ESAT INNOVATION SCHOOL | ESCUELA SUPERIOR DE ARTE Y TECNOLOGÍA

Mobile Programming 2023-2024

ANDROID APP DEVELOPMENT DOCUMENTATION

Nombre Asignatura: Mobile Programming
Módulo Asociado: 3º VGP
Profesor: Rubén Blanco García
Curso: 2023/2024
Autor/es: Ignacio Sempere Santonja

Index

1.- App Project

1.1.- Android introduction

Android, developed by Google, stands as the world's most widely used mobile operating system, powering billions of devices globally. Launched in 2008, Android has since evolved into a robust ecosystem, offering a versatile platform for smartphones, tablets, smartwatches, TVs, and other connected devices.

One of Android's defining features is its open-source nature, allowing manufacturers to customize the user interface and integrate their own apps. This flexibility has led to a diverse array of devices catering to various preferences and needs.

Android's evolution is marked by a series of version releases, each bringing new features, enhancements, and optimizations. Some notable versions include:

- **Android 1.0 (2008):** The inaugural release laid the foundation for the Android ecosystem, featuring essential applications and the iconic Google Play Store.
- **Android 2.3 Gingerbread (2010):** Introduced a refined user interface and improved performance, solidifying Android's presence in the mobile market.
- **Android 4.0 Ice Cream Sandwich (2011):** Unified the smartphone and tablet interfaces, offering a more cohesive user experience.
- **Android 5.0 Lollipop (2014):** Brought a material design overhaul, emphasizing a cleaner and more intuitive visual aesthetic.
- **Android 6.0 Marshmallow (2015):** Focused on improving system performance, battery life, and introducing granular app permissions.
- **Android 7.0 Nougat (2016):** Enhanced multitasking with split-screen mode and introduced Vulkan API for improved graphics performance.
- **Android 8.0 Oreo (2017):** Emphasized speed and efficiency with features like Project Treble, which streamlined the update process.
- **Android 9.0 Pie (2018):** Focused on AI-driven features, such as adaptive battery and adaptive brightness, tailoring the device experience to user habits.
- **Android 10 (2019):** Introduced a system-wide dark mode, improved privacy controls, and gesture navigation.
- **Android 11 (2020):** Emphasized conversations organization, improved media controls, and enhanced device control through the power menu.
- **Android 12 (2021):** Showcased a major design overhaul called Material You, allowing users to personalize the system's appearance.

The ongoing progression of Android highlights its commitment to innovation and adaptability, ensuring a dynamic ecosystem that continues to shape the mobile landscape. As new versions emerge, Android remains at the forefront of technological advancements, fostering a rich environment for developers and users alike.

1.2.- Developed Features

This is a simple application where users can log in, contact the developer, view a list of different characters of a video game and query more detailed information about each of the characters.

Login: Users can access the application through a standard login form in which they must enter their credentials, being these ones their username and their password. As the user types his password, he can change its visibility to check if it is correct before trying to access the application. If the user logs in successfully a welcome message will be displayed, on the other hand, if the credentials are not correct the user will be informed with an error message. Both of the messages are displayed in a Toast. As this is an academic project, to make it easier, there is only one valid username and password that are hardcoded in the source code. The username and password to log in the application are ***Batman*** and ***1234*** successively.

Query information: Once the user has logged in, a list of the main characters and some secondary characters of the *Batman Arkham Knight* video game can be displayed. This list is composed of an image and the name that identifies the character. The list of characters is loaded into the application through a request to an external database where a JSON file is read with all the necessary information. The JSON file, hosted in a *Firebase* database, collects the data asynchronously and fills each item of the list with the corresponding information. The images are loaded into the listing making use of the Glide framework, which loads the image through a URL. If any of the images are not found, a default image will be loaded. If the user clicks on any of the characters, either on the image or on its name, a new activity will open where additional information about the character can be consulted.

Contact form: The user is also provided with a contact form where they can send their comments to the developer. To do so, the user must fill out a standard contact form which requires a name, an e-mail address, a subject and a message.If any of the fields are left empty when sending the message, an error alert will be displayed informing the user that all fields must be filled in. Once all the fields are completed the user can send the message and the application will allow the user to decide which email client to use to send the message.

For the development of the login screen it has been used the *JetPack Compose* framework, the rest of the screens of the application have been developed with Kotlin. It is also worth mentioning that the default icon of the application has been changed and replaced with another image.

1.3.- Development patterns

Software development patterns provide proven solutions to recurring design problems, offering a structured and efficient approach to building robust and maintainable code. These patterns encapsulate best practices, enhance code readability, and promote modularity, making them integral to the software engineering process. Here, we'll explore three widely used patterns: Model-View-Controller (MVC), Observer, and Singleton.

Model-View-Controller (MVC)

- **Description:** MVC separates an application into three interconnected components. The Model manages data and business logic, the View handles presentation and user interface, and the Controller manages user input and communication between Model and View.
- **Advantages:** Clear separation of concerns, promotes code organization, facilitates code reusability, and supports scalability for large applications.

Observer Pattern

- **Description:** The Observer pattern defines a one-to-many dependency between objects, ensuring that when one object changes state, all its dependents are notified and updated automatically. This is commonly used to implement distributed event handling systems.

- **Advantages:** Decouples subject and observer, supports a loosely coupled design, and facilitates communication between objects without direct dependencies.

