GSoC 2020 Proposal
{{Support tablet interfaces}}
({{Mohamed Medhat}})

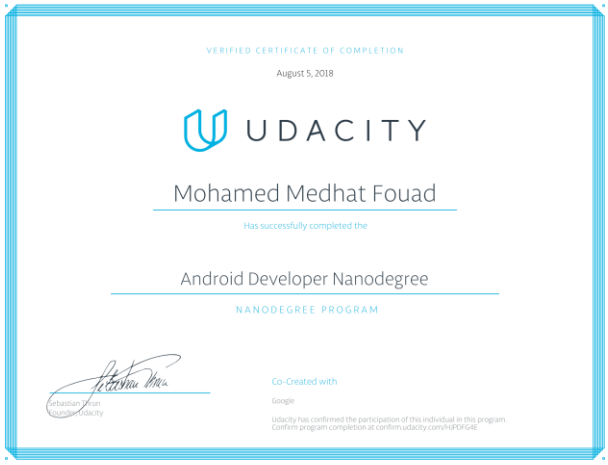
About You

Why are you interested in working with Oppia, and on your chosen project?

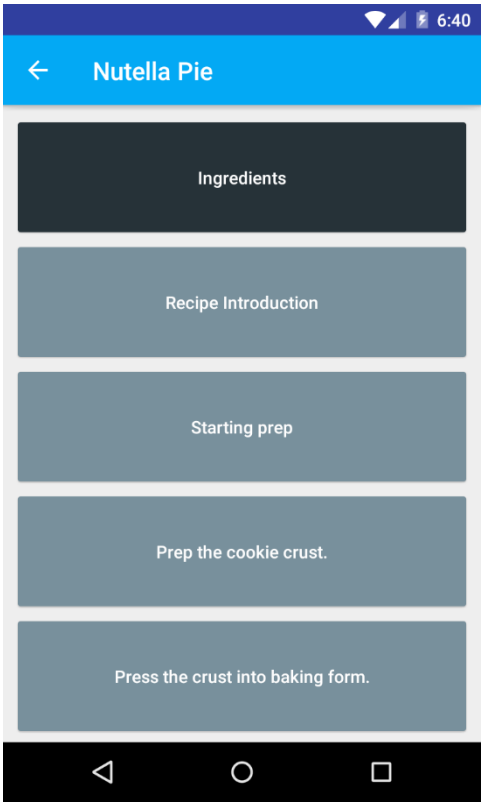
I loved the idea of the organization. It will help people learn quickly, especially those who can't go to school for any reason.
I want to work on this project because it meets my skills and it will give the users of the app a better experience.

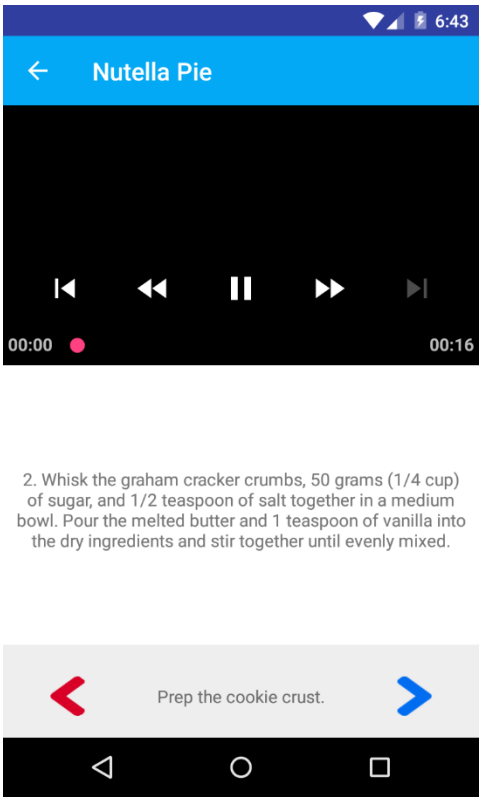
Prior experience

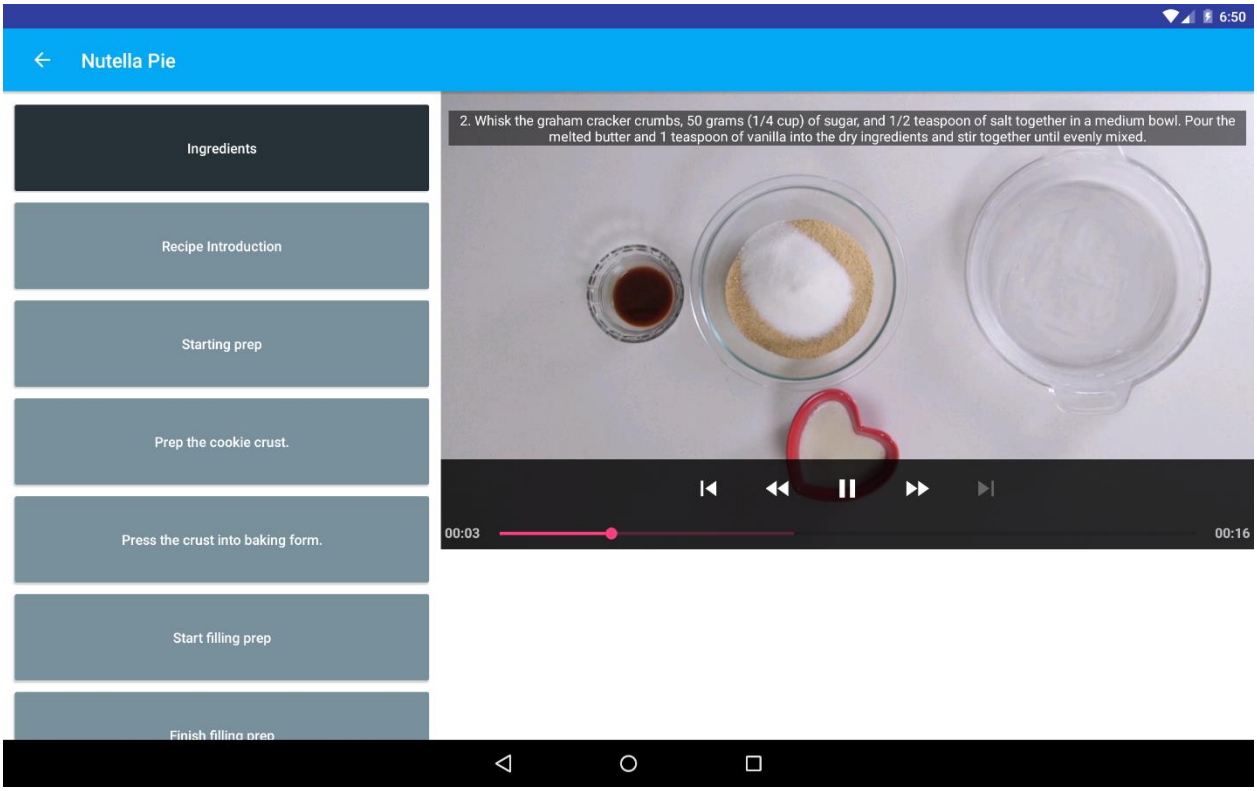
I'm a graduate of the Nanodegree program by Udacity, that is my certificate.



I have completed some large projects which I built from scratch.
I have completed a similar task to the required in this project -which is creating layouts for tablets- during the Nanodegree program by Udacity. It was to apply "master details flow" in one of the projects in the Nanodegree.
Here is a screenshot from the application:


6:40
Nutella Pie
Ingredients
Recipe Introduction
Starting prep
Prep the cookie crust.
Press the crust into baking form.


6:43
Nutella Pie
00:00
00:16
2. Whisk the graham cracker crumbs, 50 grams (1/4 cup) of sugar, and 1/2 teaspoon of salt together in a medium bowl. Pour the melted butter and 1 teaspoon of vanilla into the dry ingredients and stir together until evenly mixed.
Prep the cookie crust.


6:50
Nutella Pie
Ingredients
Recipe Introduction
Starting prep
Prep the cookie crust.
Press the crust into baking form.
Start filling prep
Finish filling prep
2. Whisk the graham cracker crumbs, 50 grams (1/4 cup) of sugar, and 1/2 teaspoon of salt together in a medium bowl. Pour the melted butter and 1 teaspoon of vanilla into the dry ingredients and stir together until evenly mixed.
00:03
00:16

I started to contribute to the oppia-android project on GitHub. I have three merged PRs, 1 successful PR but not merged due to circle ci tests, and 1 failed PR.
So, I became familiar with the project structure.
I also found that the project is combining MVVM with MVP and I know both.
The project uses DataBinding Library and I'm familiar with it, I'm not experienced with it, but I used it once before and I know how it works.
The project uses Kotlin and I switched to Kotlin early.

Here are my contributions to oppia:

- merged PRs:
  https://github.com/oppia/oppia-android/pull/857
  https://github.com/oppia/oppia-android/pull/858
  https://github.com/oppia/oppia-android/pull/964
- Successful PR but not merged:
  https://github.com/oppia/oppia-android/pull/842
- Failed PR:
  First trial: https://github.com/oppia/oppia-android/pull/753
  Second trial: https://github.com/oppia/oppia-android/pull/763

Contact info and timezone(s)

Email: mohamed.medhat0298@gmail.com
Skype: abomed7at55@gmail.com
Telegram: @M2010_1998
Discord: Abo_Med7at#0397
Mobile: +201063863298
Gitter: @MohamedMedhat1998
Hangouts: mohamed.medhat0298@gmail.com
LinkedIn: https://www.linkedin.com/in/mohamed-medhat-33952b146/
Facebook: https://www.facebook.com/abo.med7at.g.developer

Timezone: Egypt (UTC +2)

Preferred method of communication: **Email**

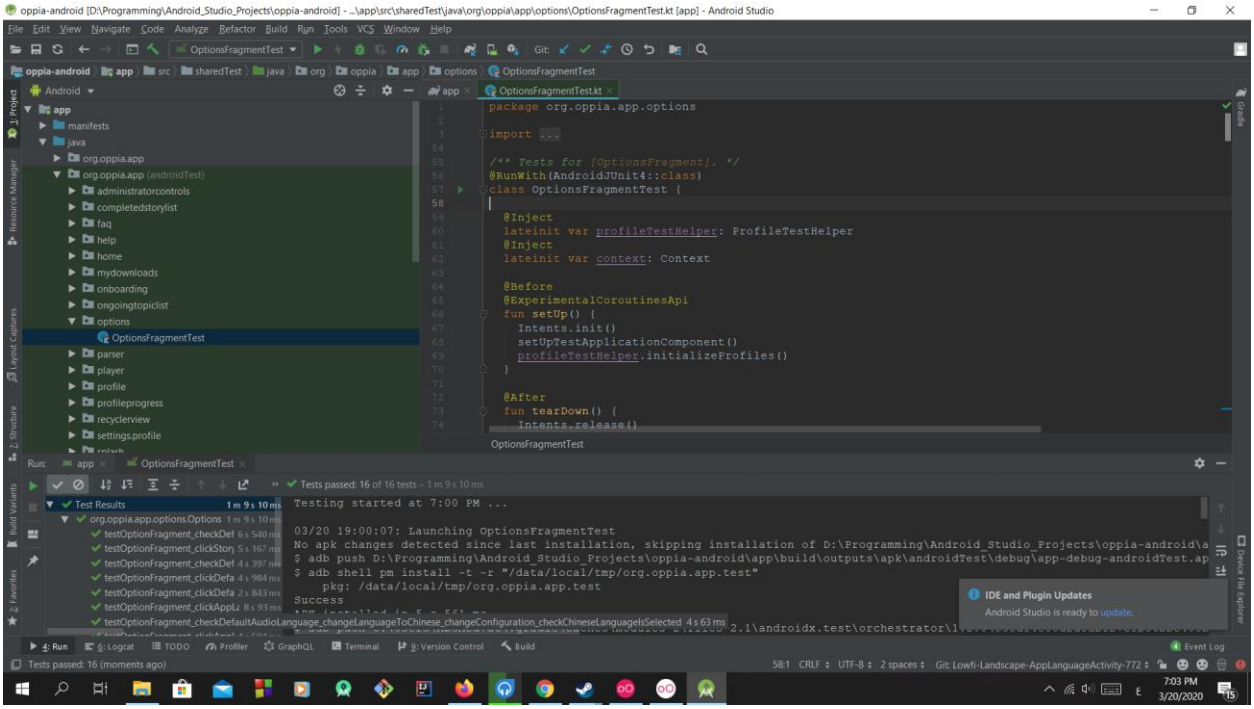
Preferred method of real-time communication: **Gitter, Hangouts**

Time commitment

7 hours/day for 5 days
3.5 hours in 1 day
1 day off
Total hours: 38.5 hours/week

Essential Prerequisites

I'm able to run unit tests on my machine



Other summer obligations

I have university exams in June.

