ESAT INNOVATION SCHOOL | ESCUELA SUPERIOR DE ARTE Y TECNOLOGÍA

APP DEVELOPMENT FRAMEWORKS

Subject Name: Mobile Programming
Associated Module: UNIT 49
Professor: Rubén Blanco
Course: 2022/2023
Author/s: Ángel Artigas Pérez

Índice/Index

1.- Android vs iOS

1.1 Introduction

Two of the most prevalent mobile operating systems around the world are Android and iOS. Google developed Android as an open-source operating system, whereas Apple developed iOS as a closed-source operating system. Android provides greater flexibility and customization, while iOS is famous for its sophisticated appearance and user-friendly interface.

Both systems have their advantages and disadvantages. The following sections will compare various aspects such as available devices, market shares, IDEs and programming languages associated with the development of applications for these systems, and finally, the corresponding implementation of the same hypothetical app for both systems will also be explained.

1.2 Devices

Android is well known for its wide range of available devices. These are manufactured by various companies, offering users a plethora of choices in terms of design, features, and price points. iOS, however, is exclusively developed and used on Apple's devices like the iPhone, iPad, and iPod Touch.

This major difference causes the two operating systems to vary widely in several key aspects:

- **Customization:** Android provides a higher level of customization compared to iOS. Android users can personalise their device's home screen, choose different default apps, and even install custom ROMs to modify the operating system itself. iOS, on the other hand, has a more restricted customization environment, offering limited options for home screen layouts and app defaults.

- **App Availability:** Both Android and iOS offer an extensive selection of apps through their respective app stores (Google Play Store for Android and the App Store for iOS). However, the App Store is known for its strict guidelines, resulting in a more curated collection of apps, while the Play Store has a more lenient approach, leading to a larger variety of apps, including some that may not adhere to strict quality standards.

- **Security:** iOS is often considered more secure than Android due to its tightly controlled ecosystem. Apple rigorously reviews apps before allowing them into the App Store, minimising the risk of malware and other security threats. Android's open nature allows for greater flexibility but also increases the risk of malicious apps

slipping through the cracks. However, both platforms regularly release security updates to address vulnerabilities.

- **Integration and Ecosystem:** iOS provides a seamless integration with other Apple devices and services like Macs, iPads, iCloud, and Apple Watch, allowing for a cohesive ecosystem where devices can communicate and share data effortlessly. Android, while compatible with other devices, does not have the same level of integration and synergy within its ecosystem.

- **User Interface:** Android and iOS have distinct user interfaces. Android typically offers more customization options for users to tailor the look and feel of their device, while iOS provides a more uniform and polished interface with a consistent user experience across devices.

- **Pricing:** Android devices are available across a wide price spectrum, ranging from budget-friendly options to high-end flagship devices. This variety makes Android more accessible to a broader range of users. In contrast, iOS devices, particularly the latest models, tend to be more expensive, targeting the premium market segment.

The difference in the variety of devices available between Android and iOS also significantly impacts the development of applications for these operating systems. One of the concepts that is most accentuated due to this difference is called: “**fragmentation”**.

Android devices come in a wide range of screen sizes, resolutions, hardware capabilities, and software versions. This fragmentation poses a challenge for app developers as they need to ensure their applications work smoothly across various devices. Developers must consider different screen sizes, aspect ratios, and performance capabilities. This requires additional testing, optimization, and compatibility checks to ensure a consistent user experience across different devices.

In contrast, iOS devices have a more limited variety, typically developed and controlled by Apple. This relative uniformity simplifies the development process, as developers can optimise their apps for a smaller range of screen sizes and hardware configurations. They can focus more on delivering a consistent experience across iOS devices.

1.3 Market Shares & Revenue Models

Android has consistently held the largest market share globally among mobile operating systems. According to Statista reports, as of Q1 2023, Android accounted for approximately 71.4% of the global smartphone market share. This dominance is attributed to the wide range of manufacturers offering Android devices across different price points and the platform's popularity in emerging markets. **[0]**

On the other hand, iOS, which is exclusive to Apple devices, held a smaller but significant market share. As of 2021, iOS accounted for 27.9% of the global smartphone market share. Despite its relatively smaller share, Apple's iOS devices have a strong presence in developed markets, where they are known for their premium design, user experience, and loyal user base.