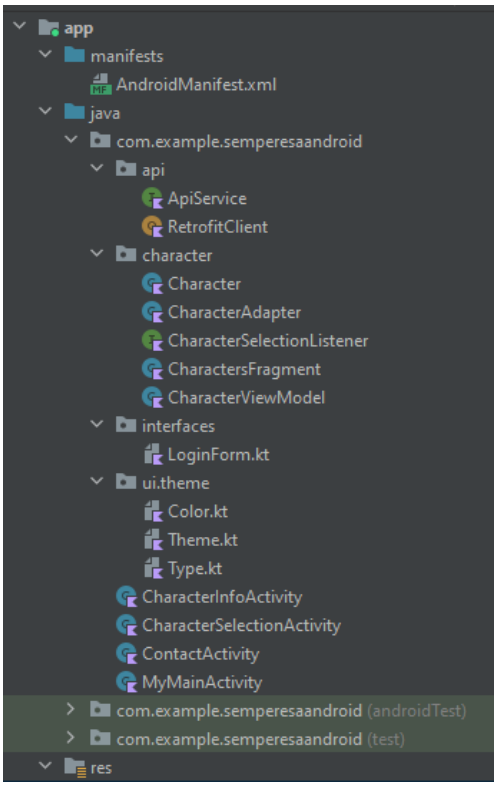
Singleton Pattern

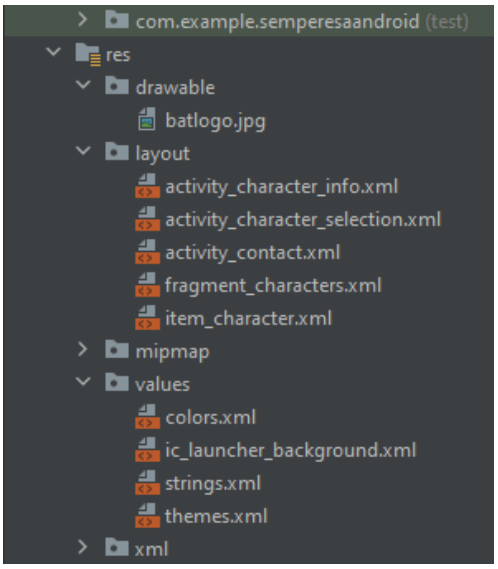
- **Description:** The Singleton pattern ensures that a class has only one instance and provides a global point of access to it. This is useful when exactly one object is needed to coordinate actions across the system, such as a single configuration manager or logging service.
- **Advantages:** Guarantees a single instance of the class, allows a global point of access, and is efficient in terms of memory usage.

For this project I have made use of the MVC pattern.

1.4.- Structure

This is the structure of the project.





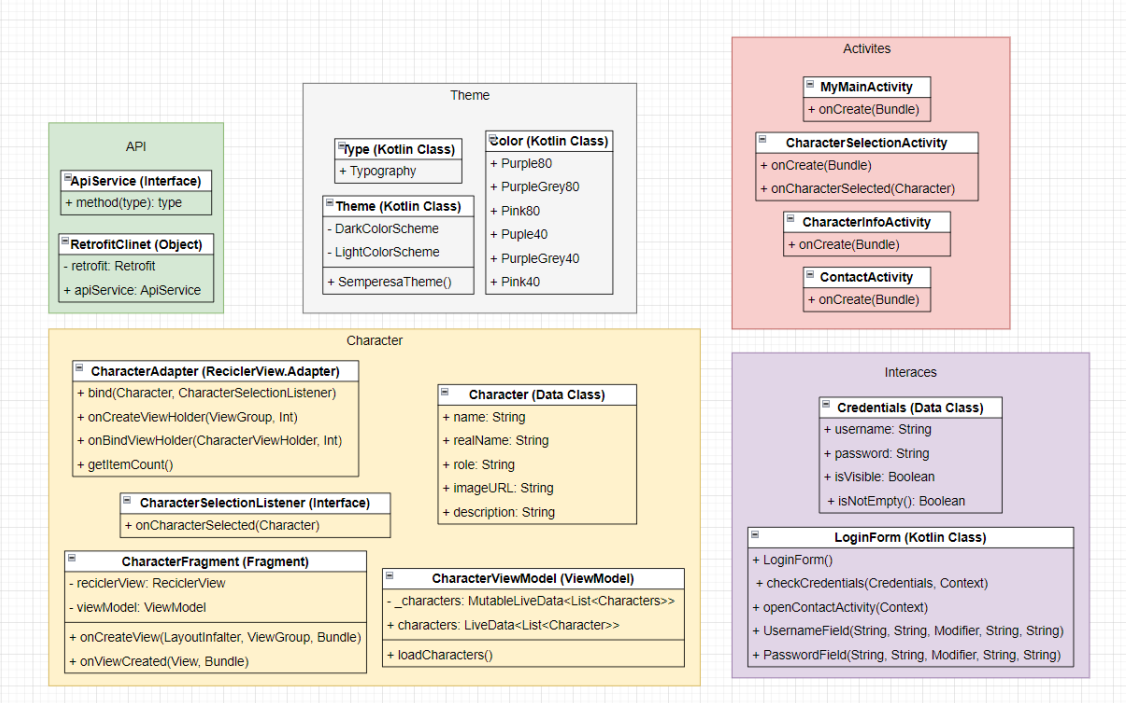
In the images above we can see the different Kotlin classes and activities, the API interface used to read the JSON located in the *Firebase* database and the different layouts used to design each activity of the application. As it can be seen, each Kotlin class and xml file is inside its own package. This allows the developer to better organize and structure the source code and the different files and classes.

The structure of the JSON used for this project is as follows:

```
{  "description": “”,
   "imageURL": “”,
   "name": "",
   "realName": "",
   "role": ""  }
```

The JSON file is an array of this structure containing all the information for several characters of the video game. The endpoint used to locate the JSON file is *Firebase*.

