Testing Policy Document

Stacks on Stacks

Group Members:

Akua Afrane-Okese U15019773

Linda Zwane U14199468

Ntiko Mathaba U14012503

Tyler Matthews U15302424

Contents

Introduction

Purpose of Document

The purpose of this document is to provide a clear statement as to the testing process for software and to outline who is responsible for which parts of that process.

Purpose of Policy

Testing is needed in order to detect potential problems within the software as early as possible, so that they can be corrected at minimum cost.

Mission of Testing

- To strive towards perfection in the delivery of a software product which is as defect-free as possible.
- To constantly seek better methods and processes to help insure the delivery of a quality product.
- To fulfil software requirements as stipulated in the systems requirement documentation

Description of the test process

For testing, the focus is on making sure functions run with the different possible cases. We have decided to use Test-Driven Development. Test driven development allows us to continuously test our code throughout the development cycle of our application. Tests are performed immediately after coding the classes to mitigate errors in advance together with each time an update is made. The following are the steps that we have used for our testing process:

1. Prepare for test-driven development

The skeleton code for the class is generated. The team member that is in charge of testing the class needs to fully understand the classes implementation. A UML diagram will aid the team member in understanding the class.

Testing Tools

- JUnit Testing Framework (4.12)

  JUnit allows for programmers to work on applications and their testing from one platform, which allows access to package private elements within the application. It provides annotations to easily distinguish between the application and testing

  The statement to include its functionality is: testImplementation 'junit:junit:4.12' (see in Figure 1).To use JUnit, its statement needs to be included in the Android Studio project gradle, in the section called dependency.

- Mockito (1.10.9)

  Mockito is used to mock interfaces so that dummy functionalities (or functionality) can be added to a mock interface that can be used in unit testing so that classes can be tested on their own, without testing dependencies to verify that the code being tested works without dependencies. It also caters for resources that are not actually available.

  The statement to include its functionality is: testImplementation 'org.mockito:mockito-core:1.10.19' (see in Figure 1). To include its functionality, its statement needs to be included in the Android Studio project gradle, in the dependency section.

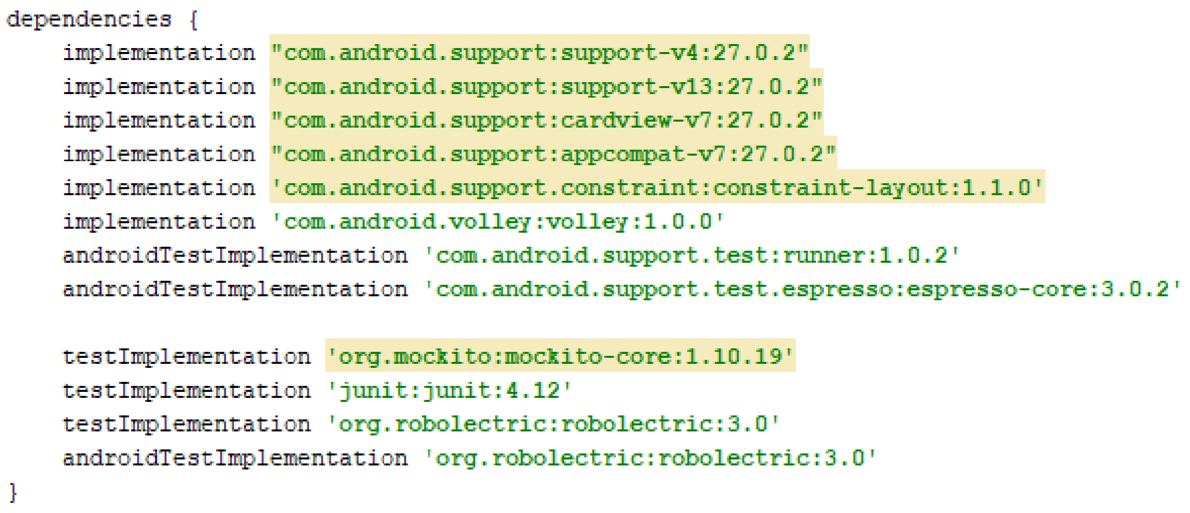
- Espresso (3.0.2)

  Espresso is used to create automated user interface tests. It is based on JUnit, thus easy to include. Espresso also needs to be stated in the dependency section. Its statement being: androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2' (see in Figure 1).

- Robolectric (3.0)

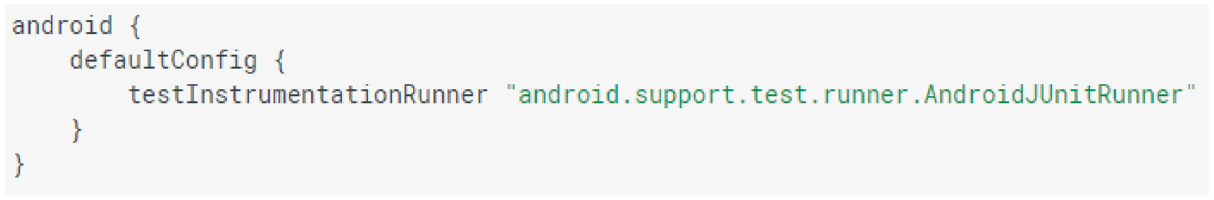
  Robolectric also needs to be stated in the dependency section. Here is its statement: androidTestImplementation 'org.robolectric:robolectric:3.0'. Not all areas need to be mocked (use mocking frameworks like Mockito). Robolectric lets you run your tests on your workstation, or on your Continuous Integration environment in a regular JVM, without an emulator.
- AndroidJUnitRunner

"Instrumentation tests are used for testing Android Frameworks such as UI, SharedPreferences and so on. Since they are for Android Framework they are run on a device or an emulator" (Ajesh, 2018).



```
dependencies {
    implementation "com.android.support:support-v4:27.0.2"
    implementation "com.android.support:support-v13:27.0.2"
    implementation "com.android.support:cardview-v7:27.0.2"
    implementation "com.android.support:appcompat-v7:27.0.2"
    implementation 'com.android.support.constraint:constraint-layout:1.1.0'
    implementation 'com.android.volley:volley:1.0.0'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'

    testImplementation 'org.mockito:mockito-core:1.10.19'
    testImplementation 'junit:junit:4.12'
    testImplementation 'org.robolectric:robolectric:3.0'
    androidTestImplementation 'org.robolectric:robolectric:3.0'
}
```

Figure 1.1: StacksonStacks Part 1 Project Gradle in Android Studio



```
android {
    defaultConfig {
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
}
```

Figure 1.2: StacksonStacks Part 2 Project Gradle in Android Studio

Levels of Testing and approaches used

The different levels of testing promote mitigation and quality risk as early as possible.

Level	Owner	Objective	Key Areas of Testing
Unit	Developer	• Detect defective units of code • Reduce the risk of unit failure in production	• Functionality • Resource Utilization
Integration	Developer	• Detect defects in unit interfaces • Reduce risk of dataflow and workflow failures in production	• Functionality • Unit interoperability and compatibility • Performance
Acceptance	Business owner	• Detect in user workflows	• Functionality

		• Demonstrate that the product works as expected • Ensure that all the requirements are met	• Operational processes

2. Write Tests

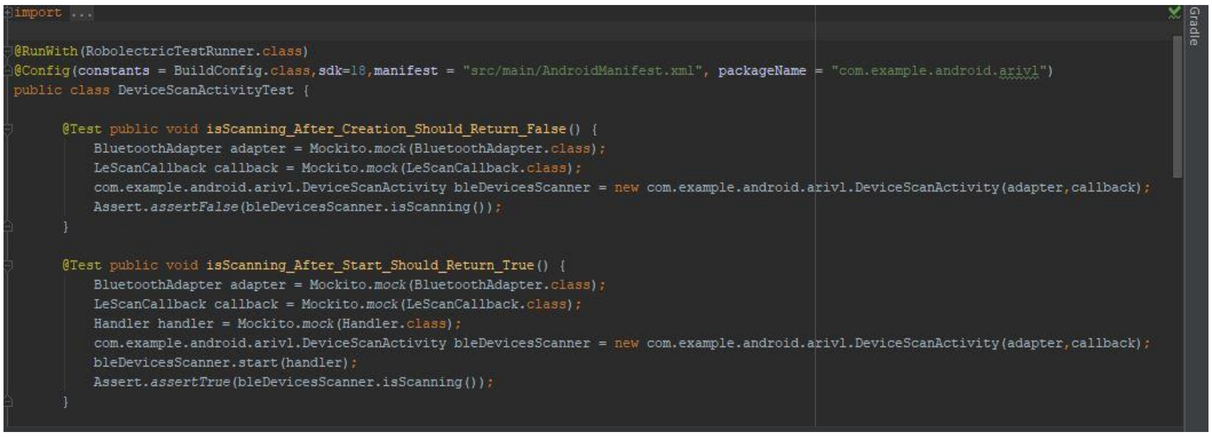
2.1. Unit Testing

The following scenarios are tested on (unit testing):

