to\_2*)).perform(click())  
 checkActivities(2)  
 fromActivityToAboutAndBack(2)  
  
 onView(withId(R.id.*from\_2\_to\_1*)).perform(click())  
 checkActivities(1)  
 fromActivityToAboutAndBack(1)  
  
 onView(withId(R.id.*from\_1\_to\_2*)).perform(click())  
 onView(withId(R.id.*from\_2\_to\_3*)).perform(click())  
 checkActivities(3)  
 fromActivityToAboutAndBack(3)  
  
 onView(withId(R.id.*from\_3\_to\_2*)).perform(click())  
 checkActivities(2)  
  
 onView(withId(R.id.*from\_2\_to\_3*)).perform(click())  
 onView(withId(R.id.*from\_3\_to\_1*)).perform(click())  
 checkActivities(1)  
 fromActivityToAboutAndBack(1)  
}

После запуска все тесты были пройдены без каких-либо ошибок (Рис. 2), что означает, что приложение в каждом из тестов достигло ожидаемого результата. Тесты покрывают всю навигацию приложения и все переходы, в каждом из activity проверяется что при повороте экрана ничего не ломается.

Изображение выглядит как текст

Автоматически созданное описание

Рис. 2 Результат прохождения тестов

**ЗАКЛЮЧЕНИЕ**

В результате данной работы было изучено тестирование Android приложений с помощью фреймворка Espresso. Была проверена навигация в приложении из лабораторной работы 3, а также создано новое приложение, в котором также была проверена работа при различных действиях с устройством. Были получены практические навыки тестирования Android приложений

Исходный код для лабораторных работ находится на [github](https://github.com/ADsty/labs-android_labs) (<https://github.com/ADsty/labs-android_labs> )

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

<https://developer.android.com/training/testing/espresso>