ESAT INNOVATION SCHOOL | ESCUELA SUPERIOR DE ARTE Y TECNOLOGÍA

MOBILE PROGRAMMING
Application Companion

Subject Name: Mobile Programming
Associated Module: APP DEVELOPMENT FRAMEWORKS (L5)
Teacher: Rubén Blanco García
Course: 2022/2023
Author: Sebastián Valdés Sánchez

Index

Contents

1.- App Project

1.1 App Introduction

The app that I have developed deals with all the "Amiibo" figures released to date from the video game series Super Smash Bros.

My app has the following features:

- Access and use an API
- Validate user Login and Register
- Give feedback to the user using pop-up messages
- Parsing the data from JSON from the API
- Use activities, fragments, buttons, images…etc.
- Save the user data with Shared Preferences
- Get crash errors using Firebase Crashlytics

My App has five activities and one fragment. I made my own background using Photoshop and I also remade a JSON file and uploaded it to Beeceptor, which is a website that allow you create endpoints to upload for example JSON files and create an own API…

When starting the App, we will find the login menu in which we can log in if we have a user created, you can also log in with a default user (user: Sebas, password: 1234). It is also possible to register in the App if the user wishes and the registration data will not be deleted since thanks to “Shared Preferences” they can be saved.

Once you have passed the login screen, a list will be displayed with all the "Amiibo" of the App's API, with their respective names and images...

There is also an “About Us” screen to know the name of the developer and a text block where feedback can be written to the developer and sent by pressing the send button.

1.2 Technical Demo

1.2.1 Tools Used:

-Volley:

Volley has been used to make requests to my API that is stored in Beeceptor.com.

Used version: *"com.android.volley:volley:1.2.1"*

-Gson:

Gson has been used to parse the API JSON.

Used version: *"com.google.code.gson:gson:2.10"*

-Picasso:

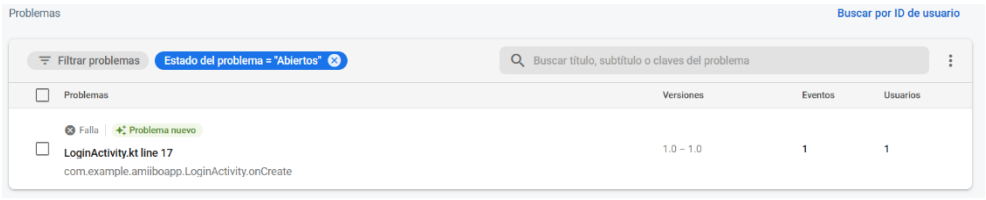
Picasso has been used to load the images from URLs that were in the JSON.

Used version: *"com.squareup.picasso:picasso:2.5.2"*

-Firebase Crashlytics:

Crashlytics has been used to find out where the program has crashed...

Example of Crashlytic in my App:



Used version: *"com.google.firebase:firebase-crashlytics-ktx:18.2.9"*

-Shared Preferences:

Shared Preferences have been used to save the registry of the users created in order to be able to log in without having to register again.

1.2.2 Technical Demo

-Login Activity: In this activity I made use of variables to store the name and password that the user has entered in the input text, later I check if the length of the name is greater than one and thus be able to return an error to the user so that they can enter the user correctly. If the user has been entered correctly but does not exist in the Shared Preferences, an error message is also returned.

Key tools used:

- Shared Preferences
- Toast for feedback
- Edit Text and Button

```
val password : EditText = findViewById(R.id.password)
val login : Button = findViewById(R.id.login)
val register : Button = findViewById(R.id.register)
val currentPreferences : SharedPreferences = getSharedPreferences( name: "UserInfo",  mode: 0)
val aboutusbut : Button = findViewById(R.id.AboutUsButton)

login.setOnClickListener(){  it: View!
    val usernameValue : String = username.text.toString()
    val passwordValue : String = password.text.toString()

    val registeredUsername : String = currentPreferences.getString("username", "").toString()
    val registeredPassword : String = currentPreferences.getString("password", "").toString()

    if(usernameValue.length > 1) {
        if (usernameValue.equals(registeredUsername) && passwordValue.equals(registeredPassword)
            || usernameValue.equals("Sebas") && passwordValue.equals("1234")) {
            val AppIntent: Intent = Intent( packageContext: this, HomeActivity::class.java)
            startActivity(AppIntent)
        }else{
            Toast.makeText( context: this, HtmlCompat.fromHtml( source: "<font color= '#FF2400'> <b>"
                    + "USER NOT FOUND D:" + "</b></font>",  flags: 0),
                Toast.LENGTH_SHORT).show()
        }
    }else{
        Toast.makeText( context: this, HtmlCompat.fromHtml( source: "<font color= '#FF2400'> <b>"
                + "ENTER A VALID VALUE!" + "</b></font>",  flags: 0),
            Toast.LENGTH_SHORT).show()
    }
}
```

-Register Activity: As in the Login activity, in this activity I also use variables to store the name and password that the user has entered in the input text, I also check if the user to register is valid and I return a message notifying the user if has done the process well or badly.

Key tools used:

- ·Shared Preferences
- ·Toast for feedback
- ·Edit Text and Button

```
val username : EditText = findViewById(R.id.username)
val password : EditText = findViewById(R.id.password)
val register : Button = findViewById(R.id.register)
val cancel : Button = findViewById(R.id.cancel)
val currentPreference : SharedPreferences = getSharedPreferences( name: "UserInfo",  mode: 0)

register.setOnClickListener() {  it: View!
    val usernameValue : String = username.text.toString()
    val passwordValue : String = password.text.toString()

    if(usernameValue.length > 1){
        val currentEditor : SharedPreferences.Editor = currentPreference.edit()
        currentEditor.putString("username", usernameValue)
        currentEditor.putString("password", passwordValue)
        currentEditor.apply()
        Toast.makeText( context: this, HtmlCompat.fromHtml( source: "<font color= '#228B22'> <b>" +
                "USER REGISTERED :D" + "</b></font>",  flags: 0),
            Toast.LENGTH_SHORT).show()
        //Toast.makeText(this, "USER REGISTERED :D", Toast.LENGTH_LONG).show()
        val intent : Intent = Intent( packageContext: this, LoginActivity::class.java)
        startActivity(intent)
    }else{
        Toast.makeText( context: this, HtmlCompat.fromHtml( source: "<font color= '#FF2400'> <b>" +
                "ENTER A VALID VALUE!" + "</b></font>",  flags: 0),
            Toast.LENGTH_SHORT).show()
        //Toast.makeText(this, "Enter Value in the fields!", Toast.LENGTH_LONG).show()
    }
}
```

-About Us Fragment: In this fragment I check that the feedback message that the user is going to send is valid, this will be checked when the user presses the send button and will be notified with a message using Toast.

