Testing Procedures

B.Sc in Computer Applications

Third Year Project

James Nolan & Kealan Thompson

Communal Costs developed for

Table of Contents

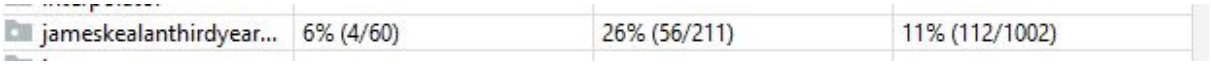
Local Unit Testing

What is it?

In android, local unit tests are tests which are carried out on the local JVM. You do not need to connect your phone or a virtual device to the computer to carry out these tests. As a result, these tests complete extremely fast. A mocking framework is used to mock any dependencies, such as Firebase Database references or Firebase Authentications.

A downside of these tests is that they cannot be used on any activities or services. To test these aspects of our app, we must carry out instrumentation unit testing.

The Local Unit Tests provide 26% coverage of the app.



Note: *See our project code to see the code for these tests (app<src<test<java<jameskealanthirdyearproject<communalsosts_client_app)*

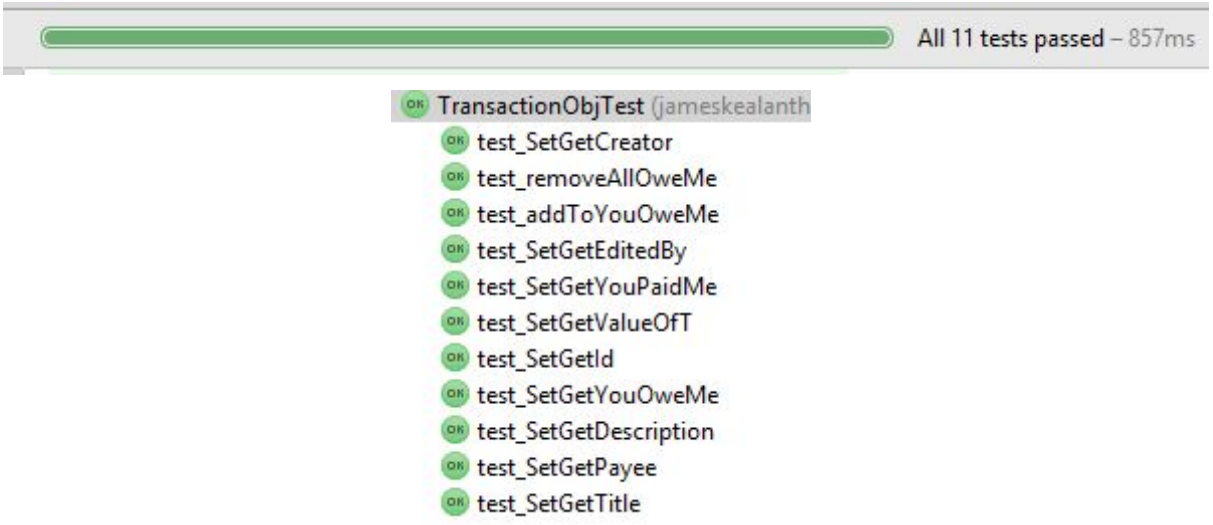
TransactionObj.class

setTitle()

```
public void setTitle(String t) {
    this.title = t;
}
```

We had a stupid error here. t = this.title instead of this.title = t; Local unit testing on this sprung this error.

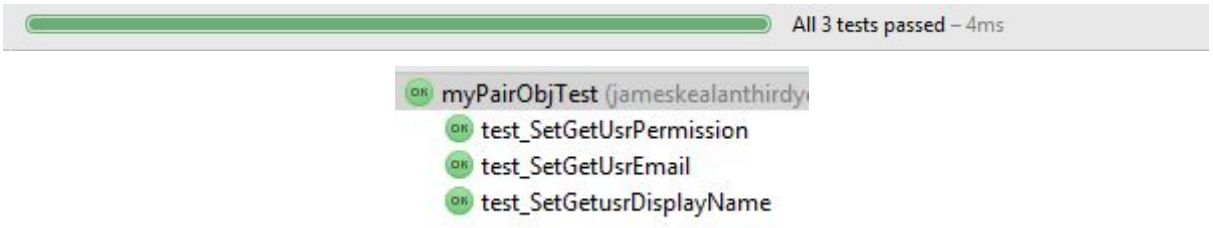
Result



We were satisfied with the code in TransactionObj.class after doing the local unit test.