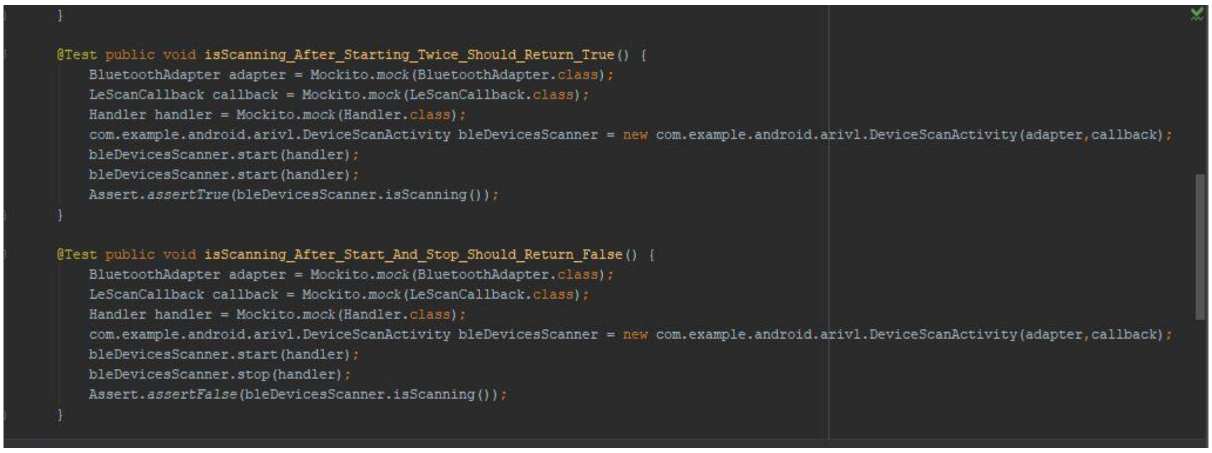
1. Device Scanning
2. Bluetooth Connectivity and Bluetooth Initialization
3. Device Control
4. Gatt Attributes
5. Login

1. Device Scanning

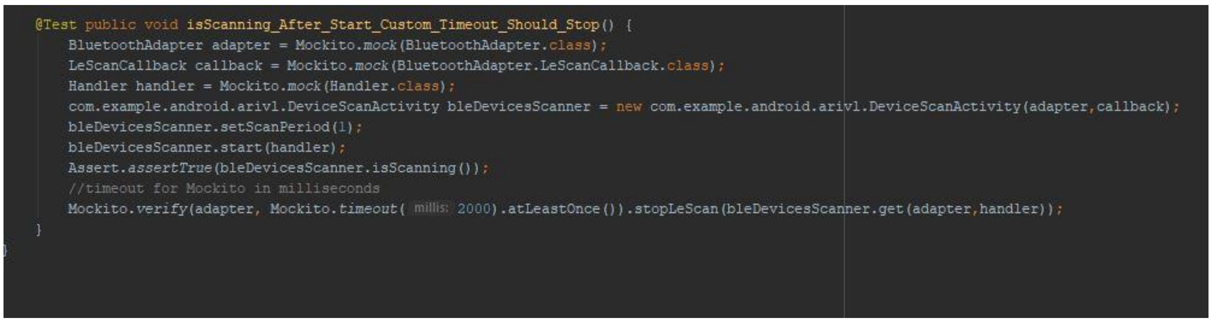
- Testing if it is scanning only after creation (Expected Result: False)
- Testing if scanning after creation and instructed to scan for BLE device(Expected Result: True)
- Testing if device continues scanning after instructed to twice (Expected Result: True)
- Testing if device is scanning after instructed to start and then stop scanning(Expected Result: False)
- Testing to verify if device is scanning a custom timeout period(Expected Result:False)



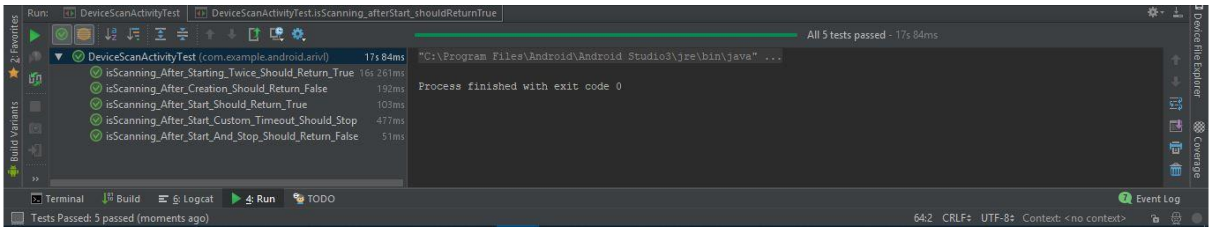
```java
import ...

@RunWith(RobolectricTestRunner.class)
@Config(constants = BuildConfig.class,sdk=18,manifest = "src/main/AndroidManifest.xml", packageName = "com.example.android.arivl")
public class DeviceScanActivityTest {

    @Test public void isScanning_After_Creation_Should_Return_False() {
        BluetoothAdapter adapter = Mockito.mock(BluetoothAdapter.class);
        LeScanCallback callback = Mockito.mock(LeScanCallback.class);
        com.example.android.arivl.DeviceScanActivity bleDevicesScanner = new com.example.android.arivl.DeviceScanActivity(adapter,callback);
        Assert.assertFalse(bleDevicesScanner.isScanning());
    }

    @Test public void isScanning_After_Start_Should_Return_True() {
        BluetoothAdapter adapter = Mockito.mock(BluetoothAdapter.class);
        LeScanCallback callback = Mockito.mock(LeScanCallback.class);
        Handler handler = Mockito.mock(Handler.class);
        com.example.android.arivl.DeviceScanActivity bleDevicesScanner = new com.example.android.arivl.DeviceScanActivity(adapter,callback);
        bleDevicesScanner.start(handler);
        Assert.assertTrue(bleDevicesScanner.isScanning());
    }
```



```java
    }

    @Test public void isScanning_After_Starting_Twice_Should_Return_True() {
        BluetoothAdapter adapter = Mockito.mock(BluetoothAdapter.class);
        LeScanCallback callback = Mockito.mock(LeScanCallback.class);
        Handler handler = Mockito.mock(Handler.class);
        com.example.android.arivl.DeviceScanActivity bleDevicesScanner = new com.example.android.arivl.DeviceScanActivity(adapter,callback);
        bleDevicesScanner.start(handler);
        bleDevicesScanner.start(handler);
        Assert.assertTrue(bleDevicesScanner.isScanning());
    }

    @Test public void isScanning_After_Start_And_Stop_Should_Return_False() {
        BluetoothAdapter adapter = Mockito.mock(BluetoothAdapter.class);
        LeScanCallback callback = Mockito.mock(LeScanCallback.class);
        Handler handler = Mockito.mock(Handler.class);
        com.example.android.arivl.DeviceScanActivity bleDevicesScanner = new com.example.android.arivl.DeviceScanActivity(adapter,callback);
        bleDevicesScanner.start(handler);
        bleDevicesScanner.stop(handler);
        Assert.assertFalse(bleDevicesScanner.isScanning());
    }
```