This is a class diagram showing all the classes used in the project



1.5.- Unit tests

Unit testing in Android Studio is a fundamental practice for ensuring the reliability and functionality of code within Android applications. These tests focus on individual units of code, typically isolated functions or methods, to verify that they produce the expected output for given inputs. Its main key points are:

Isolation of Code Units: Unit tests allow developers to isolate specific sections of code, assessing their behavior independently of the rest of the application. This isolation enhances the ability to detect and fix bugs or unexpected behavior at an early stage.

Automation of Testing: Android Studio provides a robust testing framework that enables the automation of unit tests. Developers can create test cases and run them automatically, saving time and ensuring consistent testing across different parts of the codebase.

Regression Testing: Unit tests serve as a form of regression testing, ensuring that changes or additions to the codebase do not unintentionally break existing functionality. This is crucial for maintaining the stability of the application during development and updates.

Enhanced Code Maintainability: By incorporating unit tests, developers can write modular and maintainable code. Testing individual units promotes a clean and organized code structure, making it easier to understand, modify, and extend the application over time.

Continuous Integration Support: Android Studio integrates with continuous integration tools, allowing developers to incorporate unit tests into their continuous integration pipelines. This ensures that tests are run automatically with each code change, providing rapid feedback on the code's health.

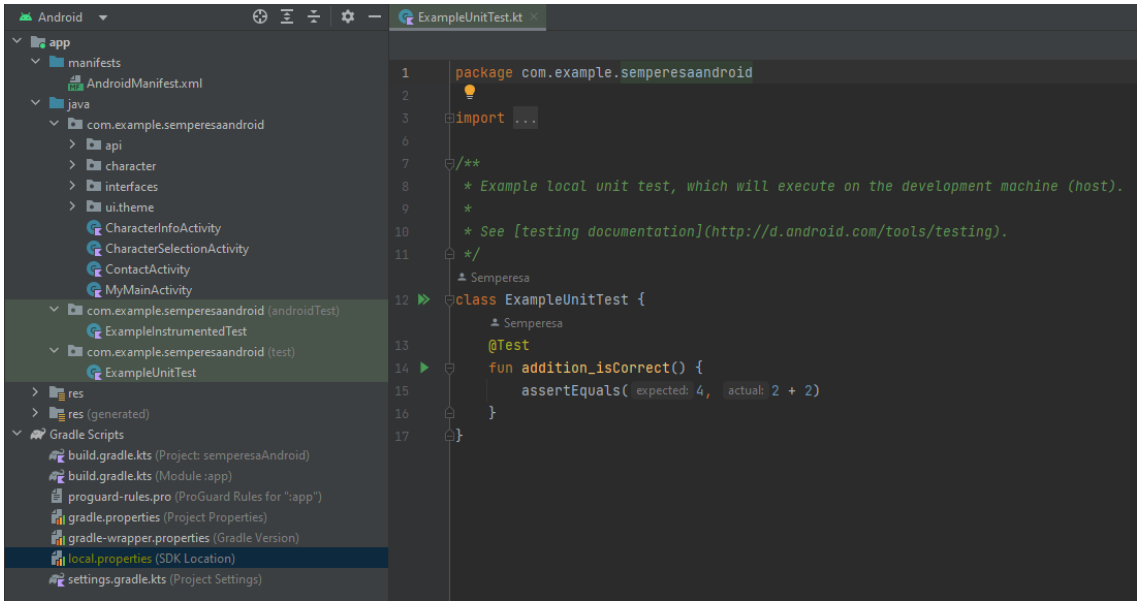
Mocking Dependencies: Unit testing in Android Studio often involves the use of mocks to simulate the behavior of external dependencies, such as databases or network calls. This enables thorough testing of specific components without relying on the actual implementation of external services.

Test-Driven Development (TDD): Some developers adopt a test-driven development approach, writing tests before implementing the actual code. This methodology ensures that the code meets the specified requirements and facilitates a more systematic development process.

In summary, unit testing in Android Studio is a critical practice for maintaining code quality, identifying issues early in the development process, and supporting the overall stability and reliability of Android applications. It is an integral part of the software development lifecycle, contributing to the creation of robust and maintainable software.

Despite all the advantages that the Android Studio unit test brings to our work I have decided to not use them in my project as this is only an academic one and I don't think I will continue developing this application once it's finished. If I decide to develop an application on my own for my personal interest I'm completely sure that I will make use of them.