Communication channels

I will communicate with my mentor using email and Gitter.
I usually check my email several times a day.
Gitter will be open while I'm working.

Project Details

Product Design

Target users: Tablet owners

Why this project matters?

Tablets are very easy to use and have larger screens than normal mobile phones. And a lot of people prefer to buy their kids a tablet device in order to help them learn through educational applications.
Oppia is one of these apps but it doesn't support layouts for tablet devices yet -**it doesn't look good on tablet devices**- and this project meets this problem.
It aims to support tablet layouts so that the app looks good and comfortable for both kids and parents.
Also, this project will improve the discoverability of the app and increase its users.
The application will make use of the extra space that tablets have, which will give the users a better user experience.

Technical Design

Architectural Overview

This project doesn't affect the architecture of the project too much. It only adds new UI elements to the project and modifies the **presenter layer** to cope with the new UI changes.
There are some activities that need to be modified. Some of these activities have related activities, but here I'm just showing the main activities. These activities are:
-OnboardingActivity
-ProfileActivity
-ProfileProgressActivity
-CompletedStoryActivity
-OngoingTopicActivity
-HintsAndSolutionActivity
-ConceptCardFragment
-RecentlyPlayedActivity
-HomeActivity
-TopicActivity
-OptionsActivity
-ExplorationPlayerActivity
-QuestionPlayerActivity
-Admin-related activities

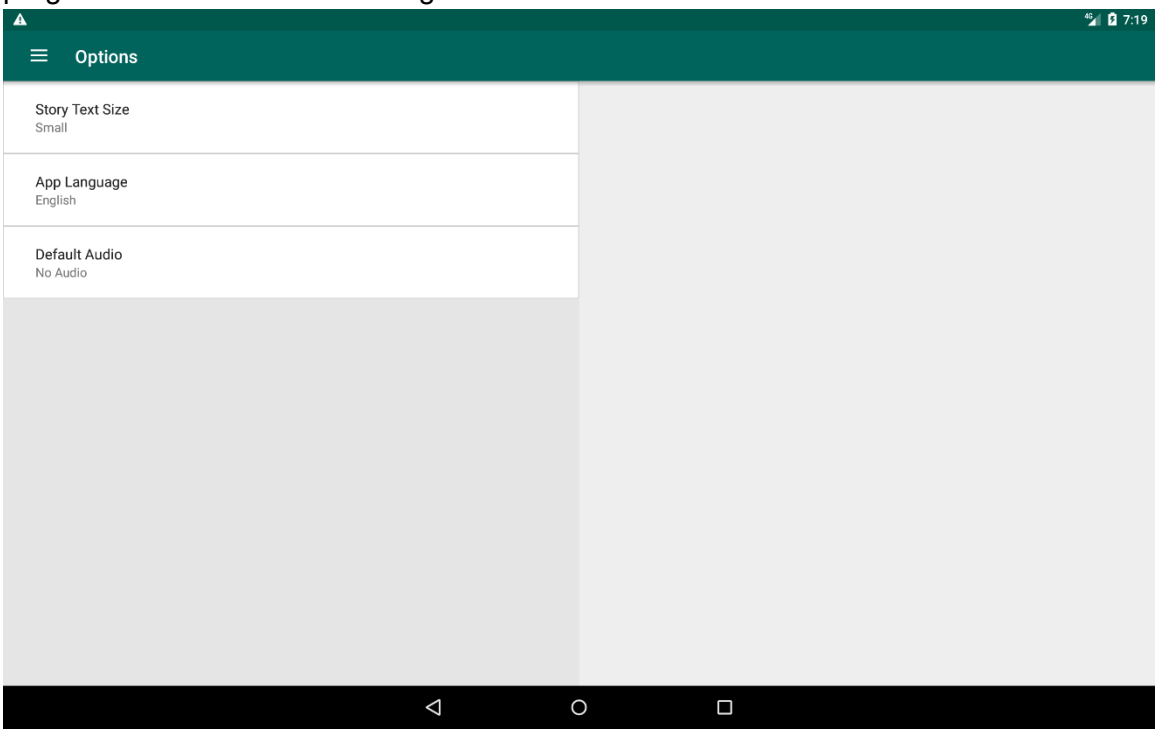
Implementation Approach

First: Writing XML code of tablet layouts following the tablet mocks.
Second: In the **app** module, in **presenter** classes, especially in **handleOnCreate** function, we'll check the existence of some UI element that exists in the tablet layout but doesn't exist in the normal layout so that we can determine whether the current application is running on a tablet or on a normal device.
Third: After determining the type of the device, we can modify the Kotlin to act correctly in both scenarios. There will be two paths:
one for the normal devices -*That will be the existing code of the presenter*-
and the other will be the path of tablet devices -*That can be achieved by creating a function that can use the new UI elements which appeared in the new XML layout for tablets*-.

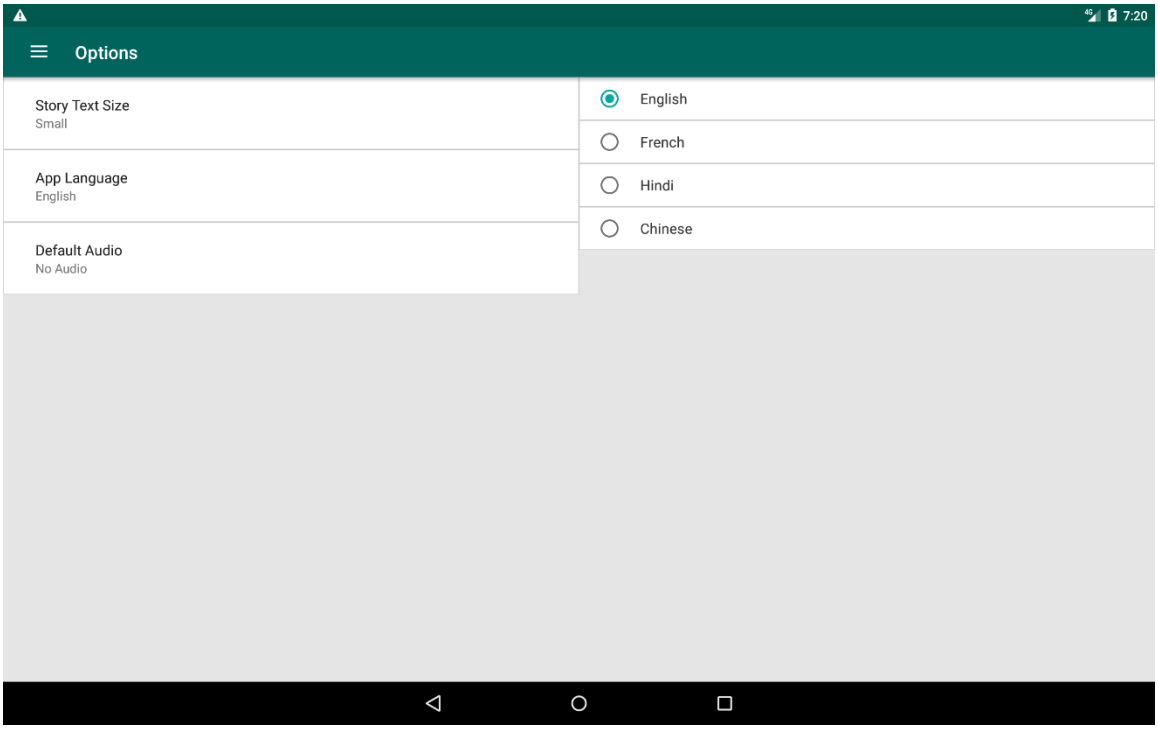
This is the general implementation process for each Activity.

Converting to tablet layouts example:

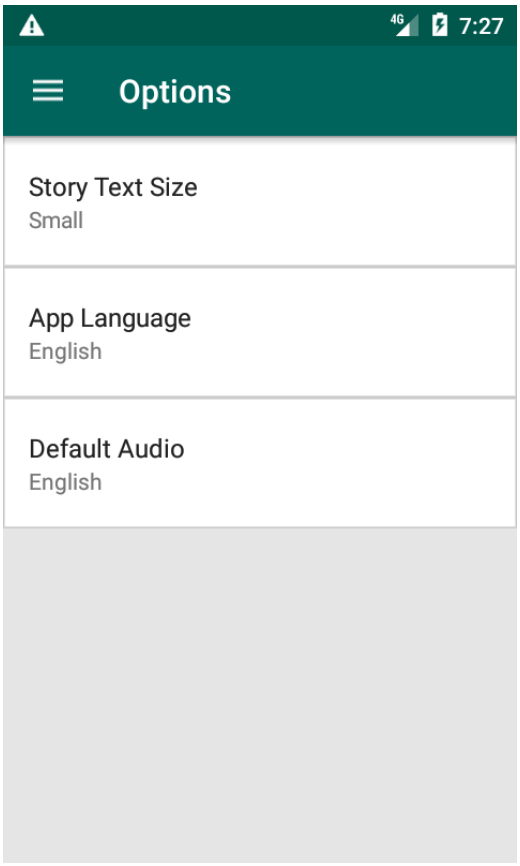
First, here is the result of converting to tablet layout, of course this is only an illustrative example and the design may be different. Also, the technical approach may change during the gsoc program based on the mentors' guides.

