ESAT INNOVATION SCHOOL | ESCUELA SUPERIOR DE ARTE Y TECNOLOGÍA

ANDROID STUDIO APP

Subject Name: Mobile Programming
Associated Module: UNIT
Professor: Rubén Blanco
Course: 2022/2023
Author/s: Ángel Artigas Pérez

Índice/Index

1.- App Project

1.1 Introduction and Technical Demo

A companion app for the Souls Saga has been developed in this project. This app shows stats and a peak at the lore of some bosses found in Demon's Souls and Dark Souls. The information is gathered from an API that presents a JSON file which content is parsed into the app.

For the development of the app the platform Android Studio Dolphin has been used, version 2021.3.1 Patch 1. The chosen coding language was Kotlin.

1.2 Development

1.2.1 Features

The features that can be found in the application are the following:

- User feedback
- Login screen with validation and persistence of user credentials
- Information gathering from an API
- Data parsing from a JSON file
- Usage of fragments and a fragment list
- Values transition between different activities
- Loading of images from URLs
- Connection of the project with FireBase

User feedback has been reached by the usage of toasts and alert dialogs and it will be explained in more detail in section 2.2 User-App Behaviour.

The activity dedicated to the login of the user presents two text fields, one for the user and one for the password, and a checkbox to mark whether the user credentials should be stored for future sessions. This has been accomplished by the usage of SharedPreferences which will save the values of user and password locally in the user's device.

All the information about the bosses of the souls saga that appears in the app is collected from an external mock API built at Beeceptor. The requests to this API have been made possible thanks to a library called Volley which allows to launch threads outside the main one to avoid the blocking of the application.

The API presents different endpoints. To decide which one should be called there is an activity prior to where the request is made in which the user chooses the game he wants to see the bosses of. This activity passes

the player's choice to the next one thanks to the use of "extra" of the intents. The data stored in this extra will allow us to call the corresponding endpoint.

Every endpoint contains a list of bosses and its information in the form of a JSON file. File's structure has been created from scratch and validated using JSONLint. To parse the content of the JSON received from the request into a data class called BossEntity GSON, a library created by Google, has been used.

Once the information of the bosses is parsed, we proceed to load in a fragment list several fragments with the name of those bosses. Each one has the same event associated with it when it is clicked that causes the next activity to be loaded with the information of the corresponding BossEntity. The data of the boss is passed through a "Bundle" added to the intent.

The BossInfoActivity gets the information from the Bundle and displays it in its Views. This activity features an ImageView which image is loaded from a variable of the BossEntity which is a String corresponding to an URL. This has been achieved thanks to the usage of the library Glide.

Last but not least, the project has been connected to Firebase, a development platform that provides the developer with a large number of utilities. In my case I have implemented one called Crashlytics, a very helpful tool to track down what is causing the application to malfunction during development stages and even once released.

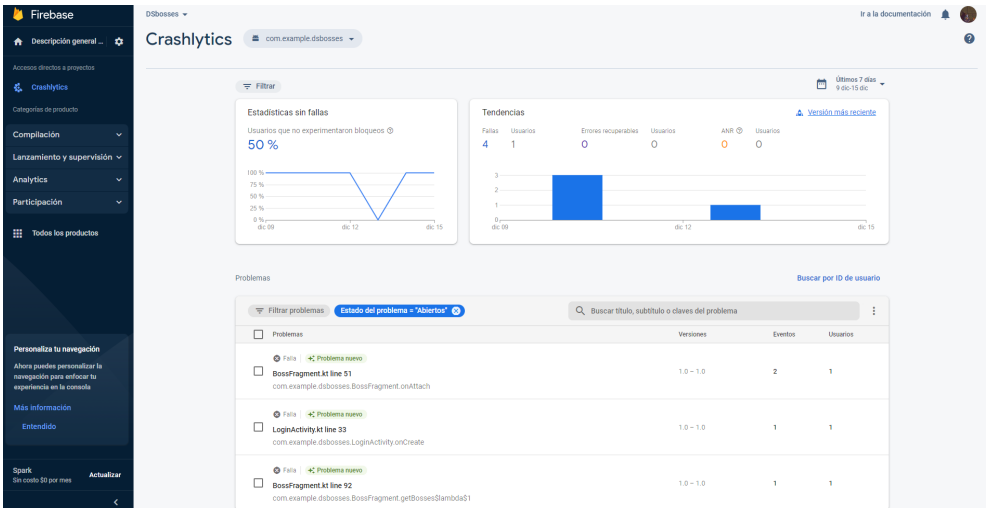

Fig.01 Crashlytics panel

1.2.2 Design patterns

In the industry there are different design patterns that have been conceived with the aim of tackling the problems that developers have to face over and over again. One of the most popular is MVC. **[1]**

Model View Controller design pattern establishes that an application must present three separated objects in order to follow the pattern. These objects are the following: **[2]**

- The Model does not include any logic describing how to present the data to a user; it only contains the pure application data.
- The View shows the user the data from the model. However the view is aware of how to get at the data in the model, but it is unaware of what this data means or how the user might be able to alter it.
- The Controller that exists as a link between the view and the model. It listens to events triggered by the view and responds to them appropriately. The response is typically to call a method on the model. The outcome of this action is then automatically reflected in the view because the view and the model are linked through a notification mechanism.