Key tools used:
- ·Toast for feedback
- ·Edit Text and Button

```
sendButton.setOnClickListener(object: View.OnClickListener{
    override fun onClick(p0: View?) {
        if(valText.length() > 1){
            valText.text = ""
            Toast.makeText(activity, HtmlCompat.fromHtml( source: "<font color= '#228B22'> <b>"
                    + "MESSAGE SENDED :D" + "</b></font>",  flags: 0),
                Toast.LENGTH_SHORT).show()
        }else{
            Toast.makeText(activity, HtmlCompat.fromHtml( source: "<font color= '#FF2400'> <b>"
                    + "ENTER A VALID TEXT!" + "</b></font>",  flags: 0),
                Toast.LENGTH_SHORT).show()
        }
    }
})
```

-Smash Bros Data Class: I have used the following Data Class to store the information of the Amiibo coming from the API, among them is their corresponding name, the video game saga to which they belong, the URL of their image and their release date.

```
data class SmashBrosEntity(
    var Name : String,
    var Series : String,
    var Image: String,
    var Release : String,
)
```

-Home Activity: In this activity is where I make the request to the API using "Volley" and I also parse the JSON using "Gson", once this is done, I show the List View of characters with their names and images. For uploading images from URLs, I had to use the “Picasso” tool.

Key tools used:

- ·Access API with Volley
- ·Parsing JSON with GSON
- ·Load URL Image with Picasso
- ·List view

```
fun getSmashCharacter(callback: (MutableList<SmashBrosEntity>) -> Unit){
    val url = "https:///smashbrosprueba.free.beeceptor.com/characters"

    val jsonObjectRequest = JsonObjectRequest(
        Request.Method.GET, url,  jsonRequest: null, {
                response ->
            Log.ERROR

            //Parsing JSON:
            val jsonList = response.getJSONArray( name: "amiibo").toString()
            val mutablelistType = object : TypeToken<MutableList<SmashBrosEntity>>(){}.type
            val CharactersList = Gson().fromJson<MutableList<SmashBrosEntity>>(jsonList, mutablelistType)

            callback(CharactersList)
        },
        { error ->
            // TODO: Handle error
            Log.ERROR
        }
    )

    val requestQueue = Volley.newRequestQueue( context: this)
    requestQueue.add(jsonObjectRequest)
}
```

-Amiibo Info Activity: In this last activity, I set the variables for the name, image, video game saga, and release date of the characters.

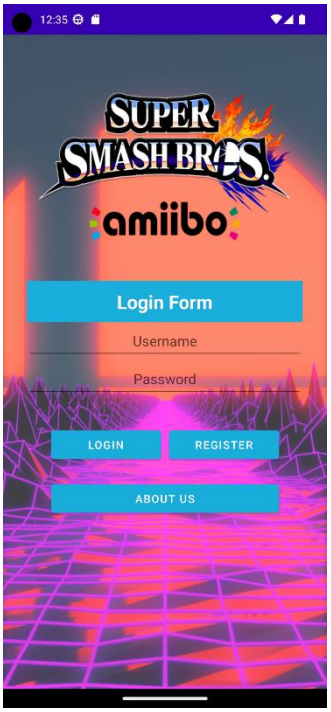
Key tools used:

·Load URL Image with Picasso

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    requestWindowFeature(Window.FEATURE_NO_TITLE)
    supportActionBar?.hide()

    binding = ActivityAmiiboInfoActivityBinding.inflate(layoutInflater)
    setContentView(binding.root)

    val img = intent.getStringExtra( name: "img")
    Picasso.with( context: this).load(img.toString()).into( binding.imageView);