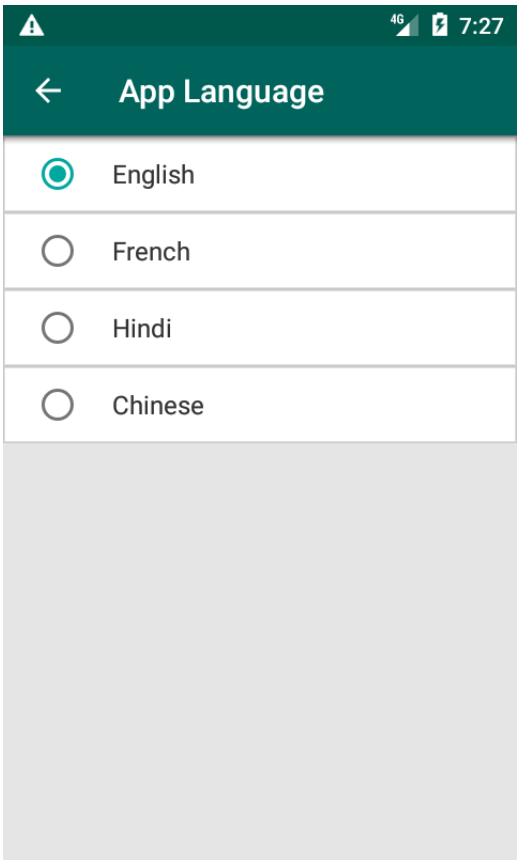


When the user clicks “App Language”, instead of navigating to a new activity, a new fragment will be placed in the right section as shown in the next picture.



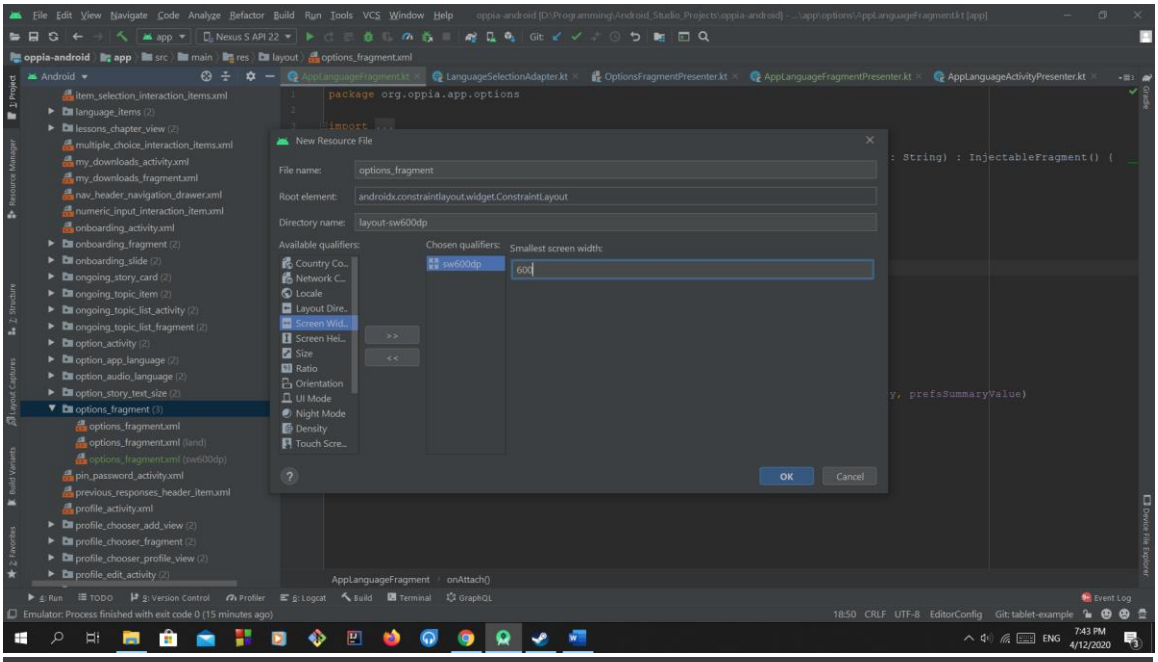
This appears in the normal phones as follows.

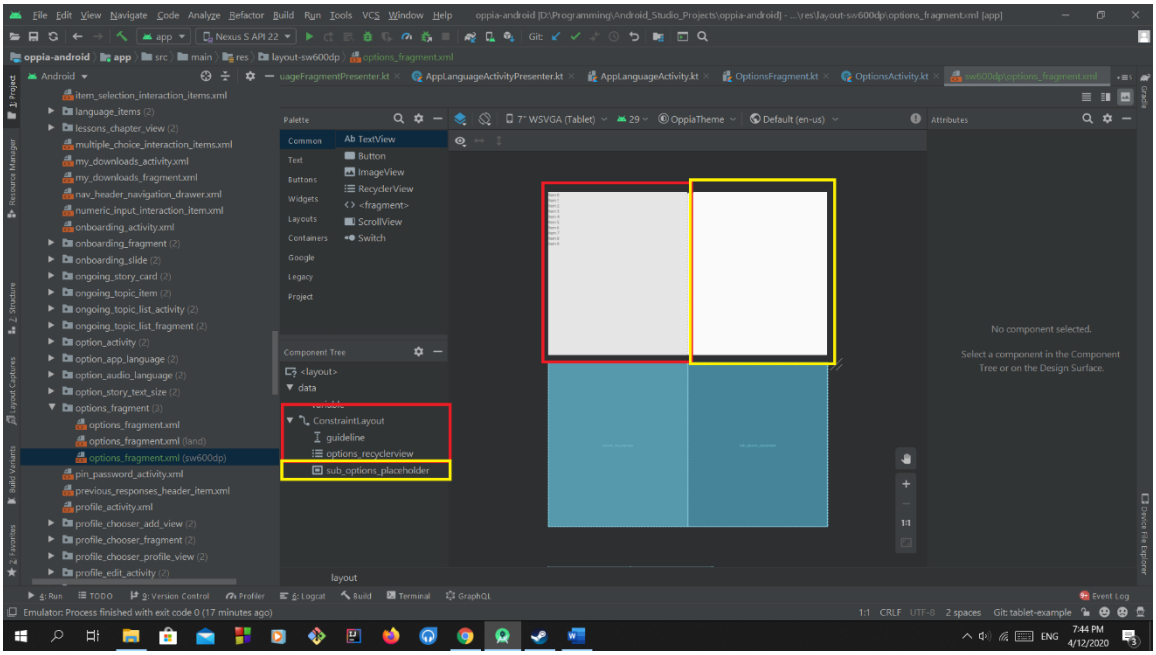




Code walkthrough:

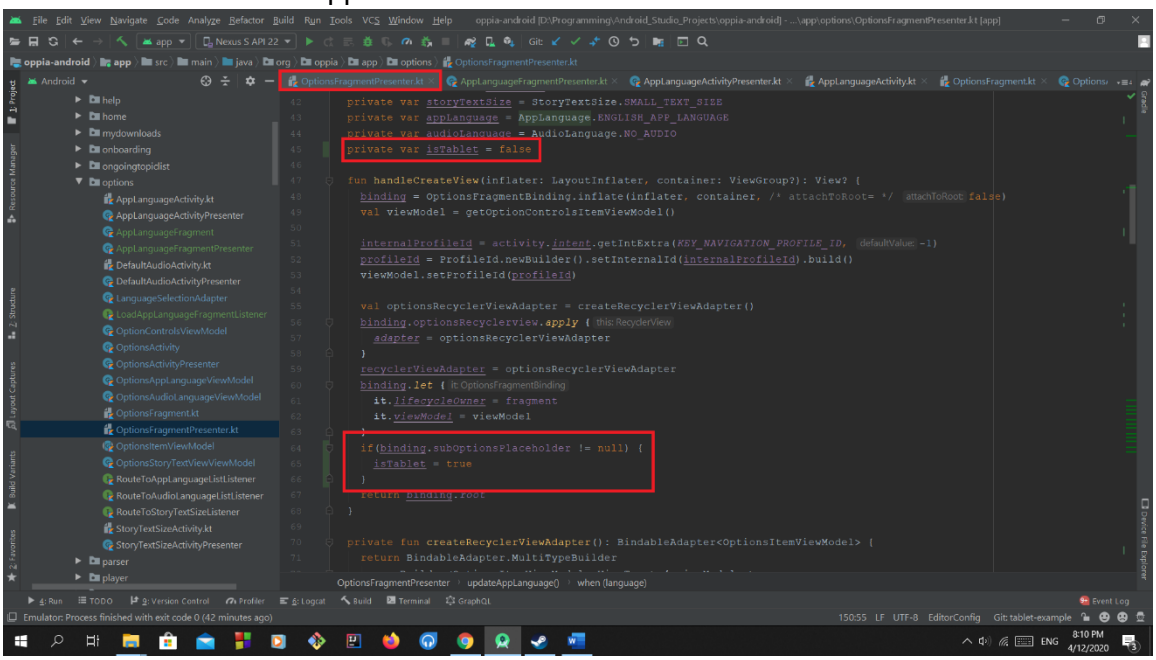
First, we need to create a new layout file for the tablet configuration by setting the smallest screen width to 600dp as shown in the picture:





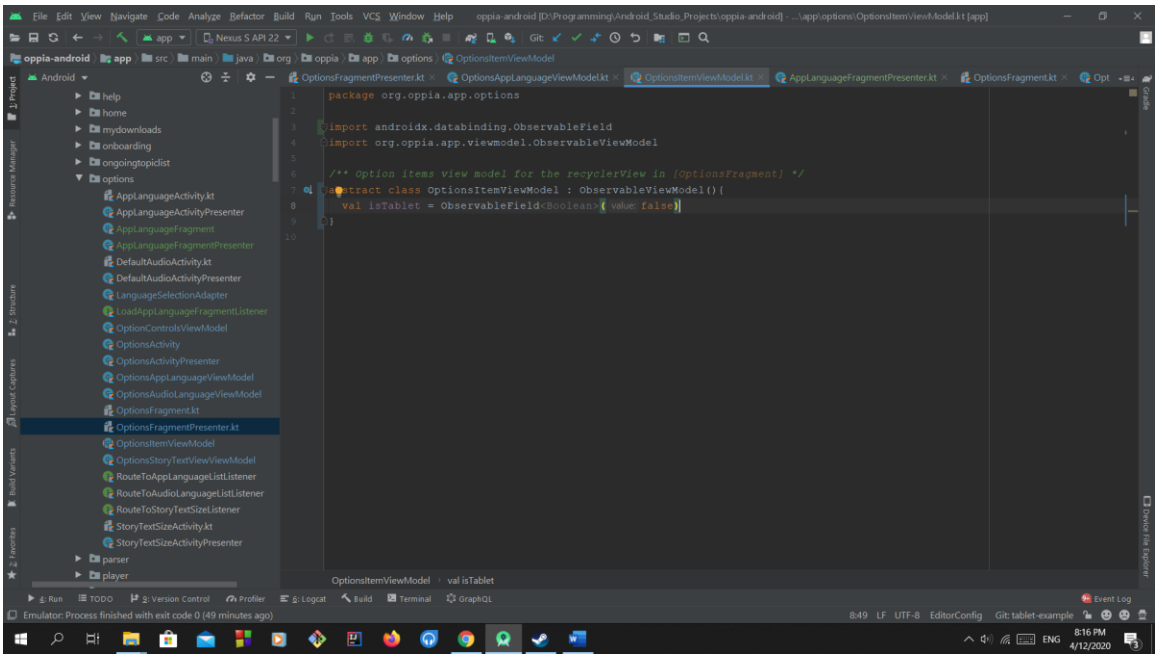
The red section in this new configuration is the same as the one in the normal phone layout.
The yellow section will contain the new change we want to make in case of tablet layout. It is simply a **FrameLayout** that will contain a new fragment.
The yellow section will be used in the kotlin code to determine whether the app is running on a tablet device or a normal device, if it exists, then the app is running on a tablet, else, the app is running on a normal device.