The project does not implement any design pattern due to the fact that it has not been the main focus of the subject.

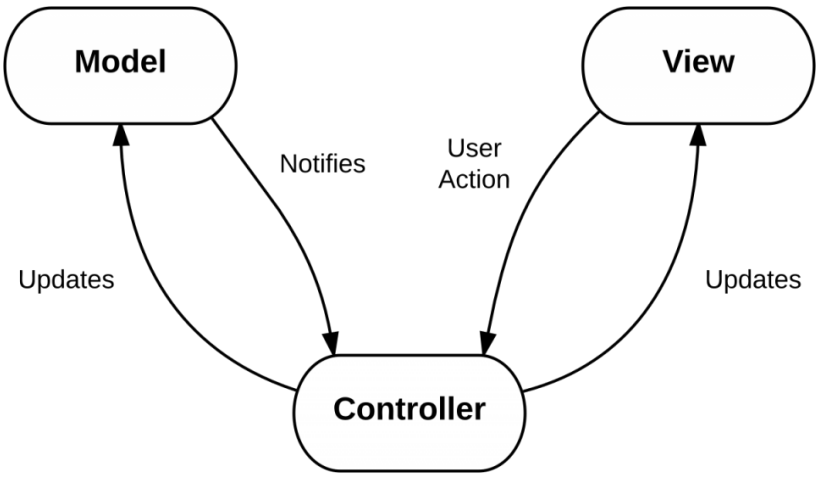

Fig.02 MVC Scheme [3]

Instead, it has been sought to generate a clean and reusable code. To achieve this it has been deleted any unnecessary comments or code never used. Also, the variables and methods are named with their first letter

being lowercase and in the case of a name with multiple words adding a capital letter to the first letter of every contiguous word.

In addition, two object classes have been created and can be accessed from any class in the project. One of them is called "Constants" and it contains the constant variables of the project. The other one is called "Utils" and it presents multiple functions that are called many times throughout the program execution.

1.2.3 Tests

No tests have been used during the development of the app, but below I will shortly explain its types and how these tests are implemented:

The usage of tests is a fundamental part of the development process of a project in any of its stages. Every project should be Test-Driven nowadays. These tests make it possible to find malfunctions of all kinds and in an automated-not tedious way.

These tests can be classified according to their scope: **[4]**

- Lower-level tests or Unit tests are tests that verify the app behaviour one class at a time.
- Intermediate level tests or Integration tests are tests that validate interactions between stack levels within a module or interactions between related modules.
- Top-level tests or UI tests are end-to-end tests that validate the path of users in various modules of the app.

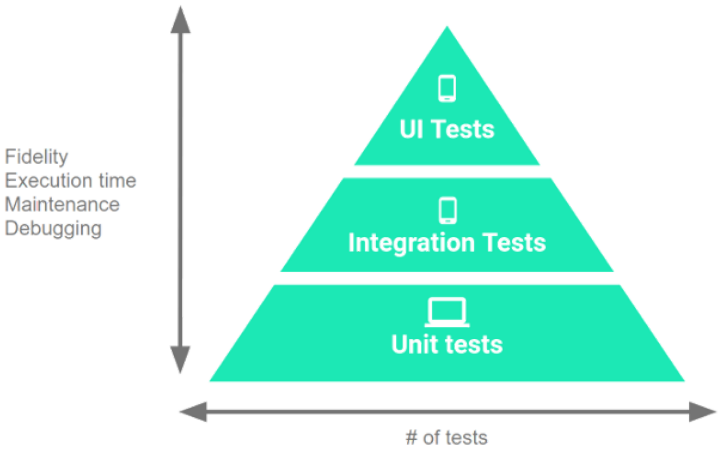

Fig.03 Tests Pyramid [4]

As stated in the Android developers documentation we have to be aware of the next facts to accomplish the implementation of our first test:

By default, the local test are stored at this path of the project: “module-name/src/test/”

Before the addition of any test to the project we have to add the corresponding dependencies:

```
dependencies {
  // Required -- JUnit 4 framework
  testImplementation "junit:junit:$jUnitVersion"
  // Optional -- Robolectric environment
  testImplementation "androidx.test:core:$androidXTestVersion"
  // Optional -- Mockito framework
  testImplementation "org.mockito:mockito-core:$mockitoVersion"
  // Optional -- mockito-kotlin
  testImplementation "org.mockito.kotlin:mockito-kotlin:$mockitoKotlinVersion"
  // Optional -- Mockk framework
  testImplementation "io.mockk:mockk:$mockkVersion"
}
```

Fig.04 Tests gradle dependencies[5]

Lastly the easiest way to add a test at any class or method of our project is following the next steps: **[6]**

1. Open the source file containing the code you want to test.
2. Place the cursor on the name of the class or method you want to test and press Control + Shift + T (Command + Shift + T on macOS).
3. In the menu that appears, click Create New Test.
4. In the Create Test dialog, choose JUnit4, edit the fields and methods you want to generate, and then click OK.
5. In the Choose Destination Directory dialog, click the source set corresponding to the type of test you want to create: androidTest for an instrumentation test or test for a local unit test. Then click OK.

