МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Белгородский государственный технологический университет

им. В.Г. Шухова»

**ИНСТИТУТ ЭНЕРГЕТИКИ, ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

**И УПРАВЛЯЮЩИХ СИСТЕМ**

**Кафедра программного обеспечения вычислительной**

**техники и автоматизированных систем**

Курсовой проект

по дисциплине: Тестирование программных систем

на тему:

Интеграционное тестирование

Работу выполнил:

студент группы ПВ-41

Мишина Ю.А.

Проверил: Поляков В. К.

Белгород 2020

**Оглавление**

[Теоретические сведения 3](#_Toc61912648)

[UI - тестирование 3](#_Toc61912649)

[Espresso Testing Framework 5](#_Toc61912650)

[Выполнение 7](#_Toc61912651)

[Вывод 20](#_Toc61912652)

[Список литературы 21](#_Toc61912653)

# Теоретические сведения

Для проведения UI - тестирования мобильного приложения будем использовать Espresso Testing Framework.

## **UI - тестирование**

**UI testing**- это тестирование графического интерфейса пользователя, которая предполагает проверить сайт на соответствия требованиям к графическому интерфейсу.

В большинстве случаев, тестирование интерфейса пользователя, осуществляется вместе со следующими видами тестирования(UI):

1. Тестирование на соответствие стандартам графических интерфейсов
2. Тестирование с различными разрешениями экрана
3. Тестирование кроссбраузерности, или совместимости с разными интернет браузерами и их версиями
4. Тестирование локализованных версий: точность перевода (мультиязычность, мультивалютность), проверка длины названий элементов интерфейса и т.д
5. Тестирование графического интерфейса пользователя на целевых устройствах (смартфоны, кпп, планшеты).

Основные плюсы UI тестирования:

* Покрывает большую часть пользовательских действий и позволяет, со стороны юзера, использовать приложение.
* Проверяет взаимодействие компонентов и сервисов между собой.
* Увеличивает надежность приложения.

Основные минусы UI тестирования:

* Огромное количество времени, потраченного на mock всех компонентов и сервисов;
* Бывают часто избыточными;
* Требуется проверка корректности работы;
* Зависимые части всей системы.

При тестировании мы учитываем критерии создания качественного интерфейса, а именно:

* минимальное время на выполнение задач пользователем;
* минимальное количество ошибок, которые допускает пользователь при работе с приложением;
* полное понимание интерфейса пользователями и отсутствие неоднозначностей при работе с ним;
* минимальный объем вводимой пользователями информации;
* простота и визуальная привлекательность интерфейса.

# Espresso Testing Framework

Espresso — это среда тестирования пользовательского интерфейса для тестирования приложений Android, разработанная на языке Java / Kotlin с использованием Android SDK. Поэтому единственным требованием к эспрессо является разработка приложения с использованием Android SDK на Java или Kotlin

В целом, тестирование мобильной автоматизации является сложной и сложной задачей. Доступность Android для различных устройств и платформ делает его утомительным для тестирования мобильной автоматизации. Чтобы сделать это проще, Google взял на себя задачу и разработал каркас Espresso. Он предоставляет очень простой, согласованный и гибкий API для автоматизации и тестирования пользовательских интерфейсов в приложениях для Android. Тесты эспрессо могут быть написаны на Java и Kotlin, современном языке программирования для разработки приложений для Android.

API-интерфейс Espresso прост и легок в освоении. Мы можем легко выполнять тестирование пользовательского интерфейса Android без сложности многопоточного тестирования. Google Drive, Карты и некоторые другие приложения в настоящее время используют Espresso.

Особенности эспрессо:

Вот некоторые характерные особенности, поддерживаемые Espresso:

* Очень простой API и так легко учиться.
* Высоко масштабируемый и гибкий.
* Предоставляет отдельный модуль для тестирования компонента Android WebView.
* Предоставляет отдельный модуль для проверки, а также макет Android-содержимого.
* Обеспечивает автоматическую синхронизацию между вашим приложением и тестами.

