Testing of Contract Tracing Applications

Yousef El-Bayoumi
1722326
Dept. of Software Engineering
yousef.elbayomi@bahcesehir.edu.tr

Sercan Emre Seviniş
1728942
Dept. of Software Engineering
sercanemre.sevinis@bahcesehir.edu.tr

İpek Yasin
1503913
Dept. of Software Engineering
ipek.yasin@bahcesehir.edu.tr

Hikmet Çağatay Yılmaz
1728860
Dept. of Software Engineering
hikmetcagatay.yilmaz@bahcesehir.edu.tr

Ghaith Nakawah
1720507
Dept. of Software Engineering
ghaith.nakawah@bahcesehir.edu.tr

Summary:

In this project, we have tried to do Acceptance Testing, Code Coverage Testing, Performance Testing, Security Testing, Mutation Testing, User Interface (UI) Testing, and Junit Testing. While we were doing these testing, we searched for many tools and used some of them needed as Emma, JaCoCo, Unit Test History Coverage, JMeter, Android 3.0 Profiler, PITest, Kit, etc. We used Android Studio and IntelliJ IDEA frameworks for testing.

Keywords:

Contract tracing, JUnit, Performance test, Security test, UI test, Acceptance test, Mutation test, Code coverage testing, Kit, Emma, JaCoCo, Android studio, IntelliJ idea, Tools, Java, Unit test history coverage, JMeter, Android 3.0, PITest.

TABLE OF CONTENT

LIST OF FIGURES

1. Introduction

Covid tracking applications are important tools both to raise awareness of the society and to use our technology effectively against a real threat, Covid-19. Therefore, these applications should be tested in detail and their deficiencies should be corrected. For this reason, we tried to understand the tracking practices of three different countries by trying them with different kinds of tests as possible. In this project, we tried to select the contract applications related to their languages as mostly written in JAVA.

Tools:

- Unit Testing and Integration Testing: Mockito framework
- Security Testing: Quick Android Review Kit (Qark), Mobile Security Framework (MobSF)
- Performance Testing: Android Profiler plugin (provided by Android Studio)
- UI Testing: Espresso framework (tool provided by Android Studio)
- Code Coverage Testing: Emma v.11.0 (had some errors), JaCoCo and Unit Test History Coverage (didn't work), Code Coverage History (tool provided by Android Studio and IntelliJ)
- Mutation Testing: PITest v.1.4.7
- Acceptance Testing: No tools needed, IntelliJ framework was used

1. CROATIA: It is 100% Java language written.

APP NAME: STOP COVID-19

- Splash Screen

Figure 1. Splash Screen in Croatia Application.

- Home Screen

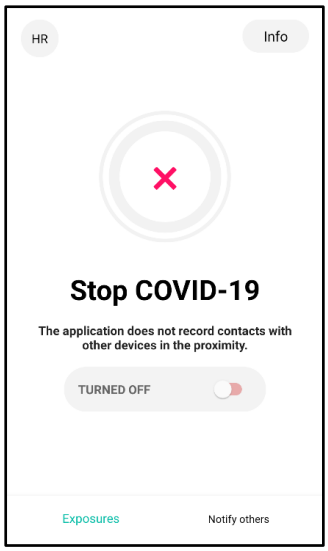


Figure 2. Home Screen in Croatia Application.

- Announcement Screen

Figure 3. Announcement Screen in Croatia Application.

- Information Screen

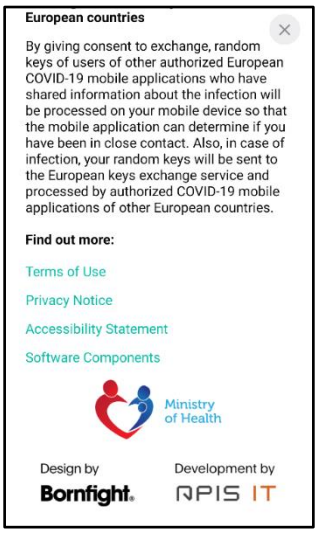


Figure 4. Information Screen in Croatia Application.

2. SPAIN: It is %100 Kotlin language written.

APP NAME: RADAR COVID

- Splash Screen

Figure 5. Splash Screen in Spain Application.

- Welcome Screen

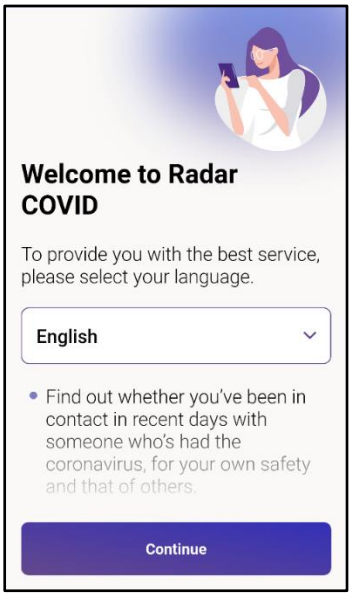


Figure 6. Welcome Screen in Spain Application.

- Settings Screen

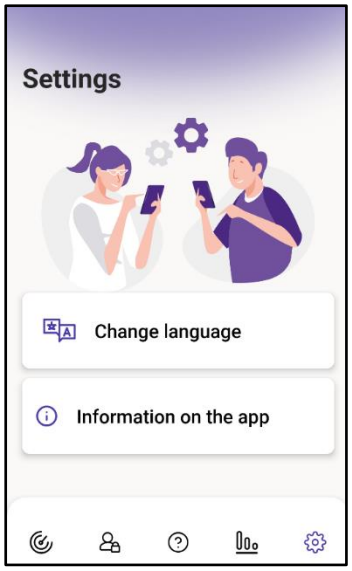


Figure 7. Settings Screen in Spain Application.

- Announcement Screen

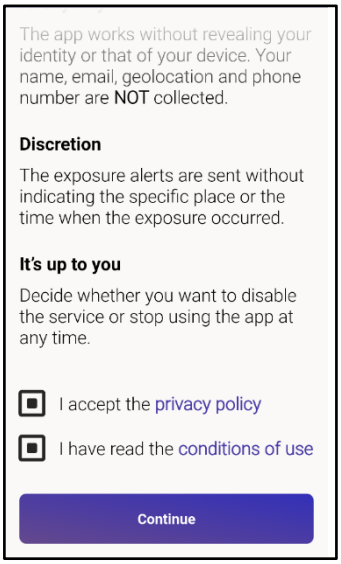


Figure 8. Announcement Screen in Spain Application.

- Information Screen

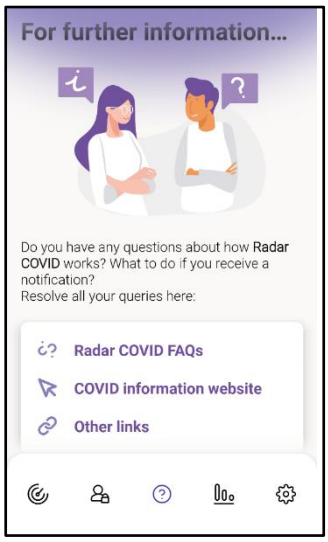


Figure 9. Information Screen in Spain Application.

3. SWITZERLAND: It is 67.8% Java, 31.7% HTML, 0.5% other language written.

In Switzerland and Spain Project, developers had already written JUnit test cases. They used the most common technique of JUnit which is assertions. As a general security protection, the app provides minimum information of user in the main servers. Authorities reach information only when user calls them. Users who have not reported positive detections cannot be tracked. Furthermore, app will destroy itself after the epidemic ends. Data in the server will be removed after 14 days as a protection for abuse of data. One of the most important protections of the app is avoiding location tracking all the time through broadcast identifiers via usage of "ephemeral identifier (EphID)". Devices generate EphIDs computed via a seed which is hash function. During the day, phones generate new EphIDs frequently to not to be tracked by the devices they broadcast

APP NAME: DP3T
We will use the name "SwissCovid" instead to make it clearer

- Splash Screen

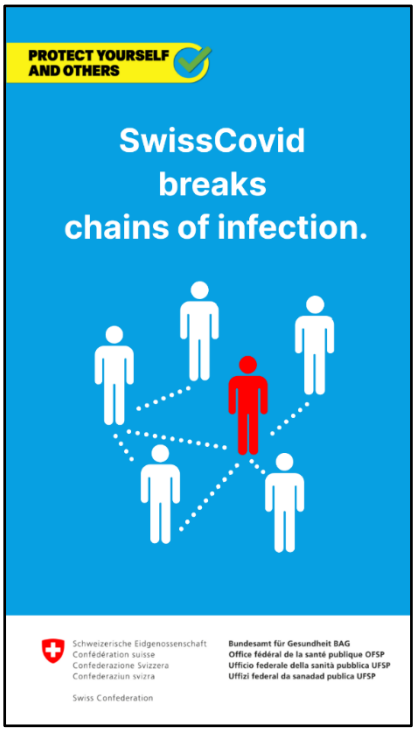


Figure 10. Splash Screen in Switzerland Application.

- Home Screen

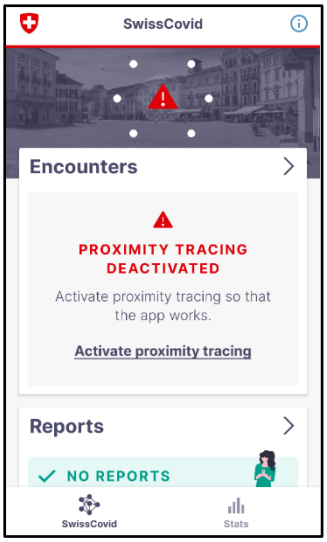


Figure 11. Home Screen in Switzerland Application.

- Feeling ill Screen

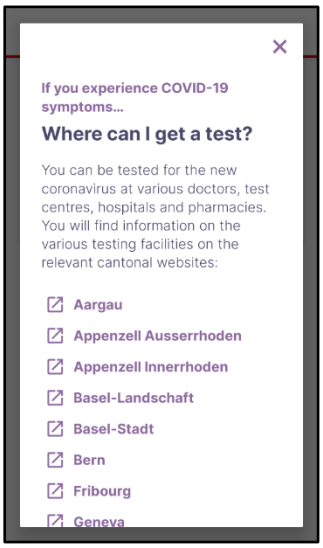


Figure 12. Feeling ill Screen in Switzerland Application.

- Entering Code Screen

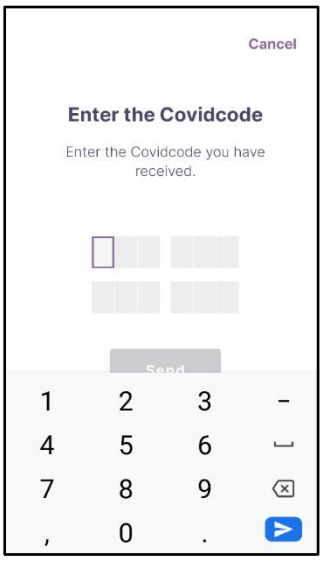


Figure 13. Entering Code Screen in Switzerland Application.

JUnit Test Case Classes:

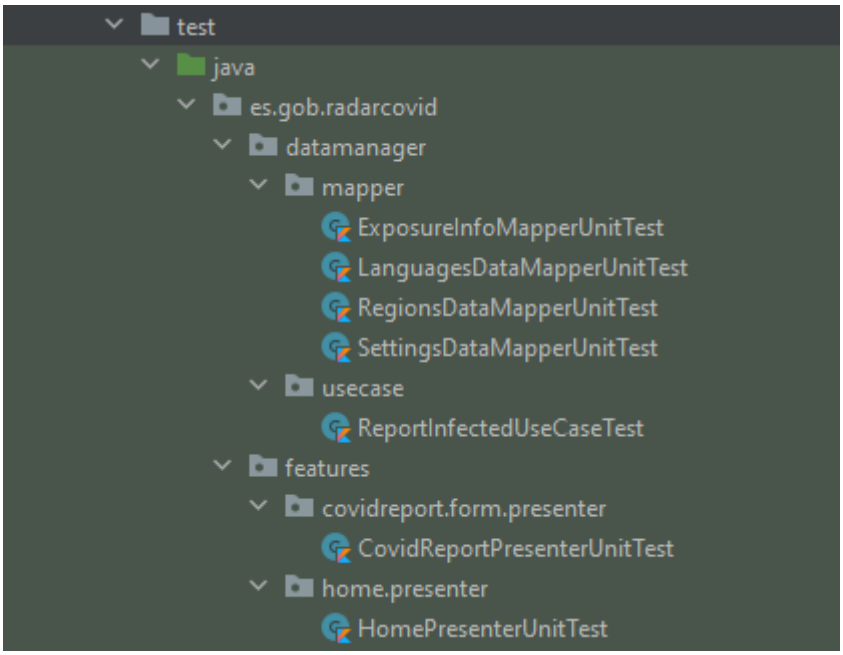


Figure 14. Junit test case classes in Spain application.

There are seven JUnit test classes named as

1. ExposureInfoMapperUnitTest
2. LanguagesDataMapperUnitTest
3. RegionsDataMapperUnitTest
4. SettingsDataMapperUnitTest
5. ReportInfecredUseCaseTest
6. CovidReportPresenterUnitTest
7. HomePresenterUnitTest.

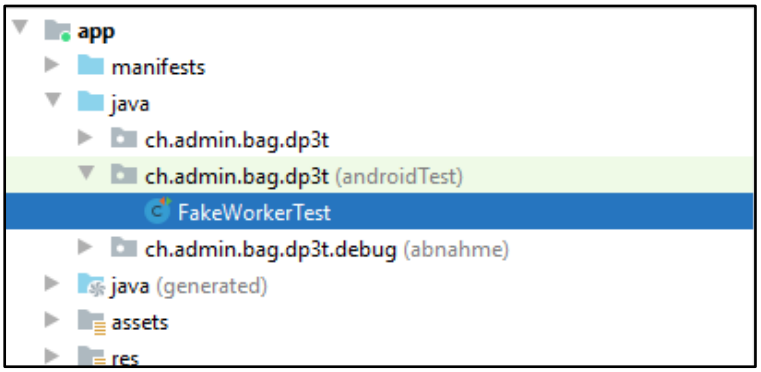


Figure 15. Junit test case classes in Switzerland application.

There is one JUnit test class named as FakeWorkerTest.