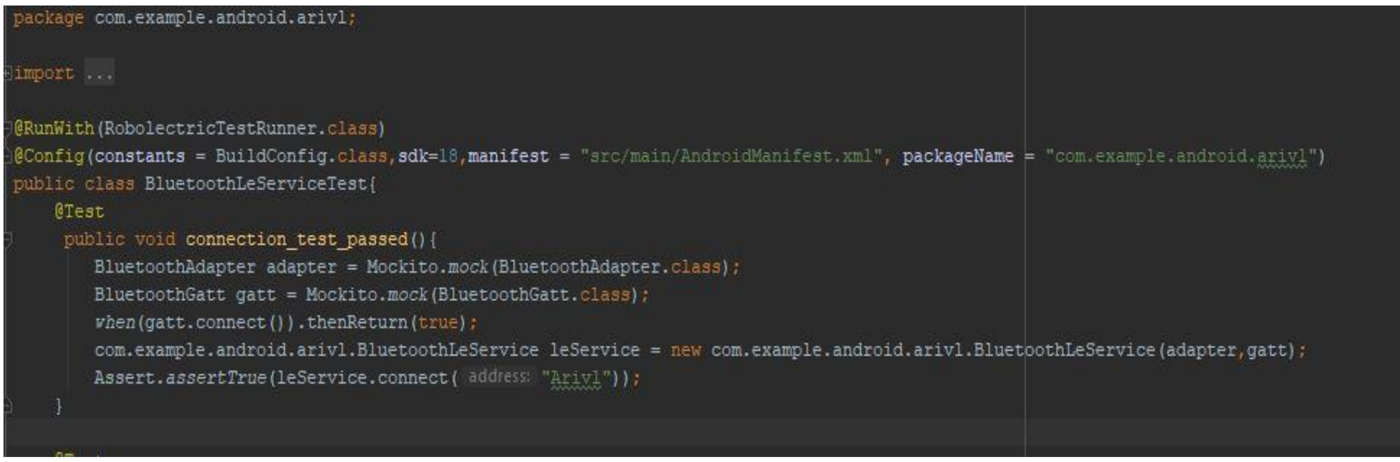
```java
    @Test public void isScanning_After_Start_Custom_Timeout_Should_Stop() {
        BluetoothAdapter adapter = Mockito.mock(BluetoothAdapter.class);
        LeScanCallback callback = Mockito.mock(BluetoothAdapter.LeScanCallback.class);
        Handler handler = Mockito.mock(Handler.class);
        com.example.android.arivl.DeviceScanActivity bleDevicesScanner = new com.example.android.arivl.DeviceScanActivity(adapter,callback);
        bleDevicesScanner.setScanPeriod(1);
        bleDevicesScanner.start(handler);
        Assert.assertTrue(bleDevicesScanner.isScanning());
        //timeout for Mockito in milliseconds
        Mockito.verify(adapter, Mockito.timeout( millis: 2000).atLeastOnce()).stopLeScan(bleDevicesScanner.get(adapter,handler));
    }
}
```



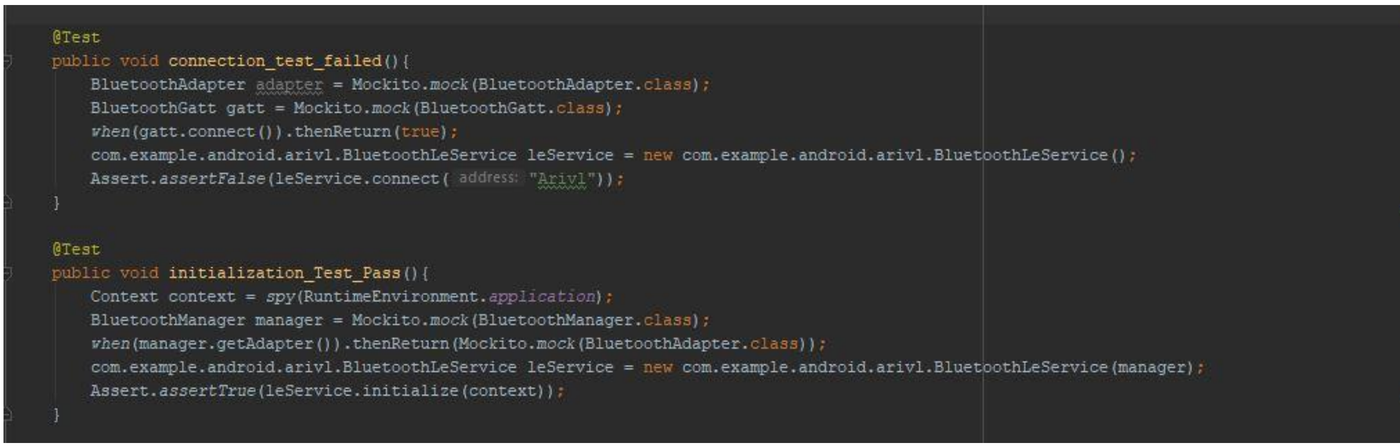
```
Run:  DeviceScanActivityTest  DeviceScanActivityTest.isScanning_afterStart_shouldReturnTrue

All 5 tests passed - 17s 84ms

DeviceScanActivityTest (com.example.android.arivl)          17s 84ms
    isScanning_After_Starting_Twice_Should_Return_True     16s 261ms
    isScanning_After_Creation_Should_Return_False              192ms
    isScanning_After_Start_Should_Return_True                  103ms
    isScanning_After_Start_Custom_Timeout_Should_Stop          477ms
    isScanning_After_Start_And_Stop_Should_Return_False         51ms

"C:\Program Files\Android\Android Studio3\jre\bin\java" ...

Process finished with exit code 0

Tests Passed: 5 passed (moments ago)          64:2  CRLF‡  UTF-8‡  Context: <no context>
```

2. Device Connectivity

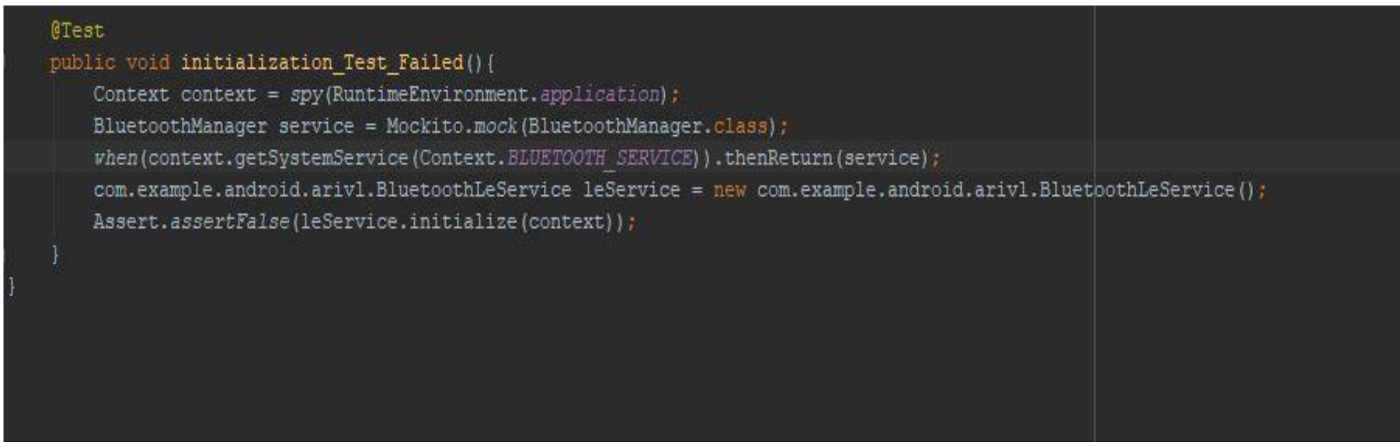
- testing if connection is established (Expected Result : True)
- testing if connection is not established (Expected Result: False)
- testing if bluetooth initialized (Expected Result: True)
- testing if bluetooth not initialized (Expected Result: False)



```
package com.example.android.arivl;

import ...

@RunWith(RobolectricTestRunner.class)
@Config(constants = BuildConfig.class,sdk=18,manifest = "src/main/AndroidManifest.xml", packageName = "com.example.android.arivl")
public class BluetoothLeServiceTest{
    @Test
     public void connection_test_passed(){
        BluetoothAdapter adapter = Mockito.mock(BluetoothAdapter.class);
        BluetoothGatt gatt = Mockito.mock(BluetoothGatt.class);
        when(gatt.connect()).thenReturn(true);
        com.example.android.arivl.BluetoothLeService leService = new com.example.android.arivl.BluetoothLeService(adapter,gatt);
        Assert.assertTrue(leService.connect( address: "Arivl"));
    }
```