Преимущества эспрессо:

Давайте теперь рассмотрим преимущества эспрессо.

* Обратная совместимость
* Прост в настройке.
* Высокостабильный цикл испытаний.
* Поддерживает тестирование вне приложения.
* Поддерживает JUnit4
* Автоматизация пользовательского интерфейса подходит для написания тестов черного ящика.

# Выполнение

Целью выполнения работы:

* Узнать об Espresso, интегрировав его в проект Android Studio.
* написать UI-тесты для экрана входа в систему и RecyclerView.
* Узнаем о целях тестирования.

Список элементов, которые должны быть правильно настроены перед началом работы в эспрессо-фреймворке, выглядит следующим образом:

* Установите последнюю версию Java JDK и настройте переменную среды JAVA\_HOME.
* Установите последнюю версию Android Studio (версия 3.2. Или выше).
* Установите последнюю версию Android SDK с помощью SDK Manager и настройте переменную среды ANDROID\_HOME.
* Установите последнюю версию Gradle Build Tool и настройте переменную среды GRADLE\_HOME.

Включить инфраструктуру тестирования эспрессо в проект так же просто, как установить среду тестирования эспрессо, как зависимость в файле gradle приложения app / build.gradle. необходимо добавить следующие зависимости из библиотеки поддержки тестирования Android в **build.gradle**

android **{** defaultConfig **{** testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"

}

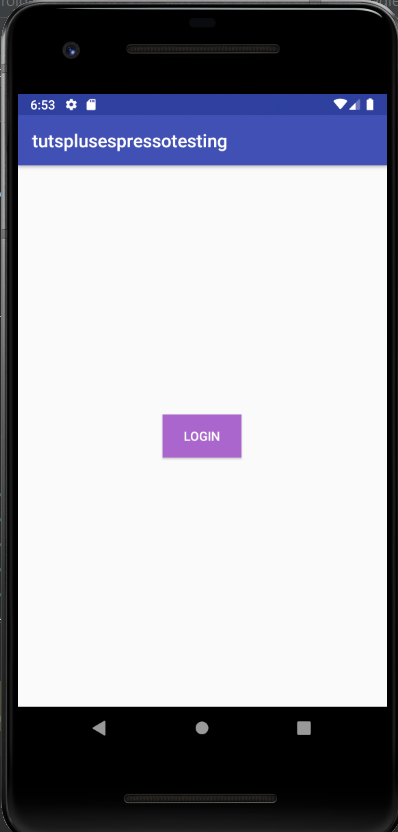
}

dependencies **{** androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'  
 androidTestImplementation 'com.android.support.test:runner:1.0.2'  
 androidTestImplementation 'com.android.support.test:rules:1.0.2'  
**}**

Instrumentation — это просто базовый класс для реализации кода инструментария приложения.

**Сначала мы начнем тестирование экрана входа в систему. Вот как начинается процесс входа в систему: пользователь запускает приложение, и первый показанный экран содержит одну кнопку входа в систему . При нажатии этой кнопки входа в систему открывается экран LoginActivity. Этот экран содержит только два EditText (поля имени пользователя и пароля) и кнопку «Отправить».**

**Вот как выглядит наш макет MainActivity:**

****

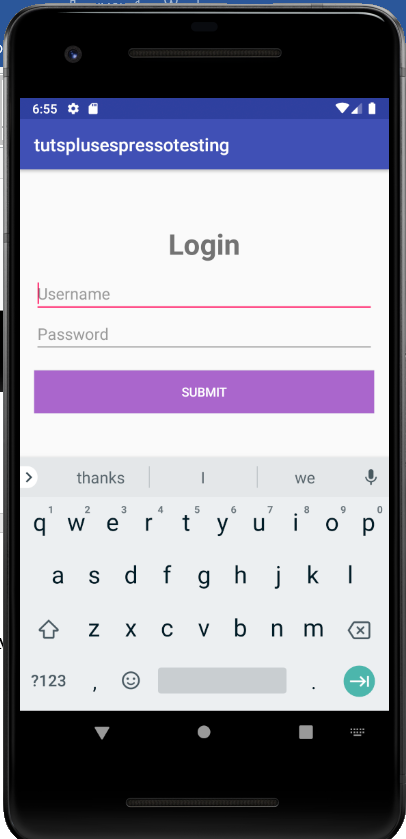
**Xml-файл MainActivity**

<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".MainActivity"  
 android:gravity="center">  
  