    binding.dataName.text = intent.getStringExtra( name: "name")
    binding.dataSaga.text = intent.getStringExtra( name: "series")
    binding.dataRelease.text = intent.getStringExtra( name: "release")
```

1.3 Application Features and User Manual

-Login

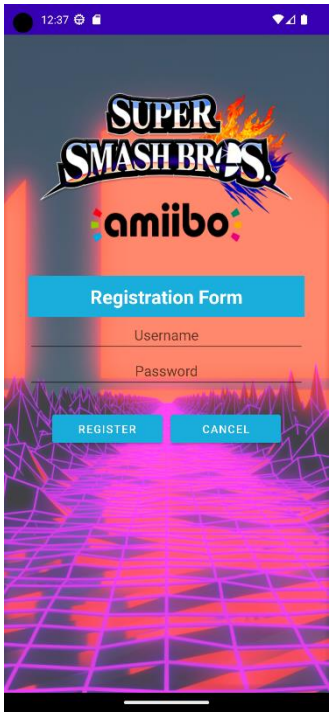


When opening the App, we will find a Log in menu where we can log in and access the App API, or we can press the registration button to go to the registration menu if we do not have a user created or go to the About Us menu by pressing the About Us button.

If the user has not been found, the following error will appear:

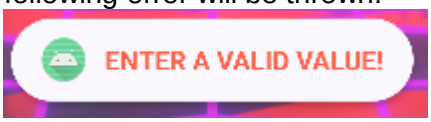
And if, on the other hand, no value has been entered, the following error will appear:

-Register

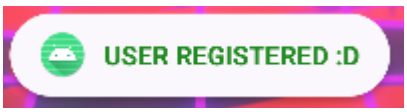


To register in the App, it is as simple as entering a username and password...

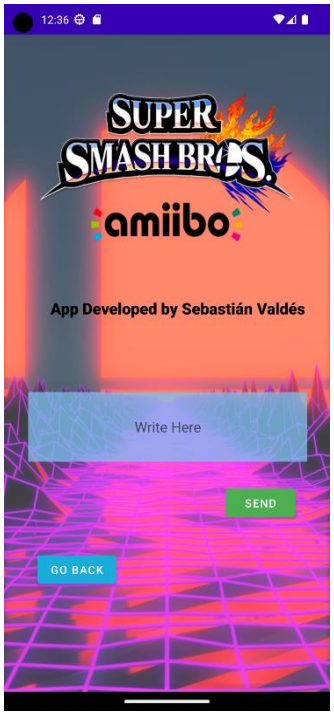
If the name or password are not valid, the following error will be thrown:



On the other hand, if everything is correct, the user will be registered and the following message will appear:



-About Us

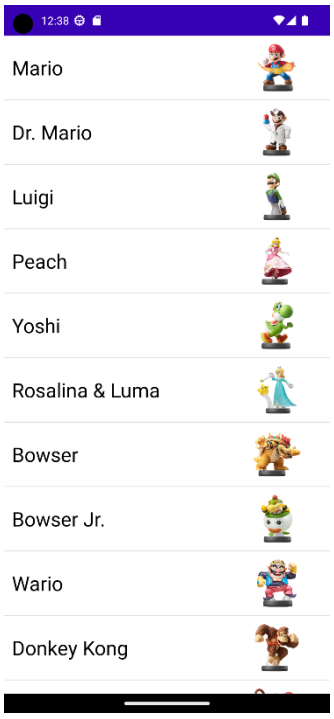


When you enter the “About Us” menu, you will see the name of the App developer, a text box to write feedback and a send button…

If when pressing the send button, the text is not valid, the following error will pop-up:

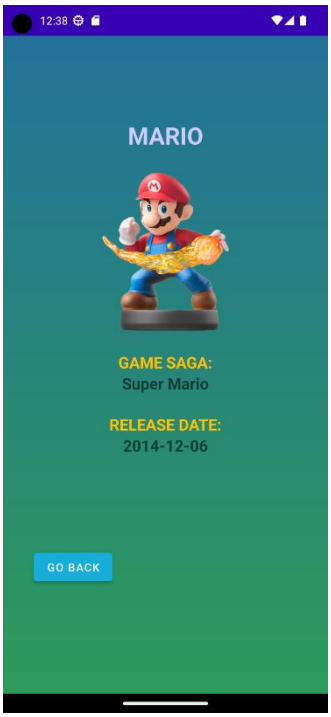
On the other hand, if the message has no errors, it will be sent and the following message will be displayed:

-Home



After logging into the App, a list of all “Amiibo” ordered by date will be displayed, showing their corresponding name and a small image on the right.

-Amiibo Info



If a component in the “Amiibo” list is pressed, a data sheet of that specific “Amiibo” will be displayed showing its Name, Image, Game Saga and release date.

1.4 Development Patterns

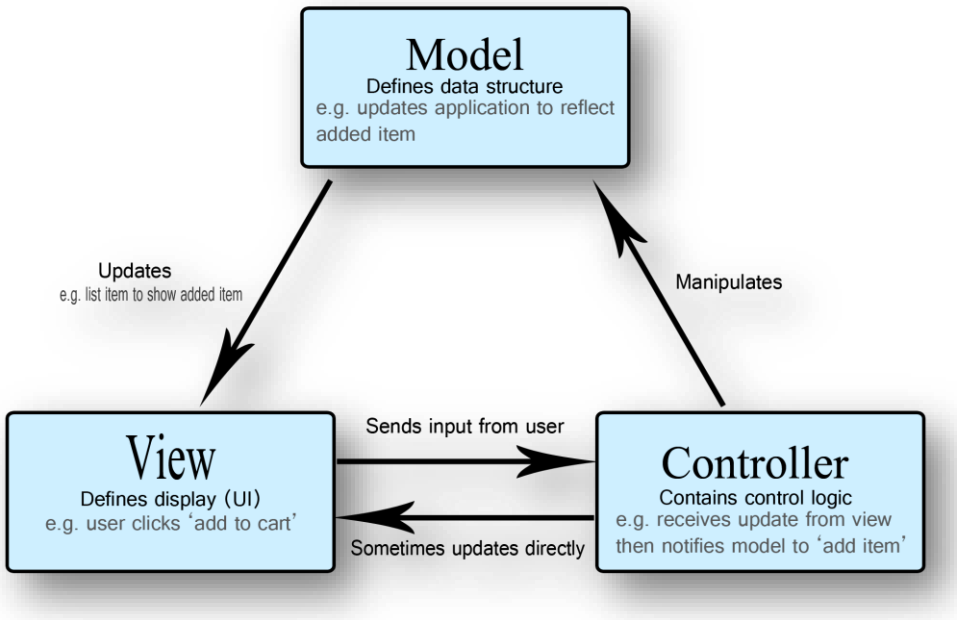
1.4.1 MVC:

MVC (Model-View-Controller) is a pattern in software design commonly used to implement user interfaces, data, and controlling logic. It emphasizes a separation between the software's business logic and display.

The three parts of the MVC software-design pattern can be described as follows:

1. Model: Manages data and business logic.
2. View: Handles layout and display.
3. Controller: Routes commands to the model and view parts.

-



1.4.2 MVVM:

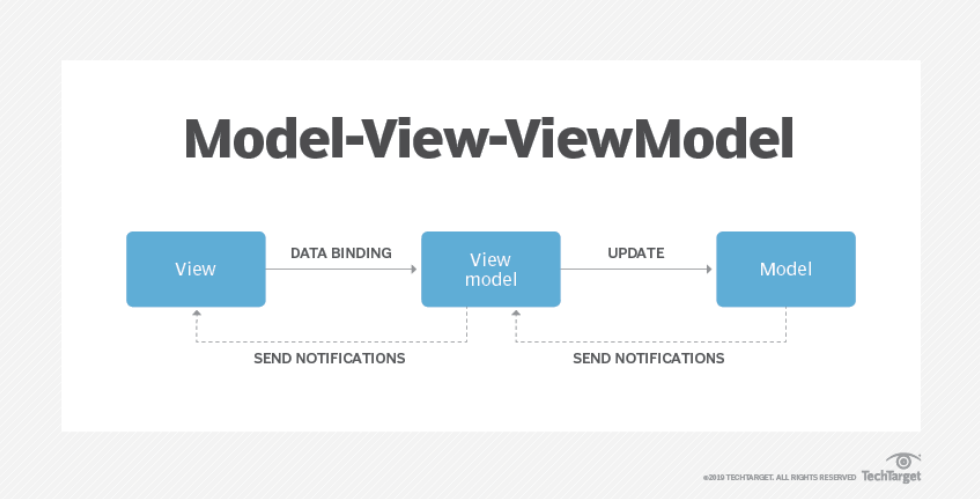
Model-View-View Model (MVVM) is a software design pattern that is structured to separate program logic and user interface controls. MVVM is also known as model-view-binder and was created by Microsoft architects Ken Cooper and John Gossman.

The separation of the code in MVVM is divided into View, View Model and Model:

·View is the collection of visible elements, which also receives user input. This includes user interfaces (UI), animations and text. The content of View is not interacted with directly to change what is presented.

·View Model is located between the View and Model layers. This is where the controls for interacting with View are housed, while binding is used to connect the UI elements in View to the controls in View Model.

·Model houses the logic for the program, which is retrieved by the View Model upon its own receipt of input from the user through View.



1.5 Unit Test

A unit test verifies the behavior of a small section of code, the unit under test. It does so by executing that code and checking the result.

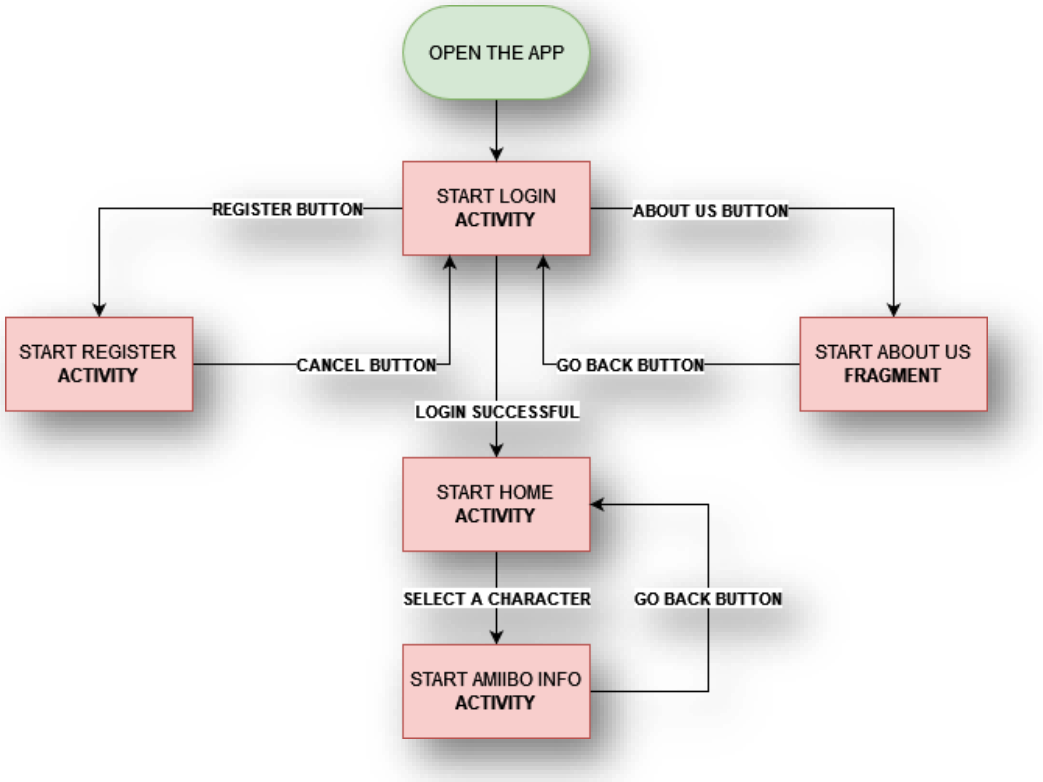
Unit tests are usually simple but their setup can be problematic when the unit under test is not designed with testability in mind:

- The code that you want to verify needs to be accessible from a test. For example, you can't test a private method directly. Instead, you test the class using its public APIs.