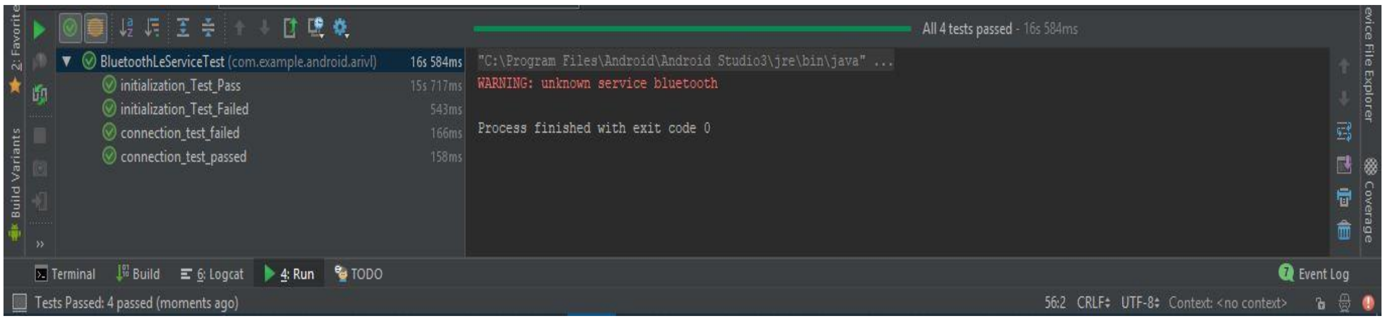


```
    @Test
    public void connection_test_failed(){
        BluetoothAdapter adapter = Mockito.mock(BluetoothAdapter.class);
        BluetoothGatt gatt = Mockito.mock(BluetoothGatt.class);
        when(gatt.connect()).thenReturn(true);
        com.example.android.arivl.BluetoothLeService leService = new com.example.android.arivl.BluetoothLeService();
        Assert.assertFalse(leService.connect( address: "Arivl"));
    }

    @Test
    public void initialization_Test_Pass(){
        Context context = spy(RuntimeEnvironment.application);
        BluetoothManager manager = Mockito.mock(BluetoothManager.class);
        when(manager.getAdapter()).thenReturn(Mockito.mock(BluetoothAdapter.class));
        com.example.android.arivl.BluetoothLeService leService = new com.example.android.arivl.BluetoothLeService(manager);
        Assert.assertTrue(leService.initialize(context));
    }
```



```
    @Test
    public void initialization_Test_Failed(){
        Context context = spy(RuntimeEnvironment.application);
        BluetoothManager service = Mockito.mock(BluetoothManager.class);
        when(context.getSystemService(Context.BLUETOOTH_SERVICE)).thenReturn(service);
        com.example.android.arivl.BluetoothLeService leService = new com.example.android.arivl.BluetoothLeService();
        Assert.assertFalse(leService.initialize(context));
    }
}
```

(Above are images of the code for the test cases)



```
BluetoothLeServiceTest (com.example.android.arivl)   16s 584ms
    initialization_Test_Pass                          15s 717ms
    initialization_Test_Failed                           543ms
    connection_test_failed                               166ms
    connection_test_passed                               158ms
```

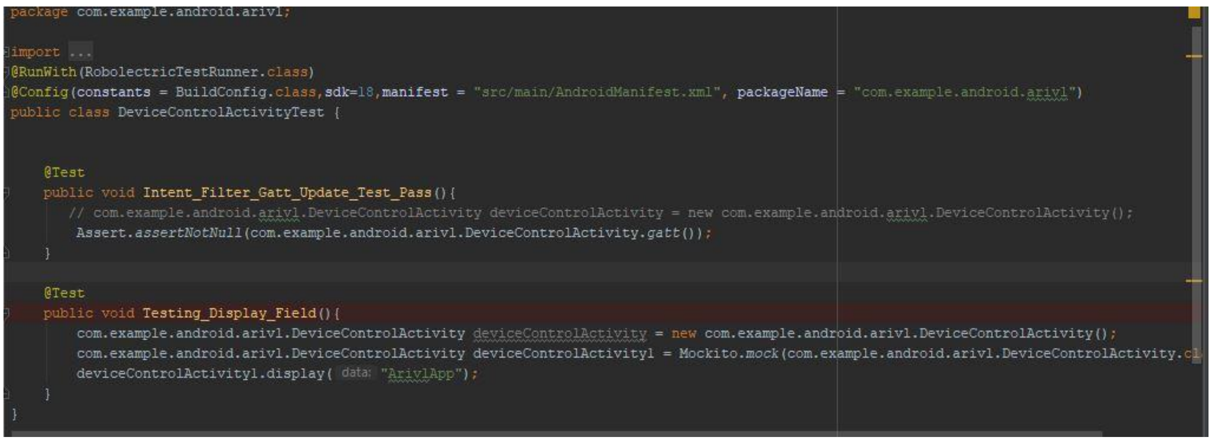
```
All 4 tests passed - 16s 584ms

"C:\Program Files\Android\Android Studio3\jre\bin\java" ...
WARNING: unknown service bluetooth

Process finished with exit code 0
```

```
Tests Passed: 4 passed (moments ago)
```

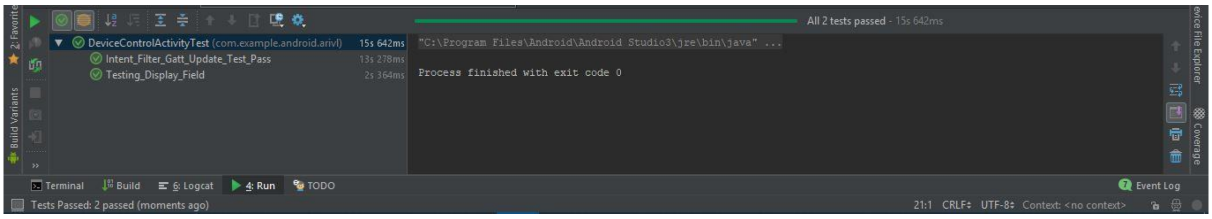
(Above is a screenshot of all 4 Test Cases Passing for Connectivity)

3. Device Control

- To ensure Intent Filter updates (Expected Result : NotNull)
- To ensure the display field is updated (Expected Result : NotNull)



```java
package com.example.android.arivl;

import ...
@RunWith(RobolectricTestRunner.class)
@Config(constants = BuildConfig.class,sdk=18,manifest = "src/main/AndroidManifest.xml", packageName = "com.example.android.arivl")
public class DeviceControlActivityTest {


    @Test
    public void Intent_Filter_Gatt_Update_Test_Pass(){
      // com.example.android.arivl.DeviceControlActivity deviceControlActivity = new com.example.android.arivl.DeviceControlActivity();
        Assert.assertNotNull(com.example.android.arivl.DeviceControlActivity.gatt());
    }

    @Test
    public void Testing_Display_Field(){
        com.example.android.arivl.DeviceControlActivity deviceControlActivity = new com.example.android.arivl.DeviceControlActivity();
        com.example.android.arivl.DeviceControlActivity deviceControlActivity1 = Mockito.mock(com.example.android.arivl.DeviceControlActivity.cl
        deviceControlActivity1.display( data: "ArivlApp");
    }
}
```