There is no test case for the Croatia application.

2 Contract Tracing Applications

First of all we got our application from the link provided by Prof. Dr. Çağatay Çatal: https://www.xda-developers.com/google-apple-covid-19-contact-tracing-exposure-notifications-api-app-list-countries/

Then we filtered the applications provided by the countries many times according to many different things. Firstly we got a list of all countries that provided the source code, then we reduced its elements by getting all apps written fully or mostly in Java. Our final filter was to get the working apps because we noticed that some apps have a source code and written in Java, but it wasn't working with us, we moved fast and changed the apps. All the source code was pulled from GitHub by HTTPS request.

The next step was to build and run the applications and for that we decided to use both of Android Studio and IntelliJ IDEA frameworks, we tried to use Eclipse, but we had issues and then we decided to find some alternatives.

Finally, one thing I need to point to, we noticed that some tools and plugins worked in Android Studio but never did in IntelliJ IDEA, and the opposite is correct as well.

3 Applied Testing Approaches

In Switzerland and Spain Project, developers had already written JUnit test cases. They used the most common technique of JUnit which is assertions. As a general security protection, the app provides minimum information of user in the main servers. Authorities reach information only when user calls them. Users who have not reported positive detections cannot be tracked. Furthermore, app will destroy itself after the epidemic ends. Data in the server will be removed after 14 days as a protection for abuse of data. One of the most important protections of the app is avoiding location tracking all the time through broadcast identifiers via usage of "ephemeral identifier(EphID)".

3.1 Unit Testing and Integration Testing

In two of the projects(Switzerland and Spain), there were some applied unit tests by the developer team. In Spain, all unit tests passed without error and in the beginning, Switzerland tests were running without error but after some time, some tests began to fail and currently 3 test methods run and 4 of them fail. In the projects, Mockito framework has been used in addition to JUnit.

Stop COVID-19 Croatia

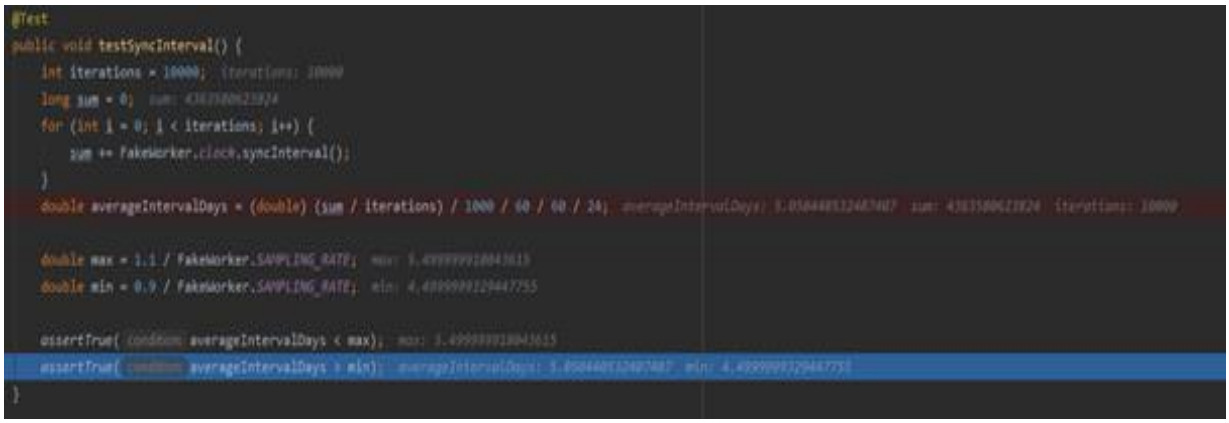


Figure 16. TestSyncInterval Test Class in Croatia.

In figure 16, if average interval days which is computed by some operations shown in figure is smaller than max day and bigger than min day, assertTrue method passes without error.

Radar COVID Spain

Figure 17. ExposureInfoDataMapper Test Class in Spain.

In Figure 17, “ExposureInfoDataMapper” class’s transform method returns default values for attributes and each of them are tested with assertNotNull and assertTrue methods. Test is passed so that means generated default values are correct(first 4 is not null and last one is True/enabled).

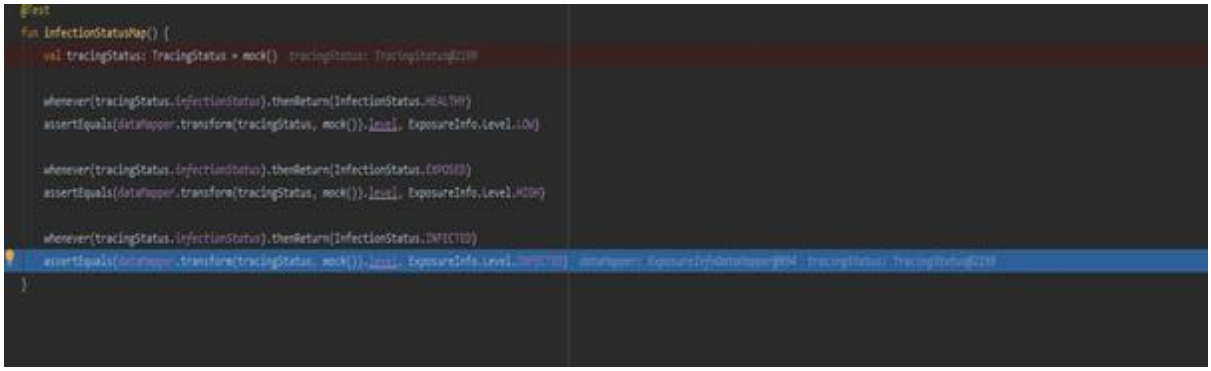


Figure 18. Assert Equals code generated.

In Figure 18, “assert equals” methods are generated to test if low, high, and infected level of exposures are working or not. If mock object status are equal to real object status, test passes. If this test had connections to related classes or databases and therefore considered modules together and tested them, it should have transformed to integration test since tracingStatus class has dependencies on APIs and server. However with this code, the thing tried to be shown is if the method is working by itself or not.

Figure 19. JUnit Explanations.

In Junit tests, since test methods should reach to different classes, different methods, databases, APIs directly; to eliminate this dependencies, there is fake object (mocking) concept. Again, this test method is executed between “@before” and “@after” notated methods. Here, from assertion methods, assertEquals is used to check given parameters are equal to each other or not. Mock class returns a fake object from “TracingStatus” class and “ExposureInfoData-Mapper”. Because exposureNotificationEnabled attribute is True, test is succeeded.

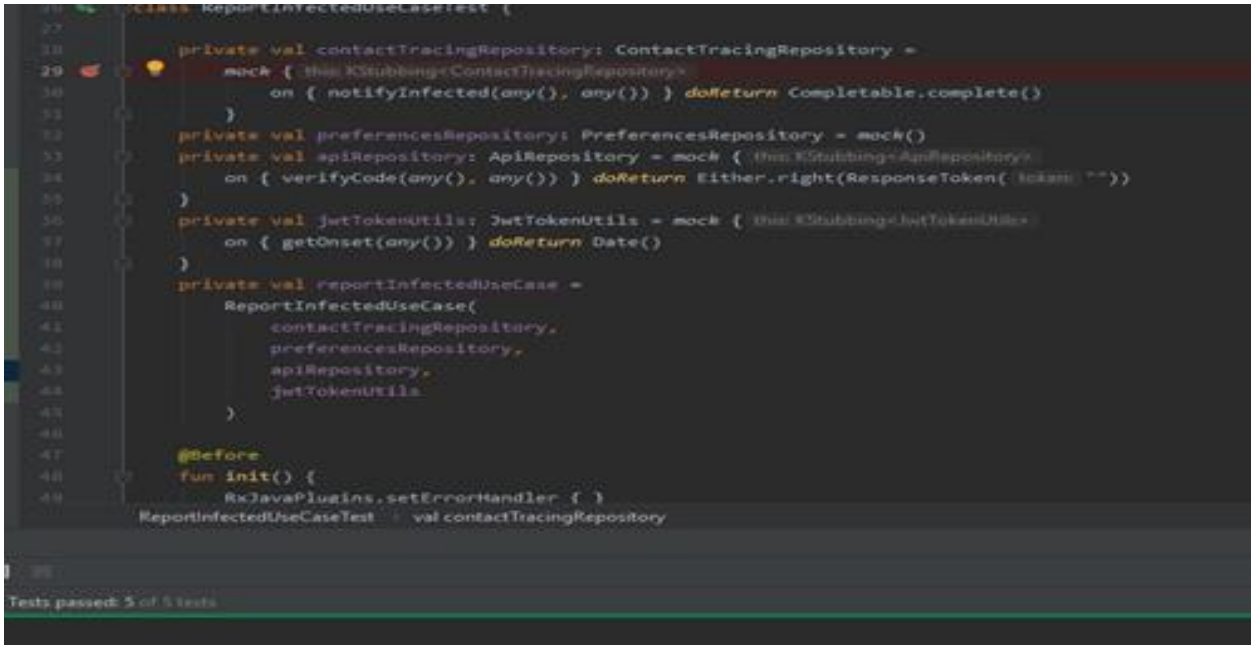
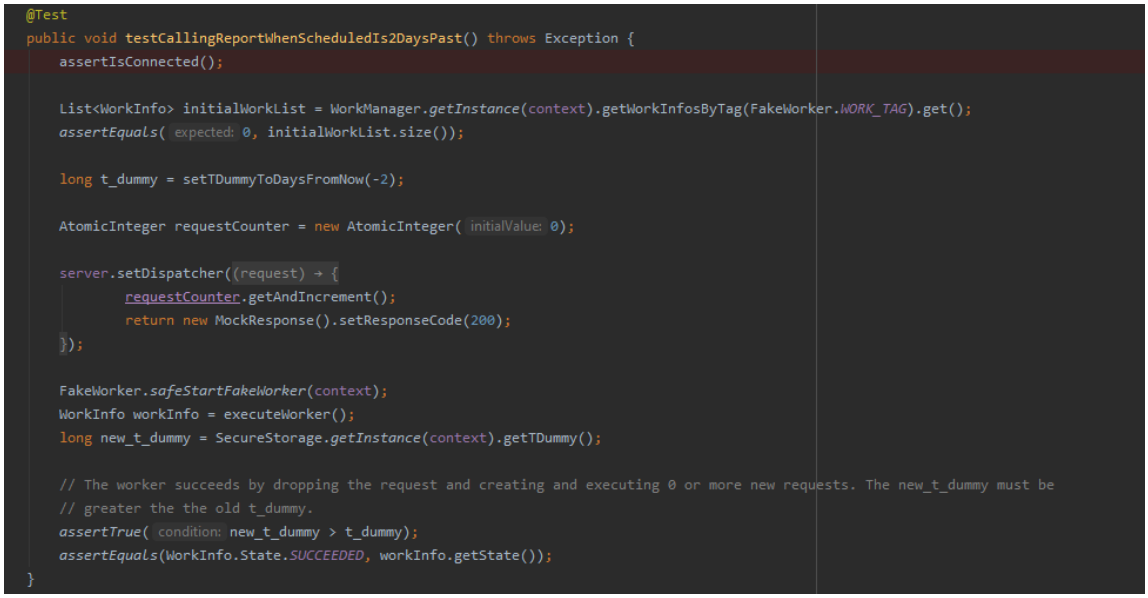


Figure 20. JUnit Explanations 2.

Figure 21. JUnit Explanations 3.

Figure 22. JUnit Explanations 4.

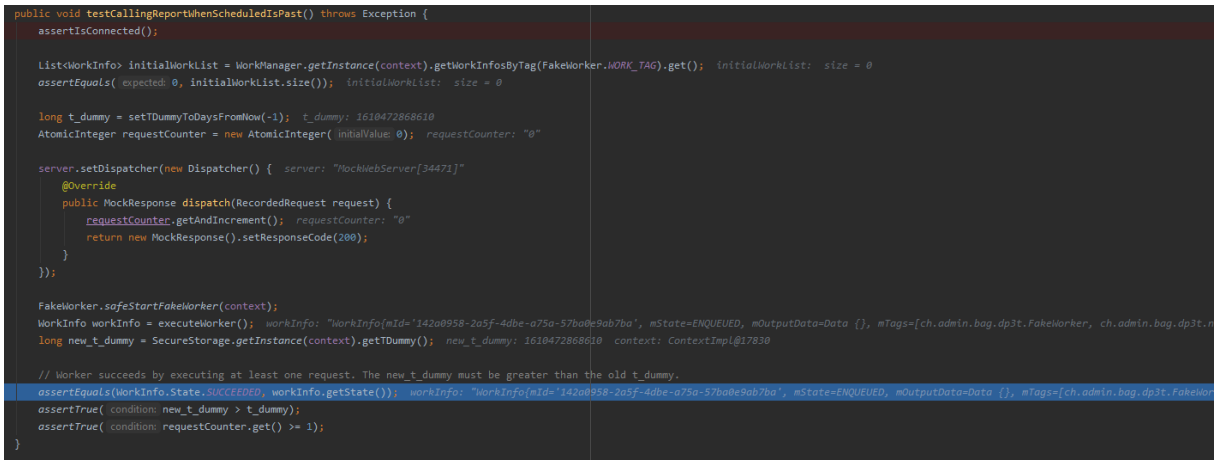
Figure 23. JUnit Explanations 5.



```
@Test
public void testCallingReportWhenScheduledIs2DaysPast() throws Exception {
    assertIsConnected();

    List<WorkInfo> initialWorkList = WorkManager.getInstance(context).getWorkInfosByTag(FakeWorker.WORK_TAG).get();
    assertEquals( expected: 0, initialWorkList.size());

    long t_dummy = setTDummyToDaysFromNow(-2);

    AtomicInteger requestCounter = new AtomicInteger( initialValue: 0);

    server.setDispatcher((request) -> {
            requestCounter.getAndIncrement();
            return new MockResponse().setResponseCode(200);
    });

    FakeWorker.safeStartFakeWorker(context);
    WorkInfo workInfo = executeWorker();
    long new_t_dummy = SecureStorage.getInstance(context).getTDummy();

    // The worker succeeds by dropping the request and creating and executing 0 or more new requests. The new_t_dummy must be
    // greater the the old t_dummy.
    assertTrue( condition: new_t_dummy > t_dummy);
    assertEquals(WorkInfo.State.SUCCEEDED, workInfo.getState());
}
```

Figure 24. Explaining Why Tests Failed.