1.3 Structure and Class Diagram

The project currently presents:

- 5 activities (classes that inherit from AppCompatActivity())
- 2 object classes that are used throughout the entire code,
- 1 data class to store the data parsed from the API
- 2 classes that have been necessary to make use of the fragmentList.

In the following diagram the disposition of the classes is shown:

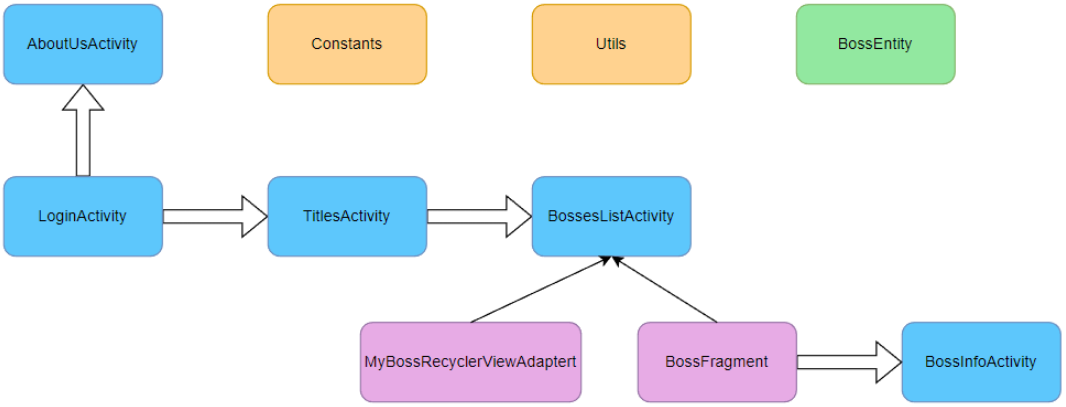

Fig.05 Class Diagram

2.- UI/UX Flow Chart

2.1 Design and Aesthetic Improvements

Figma, an interface design tool, was used for the initial design of the application. This is the scheme used as the reference blueprint:

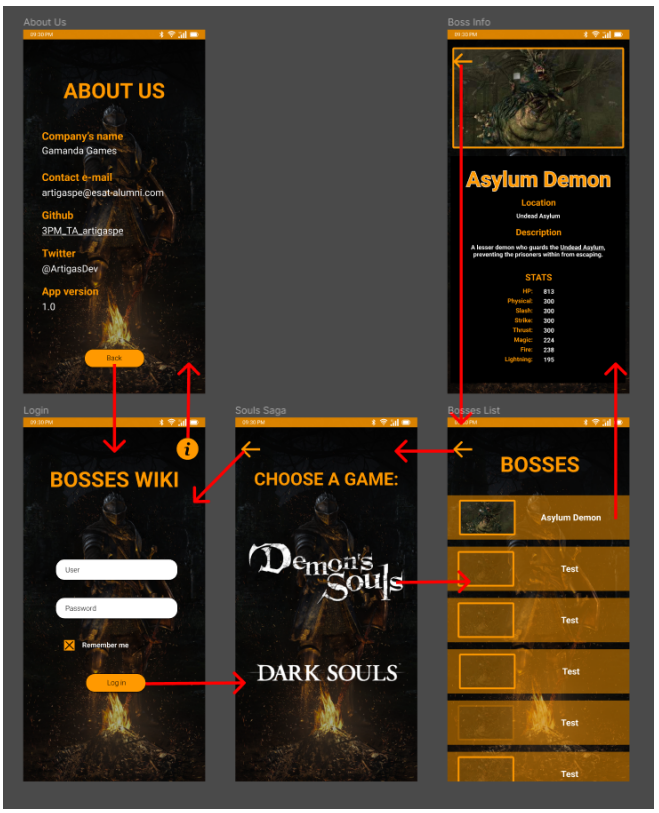

Fig.06 Figma design and flow

A custom theme has been created for the app at the "theme.xml":

```
<style name="Theme.DSbosses_Orange" parent="Theme.MaterialComponents.DayNight.NoActionBar">
    <!-- Primary brand color. -->
    <item name="colorPrimary">@color/dark_tangerine</item>
    <item name="colorPrimaryVariant">@color/tangerine</item>
    <item name="colorOnPrimary">@color/white</item>
    <!-- Secondary brand color. -->
    <item name="colorSecondary">@color/blizzard_blue</item>
    <item name="colorSecondaryVariant">@color/boston_blue</item>
    <item name="colorOnSecondary">@color/black</item>
    <!-- Status bar color. -->
    <item name="android:statusBarColor">@color/tangerine</item>
    <!-- Additional customization. -->
    <item name="android:colorBackground">@color/black</item>
    <item name="android:textColor">@color/white</item>
    <item name="android:textColorHint">@color/orange_weak</item>
    <item name="android:fontFamily">@font/ebgaramond_regular</item>
    <item name="android:windowBackground">@drawable/ds_wallpaper_pixel_3a</item>
</style>
```

Fig.07 Custom style

Also various colours has been added to the “colors.xml”:

```
<!-- Customized colors -->
<color name="dark_tangerine">#F7AD19</color>
<color name="tangerine">#F27F0C</color>
<color name="orange_dark">#E14E00</color>
<color name="orange_weak">#CCB07C</color>
<color name="blizzard_blue">#9FE7F5</color>
<color name="boston_blue">#429EBD</color>
<color name="prussian_blue">#053F5C</color>
```

Fig.08 Custom colours

The app icon has been modified too:

Fig.09 App Icon

2.2 View Flow

The app starts with the LoginActivity

Fig.10 LoginActivity

From here if the user clicks on the info button on the top-right corner of the screen the AboutUsActivity will be loaded

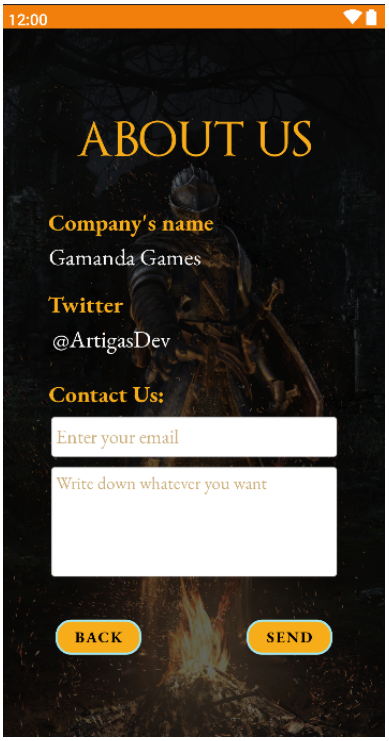

Fig.11 AboutUsActivity

On the other hand if the user enter their credentials correctly and click on the Login button the TitlesActivity will be loaded

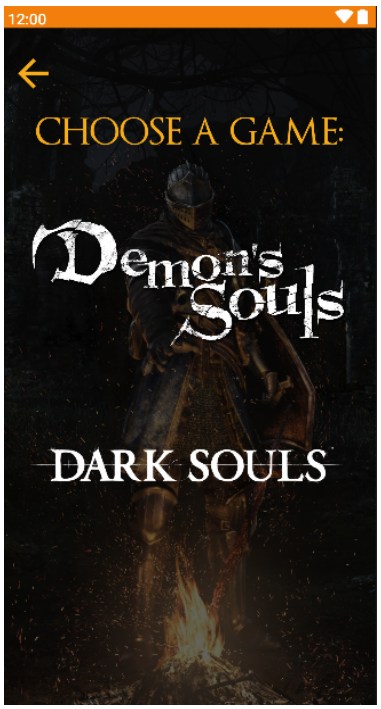

Fig.12 TitlesActivity

No matter what title the user clicks on, the next activity will be BossesListActivity, but depending on the game chosen by the user, the endpoint requested by the API will change.

Fig.13 BossesListActivity

In this activity there is a fragmentList displayed and as said before, regardless of the fragment clicked on by the user, the same activity will be started, in this case BossInfoActivity. What will change is the content of the views that will correspond to the boss chosen.

Fig.14 BossInfoActivity

2.3 User-App Behaviour

In the search for a correct user feedback it has been made use of Toast and Alerts Dialog in some activities.

There is a “Welcome” toast in the login activity:

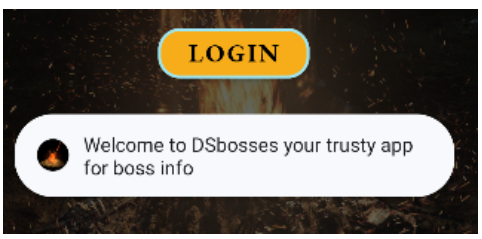

Fig.15 Welcome Toast

At this login activity there is also an alert dialog to warn the user when their credentials do not match:

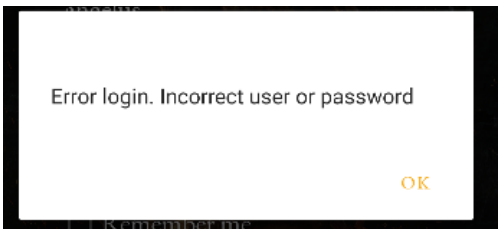

Fig.16 Error login AlertDialog

In addition, at both the LoginActivity and AboutUsActivity there is another alert dialog when the user tries to login or send their contact message if there is any blank field:

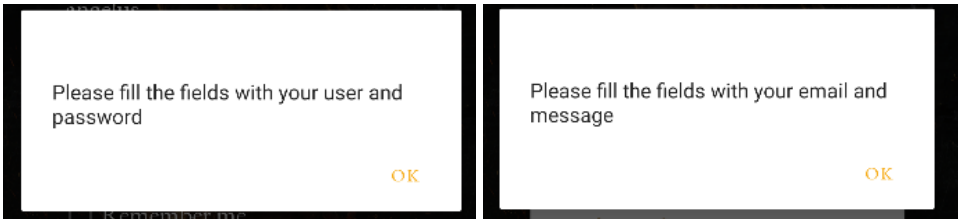

Fig.17 Empty fields AlertDialog

Also, every editText shows a hint of what is supposed to be written there:

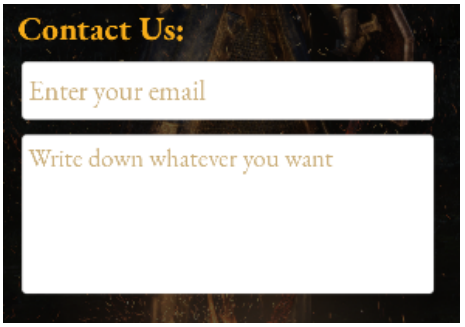

Fig.18 EditTexts hint

User login functioning

Currently, the login process works in such a way that if it is the first time the user uses the app, they will have to check the "remember me" option to be registered for a future session. Once registered, the user will be able to login in two different ways, one by entering their credentials registered without marking the "remember me" option or another by entering new credentials and marking the checkbox to override their past values.

3.- Strategies and Solutions to the Development Problems

3.1 Development Issues

I had previously worked with android studio but this occurred 3 years ago and using Java as programming language so the biggest problem during the early stages and what has caused a major slowdown in the development of the project has been the unfamiliarity with Android Studio and Kotlin.

Another concept new to me was the usage of fragments and fragmentLists due to the fact that in my first experience with Android I only worked with classes with inheritance from AppCompatActivity. This fact caused me some troubles when trying to make use of some functions inside the fragment class. For example, when a function needs the current context as a parameter, if it is a “normal activity” it is as simple as passing the keyword "this” and it works, but when we are using a fragment this is not possible due to the fact that fragments don’t live by them self, they are just pieces of layout that depend on an activity. So in order to achieve the functioning of the example stated before it is necessary to get the activity to which it is related and cast it as Context like shown here:

```
val myActivity: Activity? = activity
```

```
(myActivity as Context)
```

Fig.19 Cast of the activity from a fragment

Once the adaptation time with the IDE passed and the definition of a fragment was understood, the development of the project has been a pleasant experience.

3.2 App Enhancements

There are several improvements that I would like to develop for the app if I could dedicate additional time researching on features that I currently don't know how to implement.

The first change would be in the way login works. I would like to replace the current persistence of credentials locally with SharedPreferences by an online database that would allow the user to register using an email or their Google account.

Another improvement would be in the About Us section, currently the contact message that the user wants to write only simulates that it is sent

to the developer company. I would like those messages to arrive in my mail inbox.

Finally, another improvement I would prioritise would be the addition of images to the BossFragment. These images would be loaded from a URL just as has been done in the BossInfo activity.

4.- Personal Task Plan

This project has been worked on singly.
In the search for a better organisation and control of the tasks necessary for the progress of the project, the visual tool Trello has been used.

This is the invitation URL for the board: https://trello.com/invite/b/fYznjvPq/ATTIeb9e0f8b595d92d1021b2deaba478684B899F045/3pmtaartigaspe