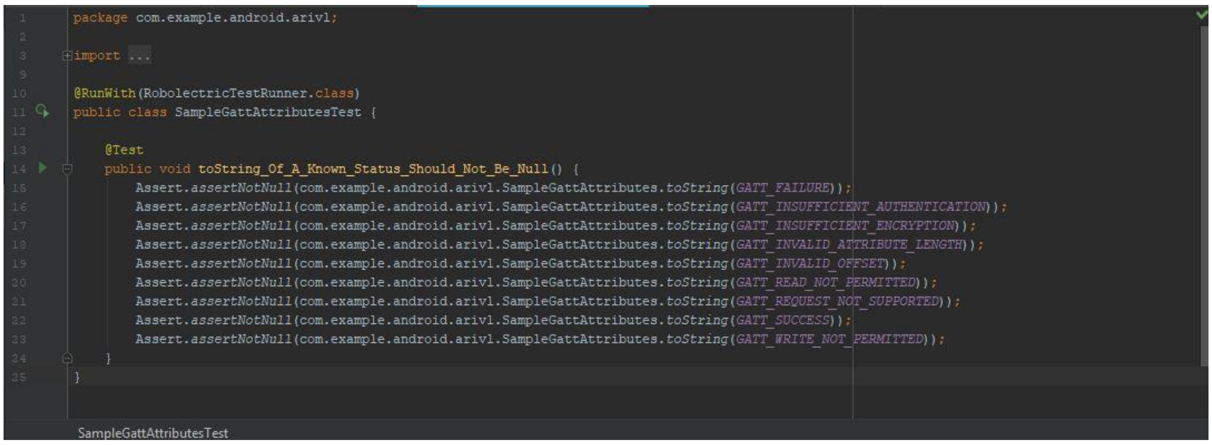
```
DeviceControlActivityTest (com.example.android.arivl)   15s 642ms
    Intent_Filter_Gatt_Update_Test_Pass                 13s 278ms
    Testing_Display_Field                                2s 364ms
```

```
All 2 tests passed - 15s 642ms

"C:\Program Files\Android\Android Studio3\jre\bin\java" ...

Process finished with exit code 0
```

```
Tests Passed: 2 passed (moments ago)
```

4. Gatt Attributes

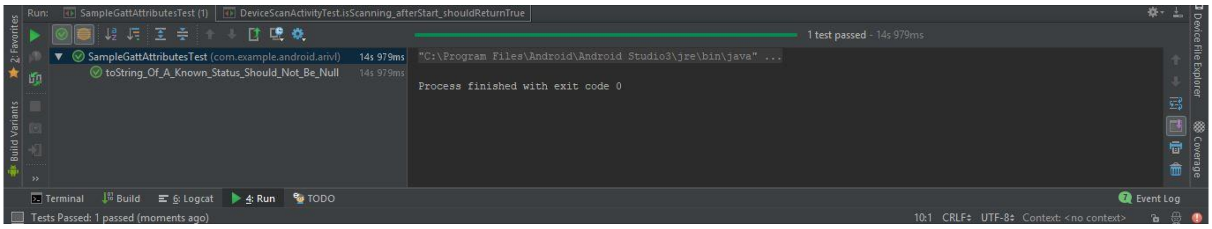
- To ensure read/write permissions (Expected Result : NotNull)
- To ensure sufficient encryption and authentication (Expected Result : NotNull)
- To ensure request permissions (Expected Result : NotNull)



```java
package com.example.android.arivl;

import ...

@RunWith(RobolectricTestRunner.class)
public class SampleGattAttributesTest {

    @Test
    public void toString_Of_A_Known_Status_Should_Not_Be_Null() {
        Assert.assertNotNull(com.example.android.arivl.SampleGattAttributes.toString(GATT_FAILURE));
        Assert.assertNotNull(com.example.android.arivl.SampleGattAttributes.toString(GATT_INSUFFICIENT_AUTHENTICATION));
        Assert.assertNotNull(com.example.android.arivl.SampleGattAttributes.toString(GATT_INSUFFICIENT_ENCRYPTION));
        Assert.assertNotNull(com.example.android.arivl.SampleGattAttributes.toString(GATT_INVALID_ATTRIBUTE_LENGTH));
        Assert.assertNotNull(com.example.android.arivl.SampleGattAttributes.toString(GATT_INVALID_OFFSET));
        Assert.assertNotNull(com.example.android.arivl.SampleGattAttributes.toString(GATT_READ_NOT_PERMITTED));
        Assert.assertNotNull(com.example.android.arivl.SampleGattAttributes.toString(GATT_REQUEST_NOT_SUPPORTED));
        Assert.assertNotNull(com.example.android.arivl.SampleGattAttributes.toString(GATT_SUCCESS));
        Assert.assertNotNull(com.example.android.arivl.SampleGattAttributes.toString(GATT_WRITE_NOT_PERMITTED));
    }
}
```

SampleGattAttributesTest

-



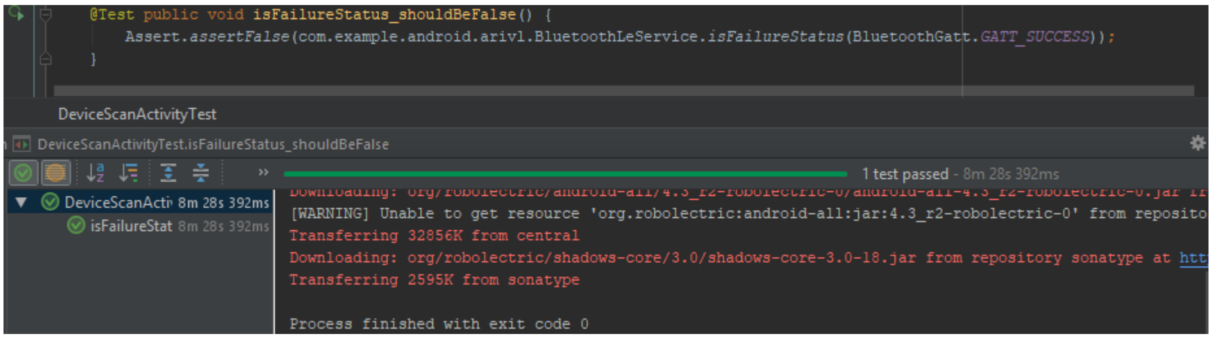
Run: SampleGattAttributesTest (1) | DeviceScanActivityTest.isScanning_afterStart_shouldReturnTrue

1 test passed - 14s 979ms

```
SampleGattAttributesTest (com.example.android.arivl)   14s 979ms
    toString_Of_A_Known_Status_Should_Not_Be_Null      14s 979ms
```

```
"C:\Program Files\Android\Android Studio3\jre\bin\java" ...

Process finished with exit code 0
```

Tests Passed: 1 passed (moments ago)



```java
@Test public void isFailureStatus_shouldBeFalse() {
    Assert.assertFalse(com.example.android.arivl.BluetoothLeService.isFailureStatus(BluetoothGatt.GATT_SUCCESS));
}
```

DeviceScanActivityTest

DeviceScanActivityTest.isFailureStatus_shouldBeFalse

1 test passed - 8m 28s 392ms

```
DeviceScanActi 8m 28s 392ms
    isFailureStat 8m 28s 392ms
```

```
Downloading: org/robolectric/android-all/4.3_r2-robolectric-0/android-all-4.3_r2-robolectric-0.jar fr
[WARNING] Unable to get resource 'org.robolectric:android-all:jar:4.3_r2-robolectric-0' from reposito
Transferring 32856K from central
Downloading: org/robolectric/shadows-core/3.0/shadows-core-3.0-18.jar from repository sonatype at htt
Transferring 2595K from sonatype

Process finished with exit code 0
```