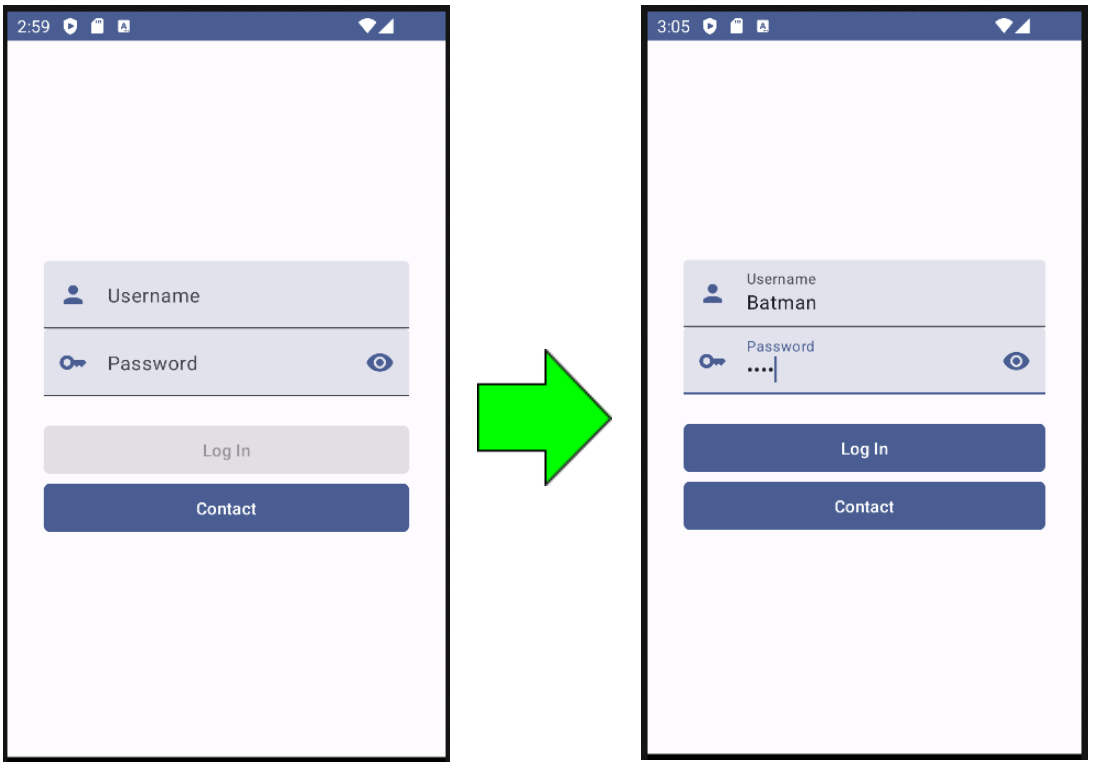
This is an example of a unit test that Android Studio provides by default to our app.



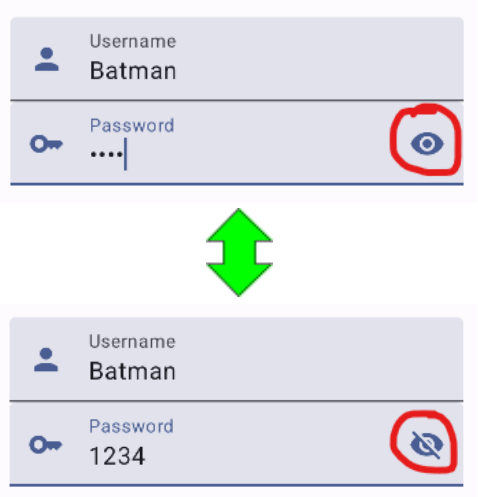
2.- UI/UX Flow Chart

This section explains how is the flow between the different screens of the application and how the user can navigate through them.

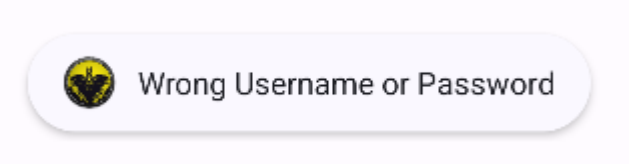
The first activity is the form to log into the application. As we can see the *LogIn* is disabled. It will only be enabled once the username and password fields are filled in.



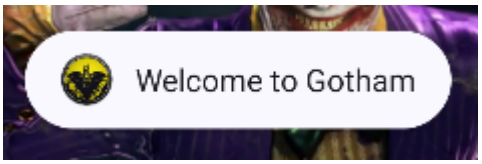
On the password field there is an eye icon that allows the user to toggle its visibility. The icon also changes its appearance.



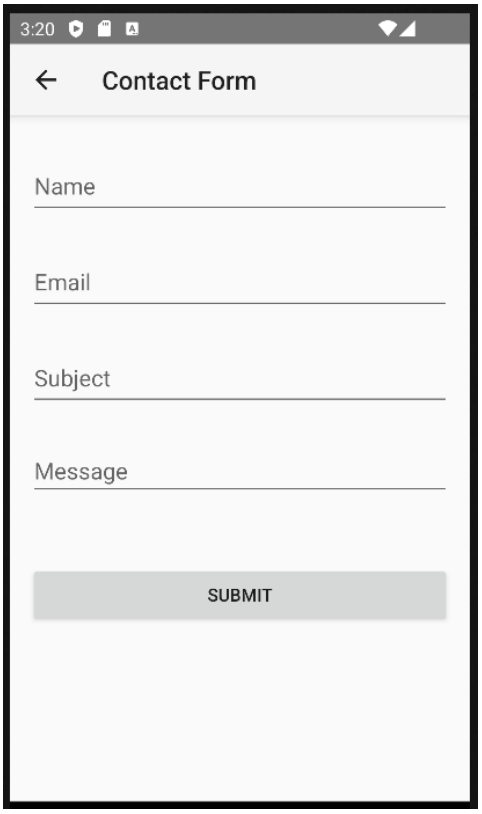
If the credentials are incorrect an error message will appear in a toast to inform the user



If the login is successful a toast message will appear greeting the user.