myPairObj.class

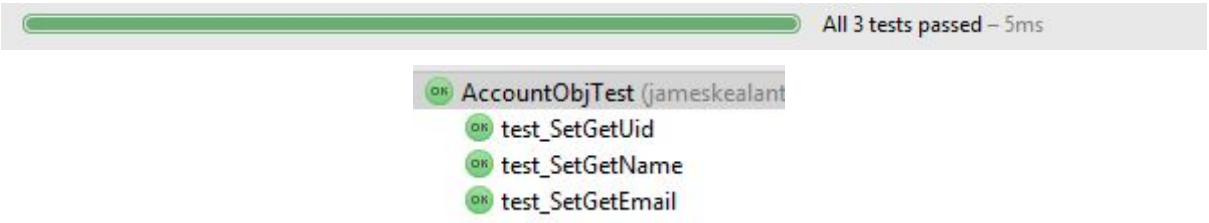
Result



We were satisfied with the code in myPairObj.class after doing the local unit test.

myAccountObj.class

Result



We were satisfied with the code in myAccountObj.class after doing the local unit test.

CollectiveObj.class

Local Unit Test Method

```
@Test
public void test_removeTransaction() throws Exception {
    ArrayList<TransactionObj> testArray = new ArrayList<>();
    assertEquals(testArray, testObj.getTransactions());

    TransactionObj t1 = new TransactionObj( descrpt: "a",  val: 1,  paying: "you");
    TransactionObj t2 = new TransactionObj( descrpt: "b",  val: 2,  paying: "me");
    testArray.add(t1);
    testArray.add(t2);
    testObj.addTransaction(t1);
    testObj.addTransaction(t2);
    for (int i = 0; i < testArray.size(); i++) {
        assertSame(testArray.get(i).getDescription(), testObj.getTransactions().get(i).getDescription());
        assertSame(testArray.get(i).getValueOfT(), testObj.getTransactions().get(i).getValueOfT());

    }
    testObj.removeTransaction(t1);
    testArray.remove(t1);
    assertSame(testArray.size(), testObj.getTransactions().size());
    for (int i = 0; i < testArray.size(); i++) {
        assertSame(testArray.get(i).getDescription(), testObj.getTransactions().get(i).getDescription());
        assertSame(testArray.get(i).getValueOfT(), testObj.getTransactions().get(i).getValueOfT());

    }

}
```

Actual Method Snippet

```
private void removeTransaction(TransactionObj transactionObj){
    this.transactions.remove(transactions);
}
```

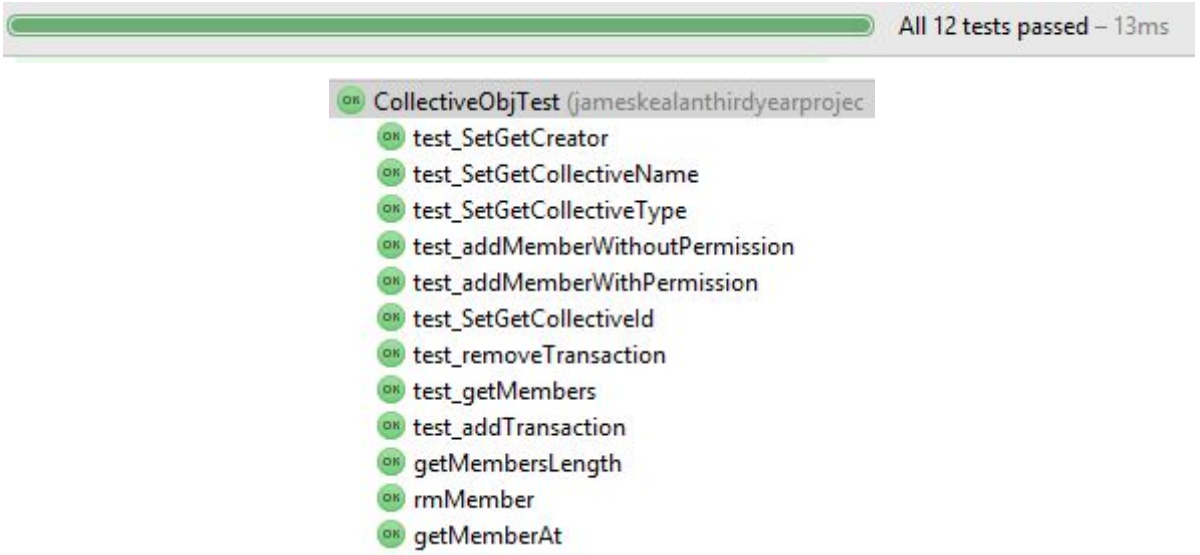
Local Unit Test Result

```
java.lang.AssertionError: expected same:<1> was not:<2>
Expected :1
Actual   :2
 <Click to see difference>
```