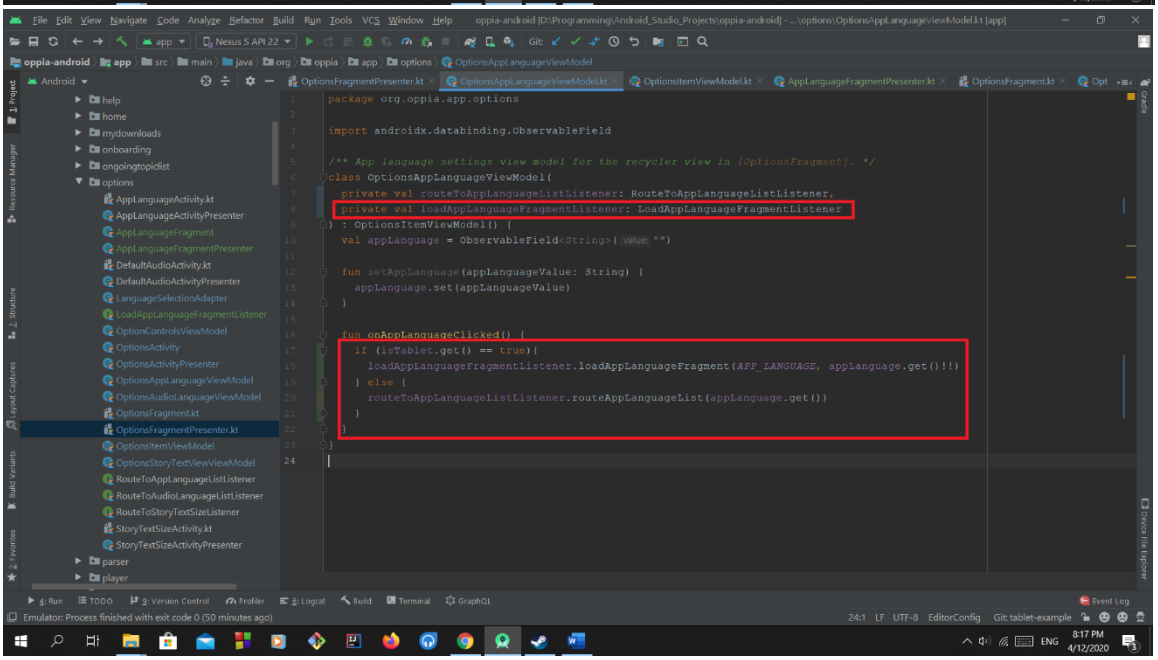
Here is the code of this approach:



Now after we have known that we are running on a tablet device, we don't want to start a new activity for the language options. Instead, we'll load a new fragment in the yellow section.
So, we'll modify the "OptionsAppLanguageViewModel" by adding a new variable that indicates whether we are on a tablet or a normal device. Then we'll use this variable to decide whether to start a new activity or to load a fragment in the yellow section.
But because this variable is not restricted to "OptionsAppLanguageViewModel", we'll create it in the parent ViewModel "OptionsItemViewModel".

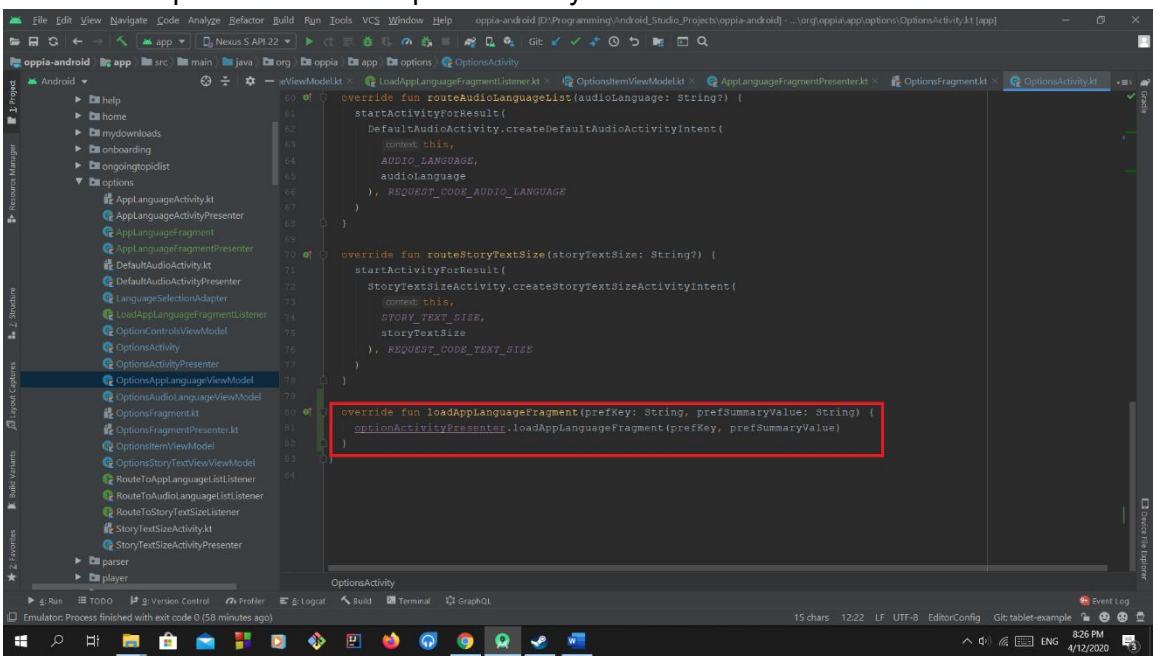
Here is the code:



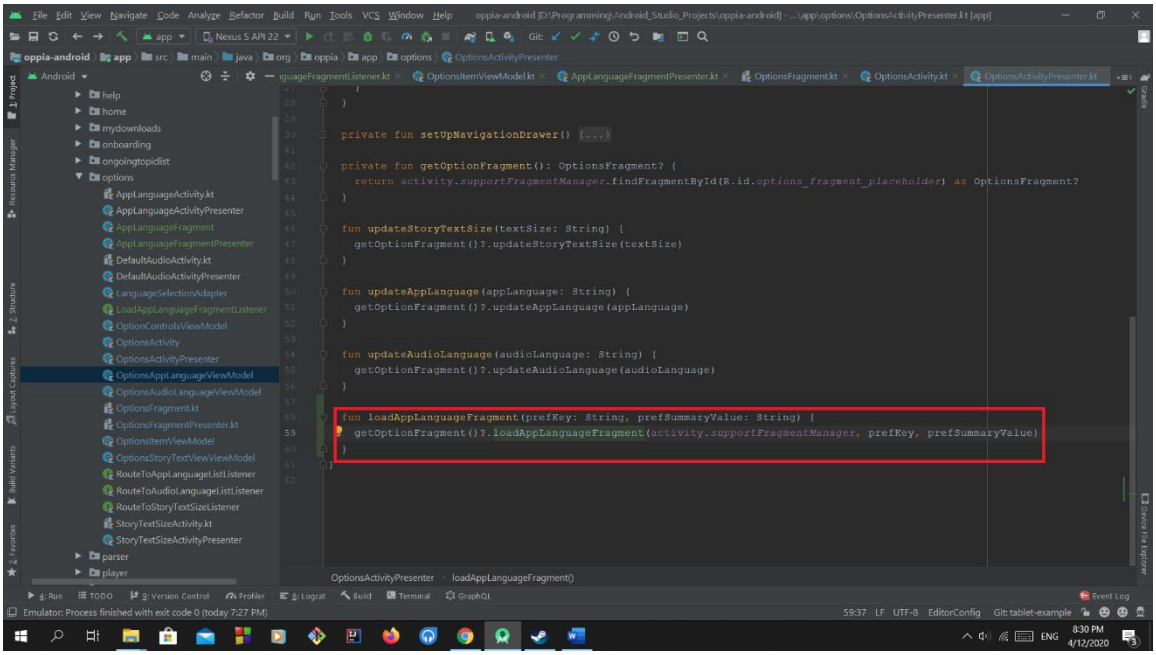


You can notice the new Interface “LoadAppLanguageFragmentListener” that will be implemented by the “OptionsActivity” to give it the ability to load the fragment in the yellow section.

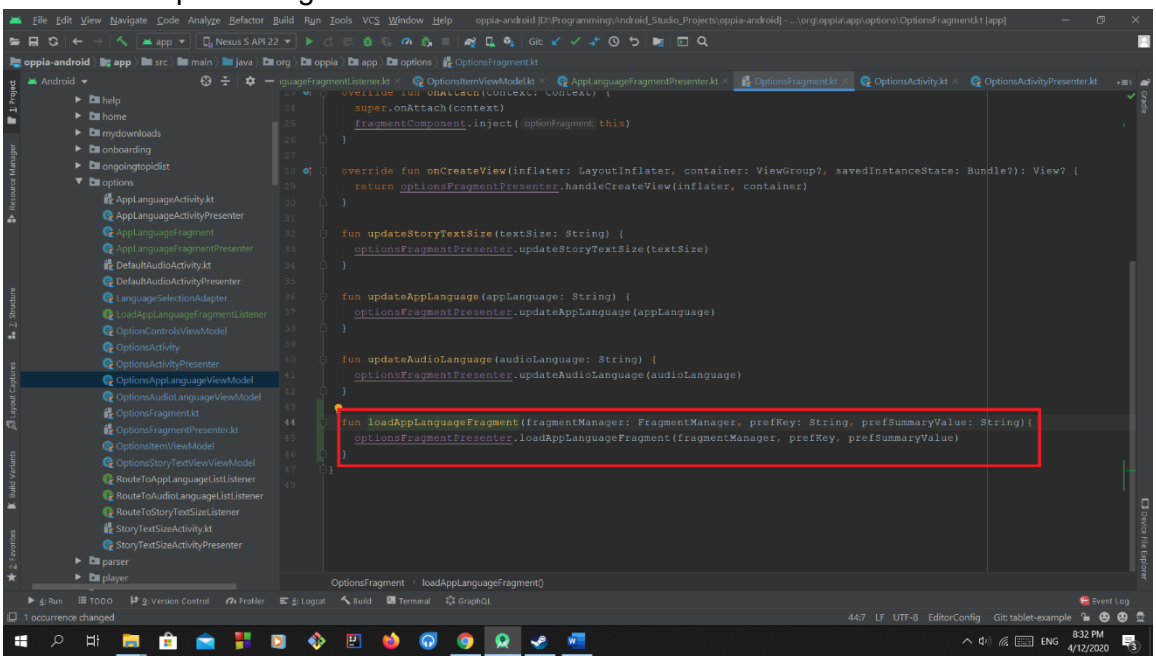
Here is its implementation in “OptionsActivity”:



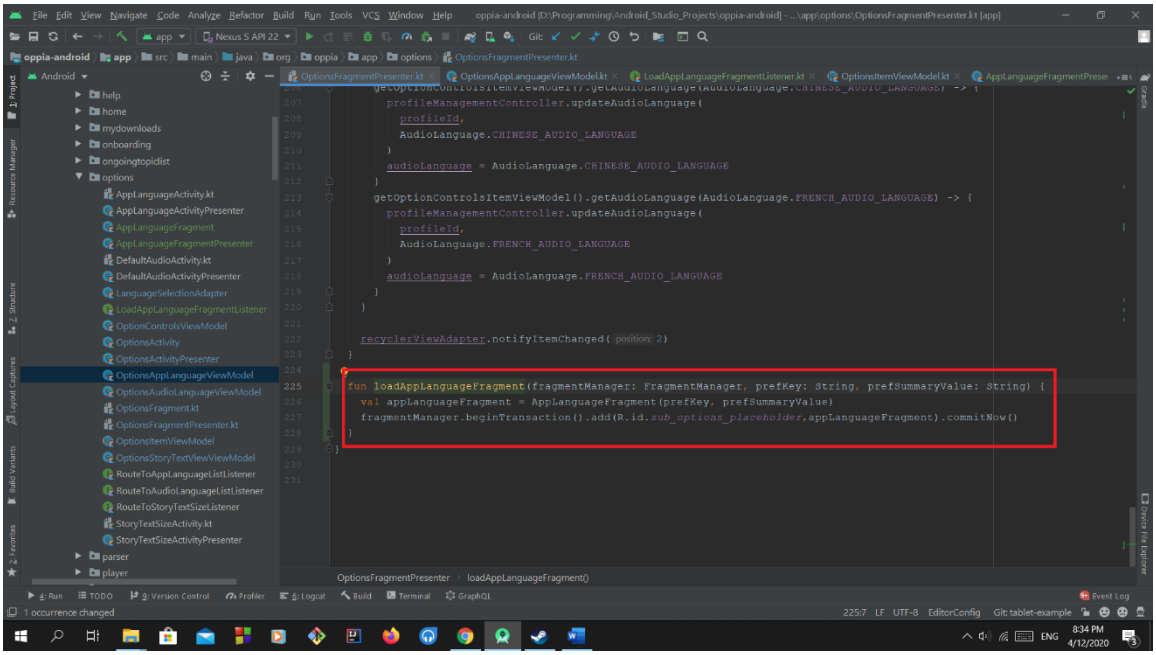
And in the “OptionsActivityPresenter”, this is the code of “loadAppLanguageFragment”:



And in the “OptionsFragment”:

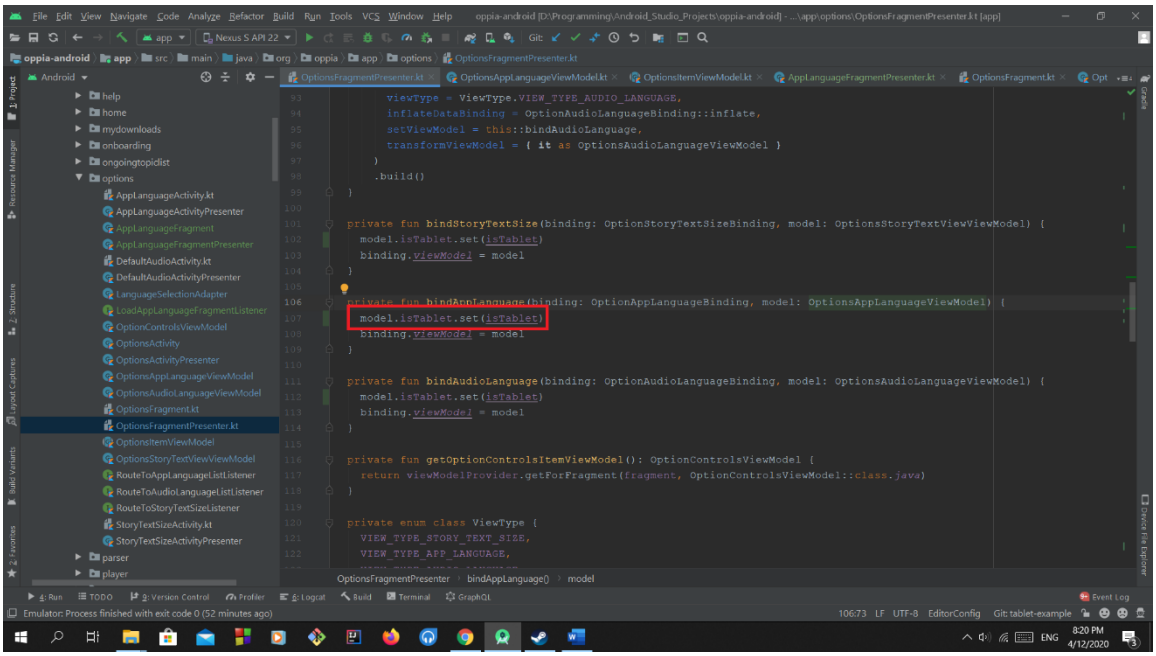


And finally, back again to the “OptionsFragmentPresenter”:

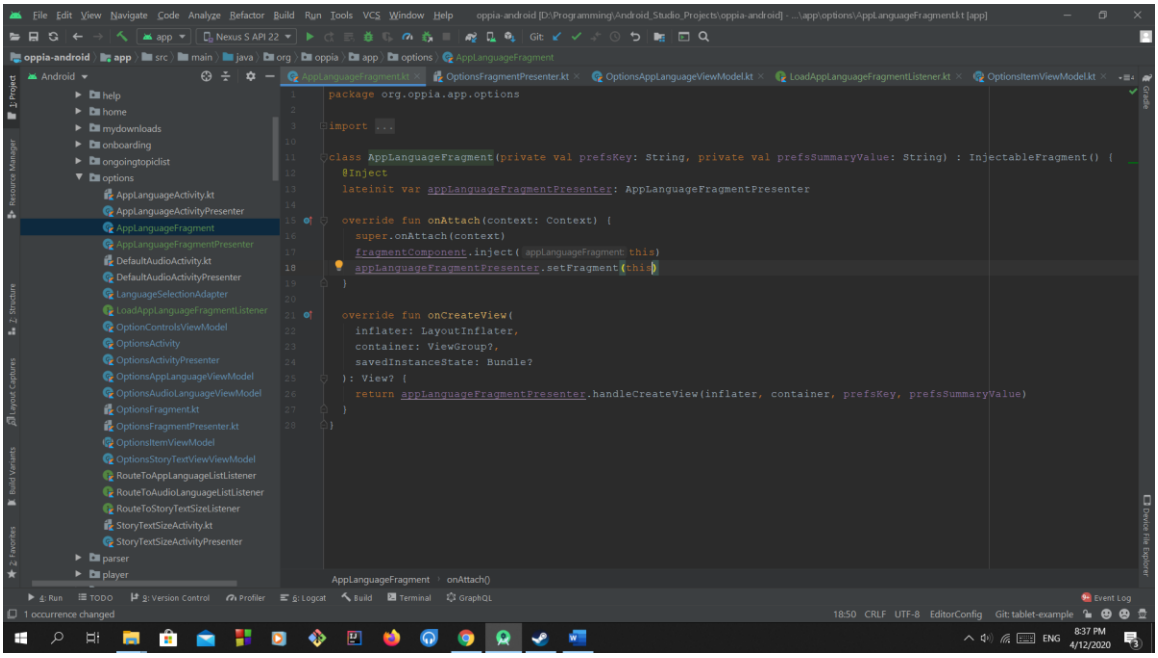


And we’ll modify the code to inform the ViewModel whether we are on a tablet or on a normal device.

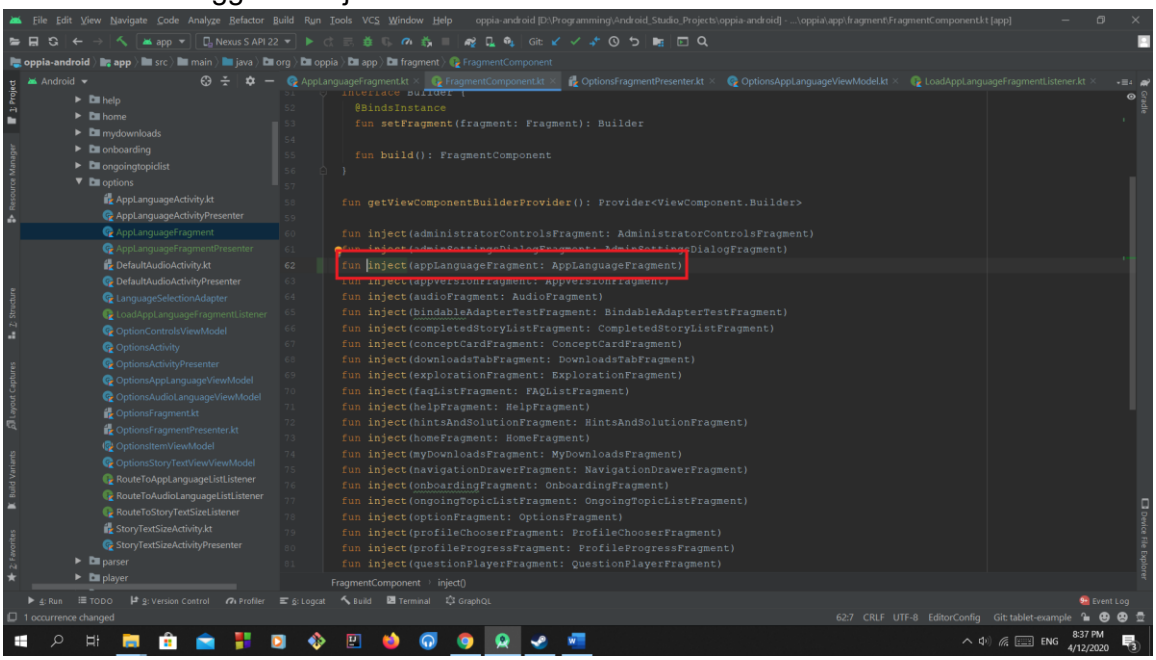
Here is the code:



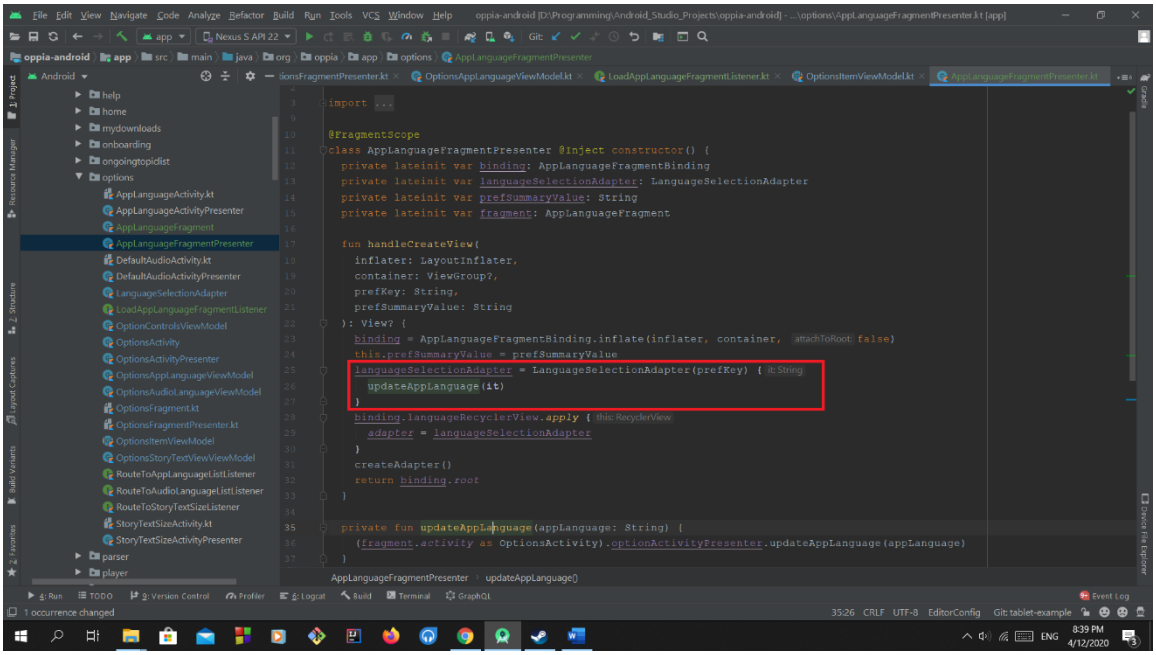
In this way, we have successfully loaded a new fragment instead of starting a new activity.
Now, let's check the code of the new fragment.
We've created a new fragment and a presenter for it.
Here is the code of the fragment:



And we told “Dagger” to inject it:

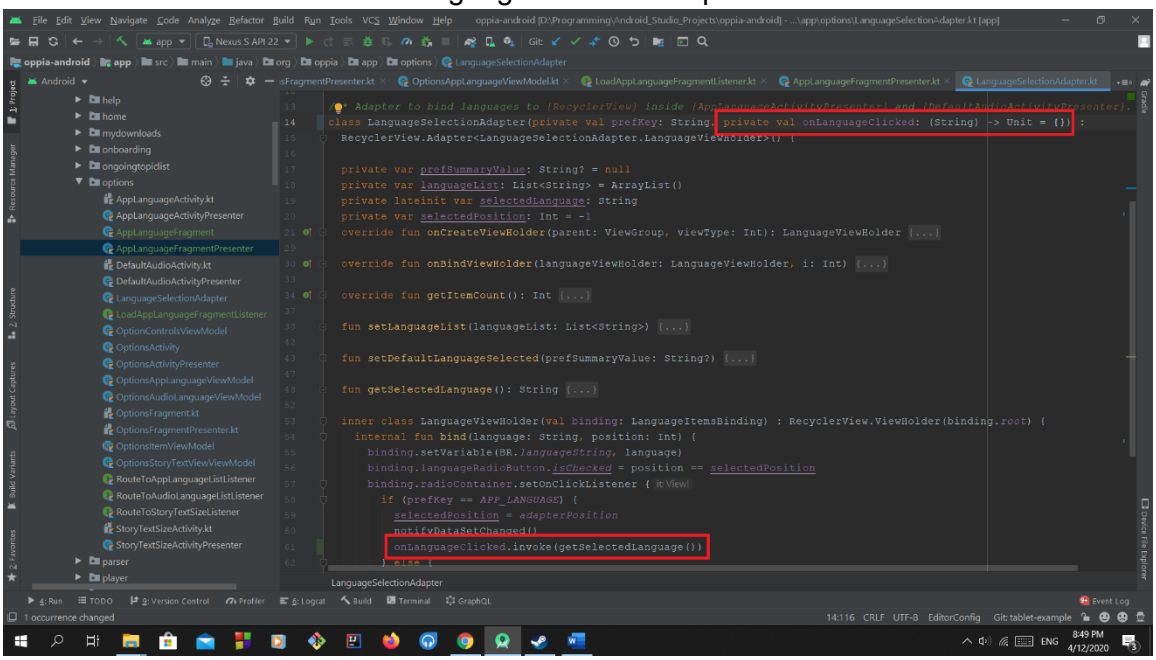


And here is its presenter:

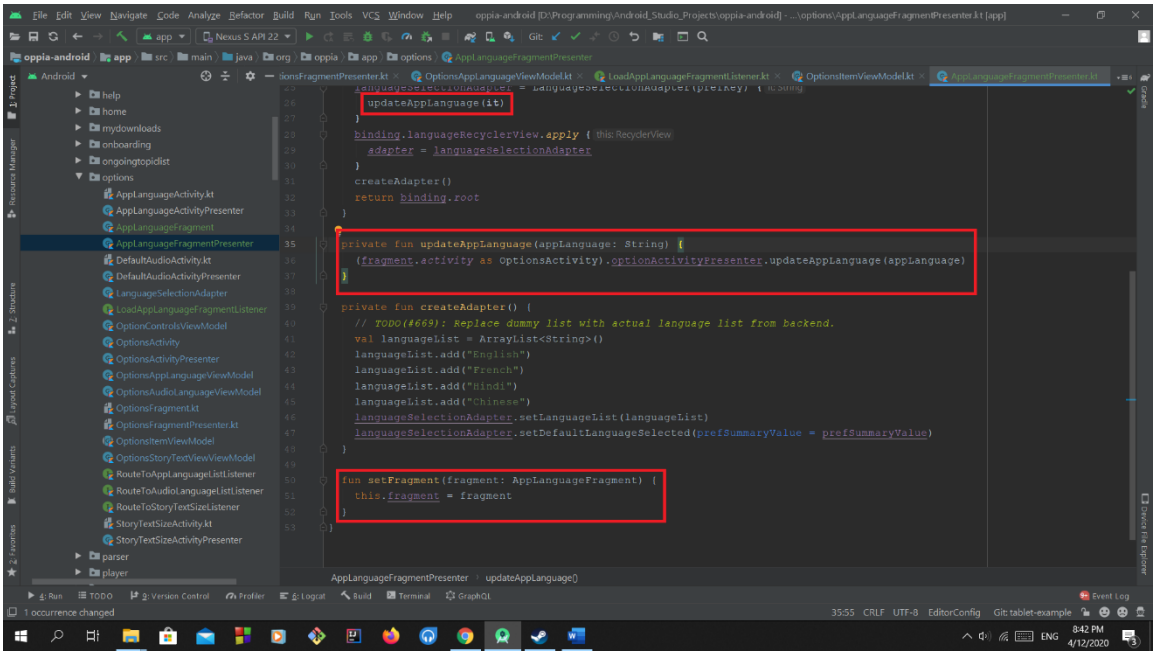


We can notice that we have modified a new lambda in the constructor of the “LanguageSelectionAdapter”. This lambda is used as a click listener in the case of loading a fragment in the yellow section. It replaces the “onActivityResult” function in case of starting a new activity.

This is the modification in the “LanguageSelectionAdapter”:



Here is the rest code of the Presenter:



We can notice the “setFragment” method which gives the presenter a reference to the “AppLanguageFragment”. We use this reference to refer to the “OptionsActivity” and its presenter in order to call the “updateAppLanguage” method which does the rest of the work.
In this way we have modified the code to work as expected in case of Tablet devices.
We didn’t change the implementation of how the language changes, we only plugged-in a new view and told the app how to deal with it.
Here is the source code of this implementation:
https://github.com/MohamedMedhat1998/oppia-android/tree/tablet-example

Testing Approach

The Facebook library for screenshot testing will be used through the following steps:

1. A code that generates the screenshots that will be used as a reference will be written
2. Run the Gradle command `record<App Variant>ScreenshotTest` which installs and runs screenshot tests, then records their output for later verification
3. Now after any code change, the Gradle command `verify<App Variant>ScreenshotTest` will be run to install and run screenshot tests, then verify their output against previously recorded screenshots to make sure that no regression occurred

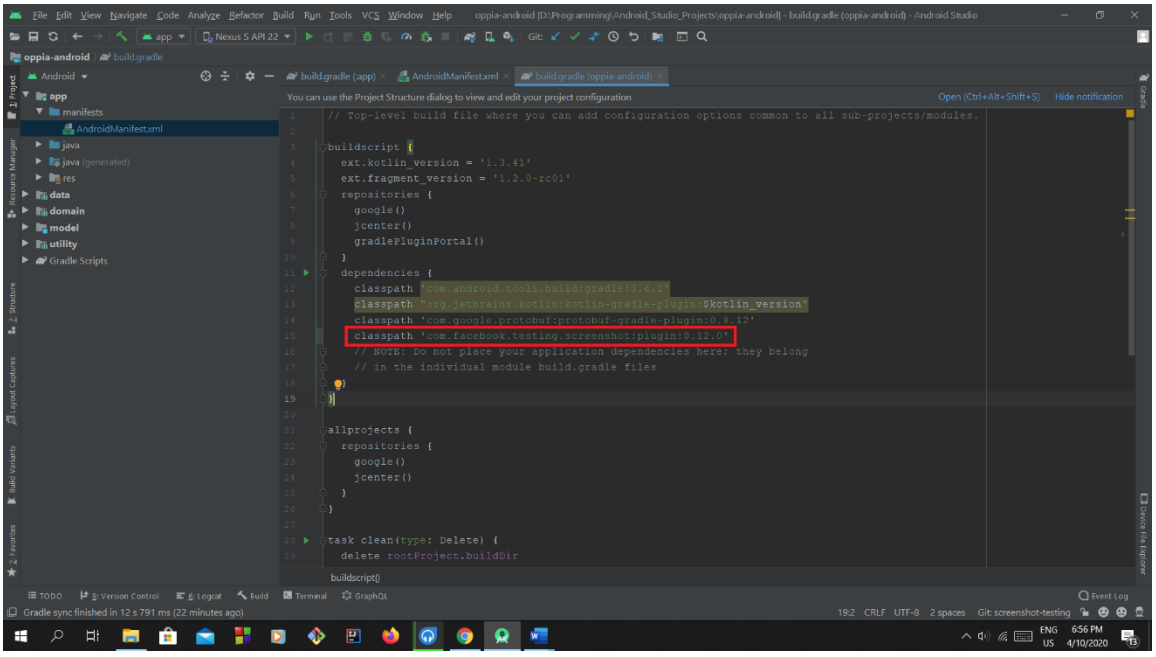
***Note:** in case that Facebook library didn't work as expected, another library will be used.