In figure 24, test fails due to last assert equals method. Here, initial work list' size comes 0 from work manager class and first assert equals method returns true. In assert true method, new_t_dummy variable which is an instance of secure storage class is bigger than t_dummy variable which is computed in "setTDummyToDaysFromNow()" method. But the last assert equals method returns false because succeed state for work information it was wanted but it returned enqueued state.



```
public void testCallingReportWhenScheduledIsPast() throws Exception {
    assertIsConnected();

    List<WorkInfo> initialWorkList = WorkManager.getInstance(context).getWorkInfosByTag(FakeWorker.WORK_TAG).get();  initialWorkList:  size = 0
    assertEquals( expected: 0, initialWorkList.size());  initialWorkList:  size = 0

    long t_dummy = setTDummyToDaysFromNow(-1);  t_dummy: 1610472868610
    AtomicInteger requestCounter = new AtomicInteger( initialValue: 0);  requestCounter: "0"

    server.setDispatcher(new Dispatcher() {  server: "MockWebServer[34471]"
        @Override
        public MockResponse dispatch(RecordedRequest request) {
            requestCounter.getAndIncrement();  requestCounter: "0"
            return new MockResponse().setResponseCode(200);
        }
    });

    FakeWorker.safeStartFakeWorker(context);
    WorkInfo workInfo = executeWorker();  workInfo: "WorkInfo{mId='142a0958-2a5f-4dbe-a75a-57ba0e9ab7ba', mState=ENQUEUED, mOutputData=Data {}, mTags=[ch.admin.bag.dp3t.FakeWorker, ch.admin.bag.dp3t.n
    long new_t_dummy = SecureStorage.getInstance(context).getTDummy();  new_t_dummy: 1610472868610  context: ContextImpl@17830

    // Worker succeeds by executing at least one request. The new_t_dummy must be greater than the old t_dummy.
    assertEquals(WorkInfo.State.SUCCEEDED, workInfo.getState());  workInfo: "WorkInfo{mId='142a0958-2a5f-4dbe-a75a-57ba0e9ab7ba', mState=ENQUEUED, mOutputData=Data {}, mTags=[ch.admin.bag.dp3t.FakeWor
    assertTrue( condition: new_t_dummy > t_dummy);
    assertTrue( condition: requestCounter.get() >= 1);
}
```

Figure 25. Explaining Why Tests Failed 2.

In figure 25, again same assert method fails but when we investigate other methods were going to pass, two assert equals methods which are after failed method work fine. New_t_dummy variable is bigger than t_dummy variable after worker succeeds with executing at least one request.

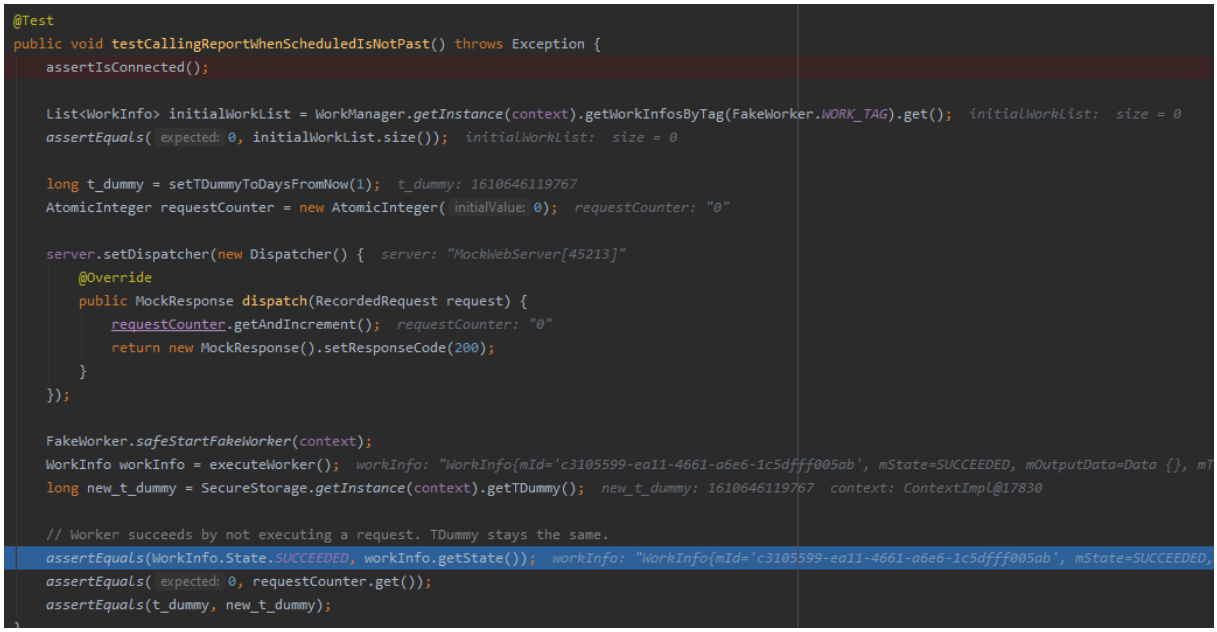


Figure 26. Special case Test Class.

In figure 26, no problem is shown but this is one of the methods that the same problem also sometimes occurs. This method is to test scheduled is not past. It checks if worker succeeds with not executing a request, but test fails here sometimes. Succeed state must be returned but instead, enqueued is returned in some cases. Other 2 methods after failed one would pass if there was no error in this one.

3.2 Security Testing

We've tried to instrument two different tools to make a security test for the applications. We had difficulties during the loading phase of one of the applications and the generating the report in the other. For this reason, I will explain which issue we had trouble with, what we tried to solve and why the problem caused.

Quick Android Review Kit (Qark): The problem is caused by the setup.py file that the tool uses to analyze an application. Pip was used as the python file manager, but in our experiment, python could not be successfully built up.

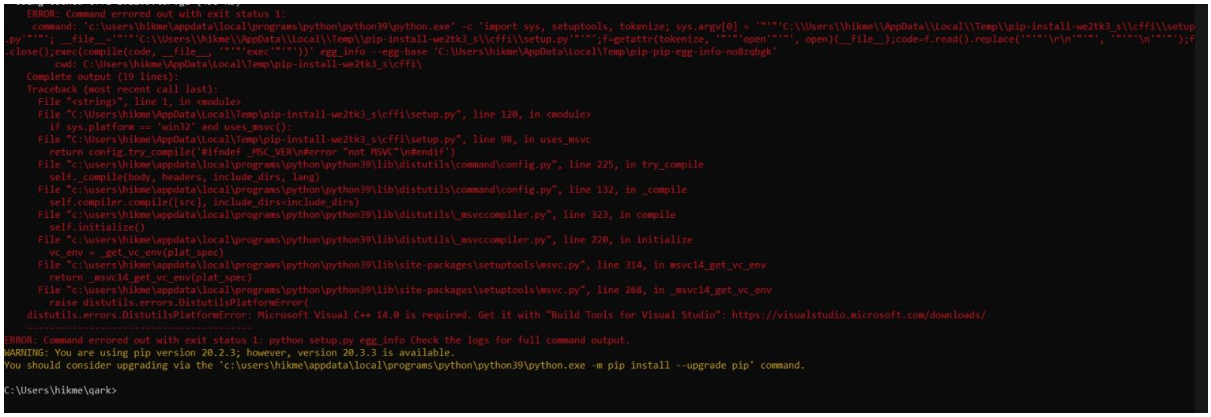


Figure 27. Pip has also been tested with version 20.3.3 but failed.

Mobile Security Framework (MobSF): The application could not be installed successfully. We think this is because the MobSF application has many requirements. We have been getting an error similar to the Quick Android Review Kit. This might be related to the setup.py file, but despite our updates we could not solve the error.

Unfortunately, we couldn't test the security. But we tried to understand what kind of security vulnerabilities contact tracing applications have by examining the security reports of the applications. We reviewed the public security test reports of the Swiss Covid-19 contact tracking application. They have classified major security concerns for contact tracing applications.

- Availability of Source Code:
- Open Source: There may not be enough time for security testing when applications are shared open source after developing quickly. These applications use APIs provided by Google and Apple. Security vulnerabilities may occur through these APIs. At the same time, these APIs may collect information on behalf of users and pose a risk to their security.
- Location of the Servers:
- Replay Attacks: In the data provided through Bluetooth can be modified maliciously. In this case, users may receive a false at-risk alert as a result of this injection attack.
- Tracking: The application can still track when tracing is on but Bluetooth is turned off.
- Interoperability: Due to the Apple / Google Exposure Notification API usage and permissions, it may be inconvenient to use the SwissCovid application in other

countries. Because of this, the other countries can make security agreements and arrange Exposure APIs accordingly.

- Data Protection: For tracing applications that processes Bluetooth signals, extra precautions should be taken for data security.
- Threats Related to Malicious Applications: It is a common problem for not only Covid-19 tracing applications but also all mobile devices.

For more details: Source

3.3 Performance Testing

We used the Android Profiler plugin offered by Android Studio for performance testing. We tried to find out how the applications' phone is related to the CPU, memory, network, and energy. Of these 4 tests, we were not successful in the network test. Because ExposureAPI used for contact tracking is closed to virtual devices.

Android Profiler has a variety of test scenarios.
Since the applications are not very complex, we used the simplest trace type throughout the test. For the memory testing, firstly we record samples and dump the result of the java heap.

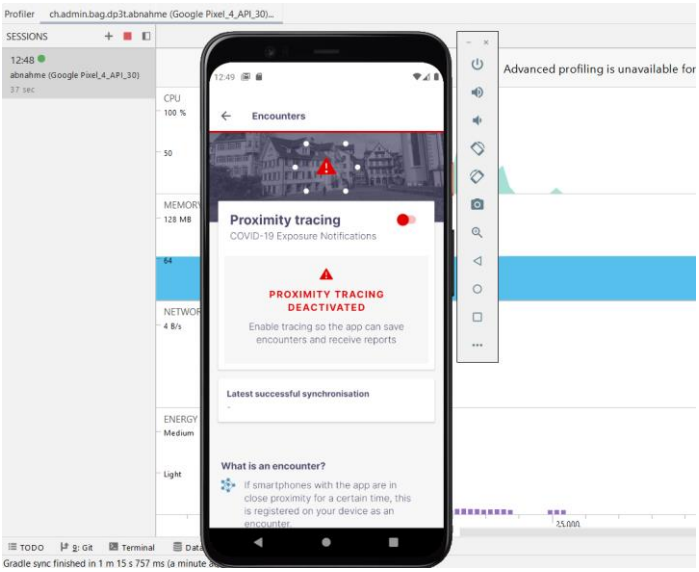


Figure 28. Android Profiler with SwissCovid Application

Test Details:
Trace Type: Sampled(Sampling Rate: 1000µs), Record Duration:30 Second with I/O Android Profiler 3.0 with Google Pixel 4 AVD
API Levels: 30, Android 11.0(R)-29, Android 10.0 (Q)

According to report, we conclude that:

1. There is no memory leak in any application.
2. In Croatia and Spanish applications, the main function is almost active throughout the application with at most 2 second delay, but there is a 10 second delay in the Swiss application.
3. CPU usage in the Croatian application is more stable than the other two applications.
4. Applications vary according to the structure of the memory. While the codes take up the most space in the Croatian application, Spain is the largest native size application.

Stop COVID-19 Croatia

CPU Profiler:

Time Range	00:00.000 - 00:30.306
Duration	30.31 s
Data Type	Thread
ID	27848

Longest running events (top 10)

Start Time	Name	Wall Duration	Self Time	CPU Duration	CPU Self Time
00:00.493	main	29.8 s	1 µs	2.94 s	1 µs
00:00.493	run	29.8 s	1 µs	2.94 s	1 µs
00:00.493	invoke	29.8 s	1 µs	2.94 s	1 µs
00:00.493	main	29.8 s	1 µs	2.94 s	1 µs
00:00.493	loop	29.8 s	1 µs	2.94 s	1 µs
00:15.367	dispatchMessage	1.53 s	0 µs	126.41 ms	0 µs
00:15.367	handleCallback	1.53 s	0 µs	126.41 ms	0 µs
00:15.367	run	1.53 s	2.82 ms	126.41 ms	1.17 ms
00:15.367	doFrame	1.51 s	0 µs	124.58 ms	0 µs
00:06.359	dispatchMessage	1.27 s	0 µs	95.82 ms	0 µs

Figure 29. CPU Profiler in Croatia App.

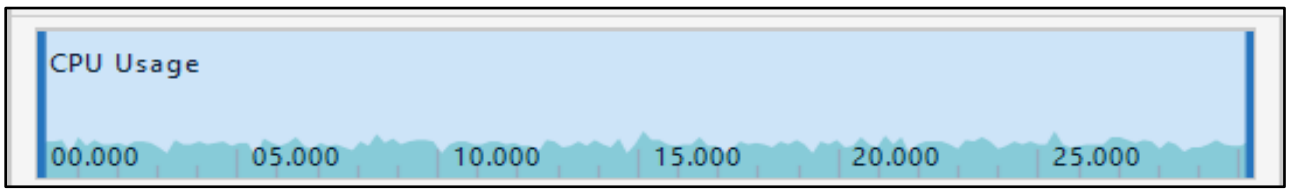


Figure 30. CPU Profiler in Croatia App 2.