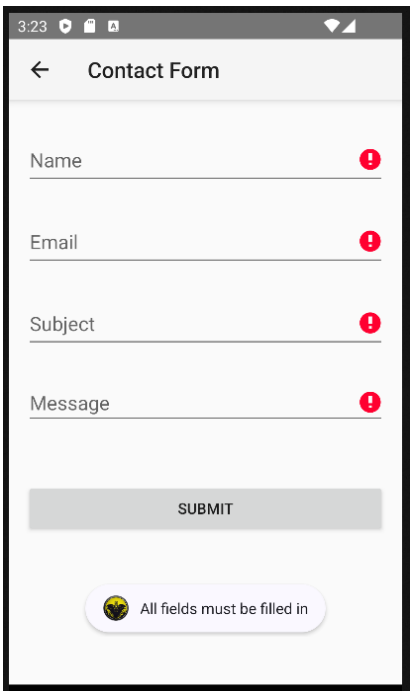


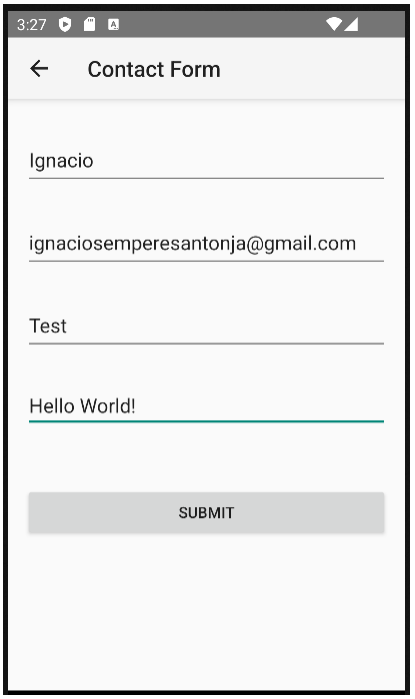
As it has been seen formerly, in the login screen there is a button to access the contact form. This is its appearance.

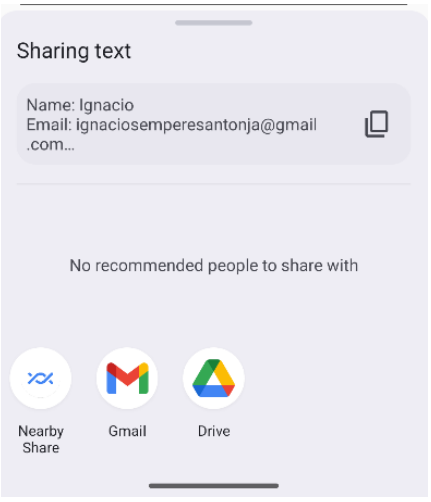


The contact form is a standard form where the users can contact the developer. If we fulfill all the fields and send the message it will be sent to my esat email address.

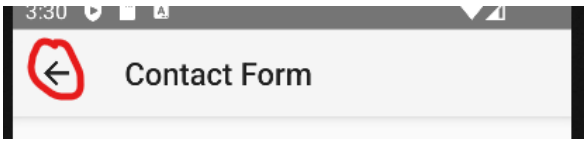
If the user tries to submit its message with one (or more) fields empty an error message will be shown and the empty fields will display an alert. However, if the user fills every field it will be able to send the message. The app lets the user choose which mail client to use.







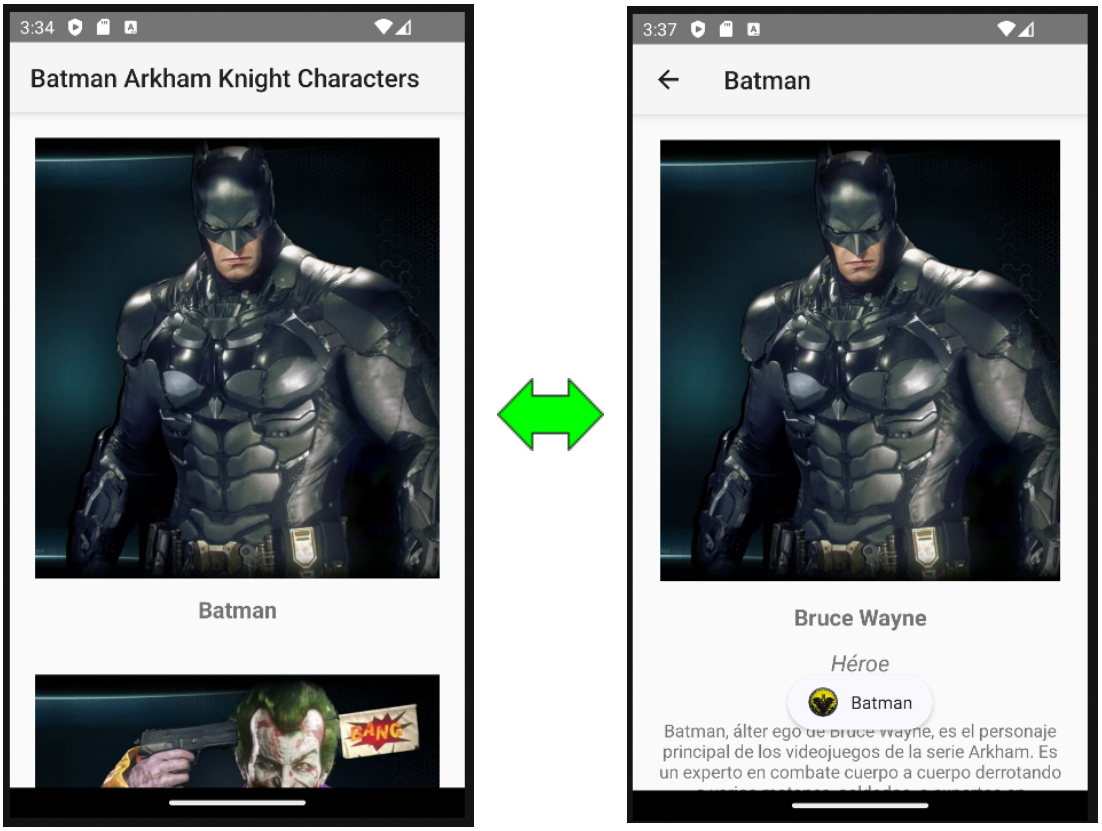
There is a button in the navbar to get back.