We realised this error on the third last assert. The size of both arrays after calling the ArrayList built in remove method and the remove method we have made for the CollectiveObj.class were not the same. This suggested the operation did not perform successfully.

Again another stupid error, the method body had an incorrect parameter, once this was changed to “transactionObj”, the test concluded successfully.

Result



We were satisfied with the code in CollectiveObj..class after doing the local unit test and applying the above fix.

Instrumented Unit Testing

What is it?

This type of testing is carried out on emulators or physical devices and gives you access to API’s and Android Testing framework. Access is granted to instrumentation information such as the Application Context which Local Unit Testing does not have access to. There is less effort required in writing mock code for API’s as you can just use the API’s in the instrumented unit testing.
Framework used: Espresso

CollectiveViewActivity.class

CreateNewCollectiveActivity.class

HomeCollectiveView.class

What's tested?

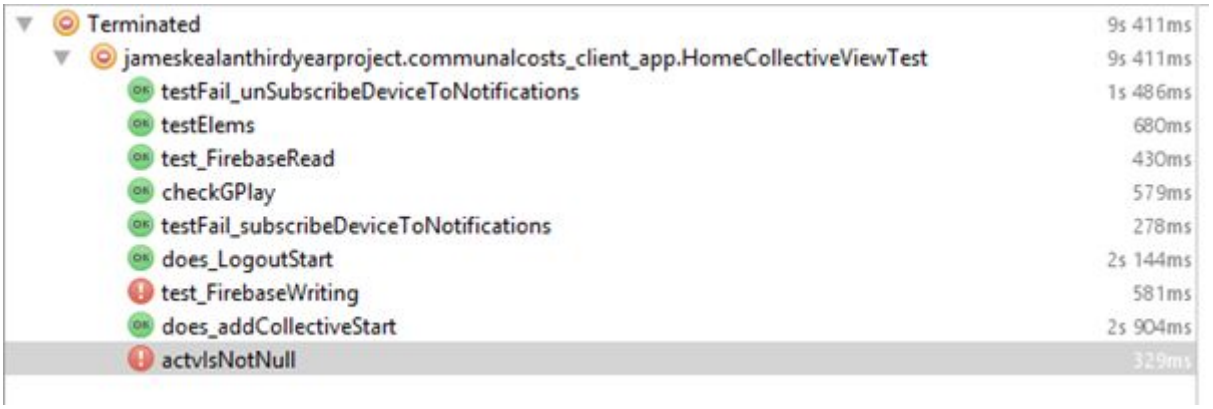
- Tested that views displayed properly
- Tested that all our objects wrote successfully to firebase
- Tested that all our objects could be read from Firebase
- Tested if Buttons start the activities they're supposed to (used Espresso intents)
- Tested Google Play Services method works for users without it
- Tested the subscription services for Firebase Notifications

Errors

- `testFail_subscribeDeviceToNotifications()`
- `testFail_unSubscribeDeviceToNotifications()`

These methods did not fail when they were given a Firebase topic that did not exist. The device would subscribe to notifications that would never be sent.