- In order to run unit tests in isolation, the dependencies of the unit under tests must be replaced by components that you control, such as fakes or other test doubles. This is especially problematic if your code depends on the Android framework.

1.6 Class Diagram

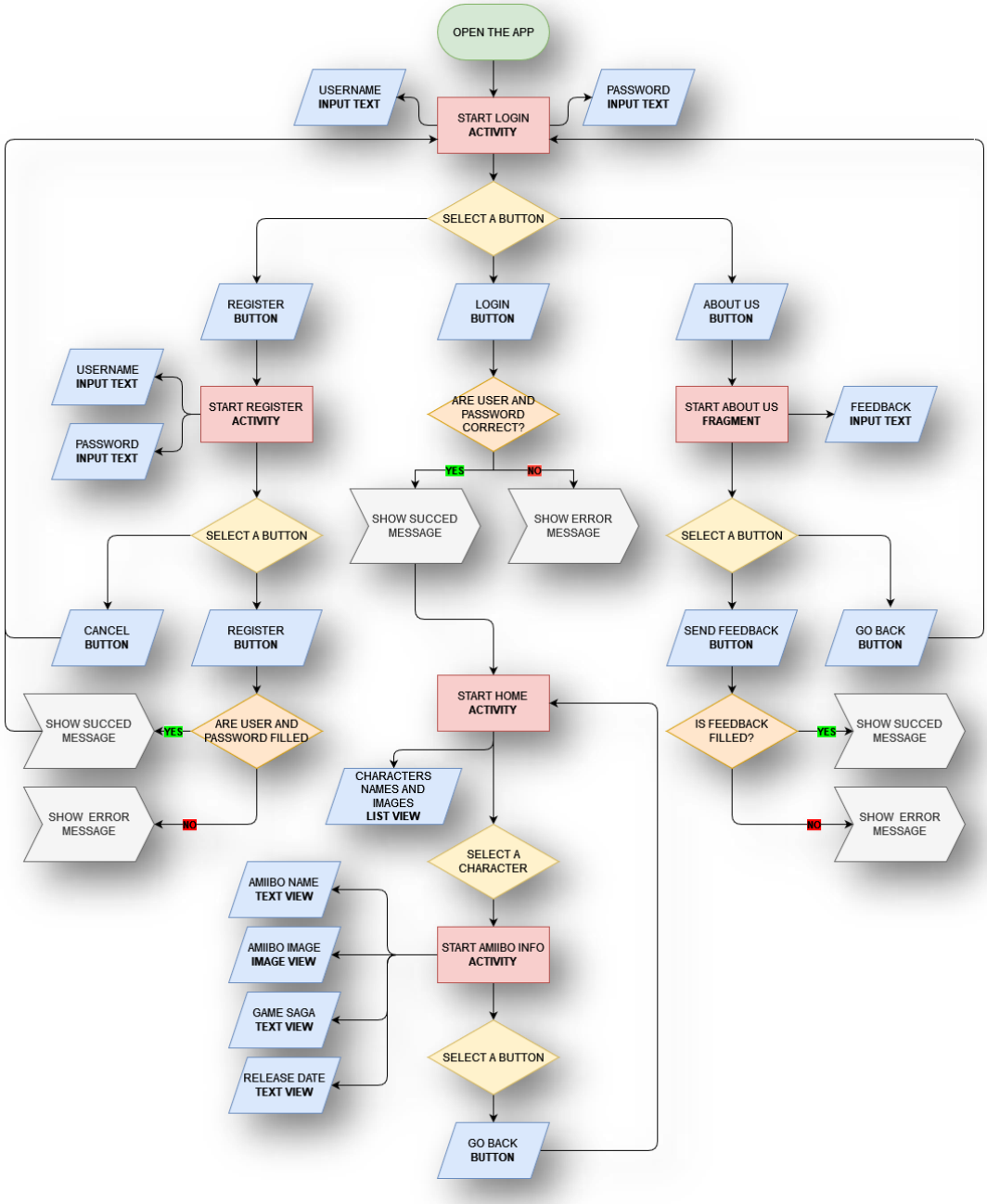
My App can be summarized in the following class diagram that I have made in Draw.io:



2.- UI/UX Flow Chart

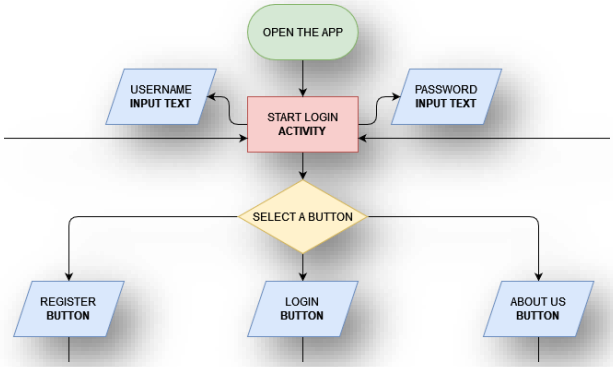
2.1 Flow Chart

For the Flow Chart, I have used the “Draw.io” tool to make the Diagram:

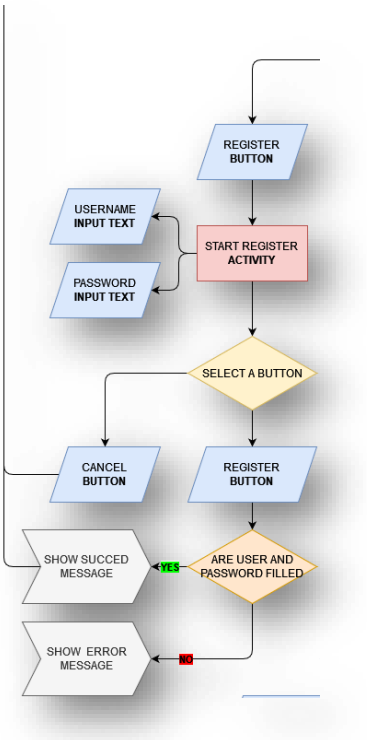


2.2 Application User Behavior

When starting the App, the Logging screen will be displayed, which has 2 Input Text, one for the username and another for the password, it also has 3 buttons, one to log in, another to go to the registration screen and the last one to go to the “About Us” screen



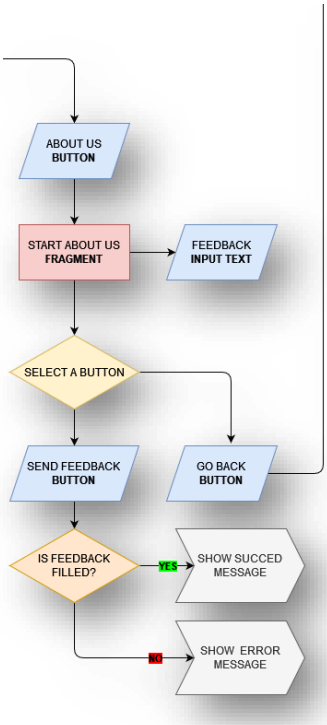
Register Button → Register Activity



If the registration button is pressed, the registration screen will open, which, like the login, has 2 Input Texts, one for the username and password.

There are also 2 buttons, one to return to the previous screen and the other to register the user's data, if everything went well, a message will appear that it has been saved correctly, otherwise, an error message will appear...

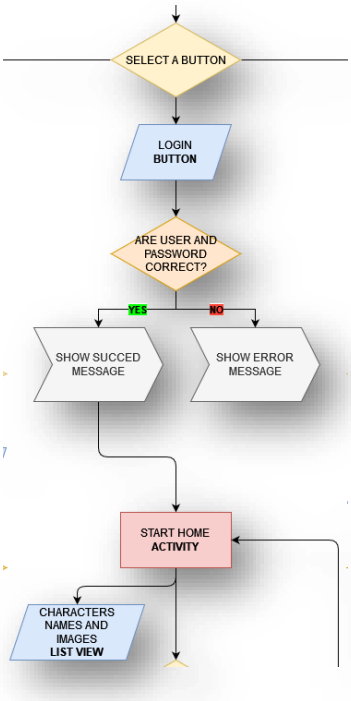
About Us Button → About Us Fragment



If the "About Us" button is pressed, the "About Us" screen will open, which has an Input Text to write the feedback to be sent to the developer

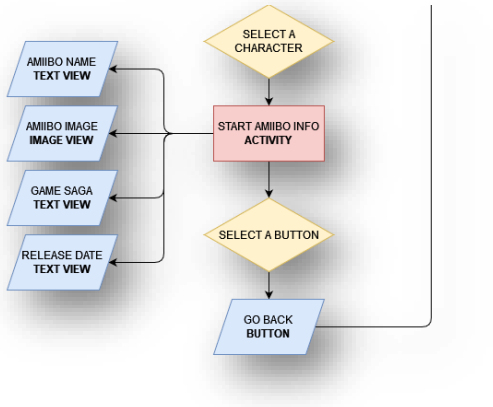
There are also 2 buttons, one to go back to the previous screen and the other to send feedback to the developer, if everything went well, a message will appear that the message has been sent correctly, if not, an error message will appear...

Login Button → Login Activity