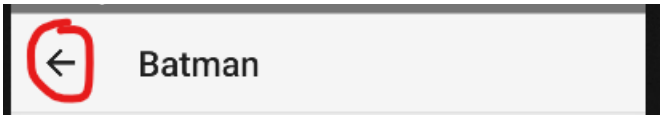
Back to the login screen, once we have logged in, we can see the main screen where the characters from the video game *Batman Arkham Knights* are displayed.

If we click in any of the characters it will open a more detailed screen with more information about said character. A toast message with the character name will also be displayed.

In the detail screen it can be seen the name of the character, its real name, its role, and a brief description of it.



Finally, in the detail activity there is also a button which will lead the user back to the characters screen.



3.- Strategies and Solutions

During the development of this project, various problems have arisen, some of which have been corrected and others that have had to be rejected, trying to find an alternative solution. These have been some of the problems:

Error importing Gradle dependencies: When importing some Gradle dependencies, it could not be synchronized correctly, which resulted in an error message. This message indicated that it was not possible to find certain dependencies or that they did not exist, so the solution to this problem was as simple as checking that the dependencies that were being tried to import were correctly written and in their latest available version. One of the dependencies that gave an error when trying to import them was the one that follows, which turned out to be not written correctly.

```
"androidx.compose.material:material-icons-extended:1.5.4"
```

Errors designing some layouts: When trying to design the layout of some activities I had problems ensuring that all the elements were positioned correctly and behaved appropriately. To repair this error I just reviewed which properties of each element were not being used correctly and changed them to the corresponding property.

Error parsing the JSON file: The first time I tried to read the JSON file, in which all the character information is located, through the firebase endpoint it resulted in the character list being empty. To correct this error I resorted to the help of classmates who had also used the same endpoint and asked them how they had done it.
The solution to the problem turned out to be to write .json, the file extension, at the end of the path where the database was being read from.

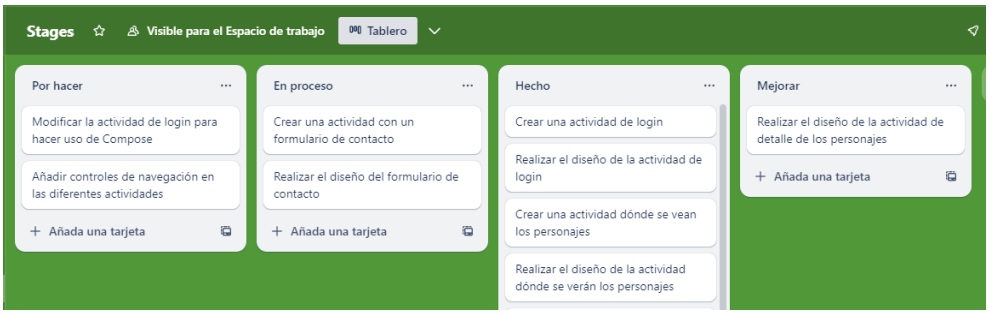
Compose: Since I had never used JetPack compose before in developing previous applications, I was not very clear about how it worked and how to start using it. To do this, I reviewed several tutorials on the internet until I managed to develop the desired functionality in my application.

Crashlytics: In this application I have tried to use *Crashlytics*, a tool provided by Google to track errors that occur in the application, but for some reason I have not been able to get *Crashlytics* to work or track any of my errors. To do this, I have created a specific button (that is currently commented on the source code) in one of the application's activities that simulates an execution error, but I still have not been able to make it work. I don't know if the dependencies that I have added to Gradle are incorrect or if, on the contrary, I am not using this tool correctly. This has been the only development error that I have dismissed and have not resolved since it was an optional section and was consuming too much production time.

4.- Personal Tasks Plan

To better organize all the tasks that I had to complete during the development of this app I have been using Trello. With Trello I have recorded all the functions that the application should have. As Trello uses panels I have used several of them to differentiate between which tasks were undone, which ones were in process, which ones were done and which ones could be improved. These are the different stages that I followed on this project.

It is worth noting that, as I developed the application and new needs arose, I also added new cards to Trello.