- `actvIsNotNull()`
- `test_FirebaseWriting()`



```
com.google.firebase.database.DatabaseException: Expected a List while deserializing, but got a class java.util.HashMap
at com.google.android.gms.internal.zzelw.zza(Unknown Source)
at com.google.android.gms.internal.zzelw.zzb(Unknown Source)
at com.google.android.gms.internal.zzelx.zze(Unknown Source)
at com.google.android.gms.internal.zzelw.zzb(Unknown Source)
at com.google.android.gms.internal.zzelw.zza(Unknown Source)
at com.google.firebase.database.DataSnapshot.getValue(Unknown Source)
at jameskealanthirdyearproject.communalcosts_client_app.HomeCollectiveView.getCollectivesList(HomeCollectiveView.java:253)
at jameskealanthirdyearproject.communalcosts_client_app.HomeCollectiveView$1.onDataChange(HomeCollectiveView.java:107)
at com.google.android.gms.internal.zzegf.zza(Unknown Source)
at com.google.android.gms.internal.zzeia.zzbyc(Unknown Source)
at com.google.android.gms.internal.zzeig.run(Unknown Source)
at android.os.Handler.handleCallback(Handler.java:746)
at android.os.Handler.dispatchMessage(Handler.java:95)
at android.os.Looper.loop(Looper.java:148)
at android.app.ActivityThread.main(ActivityThread.java:5443) <1 internal calls>
at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:728)
at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:618)
```

We encountered this error during development. We thought we had resolved it, but recent additions to the Transaction Object appeared to have brought the issue back.

Solutions

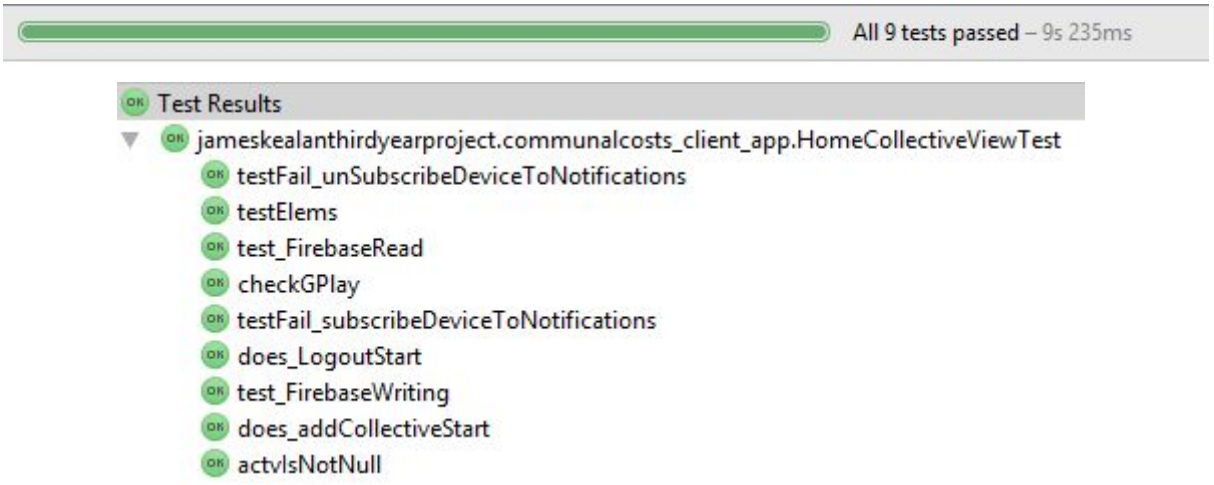
- *testFail_subscribeDeviceToNotifications()*
- *testFail_unSubscribeDeviceToNotifications()*

We introduced a custom exception “CollectiveNotFoundExeception” that throws when the user tries to subscribe to a topic that doesn’t exist.

- *actvIsNotNull()*
- *test_FirebaseWriting()*

We noticed in the transaction object that some parameters were not being pushed up correctly which meant when our app wrote the Transaction to Firebase, it crashed the app. We fixed this promptly and the test successfully passed.

Result



Firebase Cloud Testing

What is it?

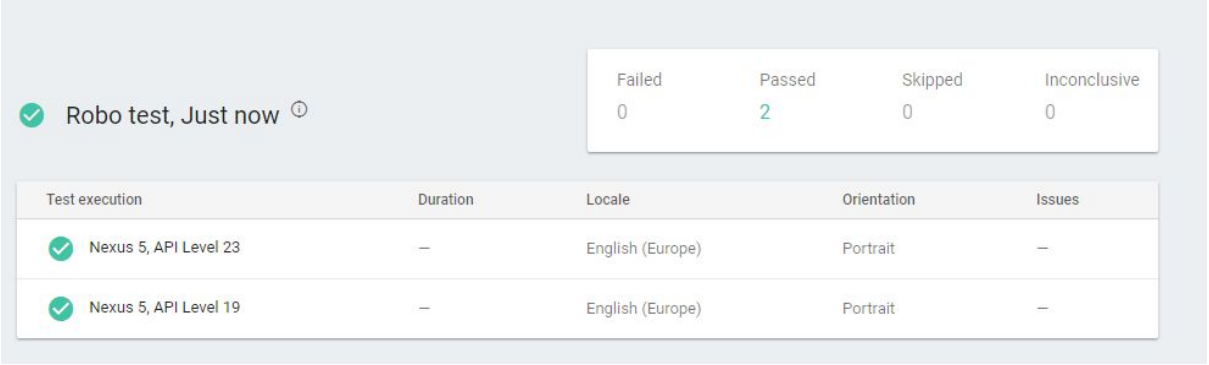
Integrated feature of the Firebase Test Lab. Analyzes UI Structure and explores it's methods through simulating user activity. These tests can be carried out on a range of physical/virtual devices across different API levels. Logs, screenshots, activity maps, videos and performance are available