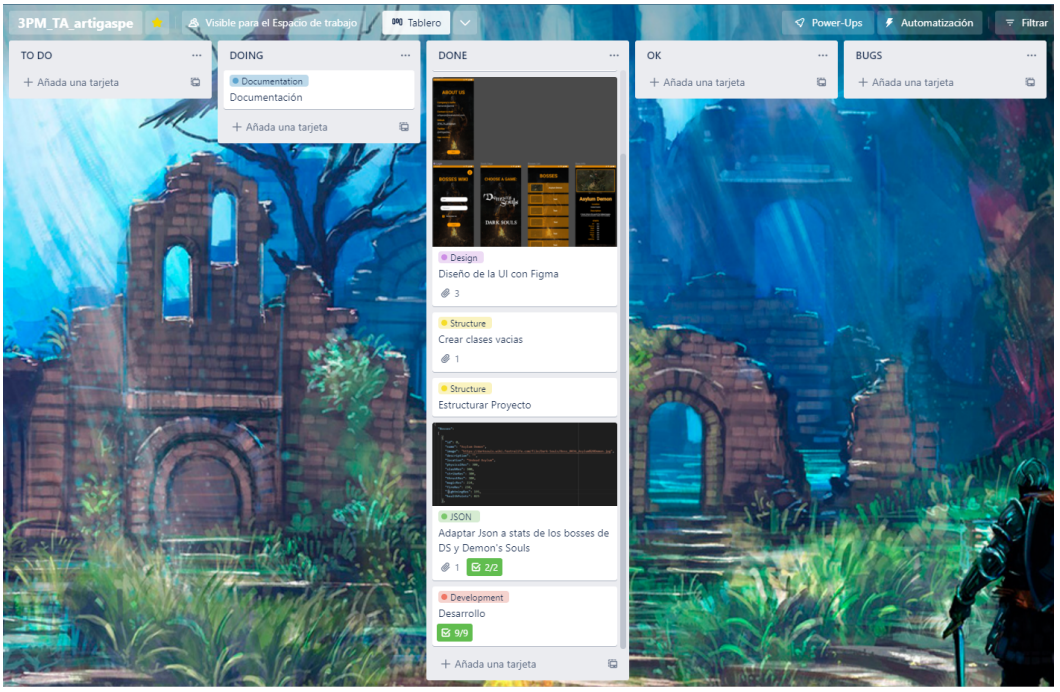

Fig.20 Trello Board

5.- Short User Manual and Demo Examples

5.1 Emulation Tools from Android Studio

If you want to emulate the application under development, the first step is to make sure that there is a virtual device on which to launch the apk: **[7]**

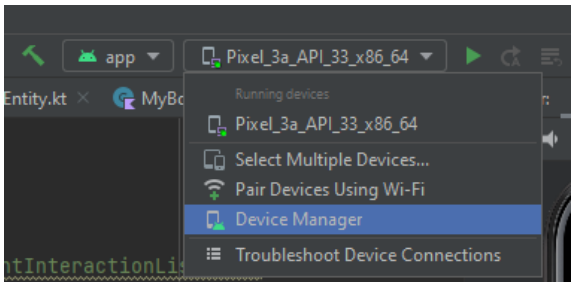

Fig.21 Device Manager

If no device exists you should click on the device manager option and create a new one:

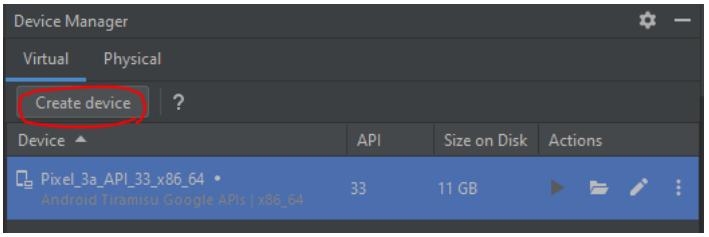

Fig.22 Create Device

Choose the device that best suits your needs:

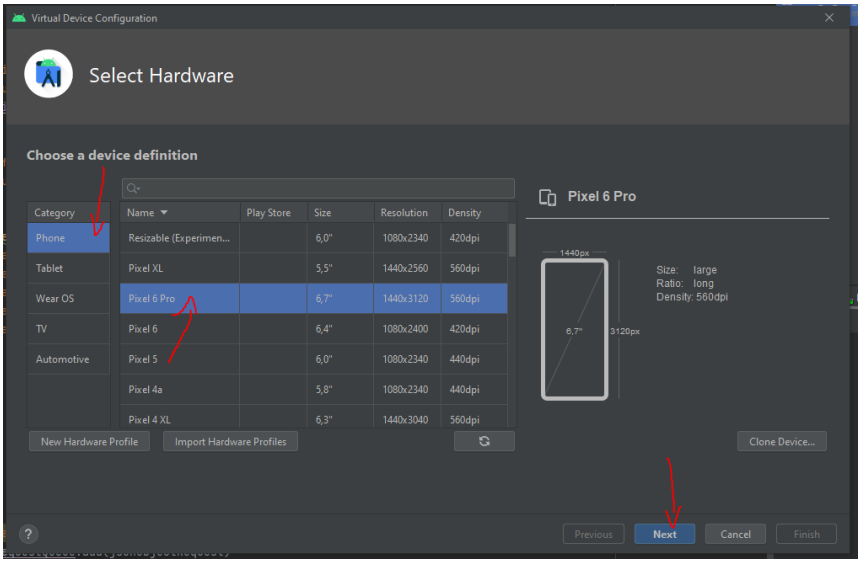

Fig.23 Choose device

Choose the API level desired for the virtual device:

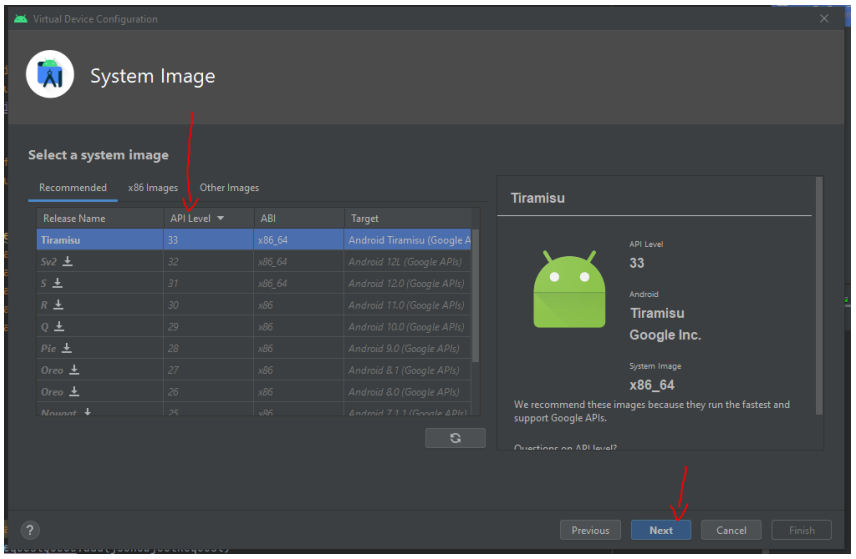

Fig.24 Choose API

Give a name for the new Android virtual device and click on “Finish”:

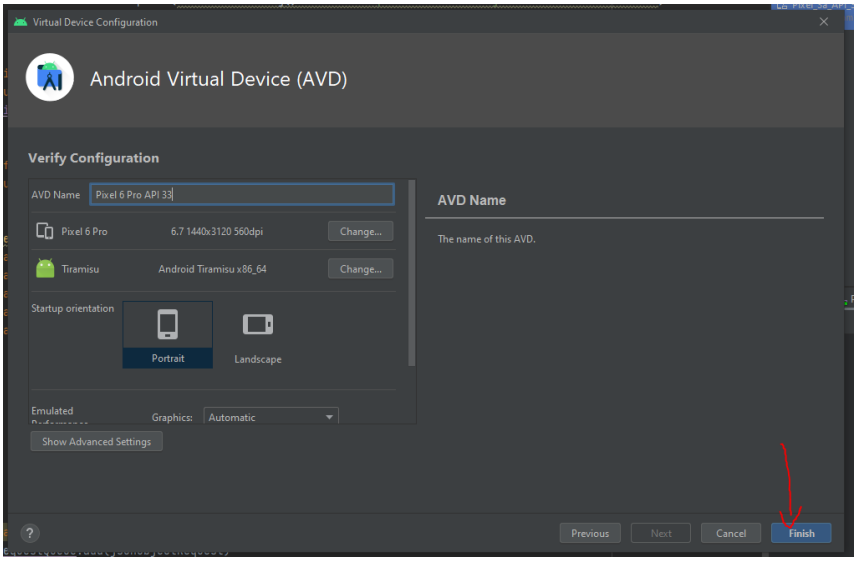

Fig.25 Creation of AVD Finished

Now select the device created and click on the “Run” button, it will launch the virtual device and generate an APK for it so the user can test their app.

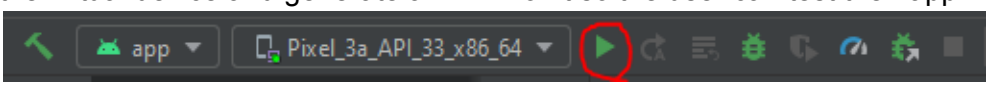

Fig.26 Run button

5.2 APK Debug Build

If you want to create a debug apk of your app you just have to click on the option show below:

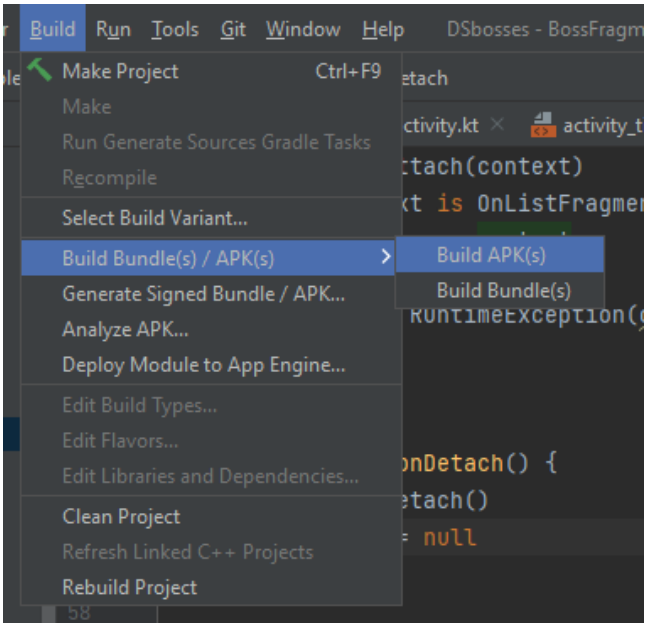

Fig.27 Build button

When the build process finishes a message will appear in the bottom-right corner of the screen:

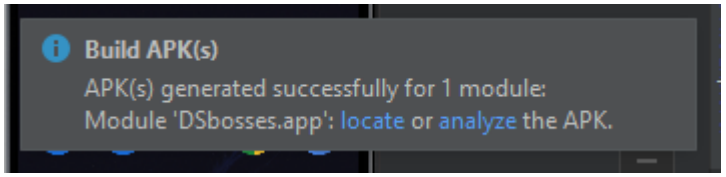

Fig.28 Build location message

Click on the “locate” option to open the directory that contains the generated apk:

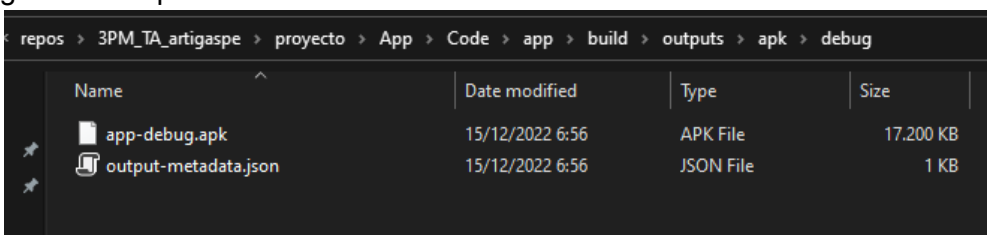

Fig.29 Build path

5.3 Demo and Versioning

Currently, an apk in debug mode has been built. If in the future it is desired to upload the app to an App Store (like Google's Play Store) it will be necessary to create a Signed App Bundle or a Signed APK. The meaning of "signed" refers to the fact that this application would be binded with a unique key that is required to be well-kept due to the fact that this key will be mandatory to perform actions like an update of the app at their corresponding App Store. This signed builds could be achieved in Android Studio:

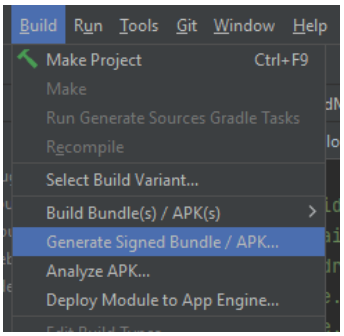

Fig.30 Signed generation

The minimum SDK of the application is 21 that corresponds to Android 5.0 (Lollipop). This choice has been made due to the fact that it will allow 98.8% of the devices to run the application.

Although the layouts of the project have been designed to be usable on the vast majority of hardwares, the virtual device used as reference has been a Pixel 3a, which means that it is possible that devices with very disparate measures may show up layout malfunctions.

Example of layout visualisation on AVD Pixel 3a:

Fig.31 Layout on Pixel 3a

6.- Bibliography

1.
Design Patterns in Android with Kotlin - GeeksforGeeks. ***GeeksforGeeks*****.** Online. 7 November 2021. [Accessed 14 December 2022]. Available from: https://www.geeksforgeeks.org/design-patterns-in-android-with-kotlin/

2.
MVC Design Pattern - GeeksforGeeks. ***GeeksforGeeks*****.** Online. 18 August 2017. [Accessed 14 December 2022]. Available from: https://www.geeksforgeeks.org/mvc-design-pattern/

3.
HTTPS://WWW.FACEBOOK.COM/WASISADMAN.ADAR. Android Architecture Pattern - MVC - AndroVaid. ***AndroVaid*****.** Online. 28 February 2019. [Accessed 14 December 2022]. Available from: https://androvaid.com/android/android-mvc-example/

4.

Aspectos básicos de las pruebas | Desarrolladores de Android | Android Developers. ***Android Developers*****.** Online. 2022. [Accessed 14 December 2022]. Available from: https://developer.android.com/training/testing/fundamentals

5.

Build local unit tests | Android Developers. ***Android Developers*****.** Online. 2022. [Accessed 14 December 2022]. Available from: https://developer.android.com/training/testing/local-tests

6.

Cómo realizar pruebas en Android Studio | Desarrolladores de Android | Android Developers. *Android Developers*. Online. 2022. [Accessed 14 December 2022]. Available from: https://developer.android.com/studio/test/test-in-android-studio

7.

Cómo crear y administrar dispositivos virtuales | Desarrolladores de Android | Android Developers. ***Android Developers*****.** Online. 2022. [Accessed 14 December 2022]. Available from: https://developer.android.com/studio/run/managing-avds#createavd