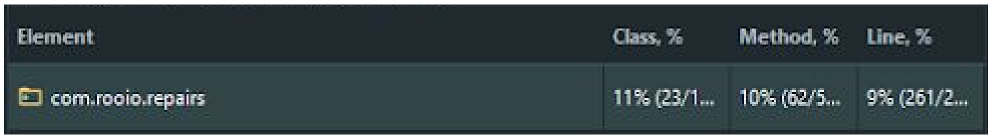
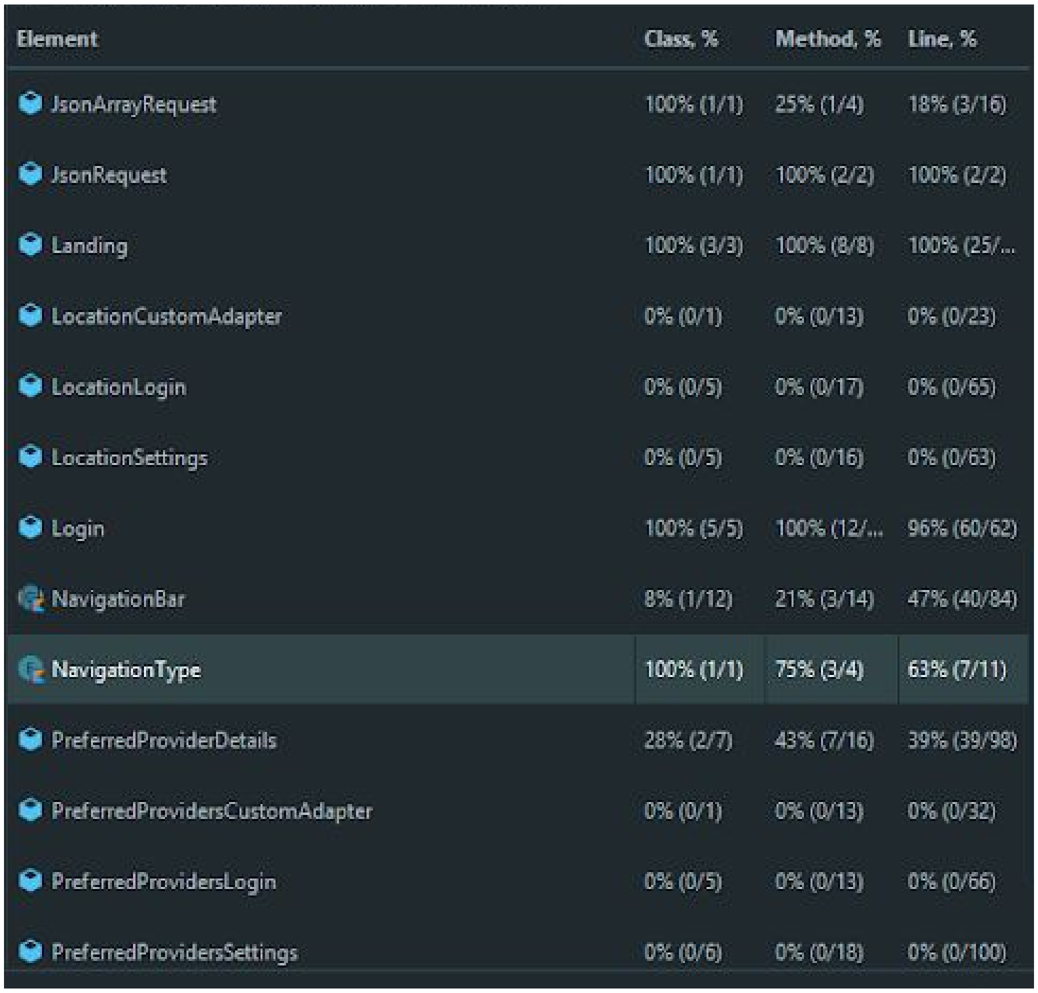
Luke Reckard
March 4th, 2020
Rooio

Code Coverage



Element	Class, %	Method, %	Line, %
com.rooio.repairs	11% (23/1...	10% (62/5...	9% (261/2...



Element	Class, %	Method, %	Line, %
JsonArrayRequest	100% (1/1)	25% (1/4)	18% (3/16)
JsonRequest	100% (1/1)	100% (2/2)	100% (2/2)
Landing	100% (3/3)	100% (8/8)	100% (25/...
LocationCustomAdapter	0% (0/1)	0% (0/13)	0% (0/23)
LocationLogin	0% (0/5)	0% (0/17)	0% (0/65)
LocationSettings	0% (0/5)	0% (0/16)	0% (0/63)
Login	100% (5/5)	100% (12/...	96% (60/62)
NavigationBar	8% (1/12)	21% (3/14)	47% (40/84)
NavigationType	100% (1/1)	75% (3/4)	63% (7/11)
PreferredProviderDetails	28% (2/7)	43% (7/16)	39% (39/98)
PreferredProvidersCustomAdapter	0% (0/1)	0% (0/13)	0% (0/32)
PreferredProvidersLogin	0% (0/5)	0% (0/13)	0% (0/66)
PreferredProvidersSettings	0% (0/6)	0% (0/18)	0% (0/100)

Our code coverage is currently at 11%. This seems like a small number, but it indicates progress towards an obtainable goal, as we are now able to test our code without limits. Before I get into the details though, I think it's important to mention that we currently use the Android Studio code coverage tool as a lot of Android UI tests do not like being run with gradle or CI, so they have to be run manually. However, I value actually having tests verify our code over automation, so that is a roadblock that our team will have to address later.

A lot of the current coverage comes from recent refactors after I found a way for structuring each Activity class to be testable. This included rewriting functions so that they are smaller and easy to unit test, as well as changing how we access and handle the REST API so that we are able to test our side of the application without calling the API. The Login class listed above is a great example of this. Each function in that class has a clear, single sentence purpose that is easily unit tested inside the class. There are only 2 lines in that class that are not covered, and this comes from an exception that is caught from a JSONObject that we are unsure of how to handle in our application. Other than that, the Login class is a testament to the ability that we are able to test most, if not all, of our other classes with ease.

Other classes that appear but are not fully covered have some issues that will be handled. For example, the JsonArrayRequest class is no longer used and our RestApi class (not listed) has a lot of code that needs to be removed but was left in until class testing refactors are complete. Once these and a multitude of other classes have code removed, our coverage will go up. Another issue comes with classes like PreferredProviderDetails. This page needs an ID for a preferred provider, but this data is encapsulated within the class, so it poses the question of how we should go about mocking data without exposing variables that should be private. I am confident that there is a solution to this problem, though, and coverage should increase to 100% for this class and similar classes after the issue has been resolved.