 <Button  
 android:id="@+id/btn\_login"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:background="@android:color/holo\_purple"  
 android:textColor="@android:color/white"  
 android:text="Login"/>  
  
</LinearLayout>

**Kotlin class MainActivity**

class MainActivity : AppCompatActivity() {  
  
 private lateinit var loginButton: Button  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_main*)  
 loginButton = findViewById(R.id.*btn\_login*)  
  
 loginButton.setOnClickListener **{** val intent = Intent(this, LoginActivity::class.*java*)  
 startActivity(intent)  
 **}** }  
}

Вот как выглядит наш макет LoginActivity:



**Xml-файл** LoginActivity

<?xml version="1.0" encoding="utf-8"?>  
<ScrollView  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent">  
  
 <LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:orientation="vertical"  
 tools:context=".MainActivity"  
 android:layout\_gravity="center"  
 android:paddingStart="16dp"  
 android:paddingEnd="16dp">  
  
 <TextView  
 android:id="@+id/tv\_login"  
 android:text="Login"  
 android:textSize="32sp"  
 android:textStyle="bold"  
 android:layout\_width="wrap\_content"  
 android:padding="12dp"  
 android:layout\_gravity="center\_horizontal"  
 android:layout\_height="wrap\_content"/>  
  
 <EditText  
 android:id="@+id/et\_username"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:hint="Username"  
 android:inputType="text"/>  
  
 <EditText  
 android:id="@+id/et\_password"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:hint="Password"  
 android:inputType="textPassword"/>  
  
 <Button  
 android:id="@+id/btn\_submit"  
 android:layout\_marginTop="18dp"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:background="@android:color/holo\_purple"  
 android:textColor="@android:color/white"  
 android:text="submit"/>  
  
 </LinearLayout>  
  
</ScrollView>

**Kotlin class** LoginActivity

class LoginActivity : AppCompatActivity() {  
  
 private lateinit var usernameEditText: EditText  
 private lateinit var loginTitleTextView: TextView  
 private lateinit var passwordEditText: EditText  
 private lateinit var submitButton: Button  
  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_login*)  
  
 usernameEditText = findViewById(R.id.*et\_username*)  
 passwordEditText = findViewById(R.id.*et\_password*)  
 submitButton = findViewById(R.id.*btn\_submit*)  
 loginTitleTextView = findViewById(R.id.*tv\_login*)  
  
 submitButton.setOnClickListener **{** if (usernameEditText.*text*.toString() == getString(R.string.*nameUser*) &&  
 passwordEditText.*text*.toString() == getString(R.string.*passwordUser*)) {  
 loginTitleTextView.*text* = getString(R.string.*txtSuccess*)  
 } else {  
 loginTitleTextView.*text* = getString(R.string.*txtFailure*)  
 }  
 **}** }  
}

Теперь напишем тест для нашего класса MainActivity

Три основных этапа написания теста для эспрессо:

* Найдите виджет (например, TextView или Button), который мы хотим протестировать.
* Выполнить одно или несколько действий с этим виджетом.
* Проверить, находится ли этот виджет в определенном состоянии.

Поскольку мы хотим протестировать действие, мы должны сделать небольшую настройку. Нам нужно сообщить Espresso, какую активность открыть или запустить перед выполнением и уничтожить после выполнения любого метода тестирования.