A Robo script can be created with Android Studio to tell the system to do more specific testing, such as specific user credentials with login. The system carries out these predetermined actions then tests as usual.

Robo Test 1- 05/03/2018

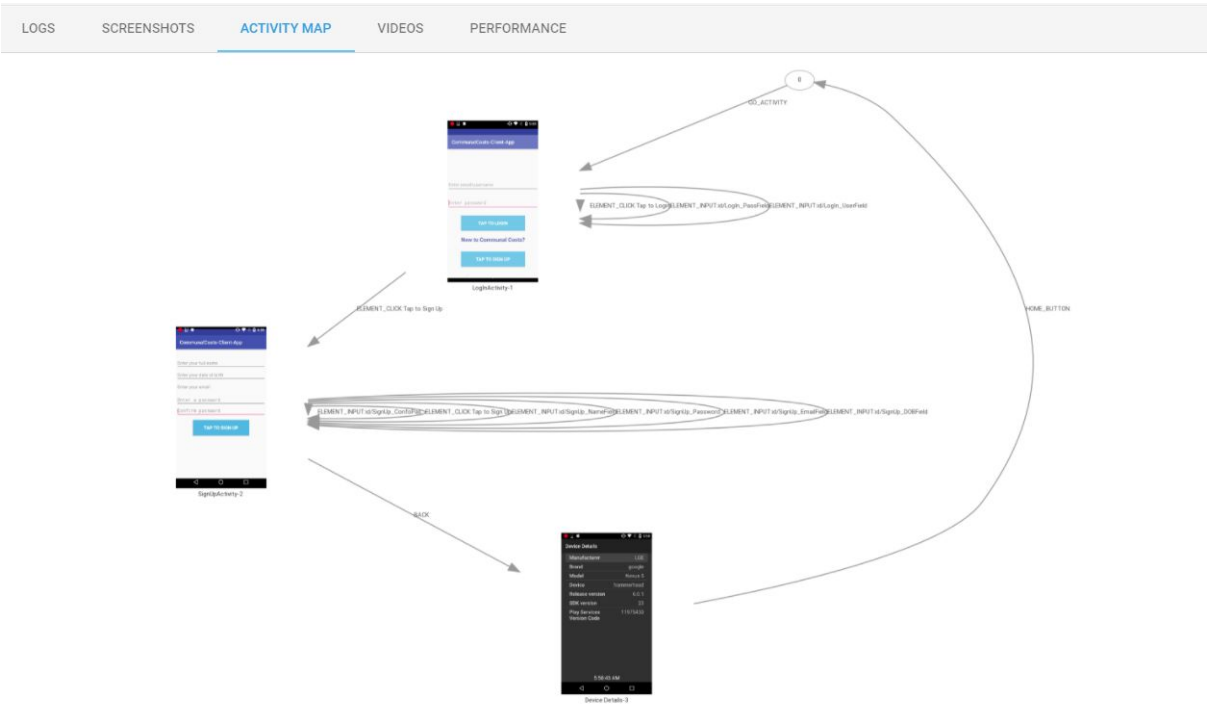
Some Screenshots

Matrix No.489735



CPU Usage throughout Testing API23

Activity Map Testing API23



Details

The first test didn't get past the login/signup screen, as credentials are required for this to work. A robo script needs to be provided to supply the system with valid credentials and inputs for both fields. However this test was useful to see how the system actually works.

Outcome

Run another test with a robo script

Robo Test 2 - 05/03/2018

Some Screenshots

Matrix No.511345