When pressing the login button, it will be validated if the username and password are valid, if the username or password does not exist, an error message will appear, if there are no errors, it will enter the "Home Activity" where the list of all the “Amiibo”…

Character List → Amiibo Info Activity



Clicking on a character from the "Amiibo" list will open the activity that shows the information of the "Amiibo", which includes name, image, video game saga and release date.

3.- Strategies and Solutions to the Development Problems

3.1 Problems found

Throughout the development of the App, I found a variety of minor errors, most of them due to the fact that it was the first time I had programmed in Kotlin, but thanks to forums and the internet I was able to solve them and learn better how Kotlin works.

One problem that I spent a long time researching how to fix was how to make my “Go Back” button return to the previous activity without having to reload the previous activity, because reloading the activity would execute an API call again ... and the API only allows 50 requests a day, so I had to find a solution to it... The solution I found is that there is a function called "onBackPressed()" that when called will return to the previous activity without being interrupted. restart... which saves me multiple unnecessary API calls.

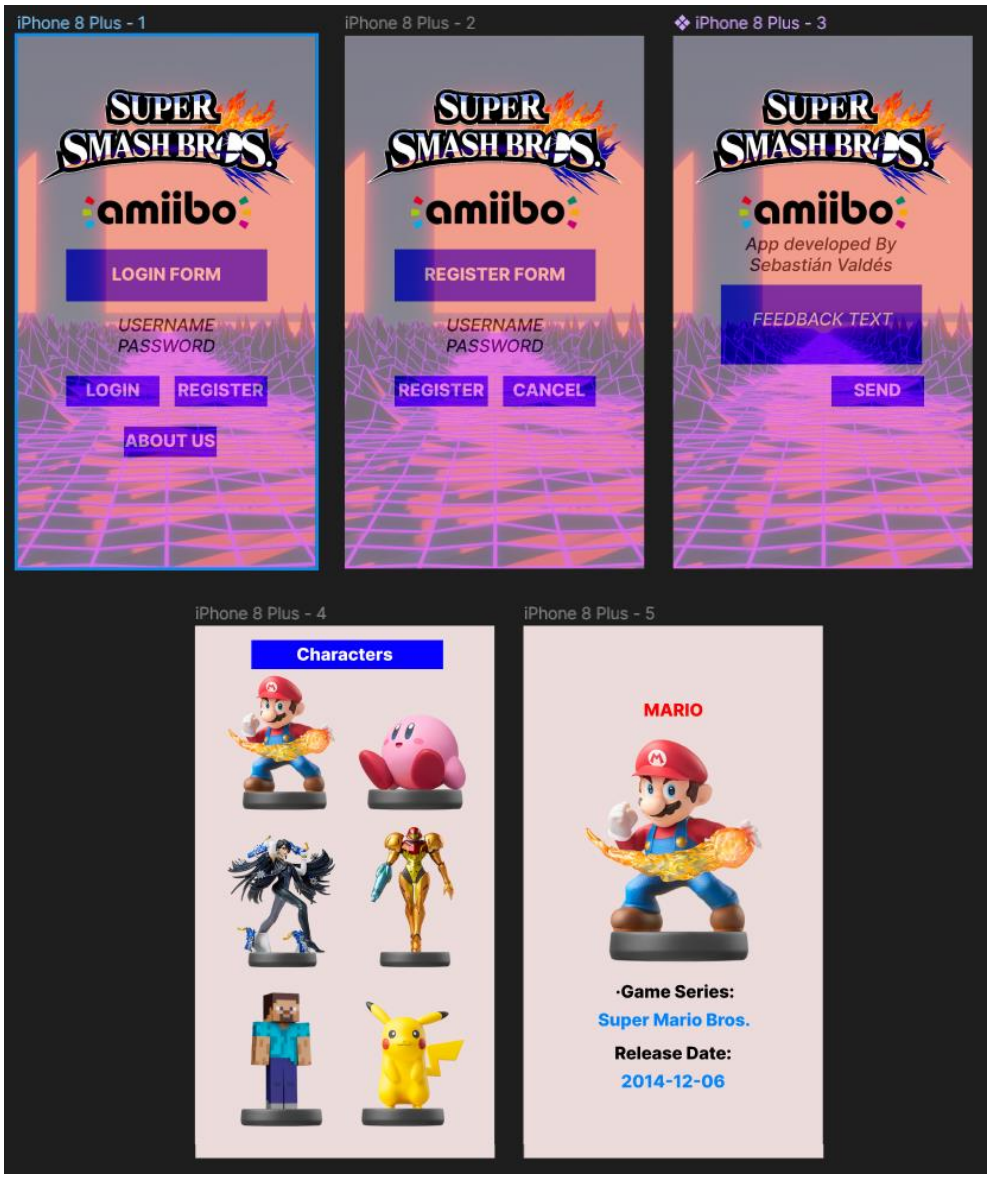
Another problem I had was that I didn't know how to import images from URLs, because my images were in the API… but I was able to find a tool called “Picasso” that allowed me to load images from URLs and it has been very useful.

4.- Personal Task Plan

4.1 Conceptualization

For the conceptualization part, read I was thinking a lot about what theme to base my App on, at first, I thought of basing it on Pokémon, then I wanted to do it on Smash Bros characters, but in the end, I ended up doing it based on the Amiibo figures from the Smash Bros Saga …

For the design of the conceptualization, I used the graphic tool "Figma", obtaining the following result:



4.2 Development