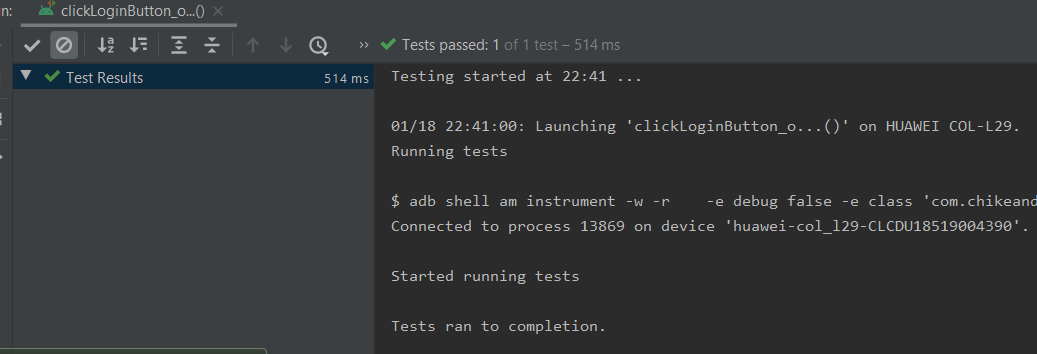
Следующие типы аннотаций могут применяться к методам, используемым в тестовом классе.

* @Rule означает, что это тестовое правило JUnit4. Правила тестирования JUnit4 запускаются до и после каждого метода тестирования (с аннотацией @Test). В нашем сценарии мы хотим запускать MainActivity перед каждым методом тестирования и уничтожать его после.
* @JvmField указывает компилятору не генерировать геттеры и сеттеры для свойства, а вместо этого выставлять его как простое поле Java.
* @BeforeClass указывает на то, что статический метод, к которому применяется эта аннотация, должен выполняться один раз и перед всеми тестами в классе. Это может быть использовано, например, для установки соединения с базой данных.
* @Before указывает, что метод, к которому прикреплена эта аннотация, должен выполняться перед каждым тестовым методом в классе.
* @Test указывает, что метод, к которому прикреплена эта аннотация, должен запускаться как контрольный пример.
* @After указывает, что метод, к которому прикреплена эта аннотация, должен запускаться после каждого метода тестирования.
* @AfterClass указывает, что метод, к которому прикреплена эта аннотация, должен запускаться после запуска всех методов тестирования в классе. Здесь мы обычно закрываем ресурсы, которые были открыты в @BeforeClass.

В нашем MainActivity макета MainActivity у нас есть только один виджет — кнопка Login. Чтобы найти виджеты в Espresso, мы используем статический метод onView().

@RunWith(AndroidJUnit4::class)  
class MainActivityTest {  
  
 @Rule @JvmField  
 var activityRule = ActivityTestRule<MainActivity>(  
 MainActivity::class.*java* )  
  
 @Test  
 fun clickLoginButton\_opensLoginUi() {  
 onView(withId(R.id.*btn\_login*)).check(matches(isDisplayed()))  
  
 }  
}

Результат запуска теста



Выполнение действия в представлении:

На объекте ViewInteraction который возвращается с помощью onView() , мы можем имитировать действия, которые пользователь может выполнять с виджетом. Например, мы можем смоделировать действие щелчка, просто вызвав статический метод click() внутри класса ViewActions

Мы выполняем событие click, сначала вызывая perform() . Этот метод выполняет заданное действие (действия) на представлении, выбранном текущим сопоставителем представления. Мы можем передать ему одно действие или список действий (выполняемых по порядку). Здесь мы дали click() . Другие возможные действия:

* typeText() для имитации ввода текста в EditText .
* clearText() для имитации очистки текста в EditText .
* doubleClick() для имитации двойного щелчка по View .
* longClick() чтобы имитировать долгое нажатие View .
* scrollTo() для имитации прокрутки ScrollView для определенного видимого представления.
* swipeLeft() для имитации пролистывания справа налево через вертикальный центр View .