Why screenshot testing is important:

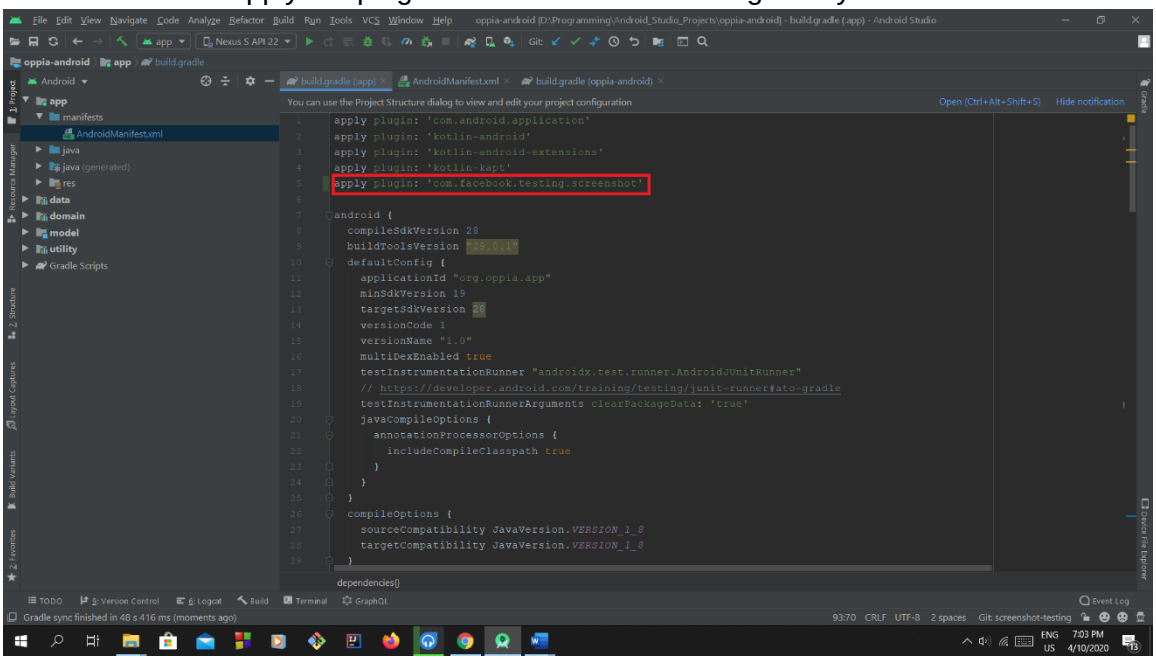
Screenshot testing prevents visual regressions in Android apps. It does that by generating reference screenshots -**from the actual xml code**- that describe how the app should look like. After that, when a developer tries to make a pull request, screenshot testing ensures that his/her pull request didn't affect the UI -**in case of non-UI related pull requests**- by generating new screenshots from the application after the code has changed and comparing them with the reference screenshots generated before.

Screenshot testing integrating:

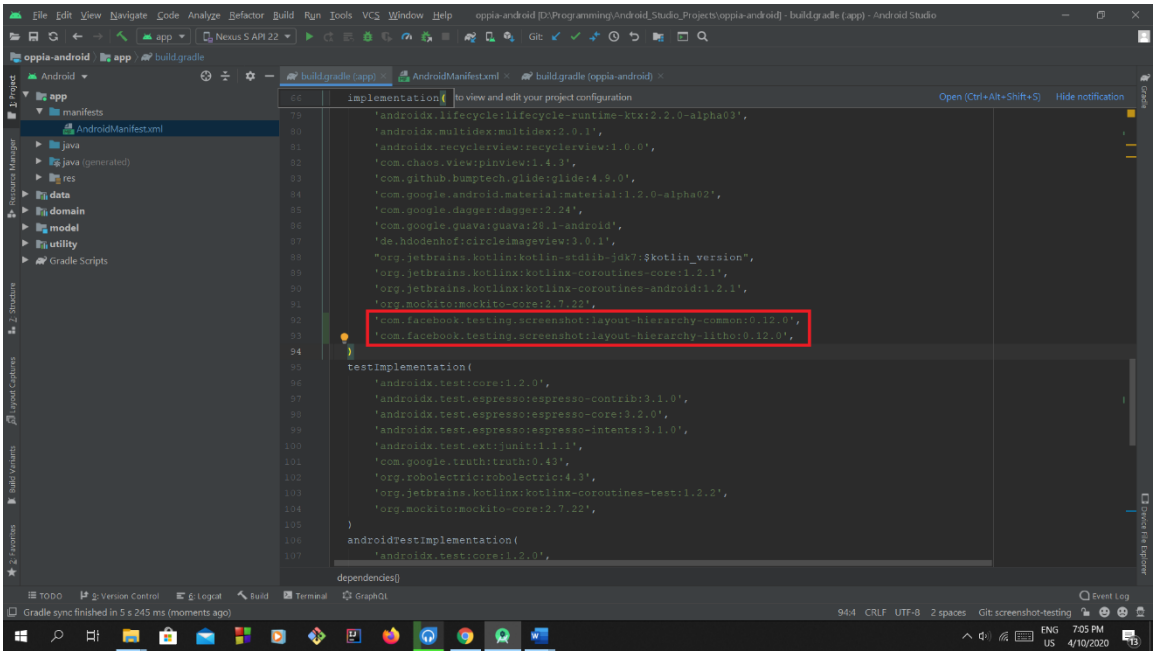
First, we need to add the classpath dependency to the project-level Gradle file.



Then we need to apply the plugin of the screenshot testing library.

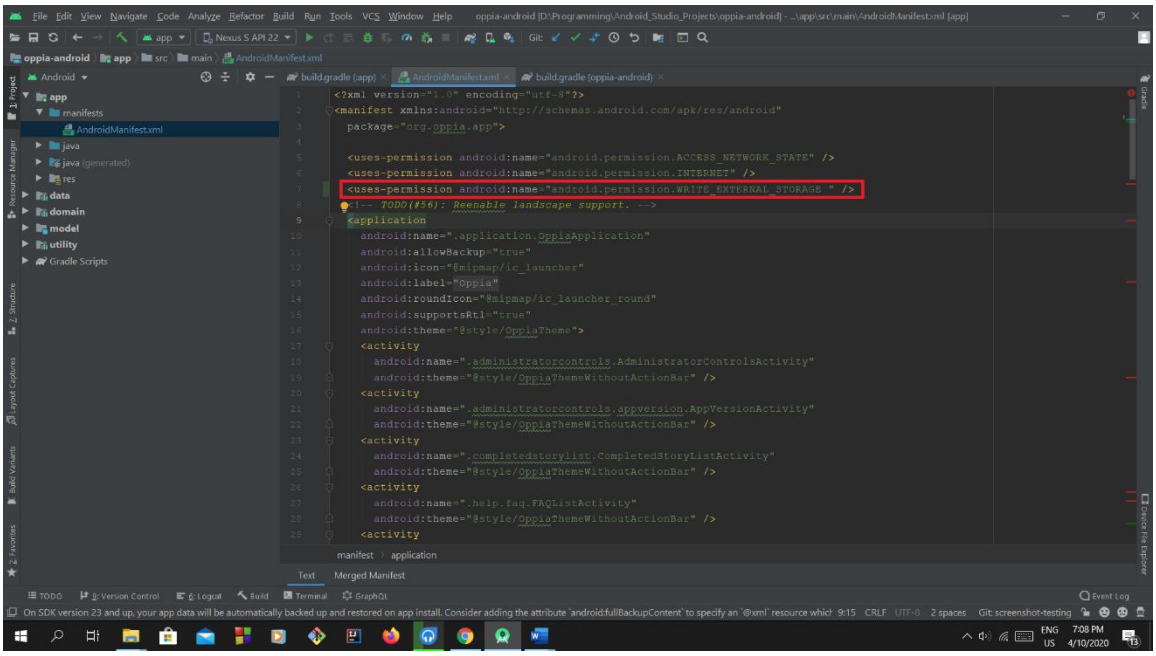


Then, we need to add the required dependency in the module-level Gradle file.

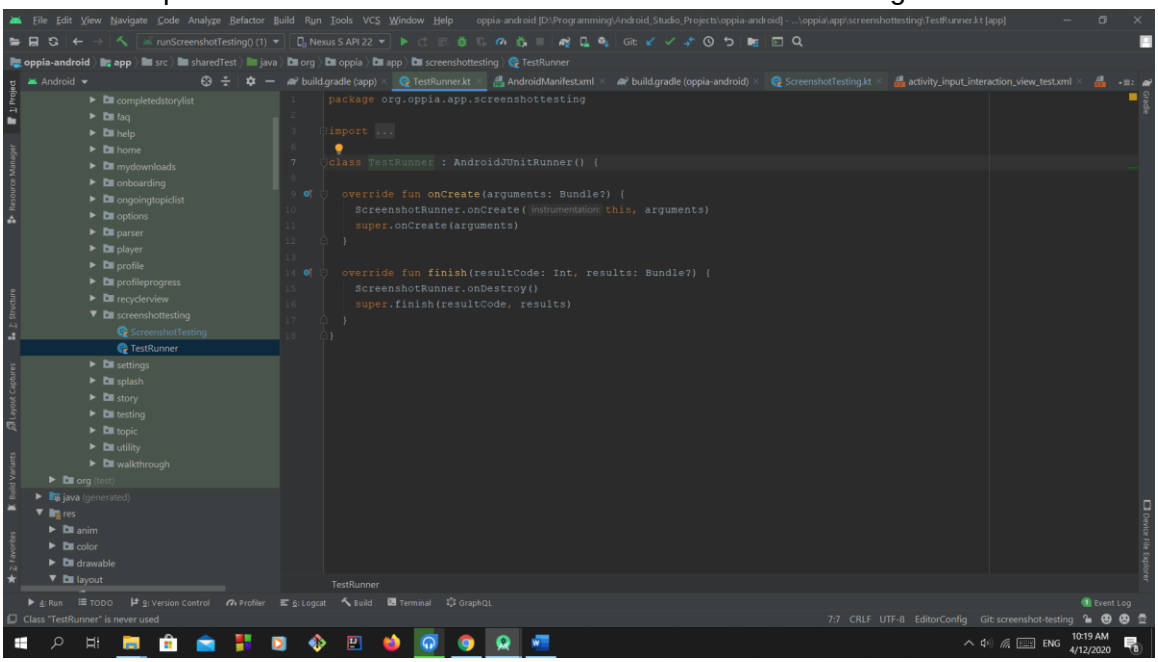


We sync the project with the Gradle file after each step of course.

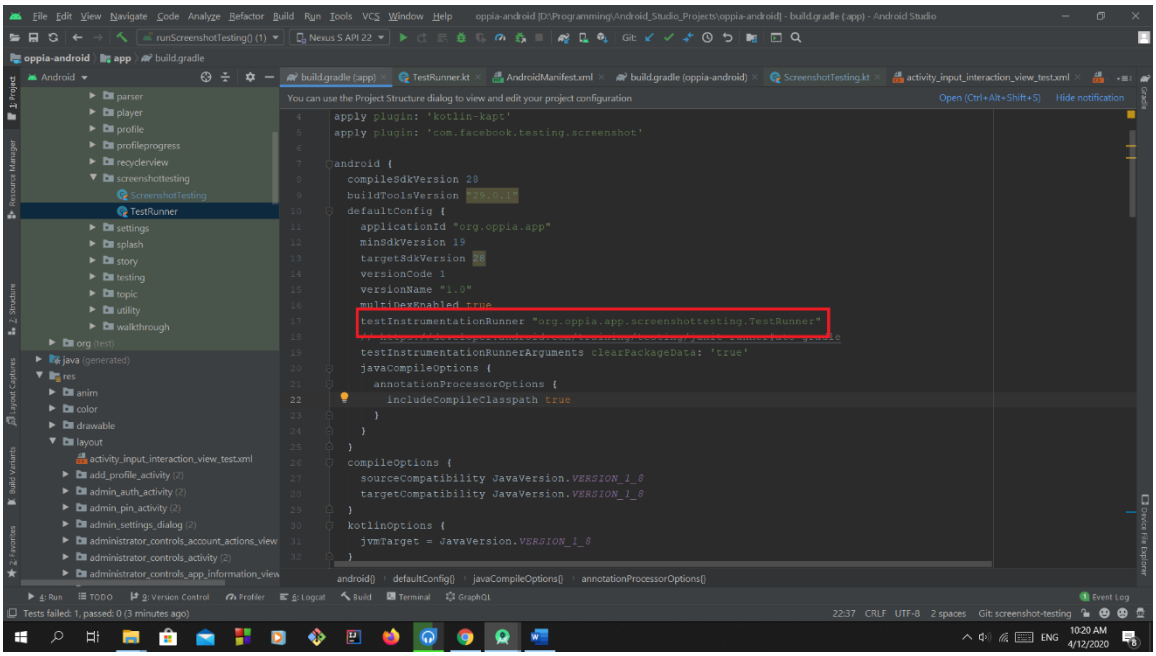
Screenshot testing requires the external storage permission, so we need to add it in our manifest file too.