Memory Profiler:

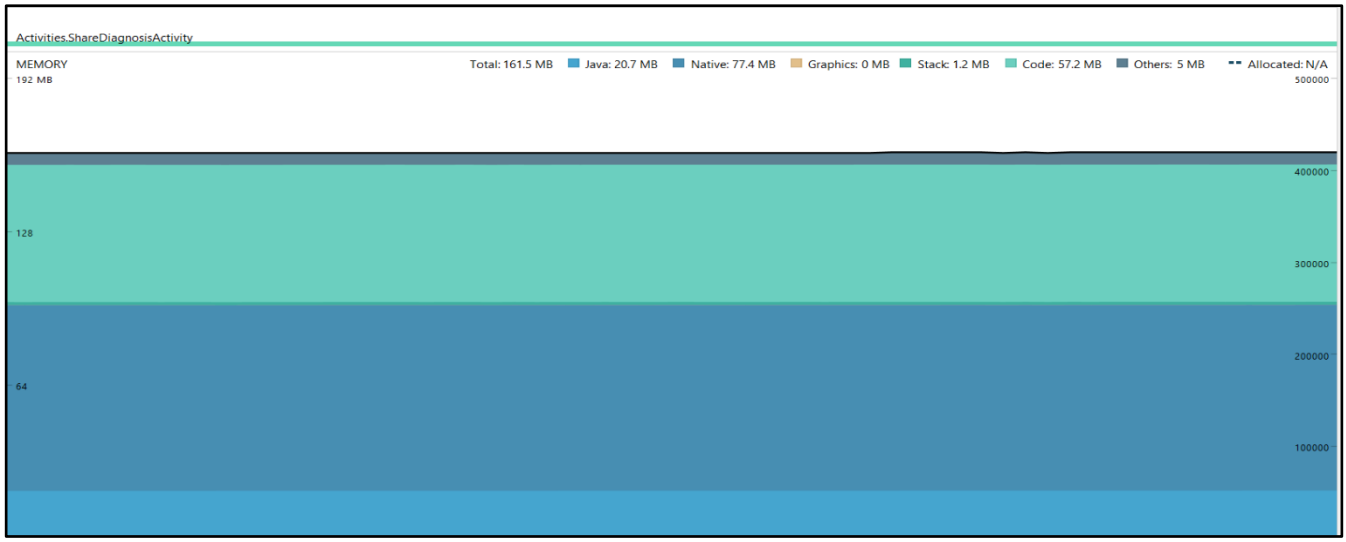
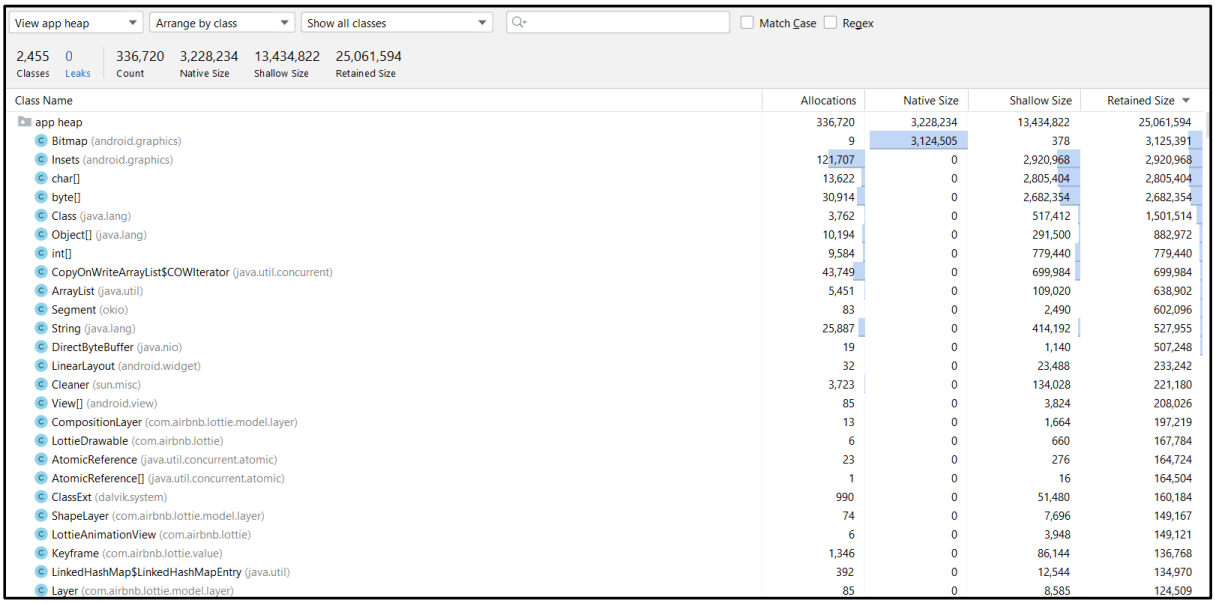


Figure 31. Dumping results of Java Heap (to examine instance properties of the objects, checking whether any memory leak or not.)



Class Name	Allocations	Native Size	Shallow Size	Retained Size
app heap	336,720	3,228,234	13,434,822	25,061,594
Bitmap (android.graphics)	9	3,124,505	378	3,125,391
Insets (android.graphics)	121,707	0	2,920,968	2,920,968
char[]	13,622	0	2,805,404	2,805,404
byte[]	30,914	0	2,682,354	2,682,354
Class (java.lang)	3,762	0	517,412	1,501,514
Object[] (java.lang)	10,194	0	291,500	882,972
int[]	9,584	0	779,440	779,440
CopyOnWriteArrayList$COWIterator (java.util.concurrent)	43,749	0	699,984	699,984
ArrayList (java.util)	5,451	0	109,020	638,902
Segment (okio)	83	0	2,490	602,096
String (java.lang)	25,887	0	414,192	527,955
DirectByteBuffer (java.nio)	19	0	1,140	507,248
LinearLayout (android.widget)	32	0	23,488	233,242
Cleaner (sun.misc)	3,723	0	134,028	221,180
View[] (android.view)	85	0	3,824	208,026
CompositionLayer (com.airbnb.lottie.model.layer)	13	0	1,664	197,219
LottieDrawable (com.airbnb.lottie)	6	0	660	167,784
AtomicReference (java.util.concurrent.atomic)	23	0	276	164,724
AtomicReference[] (java.util.concurrent.atomic)	1	0	16	164,504
ClassExt (dalvik.system)	990	0	51,480	160,184
ShapeLayer (com.airbnb.lottie.model.layer)	74	0	7,696	149,167
LottieAnimationView (com.airbnb.lottie)	6	0	3,948	149,121
Keyframe (com.airbnb.lottie.value)	1,346	0	86,144	136,768
LinkedHashMap$LinkedHashMapEntry (java.util)	392	0	12,544	134,970
Layer (com.airbnb.lottie.model.layer)	85	0	8,585	124,509

Figure 32. Memory Profiler Results in Croatia App.

Radar COVID Spain

CPU Profiler:

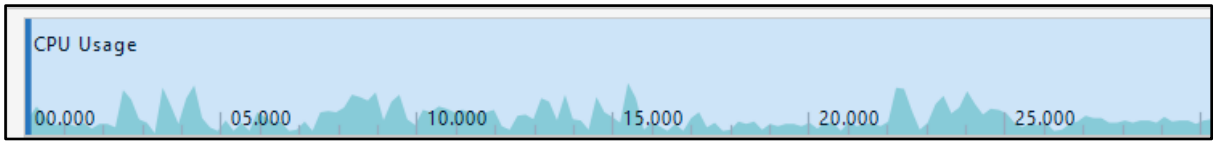


Figure 33. CPU Profiler in Spain App.

Time Range	00:00.000 - 00:31.559
Duration	31.56 s
Data Type	Thread
ID	2666

▼ Longest running events (top 10)

Start Time	Name	Wall Duration	Self Time	CPU Duration	CPU Self Time
00:00.359	main	30.78 s	1 µs	3 s	1 µs
00:00.359	run	30.78 s	1 µs	3 s	1 µs
00:00.359	invoke	30.78 s	1 µs	3 s	1 µs
00:00.359	main	30.78 s	1 µs	3 s	1 µs
00:00.359	loop	30.78 s	1 µs	3 s	1 µs
00:19.753	next	1.03 s	0 µs	42.06 ms	0 µs
00:19.753	nativePollOnce	947.28 ms	947.28 ms	38.62 ms	38.62 ms
00:06.153	next	913.81 ms	0 µs	33.91 ms	0 µs
00:06.153	nativePollOnce	913.81 ms	913.81 ms	33.91 ms	33.91 ms
00:08.086	dispatchMessa...	752.78 ms	0 µs	429.51 ms	0 µs

Figure 34. CPU Profiler in Spain App 2.

Memory Profiler:

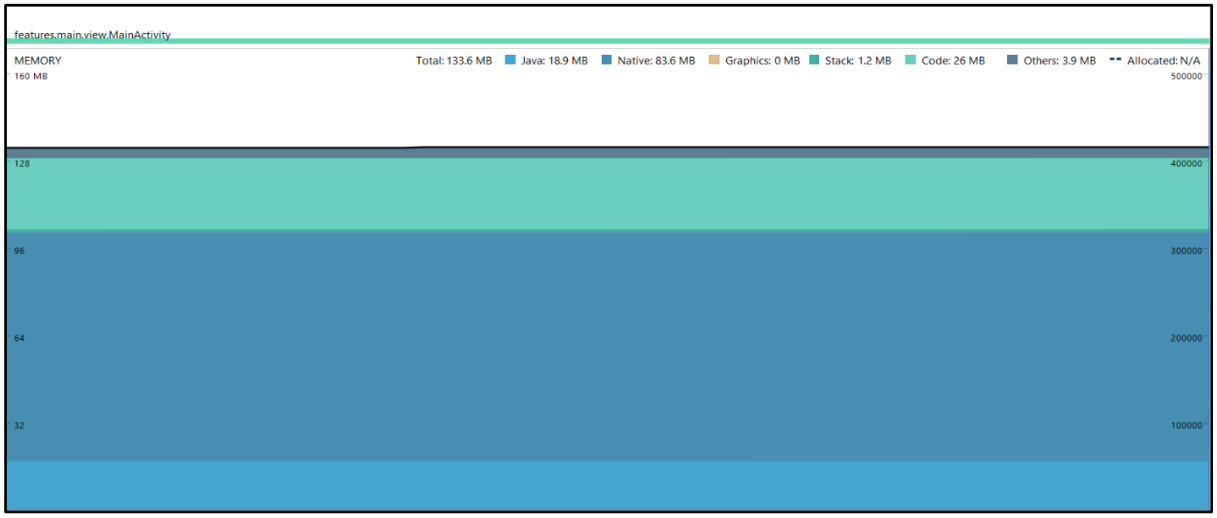
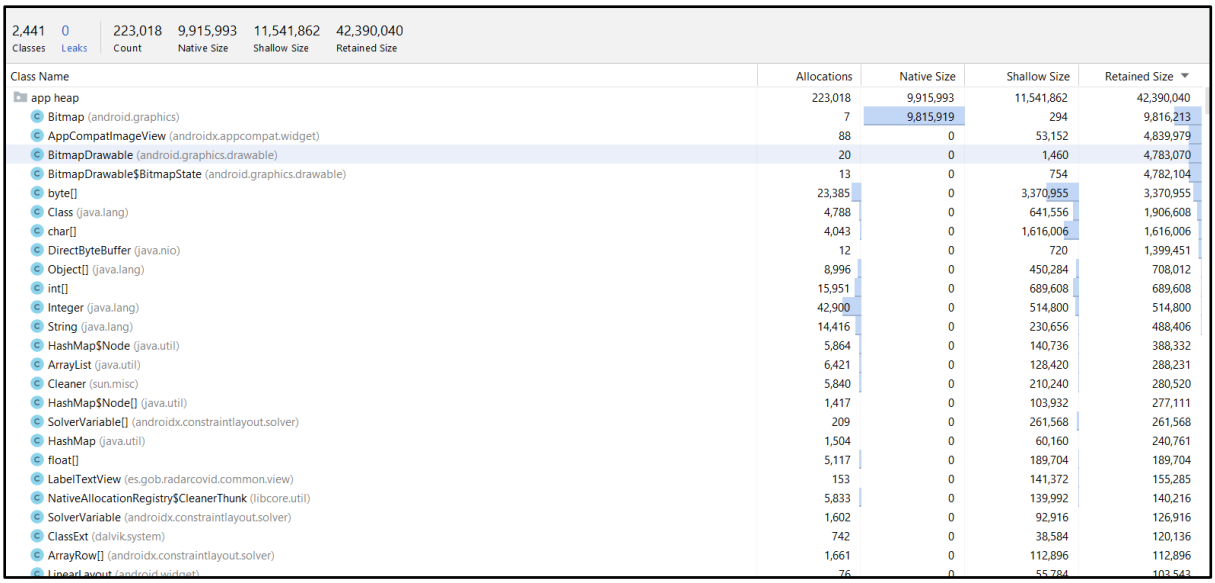


Figure 35. Memory Profiler Results in Spain App.