Проверка с утверждениями представления:

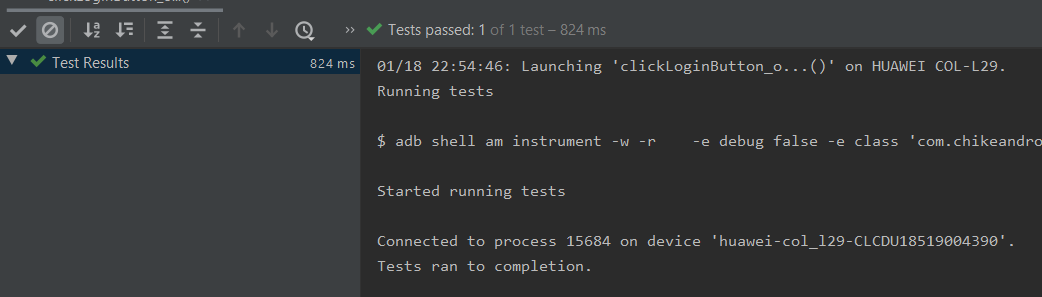
Экран LoginActivity отображается при каждом нажатии кнопки входа.

Внутри LoginActivity макета LoginActivity, кроме EditText и Button, у нас также есть TextView с идентификатором R.id.tv\_login . Поэтому мы просто делаем проверку, чтобы подтвердить, что TextView видим для пользователя.

@RunWith(AndroidJUnit4::class)  
class MainActivityTest {  
  
 @Rule @JvmField  
 var activityRule = ActivityTestRule<MainActivity>(  
 MainActivity::class.*java* )  
  
 @Test  
 fun clickLoginButton\_opensLoginUi() {

onView(withId(R.id.*btn\_login*)).check(matches(isDisplayed()))  
 onView(withId(R.id.*btn\_login*)).perform(click())  
 onView(withId(R.id.*tv\_login*)).check(matches(isDisplayed()))  
 }  
}

Результат запуска теста



В результате написания и запуска теста мы:

1. Запустили MainActivity с помощью поля activityRule.
2. Проверили, была ли кнопка **входа** в **систему** (R.id.btn\_login) видимой (isDisplayed()) для пользователя.
3. Имитировал действие нажатия (click()) на эту кнопку.
4. Проверили, отображается ли TextView с идентификатором R.id.tv\_login в LoginActivity .

Протестируем экран LoginActivity

Сценарии LoginActivity: если введенное имя пользователя — «user», а пароль — «1234», то вход успешен. Для любого другого входа это сбой.

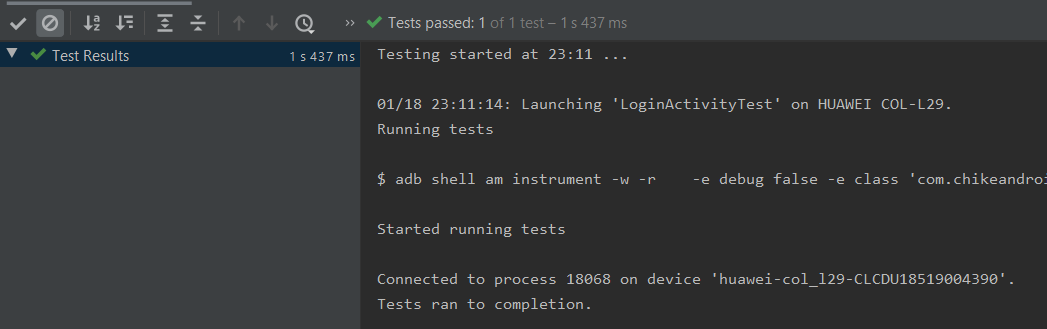
|  |  |
| --- | --- |
|  |  |

Теперь напишем тест эспрессо.

@RunWith(AndroidJUnit4::class)  
class LoginActivityTest {  
  
 @Rule  
 @JvmField  
 var activityRule = ActivityTestRule<LoginActivity>(  
 LoginActivity::class.*java* )  
  
 private val username = "user"  
 private val password = "1234"  
  
 @Test  
 fun clickLoginButton\_opensLoginUi() {  
 onView(withId(R.id.*et\_username*)).perform(ViewActions.typeText(username))  
 onView(withId(R.id.*et\_password*)).perform(ViewActions.typeText(password))  
  
 onView(withId(R.id.*btn\_submit*)).perform(ViewActions.scrollTo(), ViewActions.click())  
  
 Espresso.onView(withId(R.id.*tv\_login*))  
 .check(matches(withText("Success")))  
 }  
}

Этот тестовый класс очень похож на наш первый.  Имя пользователя и пароль вводятся в поля R.id.et\_username и R.id.et\_password соответственно. Затем Espresso нажимает кнопку «**Отправить»** (R.id.btn\_submit ). Он будет ждать, пока не будет найден View с идентификатором R.id.tv\_login для чтения текста.