Regarding the development part, I decided to start by adding all the tools that I was going to use (Volley, Gson...) and then I started to program the Login and Registration screens, once I finished them, I decided to implement "Shared Preferences" so that the user registration is saved and can be logged in with the same username...

The next thing I decided to program was reading the JSON from the API and its respective parsing, once everything was read correctly from the API, I decided to program the list of characters with their respective image and name. I had to add the “Picasso” tool to be able to load the images from URLs.

When I already had the list of characters working, I decided to implement that when a character on the list is clicked, a new activity would open where the information of the corresponding Amiibo could be seen in detail, showing name, image, game it comes from and release date…

Finally, I decided to include to the project Firebase Crashlytics in order to know in which line and where the App crashes...

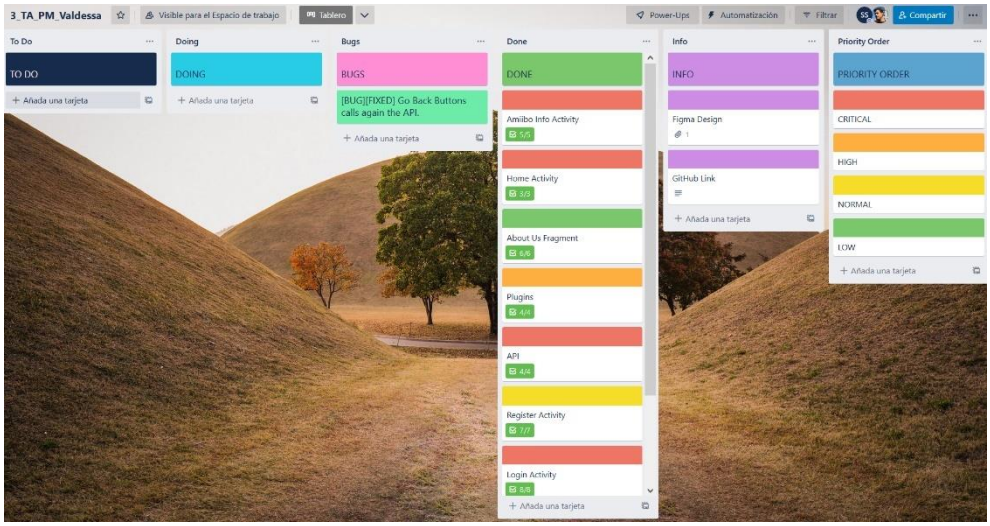
4.3 Documentation

The documentation was what I left for last, I explained all the development, tools used, ideas, problems... All accompanied by images to better understand the project.

4.4 Responsibilities

To organize my responsibilities, I used the "Trello" tool, which has been very useful for me to create boxes with the different activities and tasks that I have been doing throughout the development of the project.

Link to Trello: https://trello.com/b/jqKqQzQR/3tapmvaldessa

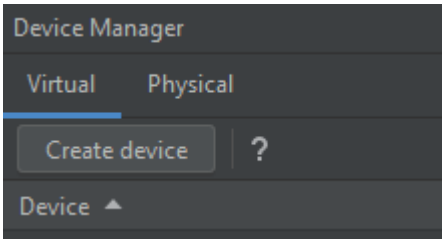


5.- Compilation Manual

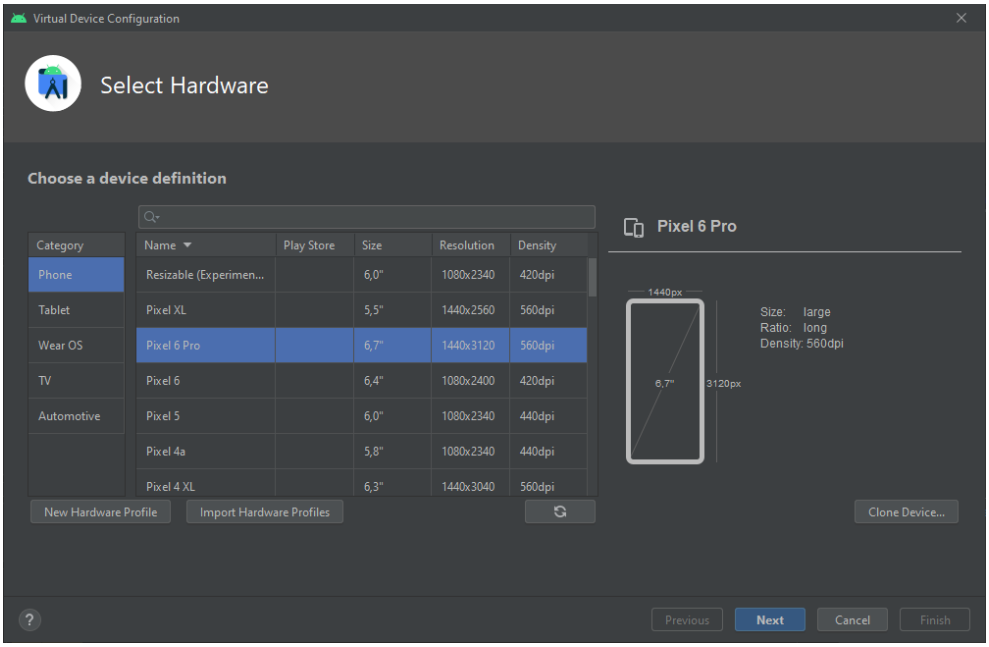
5.1 Running the App

To Run the App with Android Studio, we must follow the following steps:

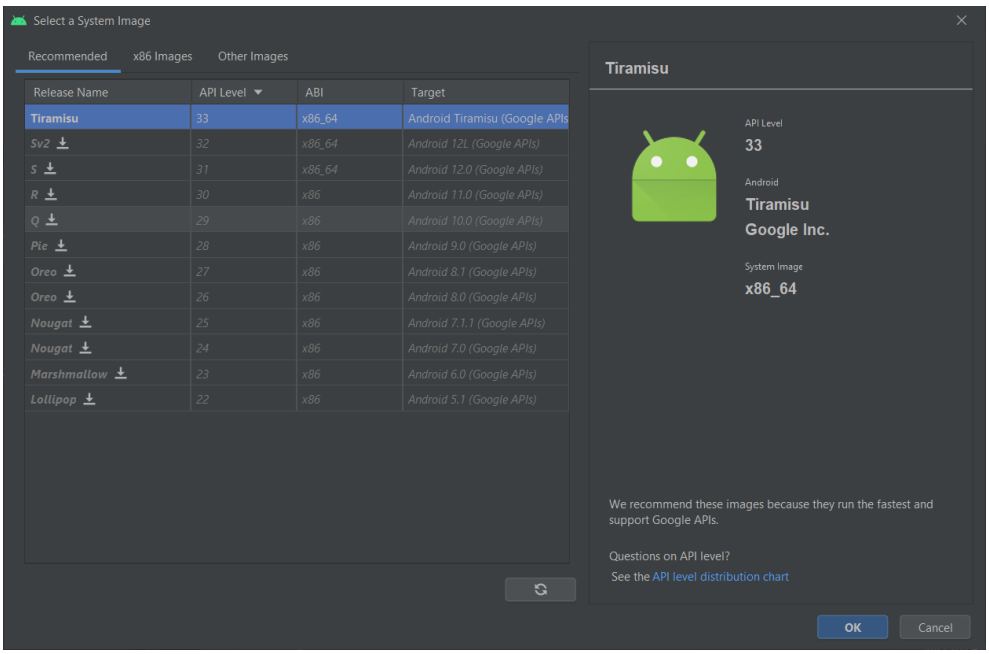
1.- Device Manager: First of all, we need to create an emulator, so we will have to select "Create a Device", then a window will open with the options for creating the device to emulate