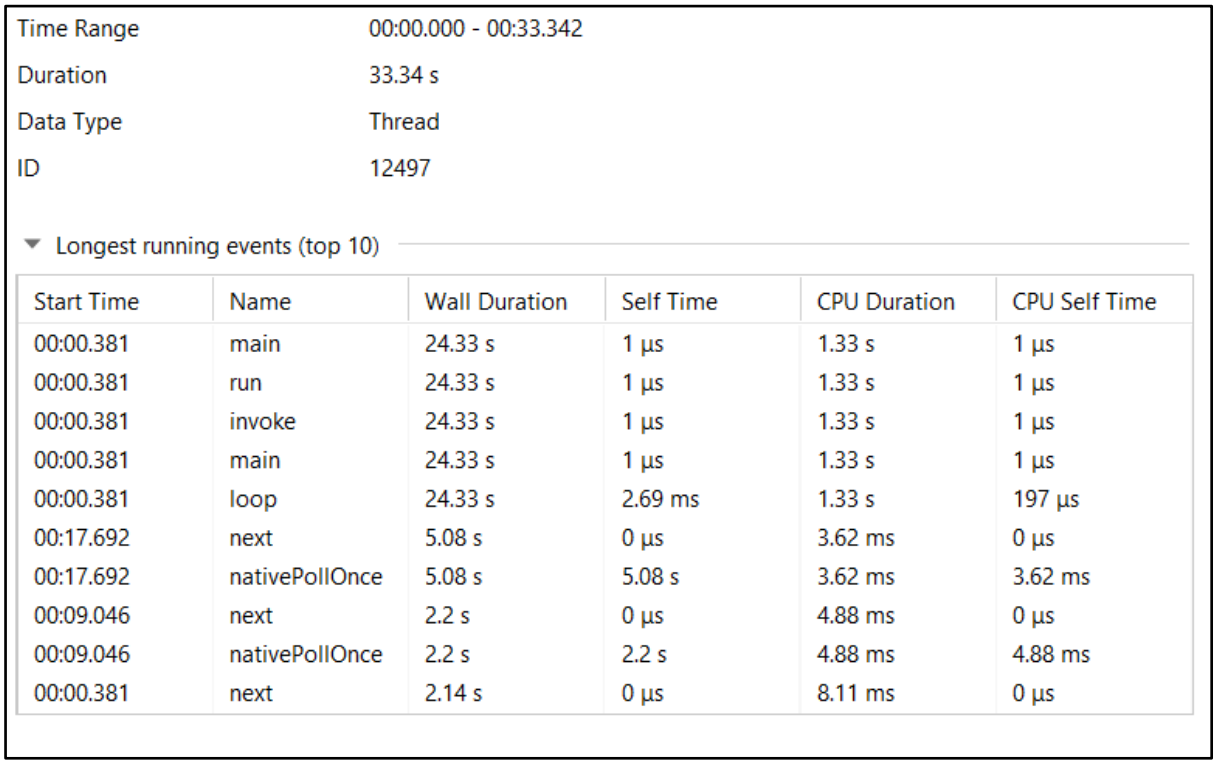
2,441 Classes	0 Leaks	223,018 Count	9,915,993 Native Size	11,541,862 Shallow Size	42,390,040 Retained Size

Class Name	Allocations	Native Size	Shallow Size	Retained Size
app heap	223,018	9,915,993	11,541,862	42,390,040
Bitmap (android.graphics)	7	9,815,919	294	9,816,213
AppCompatImageView (androidx.appcompat.widget)	88	0	53,152	4,839,979
BitmapDrawable (android.graphics.drawable)	20	0	1,460	4,783,070
BitmapDrawable$BitmapState (android.graphics.drawable)	13	0	754	4,782,104
byte[]	23,385	0	3,370,955	3,370,955
Class (java.lang)	4,788	0	641,556	1,906,608
char[]	4,043	0	1,616,006	1,616,006
DirectByteBuffer (java.nio)	12	0	720	1,399,451
Object[] (java.lang)	8,996	0	450,284	708,012
int[]	15,951	0	689,608	689,608
Integer (java.lang)	42,900	0	514,800	514,800
String (java.lang)	14,416	0	230,656	488,406
HashMap$Node (java.util)	5,864	0	140,736	388,332
ArrayList (java.util)	6,421	0	128,420	288,231
Cleaner (sun.misc)	5,840	0	210,240	280,520
HashMap$Node[] (java.util)	1,417	0	103,932	277,111
SolverVariable[] (androidx.constraintlayout.solver)	209	0	261,568	261,568
HashMap (java.util)	1,504	0	60,160	240,761
float[]	5,117	0	189,704	189,704
LabelTextView (es.gob.radarcovid.common.view)	153	0	141,372	155,285
NativeAllocationRegistry$CleanerThunk (libcore.util)	5,833	0	139,992	140,216
SolverVariable (androidx.constraintlayout.solver)	1,602	0	92,916	126,916
ClassExt (dalvik.system)	742	0	38,584	120,136
ArrayRow[] (androidx.constraintlayout.solver)	1,661	0	112,896	112,896
LinearLayout (android.widget)	76	0	55,784	103,543

Figure 36. Memory Profiler Results in Spain App 2.

SwissCovid Switzerland

CPU Profiler:



Time Range	00:00.000 - 00:33.342
Duration	33.34 s
Data Type	Thread
ID	12497

▼ Longest running events (top 10)

Start Time	Name	Wall Duration	Self Time	CPU Duration	CPU Self Time
00:00.381	main	24.33 s	1 µs	1.33 s	1 µs
00:00.381	run	24.33 s	1 µs	1.33 s	1 µs
00:00.381	invoke	24.33 s	1 µs	1.33 s	1 µs
00:00.381	main	24.33 s	1 µs	1.33 s	1 µs
00:00.381	loop	24.33 s	2.69 ms	1.33 s	197 µs
00:17.692	next	5.08 s	0 µs	3.62 ms	0 µs
00:17.692	nativePollOnce	5.08 s	5.08 s	3.62 ms	3.62 ms
00:09.046	next	2.2 s	0 µs	4.88 ms	0 µs
00:09.046	nativePollOnce	2.2 s	2.2 s	4.88 ms	4.88 ms
00:00.381	next	2.14 s	0 µs	8.11 ms	0 µs

Figure 37. CPU Profiler in Switzerland App.

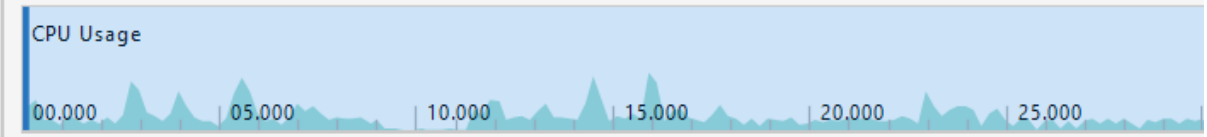


Figure 38. CPU Profiler in Switzerland App 2.

Memory Profiler:

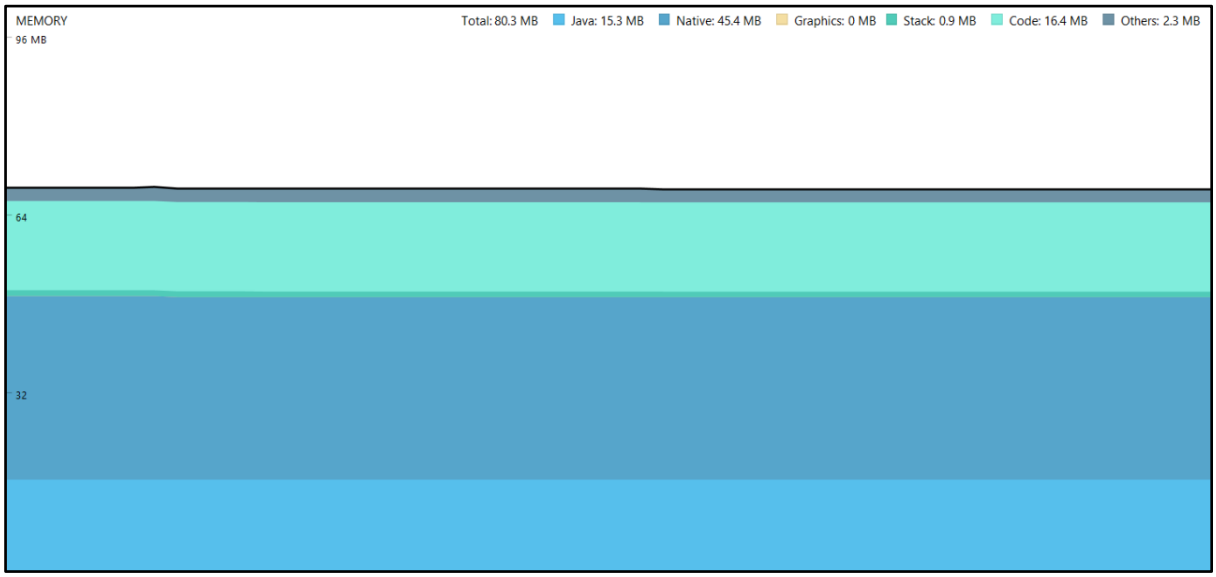
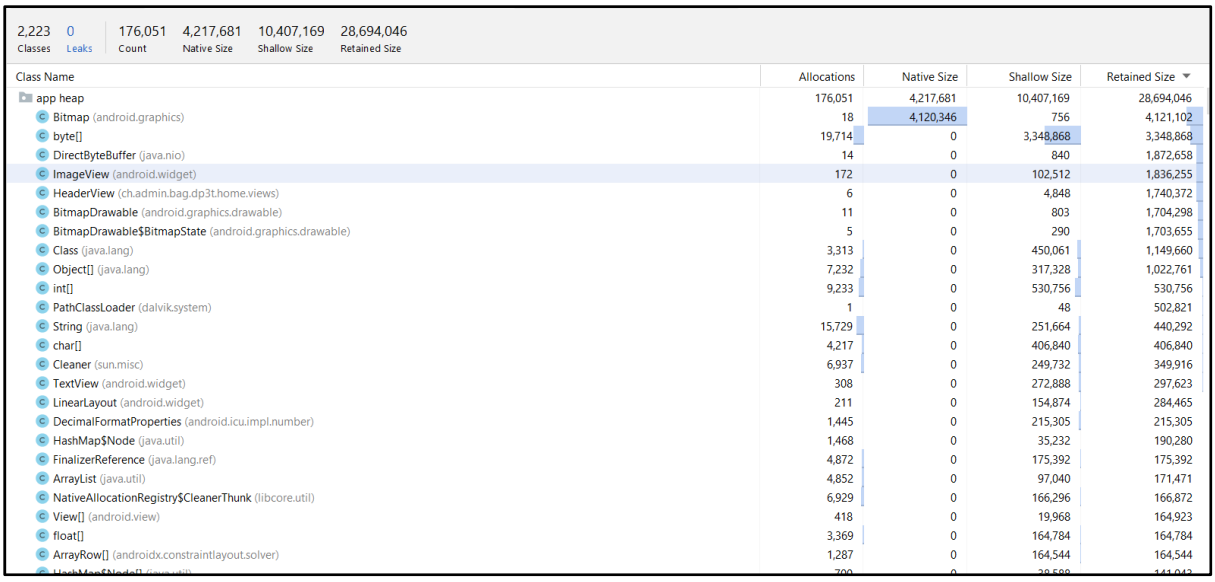


Figure 39. Memory Profiler Results in Switzerland App.



2,223 Classes	0 Leaks	176,051 Count	4,217,681 Native Size	10,407,169 Shallow Size	28,694,046 Retained Size

Class Name	Allocations	Native Size	Shallow Size	Retained Size
app heap	176,051	4,217,681	10,407,169	28,694,046
Bitmap (android.graphics)	18	4,120,346	756	4,121,102
byte[]	19,714	0	3,348,868	3,348,868
DirectByteBuffer (java.nio)	14	0	840	1,872,658
ImageView (android.widget)	172	0	102,512	1,836,255
HeaderView (ch.admin.bag.dp3t.home.views)	6	0	4,848	1,740,372
BitmapDrawable (android.graphics.drawable)	11	0	803	1,704,298
BitmapDrawable$BitmapState (android.graphics.drawable)	5	0	290	1,703,655
Class (java.lang)	3,313	0	450,061	1,149,660
Object[] (java.lang)	7,232	0	317,328	1,022,761
int[]	9,233	0	530,756	530,756
PathClassLoader (dalvik.system)	1	0	48	502,821
String (java.lang)	15,729	0	251,664	440,292
char[]	4,217	0	406,840	406,840
Cleaner (sun.misc)	6,937	0	249,732	349,916
TextView (android.widget)	308	0	272,888	297,623
LinearLayout (android.widget)	211	0	154,874	284,465
DecimalFormatProperties (android.icu.impl.number)	1,445	0	215,305	215,305
HashMap$Node (java.util)	1,468	0	35,232	190,280
FinalizerReference (java.lang.ref)	4,872	0	175,392	175,392
ArrayList (java.util)	4,852	0	97,040	171,471
NativeAllocationRegistry$CleanerThunk (libcore.util)	6,929	0	166,296	166,872
View[] (android.view)	418	0	19,968	164,923
float[]	3,369	0	164,784	164,784
ArrayRow[] (androidx.constraintlayout.solver)	1,287	0	164,544	164,544

Figure 40. Memory Profiler Results in Switzerland App 2.

3.4 UI Testing

For UI testing, we have used the Espresso framework, a tool that is provided by Android Studio itself. Espresso Test Recorder is an automated tool for the most commonly used general UI test concepts in an automated manner. In our study, we simulated user behavior and interactions with user interfaces using Espresso Test Recorder for all three countries. Espresso Test Recorder helped to prepare UI tests according to the parts such as button and text. We think this test is useful because all three covid-19 contact tracking applications are easy to use and do not have many pages.
We conducted tests using Espresso for all three applications.