Результат запуска теста

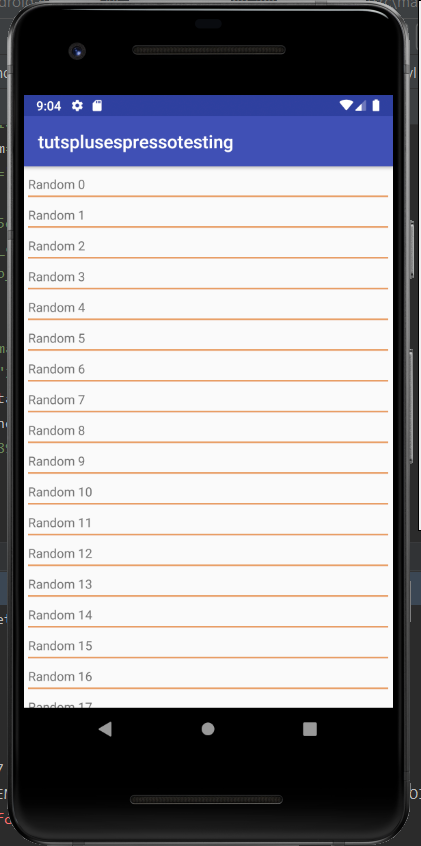


Протестируем RecyclerView

[RecyclerViewActions](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.ru&sl=en&sp=nmt4&tl=ru&u=https://developer.android.com/reference/android/support/test/espresso/contrib/RecyclerViewActions.html&xid=17259,15700019,15700186,15700191,15700259,15700271,15700302&usg=ALkJrhhdrkXg64RSNKI6jBkJg-drT7KUSw) — это класс, который предоставляет набор API для работы с RecyclerView . RecyclerViewActions является частью отдельного артефакта внутри артефакта espresso-contrib , который также должен быть добавлен в **build.gradle** :

androidTestImplementation 'com.android.support.test.espresso:espresso-contrib:3.0.2'

Вот как выглядит наш макет MyListActivity:



Xml-файл MyListActivity

<?xml version="1.0" encoding="utf-8"?>  
<android.support.v7.widget.RecyclerView xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:id="@+id/rv"  
 tools:context=".MyListActivity"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent" />

Xml-файл элемента RecyclerView

<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:orientation="vertical"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content">  
  
<TextView  
 android:id="@+id/tv\_country"  
 android:layout\_marginTop="10dp"  
 android:layout\_marginBottom="3dp"  
 android:layout\_marginStart="5dp"  
 android:textSize="14sp"  
 android:layout\_marginEnd="5dp"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"/>  
  
 <View  
 android:layout\_width="match\_parent"  
 android:layout\_height="2dp"  
 android:layout\_marginStart="5dp"  
 android:layout\_marginEnd="5dp"  
 android:background="#E89E65" />  
  
  
</LinearLayout>

Kotlin class MyListActivity

class MyListActivity : AppCompatActivity() {  
  
 private lateinit var recyclerView: RecyclerView  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_mylist*)  
  
 recyclerView = findViewById(R.id.*rv*)  
  
 var adapter = RandomAdapter(generateData())  
 val layoutManager = LinearLayoutManager(*applicationContext*)  
 recyclerView?.*layoutManager* = layoutManager  
 recyclerView?.*itemAnimator* = DefaultItemAnimator()  
  
 recyclerView?.*adapter* = adapter  
 adapter.notifyDataSetChanged()  
 }  
  
 fun generateData(): ArrayList<Random> {  
 var result = ArrayList<Random>()  
  