The final step is to add a custom test runner and refer to in the bulid.gradle file



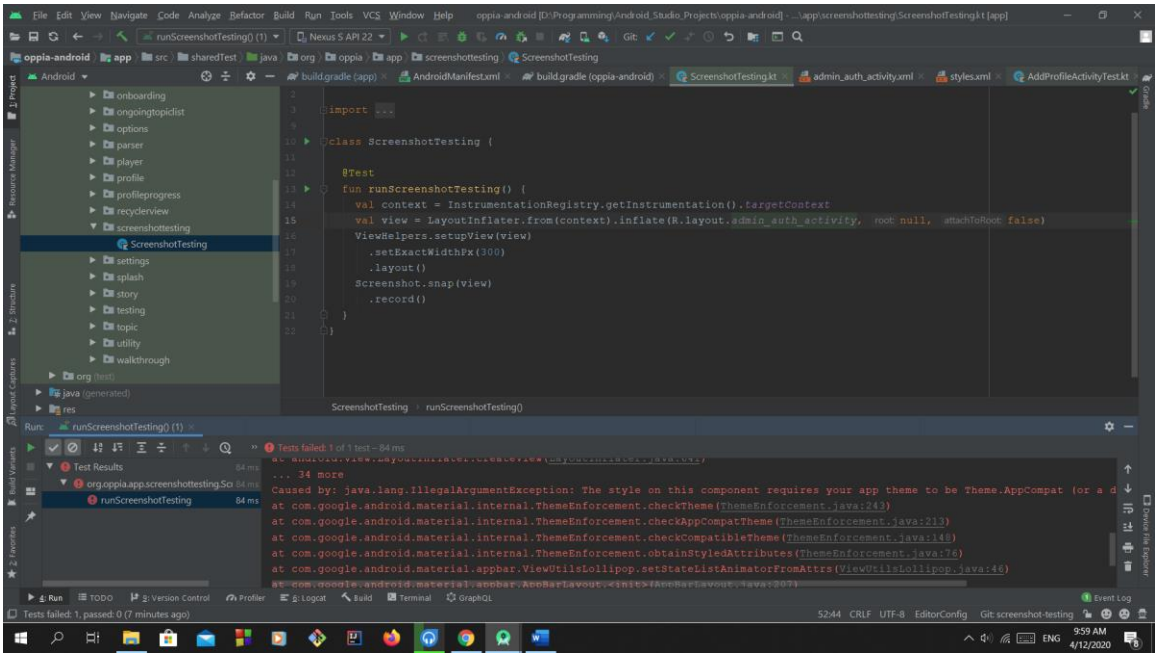
And this is how to refer to it in the Gradle file.



And sync again.

Now, we have integrated our project with screenshot testing library, the next step is to write the actual tests.

Screenshot testing example:



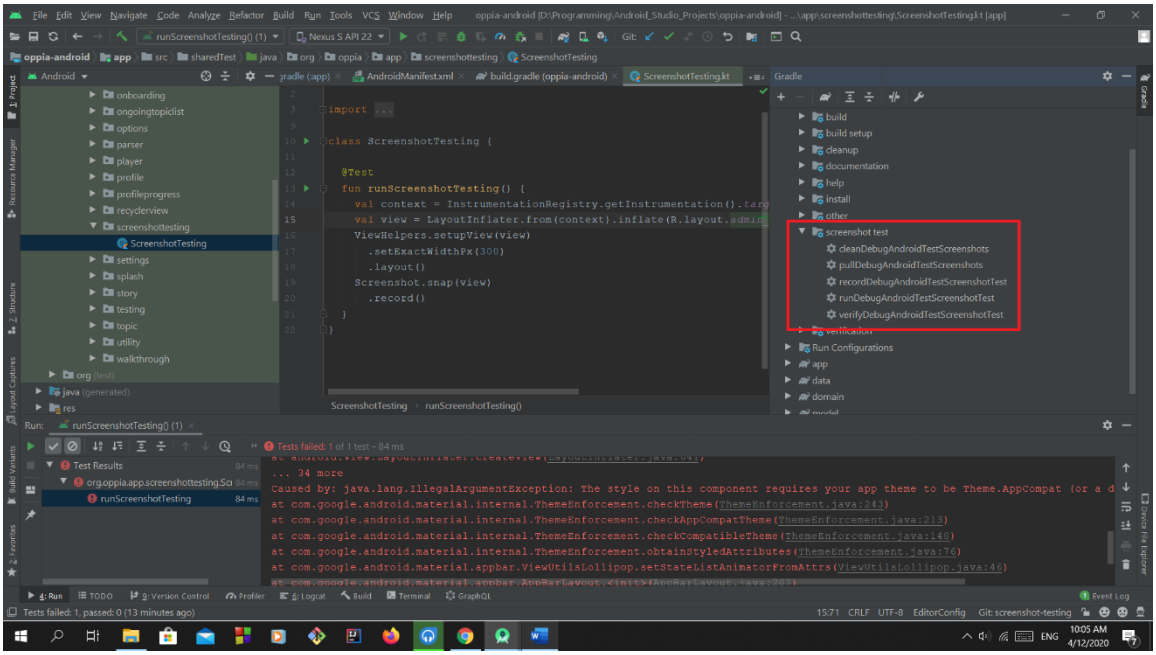
This is the code of screenshot testing. It enables us to record how the app will look like in the AdminAuthActivity.

Currently it faces an error because of the MaterialDesign library. I couldn't solve this error, I have been trying for a long time, so I will ask the mentors for support.

Error details: https://github.com/oppia/oppia-android/pull/981
But this is how the screenshot testing code will look like. I'll repeat this code for every activity, or even for every layout file.
There are 5 Gradle commands in the screenshot testing as illustrated in the docs, which are:

- Clean
- Pull
- Record
- Run
- Verify

They are shown in the next picture.



Clean: cleans last generated screenshot report
Pull: pulls screenshots from your device
Record: Installs and runs screenshot tests, then records their output for later verification
Run: Installs and runs screenshot tests, then generates a report
Verify: Installs and runs screenshot tests, then verifies their output against previously recorded screenshots

Screenshot testing usage:

Screenshot tests are used when a developer wants to submit a pull request. The screenshot-testing makes sure that the UI didn't change. If the developer made a code change that is not related to the UI, the UI should not change. That is what screenshot-testing verifies.
I think this test should be a part of the CircleCi tests, but I don't know how people will include it in CircleCi tests.

Milestones

Milestone 1

Key Objective: Review and provide suggestions for all tablet mocks. Implement low-fidelity & high-fidelity code for tablet UI in the Onboarding Flow, HomeFragment, ProfileChooser, NavigationDrawer, and RecentlyPlayedStoryList.

No.	Description of PR	Prereq PR numbers	Target date for PR submission	Target date for PR to be merged
1.1	Review the mocks and provide suggestions if there are any	-	1/6/2020	-
1.2	Onboarding Flow low-fi implementation	-	3/6/2020	6/6/2020
1.3	Onboarding Flow high-fi implementation	1.2	12/6/2020	16/6/2020
1.4	ProfileChooser low-fi implementation	-	3/6/2020	6/6/2020
1.5	ProfileChooser high-fi implementation	1.4	12/6/2020	16/6/2020
1.6	NavigationDrawer low-fi implementation	-	5/6/2020	8/6/2020
1.7	NavigationDrawer high-fi implementation	1.6	14/6/2020	18/6/2020
1.8	RecentlyPlayedStoryList low-fi implementation	-	5/6/2020	8/6/2020
1.9	RecentlyPlayedStoryList high-fi implementation	1.8	14/6/2020	18/6/2020
1.10	HomeFragment low-fi implementation	-	7/6/2020	10/6/2020
1.11	HomeFragment high-fi implementation	1.10	14/6/2020	18/6/2020

Milestone 2

Key Objective: Implement low-fidelity & high-fidelity code for tablet UI in following screens: Topic including all Tabs, ExplorationPlayer, Hints & Solution, QuestionPlayer and Concept Card.

No.	Description of PR	Prereq PR numbers	Target date for PR submission	Target date for PR to be merged
2.1	Topic-info low-fi implementation	-	23/6/2020	27/6/2020
2.2	Topic-info high-fi implementation	2.1	13/7/2020	17/7/2020
2.3	Topic-lessons low-fi implementation	-	23/6/2020	27/6/2020
2.4	Topic-lessons high-fi implementation	2.3	13/7/2020	17/7/2020
2.5	Topic-practice low-fi implementation	-	23/6/2020	27/6/2020
2.6	Topic-practice high-fi implementation	2.5	13/7/2020	17/7/2020
2.7	Topic-revision low-fi implementation	-	23/6/2020	27/6/2020
2.8	Topic-revision high-fi implementation	2.7	13/7/2020	17/7/2020
2.9	ExplorationPlayer low-fi implementation	-	27/6/2020	1/7/2020
2.10	ExplorationPlayer high-fi implementation	2.9	5/7/2020	9/7/2020
2.11	QuestionPlayer low-fi implementation	-	1/7/2020	5/7/2020
2.12	QuestionPlayer high-fi implementation	2.11	9/7/2020	13/7/2020
2.13	Hints & Solution low-fi implementation	-	5/7/2020	9/7/2020
2.14	Hints & Solution high-fi implementation	2.13	15/7/2020	19/7/2020
2.15	Concept Card low-fi implementation	-	5/7/2020	9/7/2020
2.16	Concept Card high-fi implementation	2.15	15/7/2020	19/7/2020

Milestone 3

Key Objective: Implement low-fidelity & high-fidelity code for tablet UI in following screens: Admin Controls, ProfileProgressActivity, OngoingTopicList, CompletedStoryList, Options and its 3 child screens (Audio Selection, Language Selection and StoryTextSize Selection). Introduce screen-diff/screenshot testing to the application and write test cases to cover at least 1 screen for mobile+tablet UI.

No.	Description of PR	Prereq PR numbers	Target date for PR submission	Target date for PR to be merged
3.1	Admin Controls low-fi implementation	-	26/7/2020	30/7/2020
3.2	Admin Controls high-fi implementation	3.1	8/8/2020	12/8/2020
3.3	Options low-fi implementation	-	26/7/2020	30/7/2020
3.4	Options high-fi implementation	3.3	8/8/2020	12/8/2020
3.5	ProfileProgressActivity low-fi implementation	-	26/7/2020	30/7/2020
3.6	ProfileProgressActivity high-fi implementation	3.5	8/8/2020	12/8/2020
3.7	OngoingTopicList low-fi implementation	-	30/7/2020	3/8/2020
3.8	OngoingTopicList high-fi implementation	3.7	9/8/2020	13/8/2020
3.9	CompletedStoryList low-fi implementation	-	30/7/2020	3/8/2020
3.10	CompletedStoryList high-fi implementation	3.9	9/8/2020	13/8/2020
3.11	Screenshot testing integration	-	16/8/2020	19/8/2020
3.12	Test cases code	3.11	21/8/2020	23/8/2020