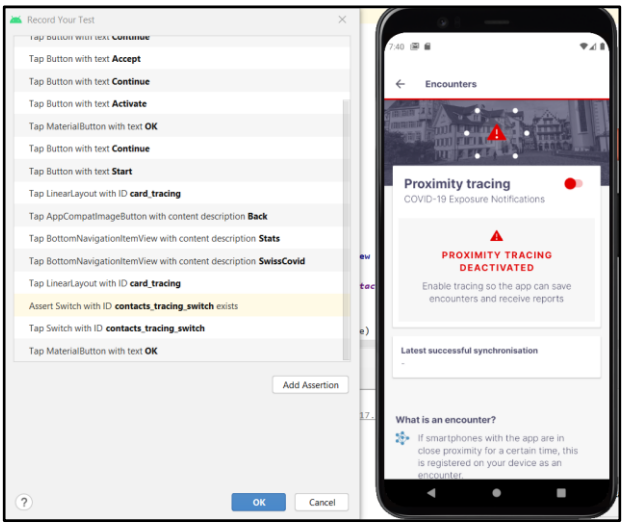


Figure 41. Espresso Framework with SwissCovid Application

Stop COVID-19 Croatia

```
@LargeTest
@RunWith(AndroidJUnit4.class)
public class UITest {

  @Rule
  public ActivityTestRule<SplashActivity> mActivityTestRule = new
ActivityTestRule<>(SplashActivity.class);

  @Test
  public void uITest() {
     ViewInteraction relativeLayout = onView(
allOf(withId(R.id.rlEnglish),
```

```
childAtPosition(
childAtPosition(
withClassName(is("android.widget.ScrollView")),
0),
4)));
        relativeLayout.perform(scrollTo(), click());

        ViewInteraction linearLayout = onView(
allOf(withId(R.id.llStart),
childAtPosition(
childAtPosition(
withClassName(is("android.widget.ScrollView")),
0),
5)));
        linearLayout.perform(scrollTo(), click());

        ViewInteraction appCompatButton = onView(
allOf(withId(R.id.btnProceed), withText("Continue"),
childAtPosition(
childAtPosition(
withClassName(is("androidx.core.widget.NestedScrollView")),
0),
8),
isDisplayed()));
        appCompatButton.perform(click());

        ViewInteraction appCompatButton2 = onView(
allOf(withId(R.id.btnSwitch),
childAtPosition(
allOf(withId(R.id.rlSwitch),
childAtPosition(
withId(R.id.llSwitch),
2)),
2),
isDisplayed()));
        appCompatButton2.perform(click());

        ViewInteraction bottomNavigationItemView = onView(
allOf(withId(R.id.navigation_notify_others), withContentDescription("instance of
android.text.SpannableString(id=19261)"),
childAtPosition(
childAtPosition(
withId(R.id.navigation),
0),
1),
isDisplayed()));
        bottomNavigationItemView.perform(click());

        ViewInteraction appCompatButton3 = onView(
allOf(withId(R.id.fragment_notify_share_button), withText("Share with others"),
```

```
childAtPosition(
allOf(withId(R.id.notify_share_section),
childAtPosition(
withClassName(is("android.widget.LinearLayout")),
1)),
2),
isDisplayed()));
        appCompatButton3.perform(click());

        ViewInteraction linearLayout2 = onView(
allOf(withId(R.id.mainContainer),
childAtPosition(
childAtPosition(
withClassName(is("android.widget.LinearLayout")),
1),
0)));
        linearLayout2.perform(scrollTo(), click());

        ViewInteraction textInputEditText = onView(
allOf(withId(R.id.shareTestIdentifier),
childAtPosition(
childAtPosition(
withId(R.id.tvShareId),
0),
0)));
        textInputEditText.perform(scrollTo(), replaceText("11111111"), closeSoftKeyboard());

        ViewInteraction appCompatButton4 = onView(
allOf(withId(R.id.shareNextButton), withText("Confirm"),
childAtPosition(
allOf(withId(R.id.mainContainer),
childAtPosition(
withClassName(is("android.widget.ScrollView")),
0)),
6)));
        appCompatButton4.perform(scrollTo(), click());
        }

    private static Matcher<View> childAtPosition(
            final Matcher<View> parentMatcher, final int position) {

        return new TypeSafeMatcher<View>() {
            @Override
            public void describeTo(Description description) {
                description.appendText("Child at position " + position + " in parent ");
                parentMatcher.describeTo(description);
            }

            @Override
            public boolean matchesSafely(View view) {
```

```
            ViewParent parent = view.getParent();
            return parent instanceof ViewGroup && parentMatcher.matches(parent)
                && view.equals(((ViewGroup)parent).getChildAt(position));
        }
    };
}
}
```

Radar COVID Spain

```
class UITest {

  @Rule
  @JvmField
  var mActivityTestRule = ActivityTestRule(SplashActivity::class.java)

  @Test
  fun UITest() {
    val labelButton = onView(
        allOf(withId(R.id.buttonContinue), withText("Continue"),
            childAtPosition(
                childAtPosition(
                    withClassName(`is`("android.widget.FrameLayout")),
                    0),
                1),
            isDisplayed()))
    labelButton.perform(click())

    val labelCheckBox = onView(
        allOf(withId(R.id.checkBoxTermsAndConditions), withContentDescription("instance of
android.text.SpannableStringBuilder(id=19858)"),
            childAtPosition(
                childAtPosition(
                    withClassName(`is`("android.widget.LinearLayout")),
                    1),
                0)))
    labelCheckBox.perform(scrollTo(), click())


    val labelButton2 = onView(
        allOf(withId(R.id.buttonAccept), withText("Continue"),
            childAtPosition(
                allOf(withId(R.id.wrapperButtons),
                    childAtPosition(
                        withClassName(`is`("androidx.constraintlayout.widget.ConstraintLayout")),
                        1)),
                0),
            isDisplayed()))
    labelButton2.perform(click())
```

```
val labelButton3 = onView(
        allOf(withId(R.id.buttonContinue), withText("Continue"),
                childAtPosition(
                        childAtPosition(
                                withClassName(`is`("android.widget.FrameLayout")),
                                0),
                        1),
                isDisplayed()))
labelButton3.perform(click())

val switchCompat = onView(
        allOf(withId(R.id.switchRadar),
                childAtPosition(
                        childAtPosition(
                                withClassName(`is`("android.widget.LinearLayout")),
                                1),
                        1)))
switchCompat.perform(scrollTo(), click())

val appCompatButton2 = onView(
        allOf(withId(R.id.buttonOk), withText("Accept"),
                childAtPosition(
                        allOf(withId(R.id.wrapperErrorMessage),
                                childAtPosition(
                                        withId(R.id.viewBackground),
                                        1)),
                        2),
                isDisplayed()))
appCompatButton2.perform(click())

val labelConstraintLayout = onView(
        allOf(withId(R.id.wrapperExposure),
                childAtPosition(
                        childAtPosition(
                                withClassName(`is`("androidx.constraintlayout.widget.ConstraintLayout")),
                                1),
                        0)))
labelConstraintLayout.perform(scrollTo(), click())

val appCompatImageButton = onView(
        allOf(withId(R.id.imageButtonBack), withContentDescription("Radar Back"),
                childAtPosition(
                        allOf(withId(R.id.layoutBackButton),
                                childAtPosition(
                                        withClassName(`is`("androidx.constraintlayout.widget.ConstraintLayout")),
                                        0)),
                        0),
                isDisplayed()))
appCompatImageButton.perform(click())
```

```
    pressBack()

    val bottomNavigationItemView = onView(
        allOf(withId(R.id.menuItemProfile), withContentDescription("instance of
android.text.SpannableStringBuilder(id=20318)"),
            childAtPosition(
                childAtPosition(
                    withId(R.id.bottomNavigation),
                    0),
                1),
            isDisplayed()))
    bottomNavigationItemView.perform(click())

    val bottomNavigationItemView2 = onView(
        allOf(withId(R.id.menuItemHelpline), withContentDescription("instance of
android.text.SpannableStringBuilder(id=20351)"),
            childAtPosition(
                childAtPosition(
                    withId(R.id.bottomNavigation),
                    0),
                2),
            isDisplayed()))
    bottomNavigationItemView2.perform(click())


  private fun childAtPosition(
      parentMatcher: Matcher<View>, position: Int): Matcher<View> {

    return object : TypeSafeMatcher<View>() {
      override fun describeTo(description: Description) {
        description.appendText("Child at position $position in parent ")
        parentMatcher.describeTo(description)
      }

      public override fun matchesSafely(view: View): Boolean {
        val parent = view.parent
        return parent is ViewGroup && parentMatcher.matches(parent)
            && view == parent.getChildAt(position)
      }
    }
  }
}
```

SwissCovid Switzerland

```
package ch.admin.bag.dp3t;
@LargeTest
@RunWith(AndroidJUnit4.class)
public class UITesting {
```

```
@Rule
public ActivityTestRule<MainActivity> mActivityTestRule = new ActivityTestRule<>(MainActivity.class);

@Test
public void uITesting() {
    ViewInteraction button = onView(
            allOf(withId(R.id.onboarding_continue_button), withText("Continue"),
                    childAtPosition(
                            childAtPosition(
                                    withClassName(is("android.widget.LinearLayout")),
                                    1),
                            0),
                    isDisplayed()));
    button.perform(click());

    ViewInteraction button7 = onView(
            allOf(withId(R.id.onboarding_gaen_button), withText("Activate"),
                    childAtPosition(
                            childAtPosition(
                                    withClassName(is("android.widget.ScrollView")),
                                    0),
                            3)));
    button7.perform(scrollTo(), click());

    ViewInteraction materialButton = onView(
            allOf(withId(android.R.id.button1), withText("OK"),
                    childAtPosition(
                            childAtPosition(
                                    withId(R.id.buttonPanel),
                                    0),
                            3)));
    materialButton.perform(scrollTo(), click());

    ViewInteraction button9 = onView(
            allOf(withId(R.id.onboarding_continue_button), withText("Start"),
                    childAtPosition(
                            childAtPosition(
                                    withClassName(is("android.widget.ScrollView")),
                                    0),
                            3)));
    button9.perform(scrollTo(), click());

    ViewInteraction linearLayout = onView(
            allOf(withId(R.id.card_what_to_do_symptoms),
                    childAtPosition(
                            childAtPosition(
                                    withId(R.id.frame_card_symptoms),
                                    0),
                            0)));
    linearLayout.perform(scrollTo(), click());
```

```
ViewInteraction button10 = onView(
        allOf(withId(R.id.wtd_symtoms_button), withText("Further information"),
                childAtPosition(
                        childAtPosition(
                                withClassName(is("androidx.cardview.widget.CardView")),
                                0),
                        3)));
button10.perform(scrollTo(), click());

ViewInteraction textView = onView(
        allOf(withText("Bern"),
                childAtPosition(
                        allOf(withId(R.id.where_to_test_links_container),
                                childAtPosition(
                                        withClassName(is("android.widget.LinearLayout")),
                                        1)),
                        5)));
textView.perform(scrollTo(), click());

ViewInteraction imageView = onView(
        allOf(withId(R.id.where_to_test_close_button),
                childAtPosition(
                        childAtPosition(
                                withId(android.R.id.content),
                                0),
                        0),
                isDisplayed()));
imageView.perform(click());

ViewInteraction appCompatImageButton = onView(
        allOf(withContentDescription("Back"),
                childAtPosition(
                        allOf(withId(R.id.wtd_symptoms_toolbar),
                                childAtPosition(
                                        withClassName(is("android.widget.LinearLayout")),
                                        0)),
                        1),
                isDisplayed()));
appCompatImageButton.perform(click());

ViewInteraction linearLayout2 = onView(
        allOf(withId(R.id.card_what_to_do_test),
                childAtPosition(
                        childAtPosition(
                                withId(R.id.frame_card_test),
                                0),
                        0)));
linearLayout2.perform(scrollTo(), click());
```

```
    ViewInteraction button11 = onView(
            allOf(withId(R.id.wtd_inform_button), withText("Enter the Covidcode"),
                    childAtPosition(
                            childAtPosition(
                                    withClassName(is("android.widget.LinearLayout")),
                                    0),
                            3),
                    isDisplayed()));
    button11.perform(click());

    ViewInteraction editText = onView(
            childAtPosition(
                    allOf(withId(R.id.trigger_fragment_input),
                            childAtPosition(
                                    withClassName(is("android.widget.LinearLayout")),
                                    3)),
                    12));
    editText.perform(scrollTo(), replaceText("123456789000"), closeSoftKeyboard());


    ViewInteraction bottomNavigationItemView = onView(
            allOf(withId(R.id.bottom_nav_stats), withContentDescription("Stats"),
                    childAtPosition(
                            childAtPosition(
                                    withId(R.id.fragment_main_navigation_view),
                                    0),
                            1),
                    isDisplayed()));
    bottomNavigationItemView.perform(click());
  }

  private static Matcher<View> childAtPosition(
      final Matcher<View> parentMatcher, final int position) {

    return new TypeSafeMatcher<View>() {
      @Override
      public void describeTo(Description description) {
        description.appendText("Child at position " + position + " in parent ");
        parentMatcher.describeTo(description);
      }

      @Override
      public boolean matchesSafely(View view) {
        ViewParent parent = view.getParent();
        return parent instanceof ViewGroup && parentMatcher.matches(parent)
            && view.equals(((ViewGroup) parent).getChildAt(position));
      }
    };
  }
}
```

3.5 Code Coverage Testing

Code coverage percentage = ((code coverage line by automation test divide by the total number of lines in source code) * 100)

Spain (Radar Covid)

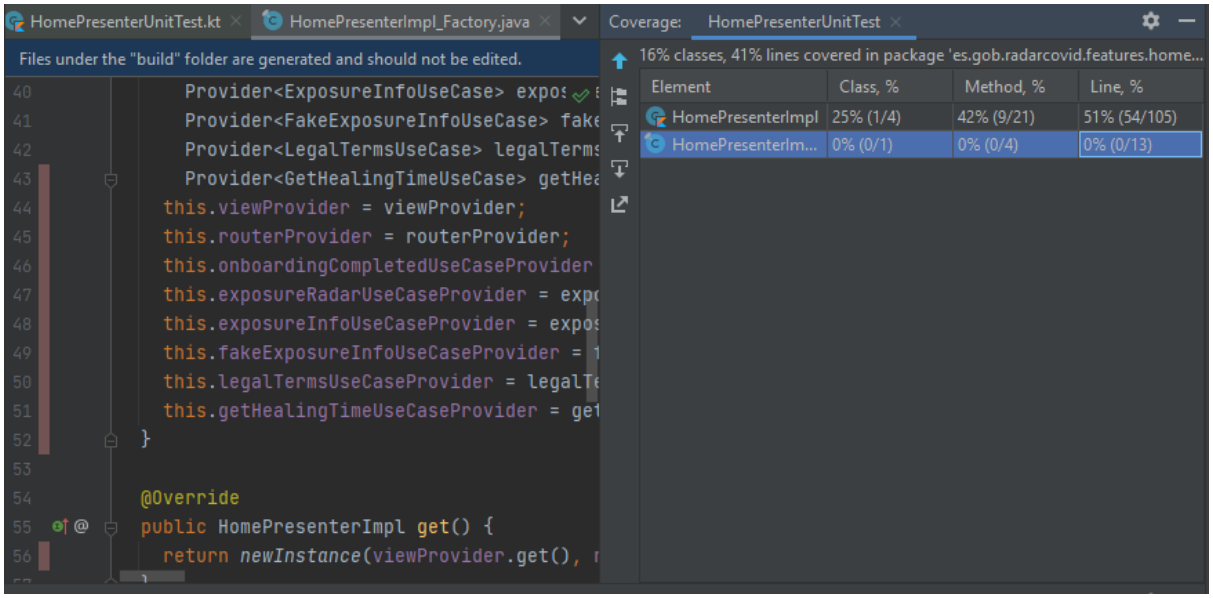


Figure 42. Code Coverage Testing for HomePresenterUnitTest.

In this unit test, there are four classes but only one class has covered codes, that's why the percentage is 25% Also, there are 21 methods and since 9 are covered, the percentage for that is 42%. Lastly, there are 105 code lines, the percentage is 51% because the covered lines are 54.