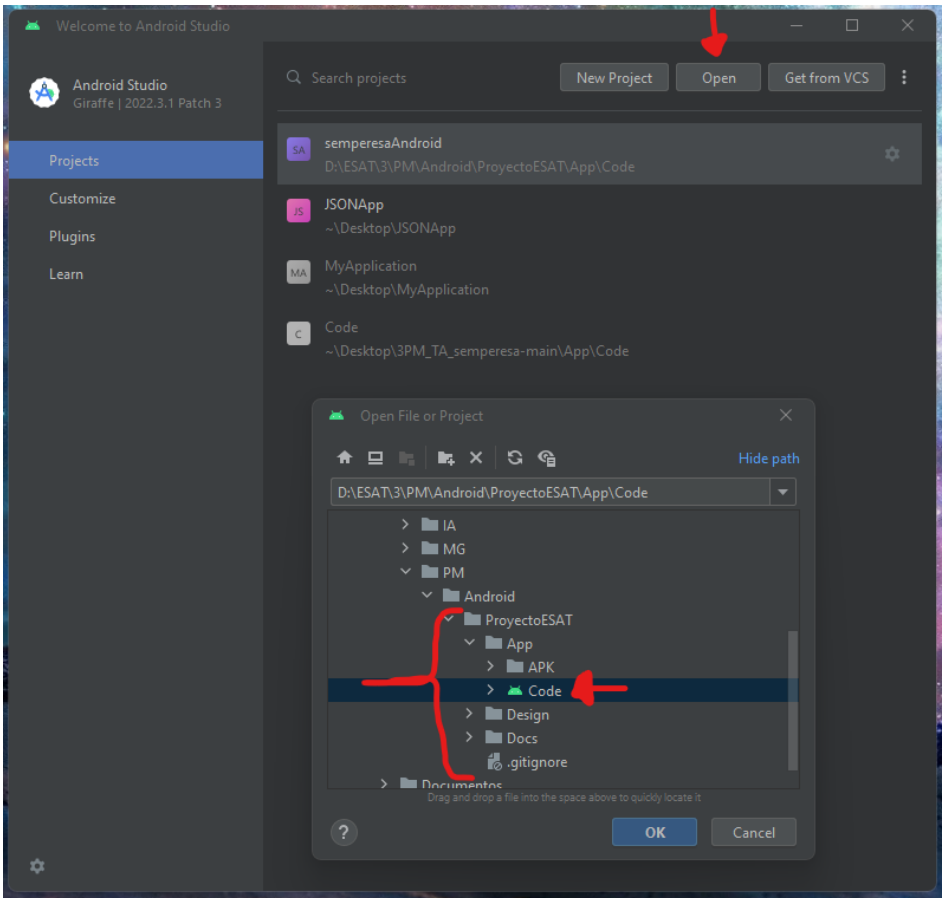


On the last stage all the tasks were on the “Hecho” panel, but I didn't screenshotted it because it was too long. Also I made some changes on the app during its development that are not reflected in the Trello tasks, but those were minimum changes, so I thought it was worthless to add them as a task to complete.

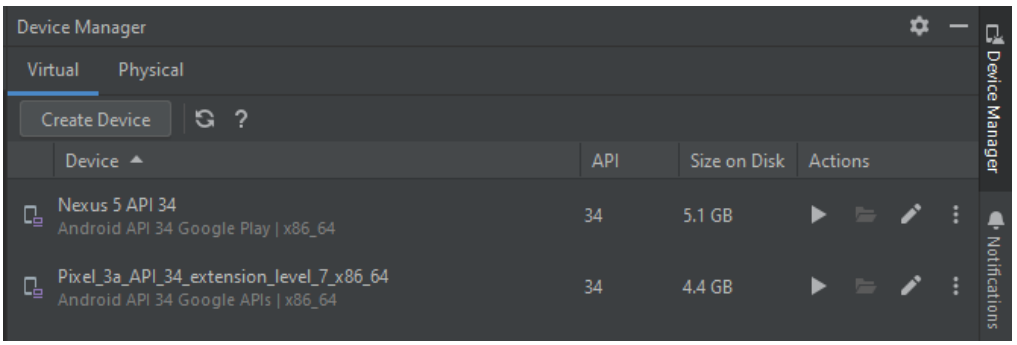
5.- Short User Manual

To execute and test this project it has to be opened in the Android Studio IDE. If we have a zip file we need to decompress it first, if we have cloned it from github we can open it directly from Android Studio. In case we have the APK we can install it directly on the phone.

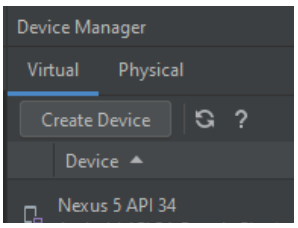
As it can be seen I have the project already in Android Studio. To add it we have to click in **Open** and find the directory **App > Code**