5. These are the following test cases that are going to be conducted to test the login functionality:

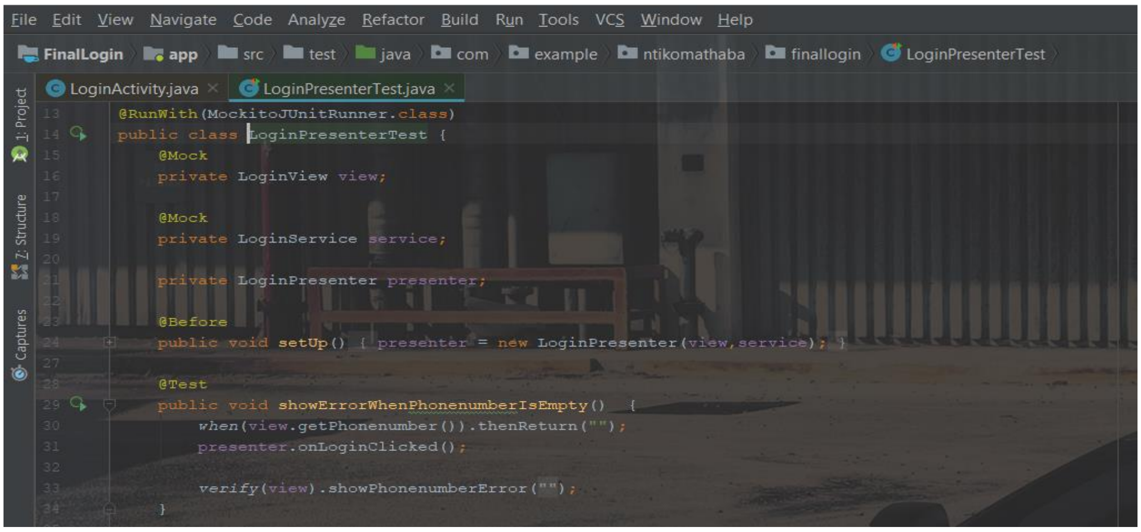
Steps	**Expected Results**
Enter a valid username and password. Click Login button	The application should display the Splash Screen
Logout. Enter a valid username and an invalid password. Click the Login button.	The application should display an error message and reopen the Login page
Enter an invalid username and an invalid password. Click Login button	The application should display an error message and reopen the Login page
Leave the username and password blank. Click Login button.	The application should display an error message and reopen the Login page

The following are the possible combinations input values the user might enter and click on the login button. A test can only be deemed passed if the expected result matches the actual result. The results are as follows:

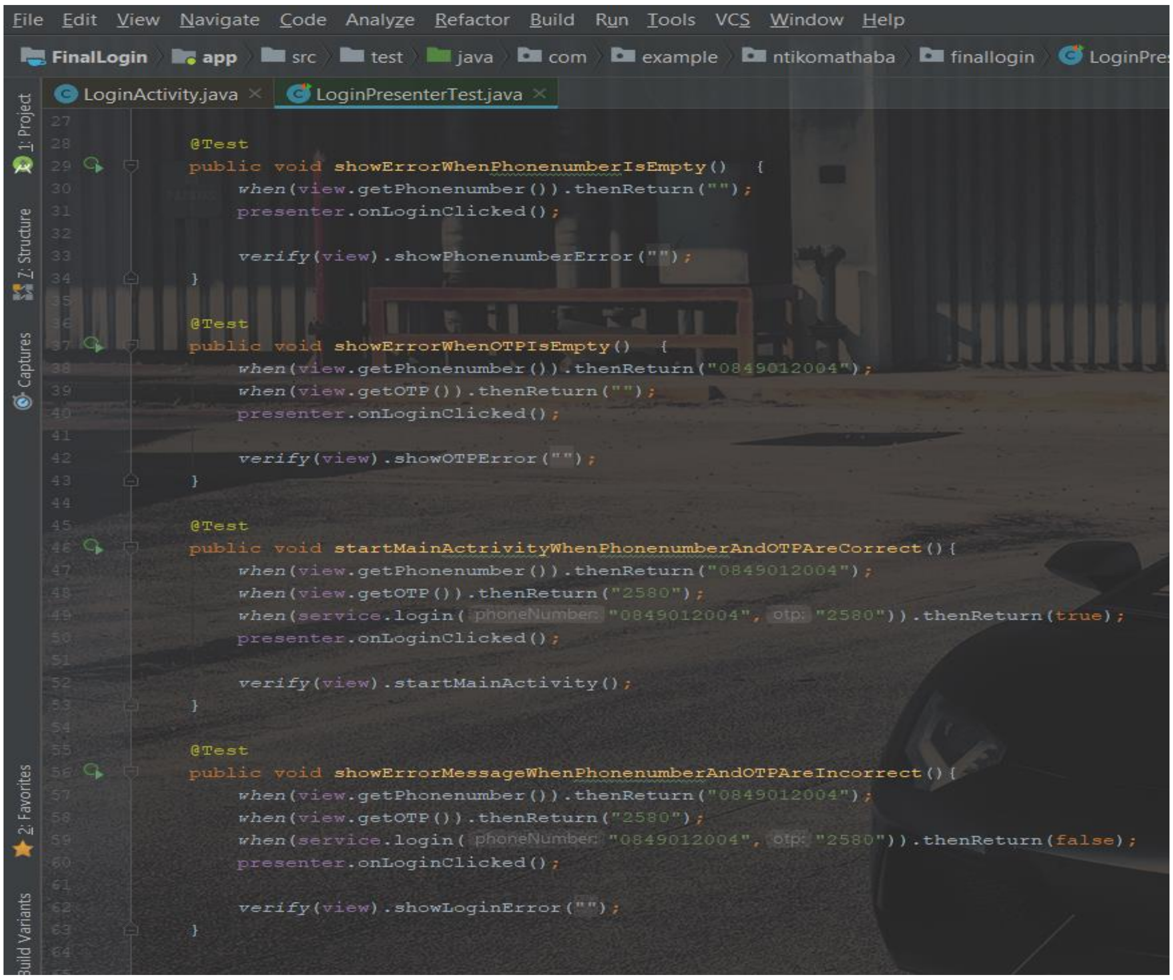
Verify username and password	**Expected Result**	**Actual Result**	**Home Page / Error**
Valid username, Valid Password	Success	Success	Home Page
Valid username, Invalid Password	Fail	Fail	Error
Valid username, Blank	Fail	Fail	Error
Invalid username, Valid Password	Fail	Fail	Error
Invalid username, Invalid Password	Fail	Fail	Error
Blank, Invalid Password	Fail	Fail	Error
Blank, Valid Password	Fail	Fail	Error
Blank, Invalid Password	Fail	Fail	Error
Blank, Blank	Fail	Fail	Error

The above table shows us the different combinations that the user might enter when trying to login into the Arivl application and the result of the input combination. It shows us that there is only one case that the user will be able to login into the Arivl application and that is when the user has entered a valid username and valid password.

Down below is a screenshot of the different tests we have conducted using Junits Automated Unit Testing framework

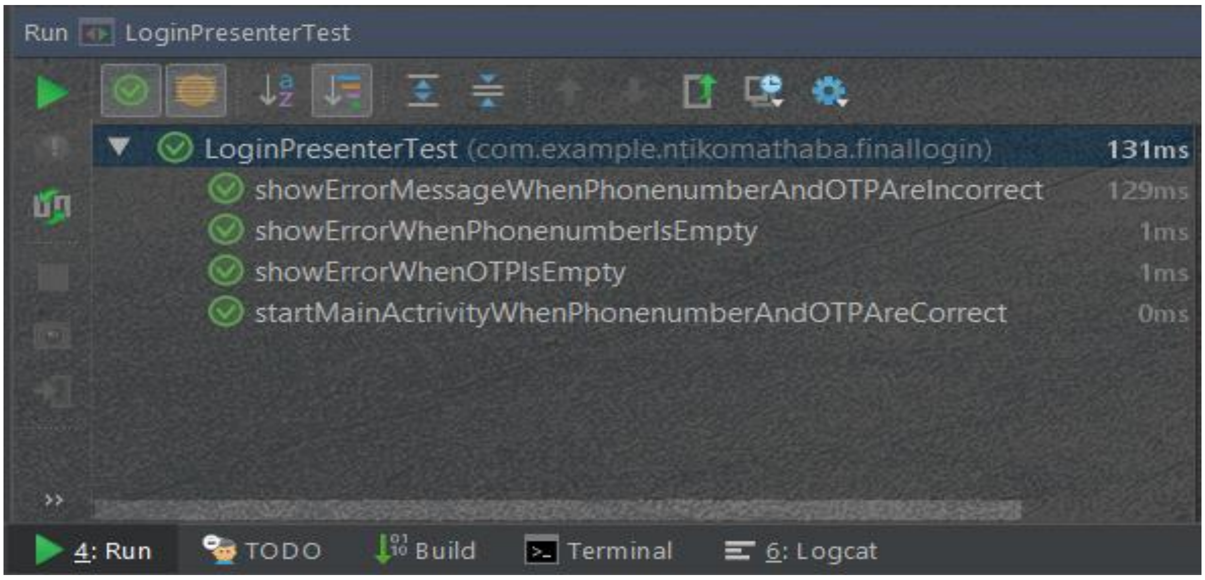

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
FinalLogin app src test java com example ntikomathaba finallogin LoginPresenterTest
LoginActivity.java
LoginPresenterTest.java
1: Project
7: Structure
Captures

```
@RunWith(MockitoJUnitRunner.class)
public class LoginPresenterTest {
    @Mock
    private LoginView view;

    @Mock
    private LoginService service;

    private LoginPresenter presenter;

    @Before
    public void setUp() { presenter = new LoginPresenter(view,service); }

    @Test
    public void showErrorWhenPhonenumberIsEmpty()  {
        when(view.getPhonenumber()).thenReturn("");
        presenter.onLoginClicked();

        verify(view).showPhonenumberError("");
    }
```