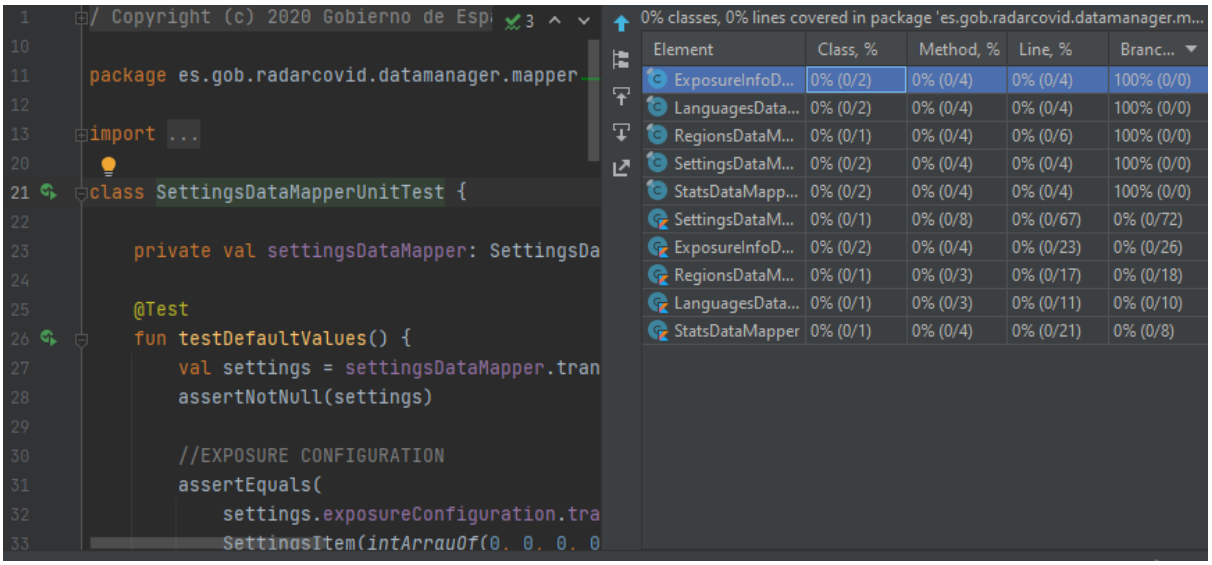


Figure 43. Code Coverage Testing for SettingDataMapperUnitTest.

In this unit test, there is one class, eight methods, and 72 code lines, but all classes are checked. That's why all percentages are 0% except for the first five ones for code lines.

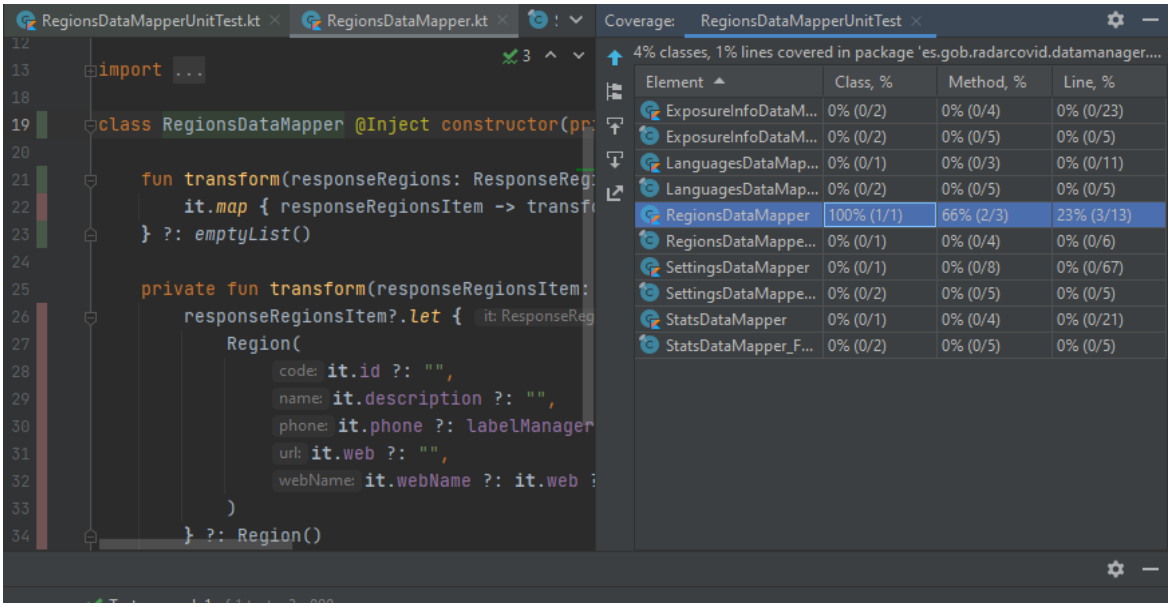


Figure 44. Code Coverage Testing for RegionsDataMapperUnitTest.

In this unit test, there is one class, three methods, and 13 code lines. The class of percentage is 100% because it has covered codes, but the method of percentage is 66% only 2 methods of 3 are covered. Code lines of percentage are 23% because 3 lines of 13 are covered.

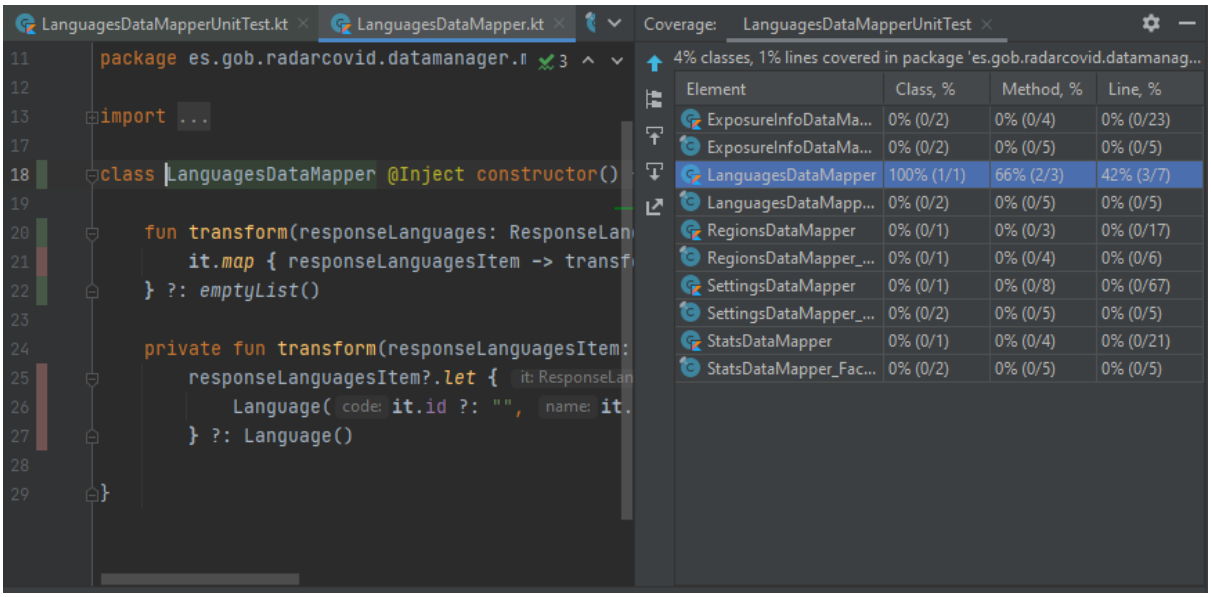


Figure 45. Code Coverage Testing for LanguagesDataMapperUnitTest.

In this unit test, there is one class, three methods, and seven code lines. The class of percentage is 100% because it has covered codes, but the method of percentage is 66% only 2 methods of 3 are covered. Code lines of percentage are 42% because 3 lines of 7 are covered.

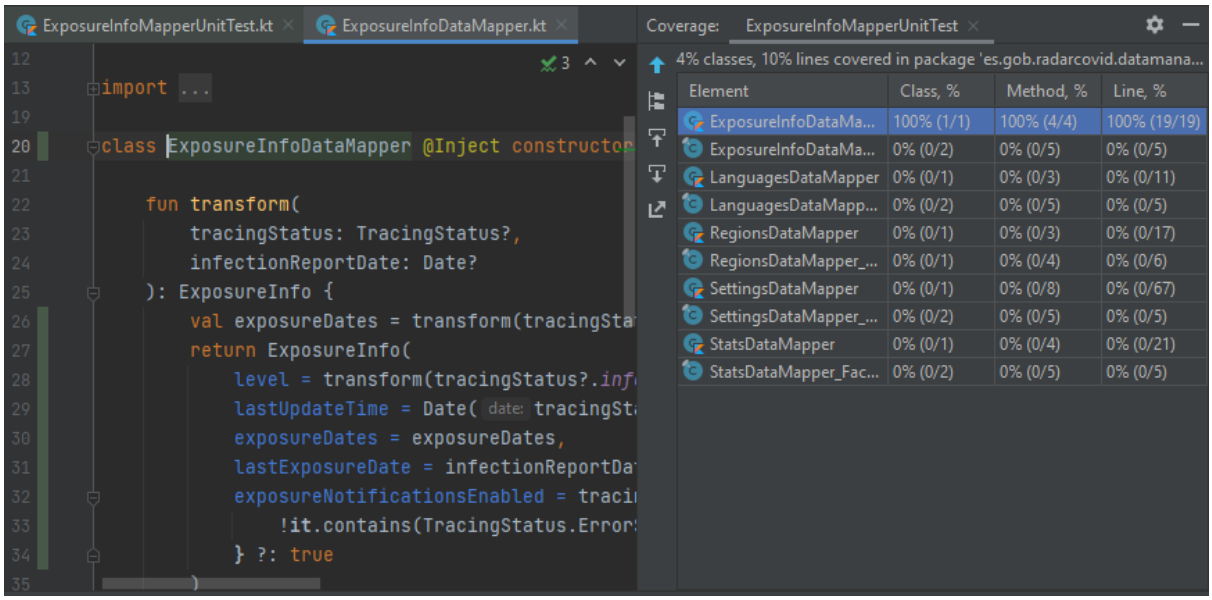


Figure 46. Code Coverage Testing for ExposureInfoMapperUnitTest.

In this unit test, there is one class, four methods, and 19 code lines. All of them are 100% because everything is fully and correctly covered.

3.6 Mutation Testing

In mutation testing we had errors in the beginning and the report wasn't generating at all. We tried different way to do it and we end up by this result:

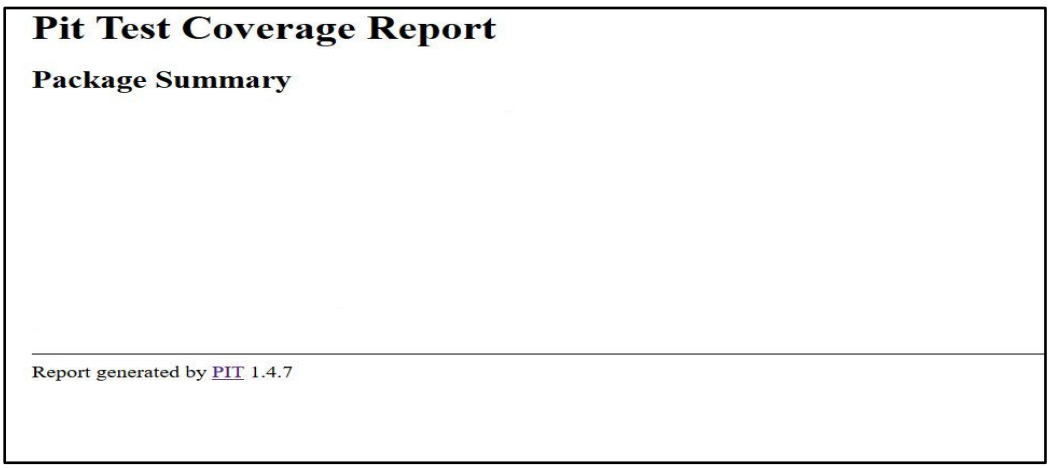
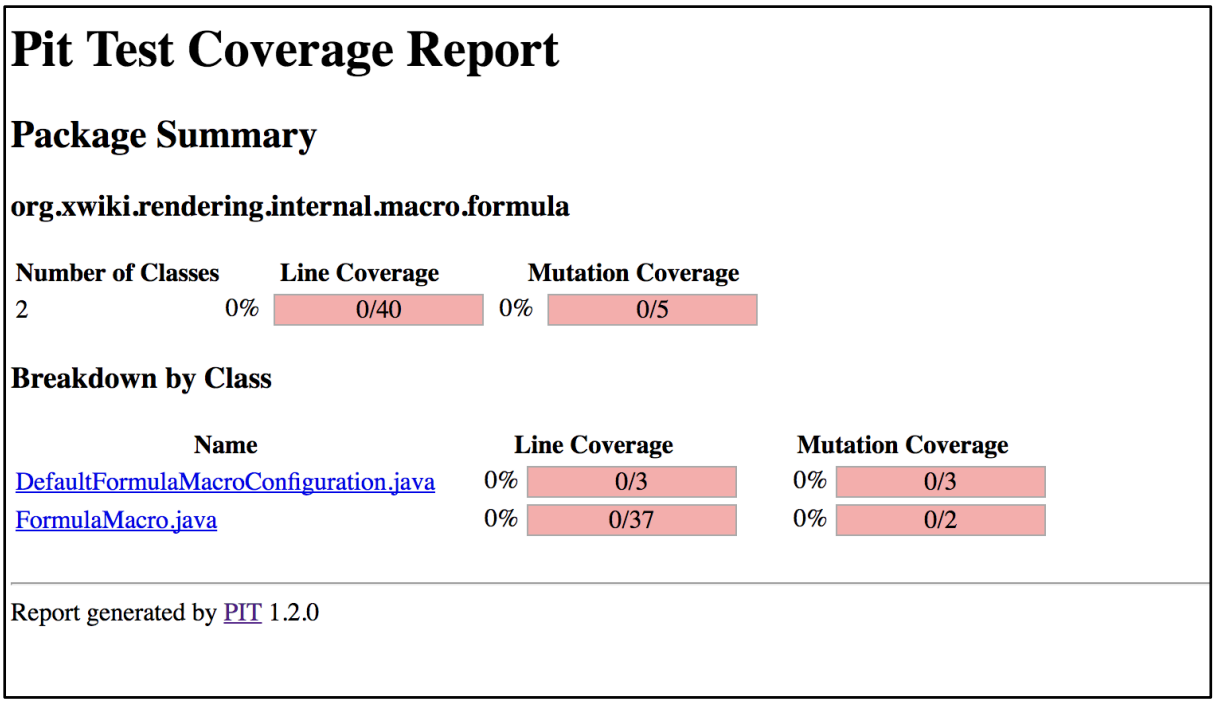


Figure 47. Actual Pit Test Coverage Report for SettingDataMapperUnitTest.