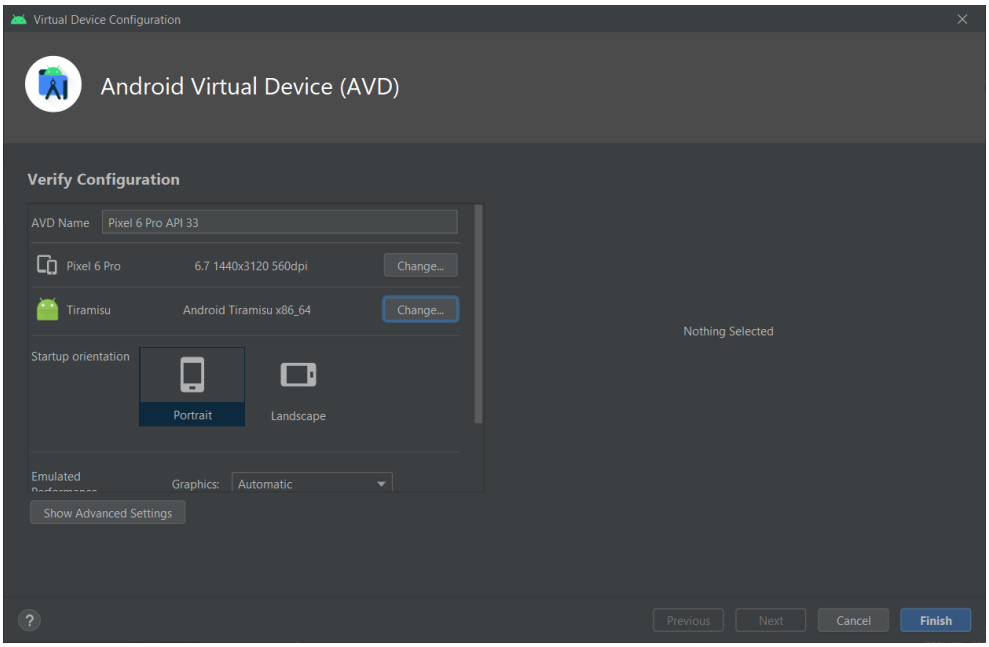
2.- Device Settings: In the window that has been opened, we will choose the type of device and the model that we want to emulate, in my case it will be a phone model "Pixel 6 Pro"



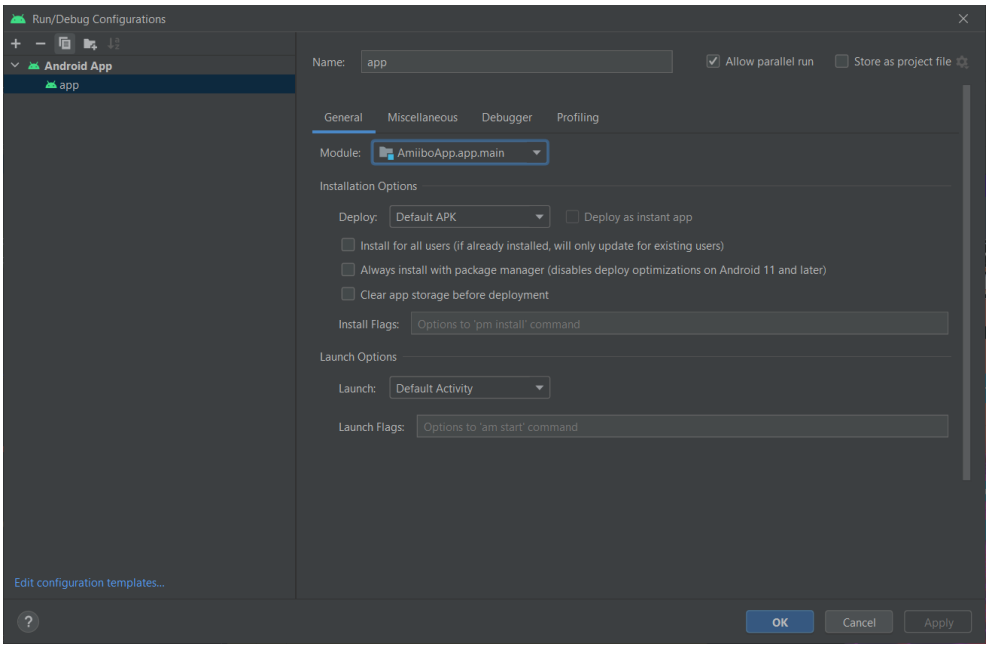
3.- Device Software: After that, a new window will be shown to select the Software of the phone, in my case I selected “Tiramisu”. If the software is not installed, a download will be necessary.



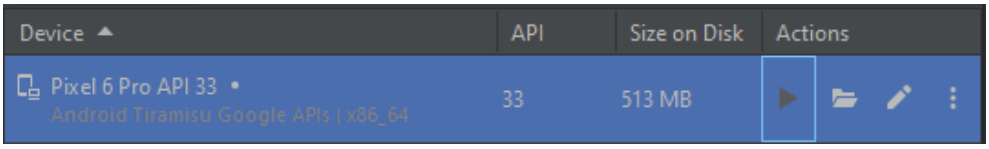
4.- Verify Configuration: Once the phone and its software are selected, a verify window will appear to check that all is correct, in this window we can choose other settings like the orientation, in my case the orientation of the phone will be “Portrait”.



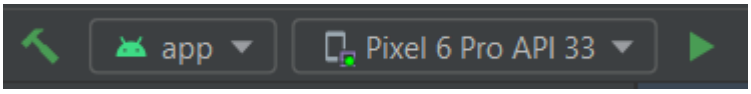
5.- App Configuration: Now that the emulator is created, a configuration for the App is needed, if we do not have a configuration created, we will click on the "green plus" button and within that configuration we will have to choose which will be the first activity that will be displayed when the App is executed.



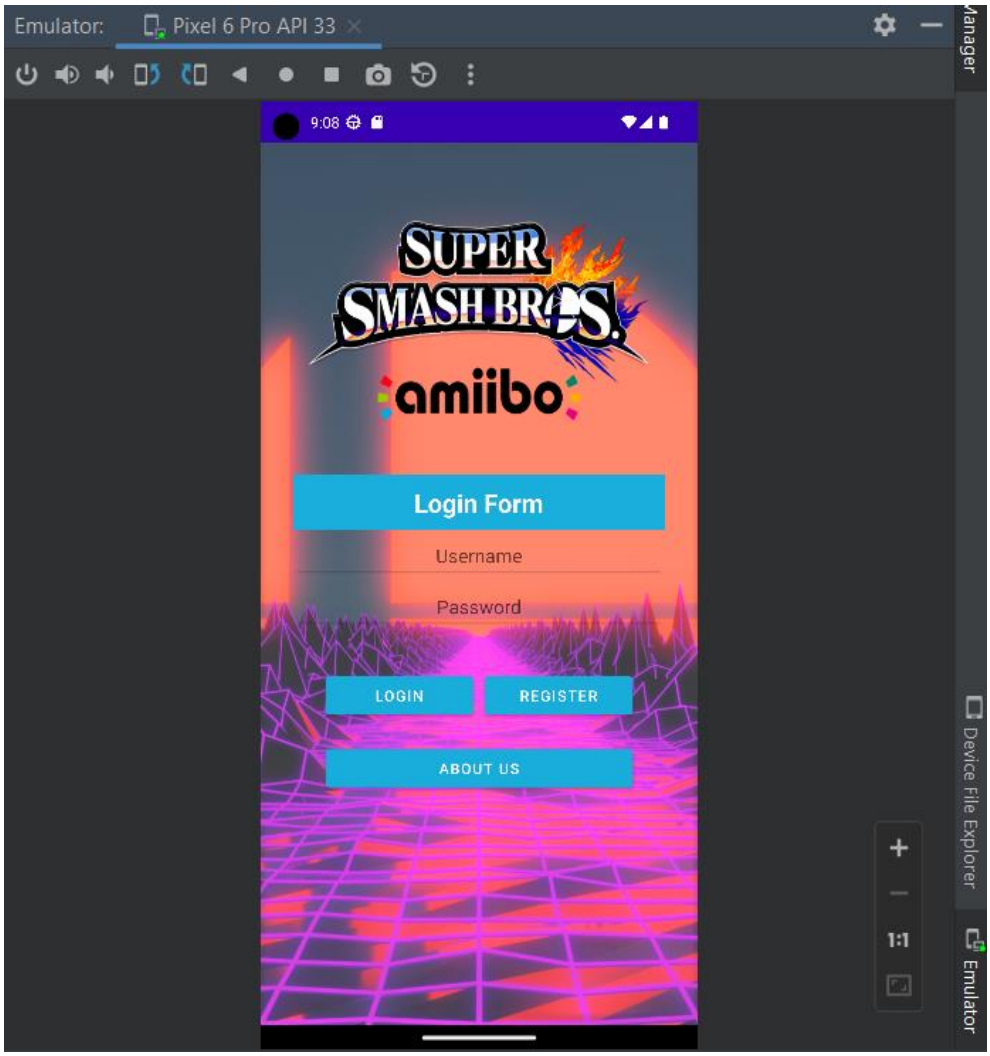
6.- Running the emulator: Now that all the configuration is set, it's time to run the emulator:



Once the emulator is running, we will be able to press the green button to execute the App



7.- App Running: If everything has gone well, we should see the home screen of the App:



6.- Bibliography

Unknown, Sep 21, 2022, MVC,
In: developer.mozilla.org [online] Available in:
https://developer.mozilla.org/en-US/docs/Glossary/MVC

TechTarget Contributor, 2022, Model-View-View Model (MVVM),
In: techtarget.com [online] Available in:
https://www.techtarget.com/whatis/definition/Model-View-ViewModel

Unknown, Sep, 9, 2022, Build local unit tests,
In: developer.android.com [online] Available in:
https://developer.android.com/training/testing/local-tests