 for (i in 0..20) {  
 var country = Random("Random $i")  
 result.add(country)  
 }  
  
 return result  
 }  
}

adapter списка

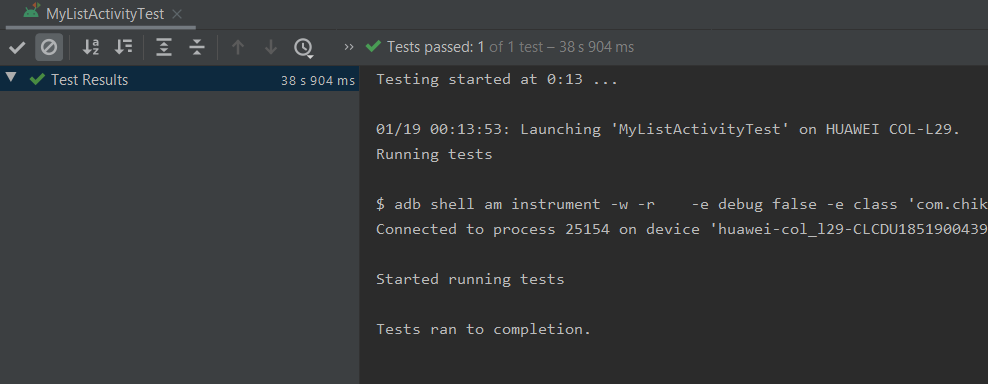
class RandomAdapter(private var items: ArrayList<Random>): RecyclerView.Adapter<RandomAdapter.ViewHolder>() {  
  
 override fun getItemCount(): Int {  
 return items.size  
 }  
  
 override fun onBindViewHolder(holder: ViewHolder, position: Int) {  
 var country = items[position]  
 holder?.txtName?.*text* = country.name  
 }  
  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {  
 val itemView = LayoutInflater.from(parent?.*context*)  
 .inflate(R.layout.*item\_country*, parent, false)  
  
 return ViewHolder(itemView)  
 }  
  
 class ViewHolder(row: View) : RecyclerView.ViewHolder(row) {  
 var txtName: TextView? = null  
  
 init {  
 this.txtName = row?.findViewById(R.id.*tv\_country*)  
 }  
 }  
}

Чтобы щелкнуть элемент в любой позиции в RecyclerView, мы вызываем actionOnItemAtPosition(). Мы должны дать ему тип. В нашем случае элемент является классом ViewHolder внутри нашего RandomAdapter. Этот метод также принимает два параметра; первая — это позиция, а вторая — действие (ViewActions.click()).

Теперь напишем тест эспрессо

@RunWith(AndroidJUnit4::class)  
class MyListActivityTest {  
  
 @Rule  
 @JvmField  
 var activityRule = ActivityTestRule<MyListActivity>(  
 MyListActivity::class.*java* )  
  
 @Test  
 fun clickItem() {  
 onView(withId(R.id.*rv*))  
 .perform(RecyclerViewActions  
 .actionOnItemAtPosition<RandomAdapter.ViewHolder>(0, ViewActions.click()))  
  
 }  
}

Результат запуска теста



# Вывод

В рамках выполнения курсового проекта было реализовано UI – тестирование мобильных приложений. Узнали, как легко использовать среду тестирования Espresso в проекте Android Studio для автоматизации рабочего процесса тестирования.

В процессе были выполнены поставленные цели, а именно:

* Были закреплены знания и навыки, полученные в рамках курса “Тестирование программных систем”;
* Был получен практический опыт проектирования и реализации интеграционных тестов.

# Список литературы

1. <https://www.tutorialspoint.com/espresso_testing/espresso_testing_intents.htm>
2. <https://espressomd.org/wordpress/documentation/>
3. <http://developer.alexanderklimov.ru/android/debug/espresso.php>
4. <https://www.fandroid.info/instrumental_testing_android_espresso_dagger2/>
5. <https://coderlessons.com/tutorials/kachestvo-programmnogo-obespecheniia/izuchite-espresso-testirovanie/espresso-testing-framework-kratkoe-rukovodstvo>