In the previous figure it shows the generated Pit Test Coverage Report without any information, this is the best result we had after many errors that will be shown at the end of report. However, in the 3.6 code coverage has been shown, one test class example was code coverage testing for SettingDataMapperUnitTest and all code coverage and mutation was 0%. Therefore the following figure is taken from internet, but it shows what we expect from Pit Test report if it was working:



Pit Test Coverage Report

Package Summary

org.xwiki.rendering.internal.macro.formula

Number of Classes	Line Coverage		Mutation Coverage	
2	0%	0/40	0%	0/5

Breakdown by Class

Name	Line Coverage		Mutation Coverage	
DefaultFormulaMacroConfiguration.java	0%	0/3	0%	0/3
FormulaMacro.java	0%	0/37	0%	0/2

Report generated by PIT 1.2.0

Figure 48. Expected Pit Test Coverage Report for SettingDataMapperUnitTest.

Finally, this was one expected result taken from the internet, other classes results can be expected the same way but the reason it's not included here is because we didn't like the idea of taking more figures from the internet that is not our own work.

3.7 Acceptance Testing

These use cases are the same for the 3 applications (Croatia – Spain – Switzerland) as all of the 3 apps work with the same logic and they all contain the 4 stages listed below.

First stage:

1. Run the application

    1.1. The splash screen must open when the user runs the app.

    1.2. The splash screen must appear for 2 seconds maximum.

    1.3. After the splash screen, the announcements screen must appear.

Second stage:

2. Terms & conditions, and notifications

    2.1. The user must “check” the checkbox in the terms and conditions screen to be able to continue in the app.

        2.1.1. (ERROR) if the user didn’t “check” the checkbox, an error message must appear.

    2.2. When the user clicks continue, a message must appear to ask the user to allow notifications.

    2.3. Whatever user selects, then must be moved to the home screen

Third stage:

3. Application usage

This is so general and wide topic to be concluded in a couple of lines, for effective mobile application usage, many test cases need to be considered as:

- Functional testing
- Performance testing
- Battery usage testing
- Usability testing
- Localization testing

- Recoverability testing

And many more tests. However, these are the most important points considered for the selected applications.

3.1. The app must work efficiently on navigation gestures.

3.2. The app must allow the user to change the application language.

3.3. The app must allow the user to enter his/her covid-19 code.

3.3.1. The app must take and store the covid-19 code entered by the user.

3.4. The app must verify the device compatibility.

3.5. The app must check the user interface (UI) interactions quickly.

3.6. The app must work on the “Standby” Mode.

3.7. The app must work efficiently with on-device permissions.

3.8. The app must perform in a normal way while other apps are running.

Fourth stage:

4. Exiting the application

4.1. The app must keep working in the background if the user hasn’t completely closed it.

4.1.1. The app must keep the user on the last opened screen.

4.2. If the user completely closes the app, the splash screen must work the next time the user starts the app.

4.2.1. The app must move the user to the home screen.

Acceptance Testing Codes:

// as the following code works the same with Spain and Switzerland, I didn't show to code for

// all of them but only for Croatia, but in the project we did it for all the 3 countries.

```
// public class ExampleSpainAcceptanceTest extends Dsl
// public class ExampleSwitzerlandAcceptanceTest extends Dsl
public class ExampleCroatiaAcceptanceTest extends Dsl
{
  @Test
  @Channel(Croatia)
// @Channel(Spain)
// @Channel(Switzerland)
  public void shouldOpenTheApplicationCorrectly() throws Exception
  {
    application.runApplication
    application.splashScreenAppear;
    waitTo(splashDisappear, 2); // 2 here is “timeoutinSeconds: 2”
    application.announcementScreenAppear;
    clickTo(Continue, 2); // 2 here is “timeoutinSeconds: 2”
  }
  public void shouldAcceptTermsAndConditions() throws Exception
  {
    application.TermsAndConditionScreen;
    clickTo(acceptPolicies, 2); // 2 here is “timeoutinSeconds: 2”

    String AcceptPolicies = announcementScreen.getPolicies();
    assertEquals(AcceptPolicies, "I have read and accepted the policies.");
  }
  public void shouldNotContinueWithoutAcceptingTheTerms() throws Exception
  {
```

```
        application.TermsAndConditionScreen;

        clickTo(acceptPolicies, 2); // 2 here is "timeoutinSeconds: 2"


        String errorMessage = announcementScreen.getPolicies.getErrorText();

        assertEquals(errorMessage, "You must 'check' the Terms and Condition checkbox.");
    }


}
```

4 Lessons Learned

1. Gradle, Maven, Android Support plugin, AVD, API30

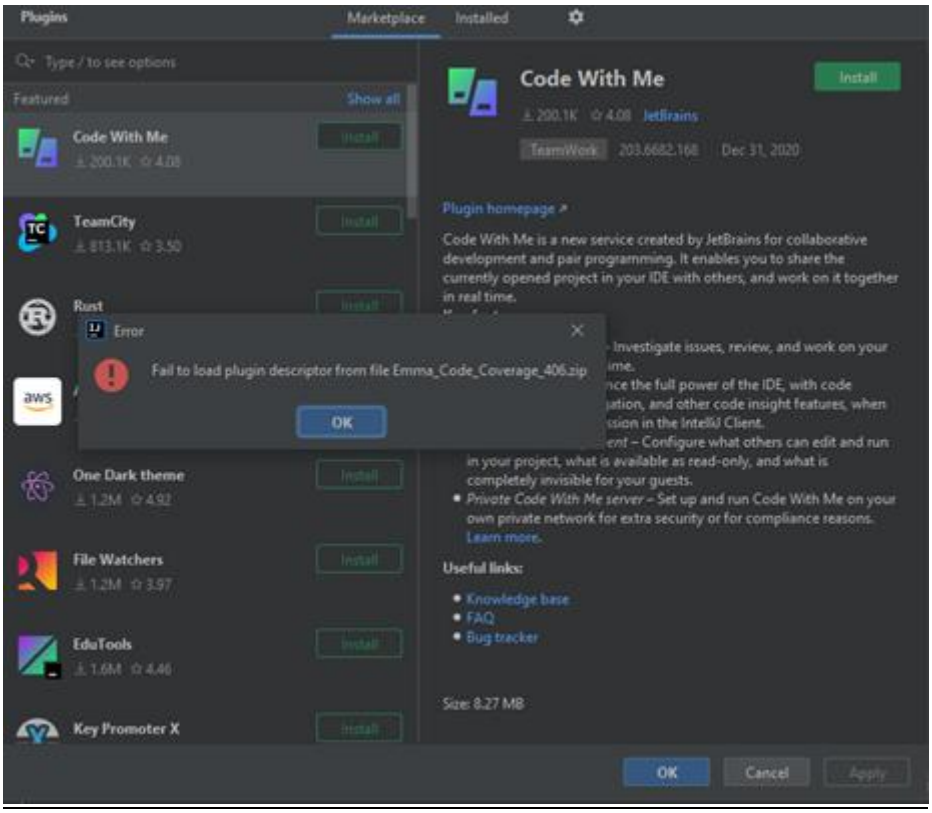


Figure 49. First Error, fail to load plugin Emma tool v.1.4.4.

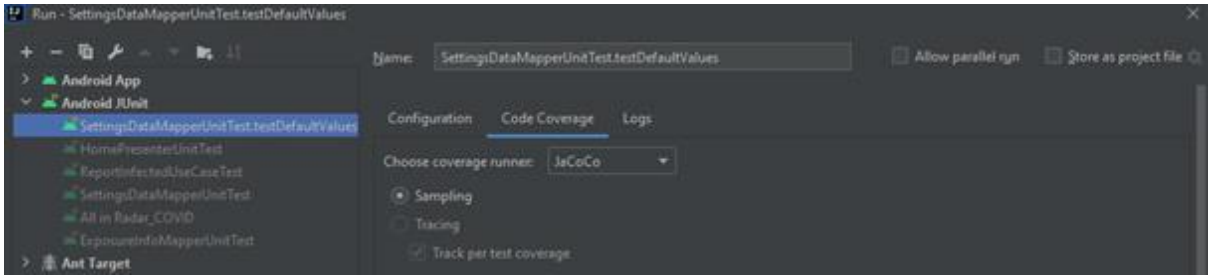


Figure 50. First try, Add plugin JaCoCo tool.

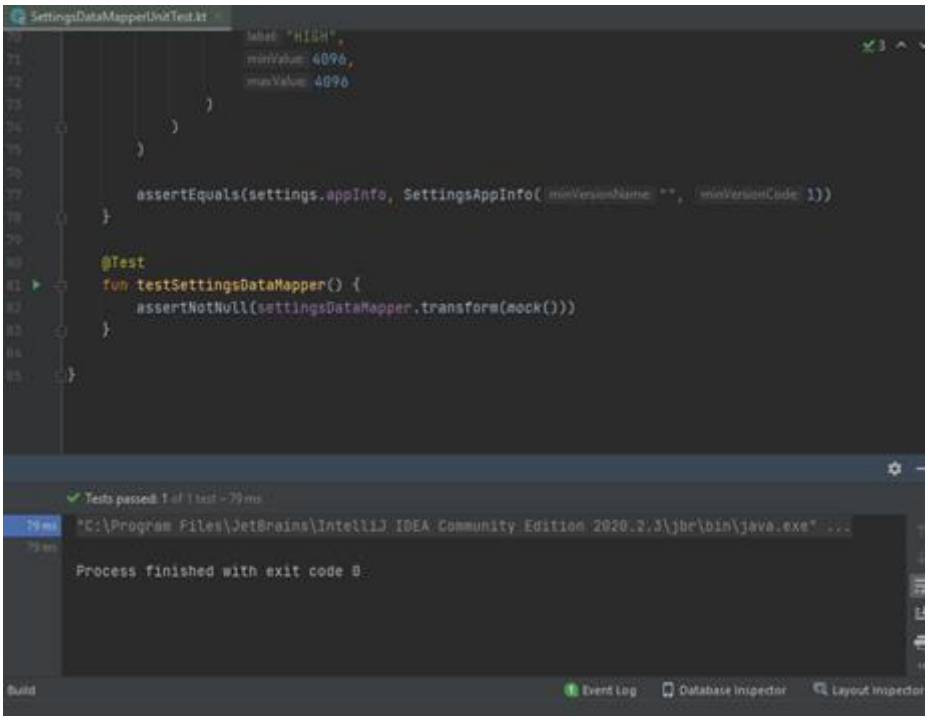


Figure 51. Second try, Checking code coverage with JaCoCo tool.

Figure 52. Third try, Checking code coverage with Unit Test History Coverage.

Figure 53. Error 2: Error appears while doing Mutation testing.

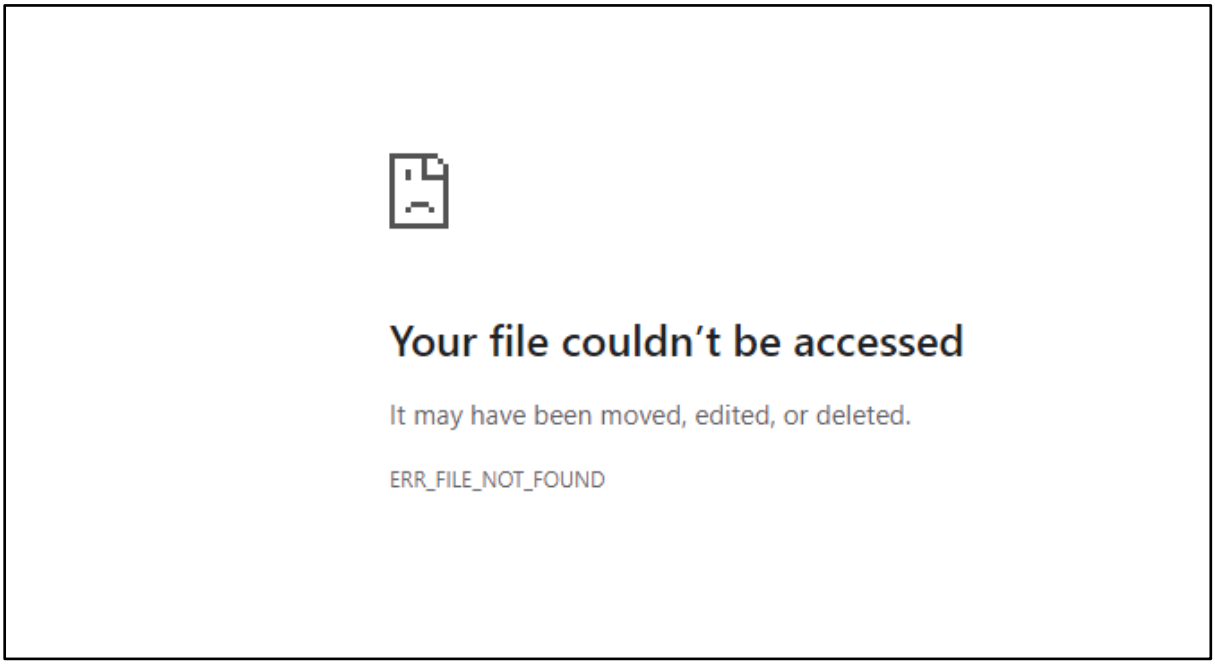


Figure 54. First try, Pit Test Coverage Report is not generating.

5. Conclusion

At the end of this report, we were able to learn a real software skill which is testing, we were able to understand many concepts in software testing and applying it as well. Moreover, we had difficulties, but we get through them and we were able to get to significant results eventually.

Applying many different tests was challenging for us, we had to divide the work clearly to make sure we finish the work correctly and most importantly on time.

6. References

1. Mobile Security Framework - MobSF Documentation. (2020). Retrieved 18 December 2020, from https://mobsf.github.io/docs/#/
2. Test apps on Android. (2020) - Android Documentation. Retrieved 30 December 2020, from https://developer.android.com/training/testing/
3. FDF, F. (2021). Current reports. Retrieved 12 January 2021, from https://www.ncsc.admin.ch/ncsc/en/home/dokumentation/covid-public-security-test/current_findings.html