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
FinalLogin app src test java com example ntikomathaba finallogin LoginPres
LoginActivity.java
LoginPresenterTest.java
1: Project
7: Structure
Captures
2: Favorites
Build Variants

```

    @Test
    public void showErrorWhenPhonenumberIsEmpty()  {
        when(view.getPhonenumber()).thenReturn("");
        presenter.onLoginClicked();

        verify(view).showPhonenumberError("");
    }

    @Test
    public void showErrorWhenOTPIsEmpty()  {
        when(view.getPhonenumber()).thenReturn("0849012004");
        when(view.getOTP()).thenReturn("");
        presenter.onLoginClicked();

        verify(view).showOTPError("");
    }

    @Test
    public void startMainActrivityWhenPhonenumberAndOTPAreCorrect(){
        when(view.getPhonenumber()).thenReturn("0849012004");
        when(view.getOTP()).thenReturn("2580");
        when(service.login( phoneNumber: "0849012004", otp: "2580")).thenReturn(true);
        presenter.onLoginClicked();

        verify(view).startMainActivity();
    }

    @Test
    public void showErrorMessageWhenPhonenumberAndOTPAreIncorrect(){
        when(view.getPhonenumber()).thenReturn("0849012004");
        when(view.getOTP()).thenReturn("2580");
        when(service.login( phoneNumber: "0849012004", otp: "2580")).thenReturn(false);
        presenter.onLoginClicked();

        verify(view).showLoginError("");
    }

```

Down below is a screen shot of the tests that we have written and they have passed. They have been sorted according to time duration. It is evident that the test that takes the longest is when the user enters an invalid username and invalid password.



```
Run  LoginPresenterTest
LoginPresenterTest (com.example.ntikomathaba.finallogin)    131ms
    showErrorMessageWhenPhonenumberAndOTPAreIncorrect       129ms
    showErrorWhenPhonenumberIsEmpty                           1ms
    showErrorWhenOTPIsEmpty                                   1ms
    startMainActivityWhenPhonenumberAndOTPAreCorrect          0ms
```

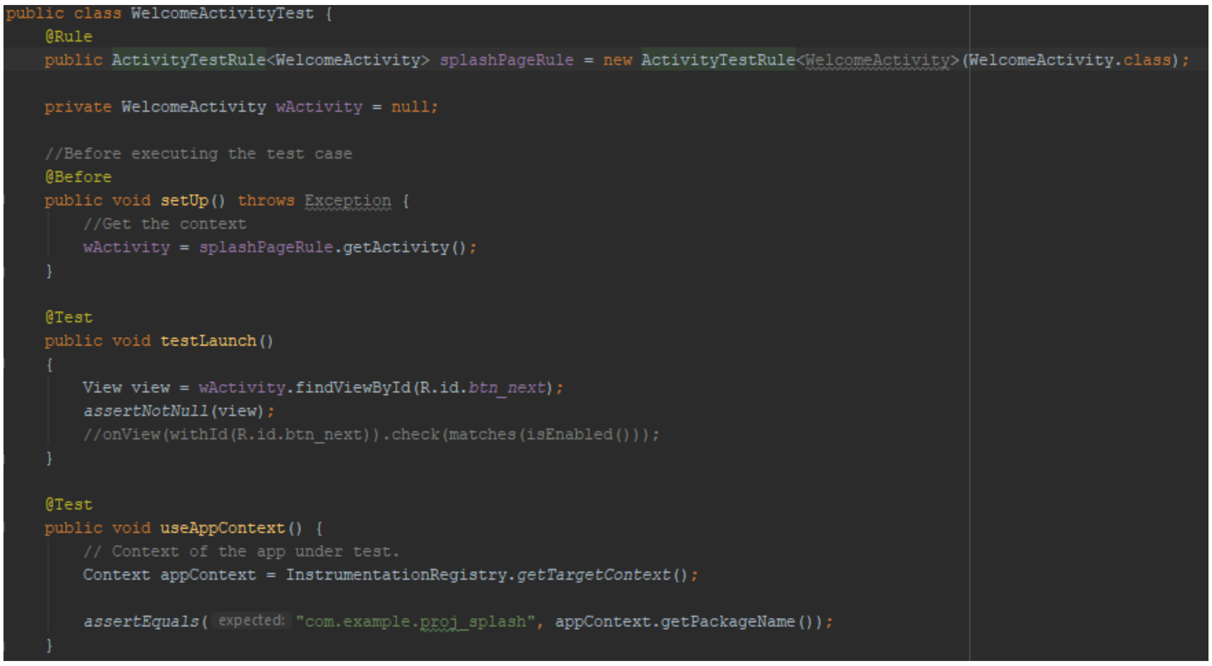
2.2. Instrumented Tests

"Under normal circumstances your application cannot control the life cycle events and the user drives the application" (Vogella, 2017). Using instrumentation, you can control these events through your test code. "An instrumentation-based test class allows you to send key events (or touch events) to the application under test" " (Vogella, 2017).