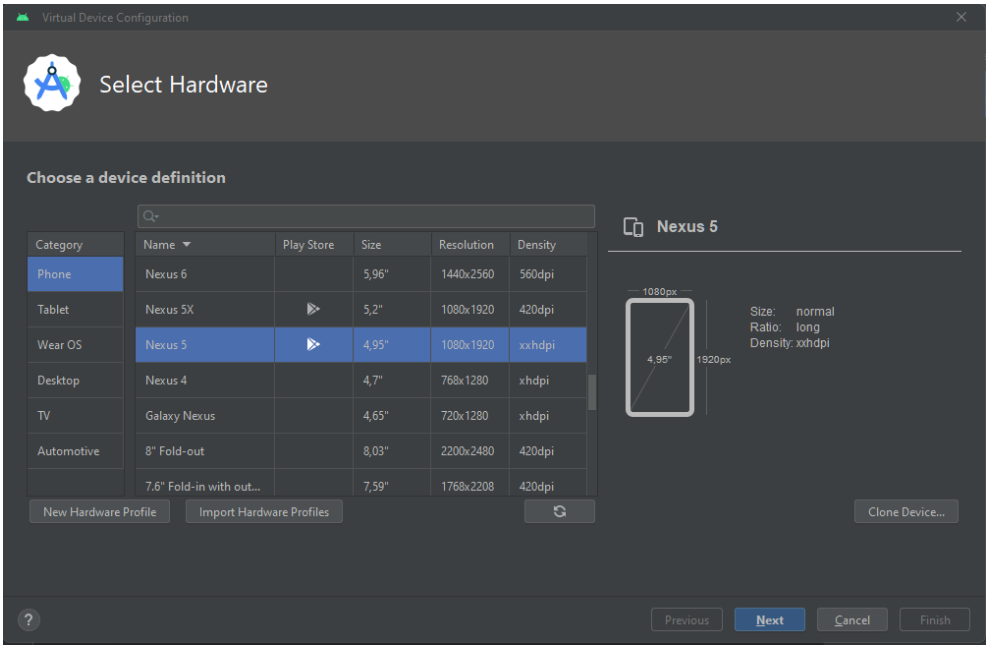


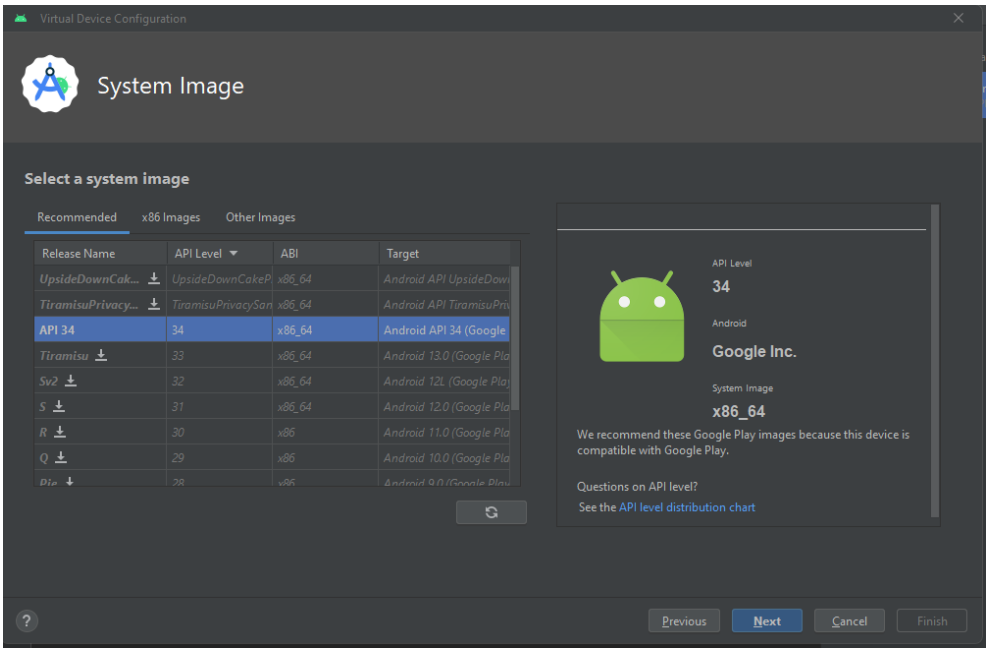
Once we have opened it we need to wait for Gradle to be built. Then, open the device manager, it is located in a tab on the left side of the IDE. If you can't see it it can be opened from the toolbar via **View** > **Tool Windows** > **Device Manager**. This is what it looks like.

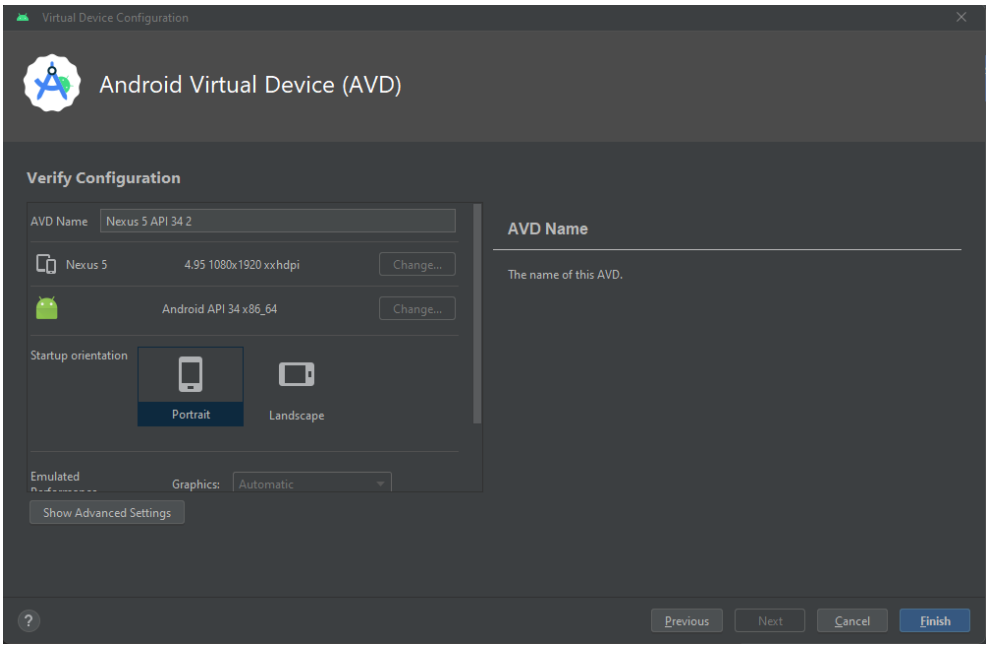


If you don’t have any device created you’ll need to create one to execute the project. To create a device just click on the **Create Device** button and follow the next images.

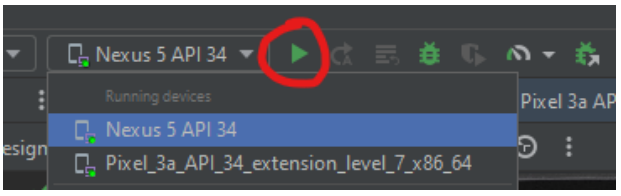


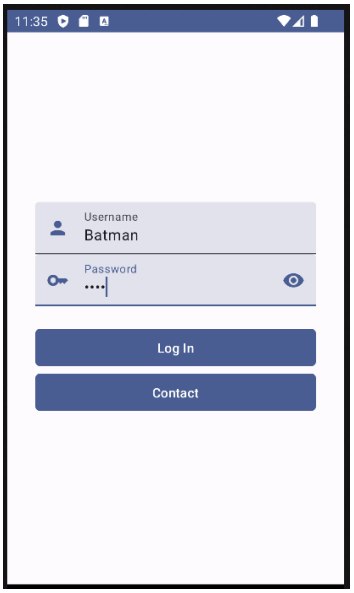






Click on the **Finish** button to create the device. Once the device is created we just need to select it and click the **Start** button.





The flow and navigation between the different activities is explained here.