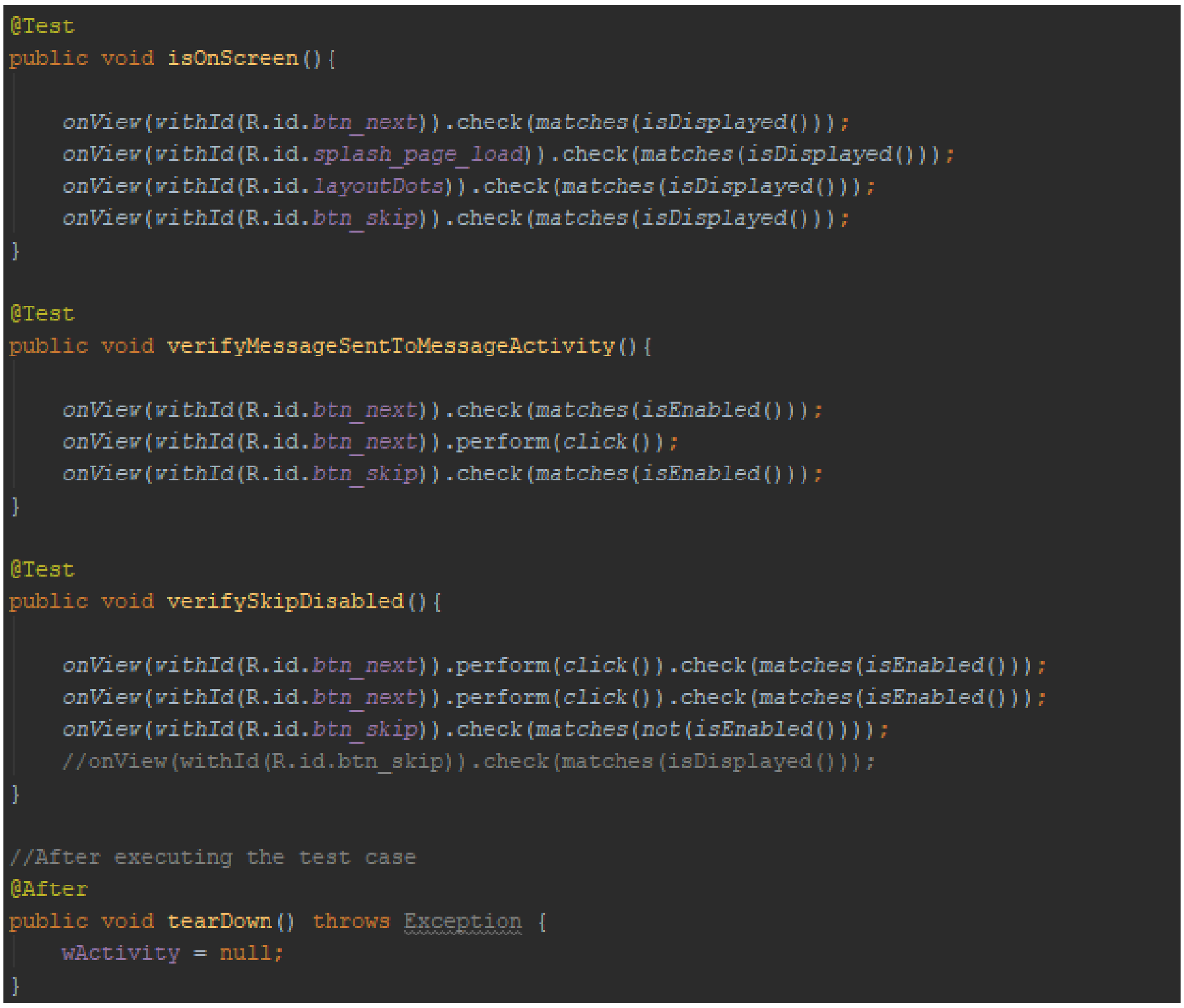
1. Splash Page

2. Login Page

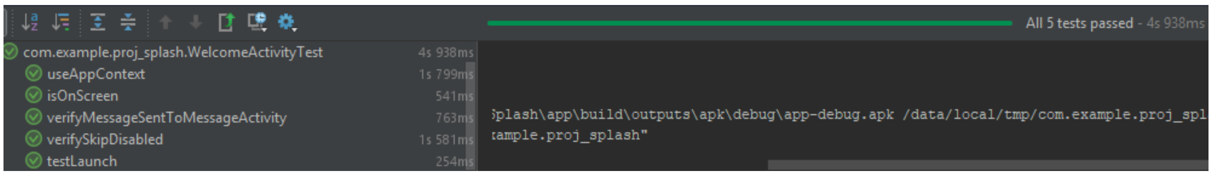
1. Splash Page



```
public class WelcomeActivityTest {
    @Rule
    public ActivityTestRule<WelcomeActivity> splashPageRule = new ActivityTestRule<WelcomeActivity>(WelcomeActivity.class);

    private WelcomeActivity wActivity = null;

    //Before executing the test case
    @Before
    public void setUp() throws Exception {
        //Get the context
        wActivity = splashPageRule.getActivity();
    }

    @Test
    public void testLaunch()
    {
        View view = wActivity.findViewById(R.id.btn_next);
        assertNotNull(view);
        //onView(withId(R.id.btn_next)).check(matches(isEnabled()));
    }

    @Test
    public void useAppContext() {
        // Context of the app under test.
        Context appContext = InstrumentationRegistry.getTargetContext();

        assertEquals( expected: "com.example.proj_splash", appContext.getPackageName());
    }
```

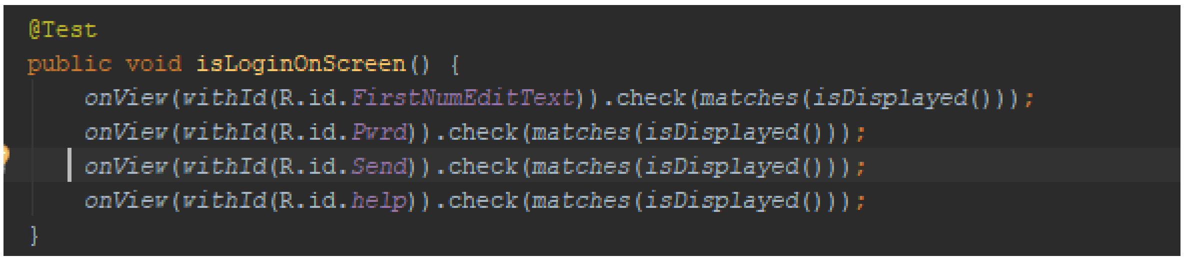


```
@Test
public void isOnScreen(){

    onView(withId(R.id.btn_next)).check(matches(isDisplayed()));
    onView(withId(R.id.splash_page_load)).check(matches(isDisplayed()));
    onView(withId(R.id.layoutDots)).check(matches(isDisplayed()));
    onView(withId(R.id.btn_skip)).check(matches(isDisplayed()));
}

@Test
public void verifyMessageSentToMessageActivity(){

    onView(withId(R.id.btn_next)).check(matches(isEnabled()));
    onView(withId(R.id.btn_next)).perform(click());
    onView(withId(R.id.btn_skip)).check(matches(isEnabled()));
}

@Test
public void verifySkipDisabled(){

    onView(withId(R.id.btn_next)).perform(click()).check(matches(isEnabled()));
    onView(withId(R.id.btn_next)).perform(click()).check(matches(isEnabled()));
    onView(withId(R.id.btn_skip)).check(matches(not(isEnabled())));
    //onView(withId(R.id.btn_skip)).check(matches(isDisplayed()));
}

//After executing the test case
@After
public void tearDown() throws Exception {
    wActivity = null;
}
```

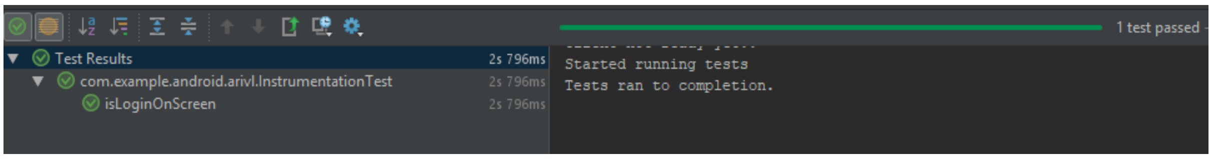


```
All 5 tests passed - 4s 938ms
com.example.proj_splash.WelcomeActivityTest      4s 938ms
  useAppContext                                  1s 799ms
  isOnScreen                                       541ms
  verifyMessageSentToMessageActivity               763ms
  verifySkipDisabled                             1s 581ms
  testLaunch                                       254ms

;plash\app\build\outputs\apk\debug\app-debug.apk /data/local/tmp/com.example.proj_spl
:ample.proj_splash"
```

2. Login Page



```
@Test
public void isLoginOnScreen() {
    onView(withId(R.id.FirstNumEditText)).check(matches(isDisplayed()));
    onView(withId(R.id.Pwrd)).check(matches(isDisplayed()));
    onView(withId(R.id.Send)).check(matches(isDisplayed()));
    onView(withId(R.id.help)).check(matches(isDisplayed()));
}
```



```
1 test passed
Test Results                                   2s 796ms
  com.example.android.arivl.InstrumentationTest  2s 796ms
    isLoginOnScreen                              2s 796ms

Started running tests
Tests ran to completion.
```

2.3. Repeat until all features are correctly implemented

Iterate through step 2 and step 3 until all the features are implemented and all the tests pass.

2.4. Accomplish test coverage

Android Studios allows us to export the results of our tests into an html file. Below is a screen shot of our test coverage:

3. Test cases

All the tests can be found on Github:
https://github.com/devawa/StacksOnStacks/tree/master/Android_Dev/Tests

4. References

Ajesh. 2018. Android Testing Part 1: Espresso Basics. [Online]. Available: https://medium.com/mindorks/android-testing-part-1-espresso-basics-7219b86c862b [1 October 2018].

Android Developers. [n.d]. Espresso. [Online]. Available: https://developer.android.com/training/testing/espresso/ [Accessed 18 July 2018].

Robolectric. [n.d]. Robolectric Test-Drive your Android Code. [Online]. Available: http://robolectric.org/ [Accessed 19 July 2018].

Tutorialspoint. [n.d]. Mockito Tutorial. [Online]. Available: https://www.tutorialspoint.com/mockito/ [Accessed 19 July 2018].

Vogella. 2017. Developing Android unit and instrumentation tests – Tutorial. [Online]. Available: http://www.vogella.com/tutorials/AndroidTesting/article.html#androidtesting_instrumentation [1 October 2018].