It's important to note that these figures represent the global market share. Regional variations exist, with different market dynamics and consumer preferences influencing the dominance of Android and iOS in specific countries or regions.

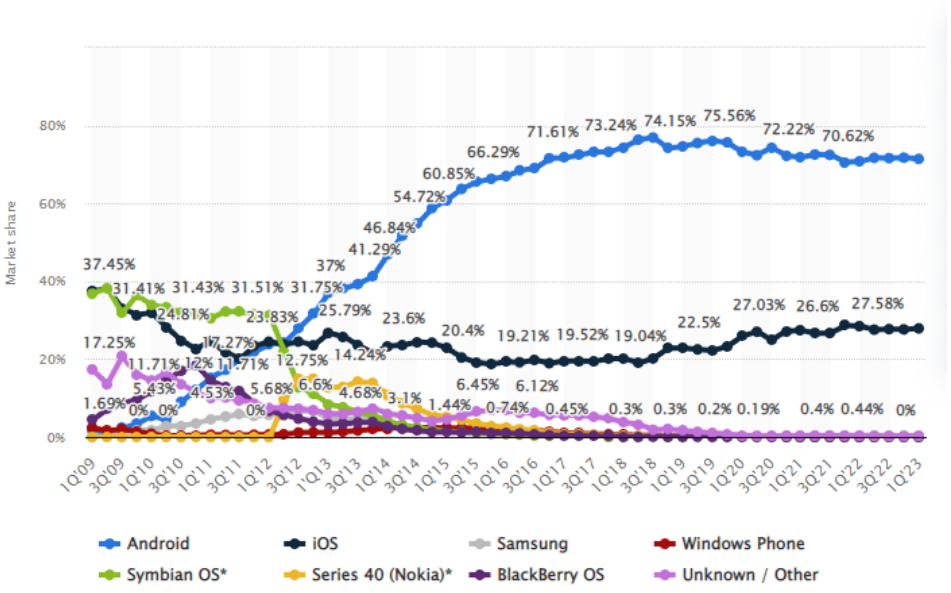


Fig.00 Statista - Mobile Market Share

While both Android and iOS support a range of revenue models, there are some differences in the prevalence and effectiveness of certain models on each operating system. Let's look at the revenue models that tend to dominate on Android and iOS:

Android Revenue Models:

In-App Advertising: Advertising is a popular revenue model on Android due to the larger user base and the availability of diverse ad networks. Ad-supported apps, including display ads, video ads, and native ads, are commonly seen on the Android platform. Developers can leverage various ad networks and integrate ads into their apps to generate revenue based on impressions, clicks, or conversions.

Freemium with In-App Purchases: The freemium model, where the app is initially offered for free with limited features and additional content available through in-app purchases, is prevalent on Android. Developers often provide a basic version of the app for free to attract users and offer premium features, virtual goods, or subscriptions as in-app purchases to unlock advanced functionality.

In-App Purchases for Virtual Goods: Android users are known to engage with games and apps that offer in-app purchases for virtual goods or currency. This revenue model works well for gaming apps where users can buy in-game items, power-ups, or extra lives. Virtual goods can also extend beyond games to apps like social networking platforms or productivity tools that offer additional features or customization options for purchase.

iOS Revenue Models:

In-App Purchases: In-app purchases are widely used on iOS and have been successful in generating revenue for developers. iOS users tend to have a higher propensity to spend on apps, making them more likely to make in-app purchases for premium content, additional features, subscriptions, or virtual goods. This model allows developers to offer a free app with enticing content and provide options for users to unlock enhanced experiences through purchases.

Paid Apps: The paid app model, where users pay an upfront fee to download the app, has traditionally performed well on iOS. Apple's App Store has historically seen a higher percentage of paid apps compared to Android's Google Play Store. iOS users are often willing to pay for high-quality apps and are perceived to have a higher willingness to purchase paid apps compared to Android users.

Subscriptions: iOS supports subscription-based revenue models, which have gained popularity in recent years. Developers can offer recurring subscriptions for access to premium content, services, or additional

features within their apps. iOS users are generally more accustomed to subscription-based models and have shown a willingness to pay for ongoing access to valuable content or services.

It's important to note that these revenue models are not exclusive to a particular operating system, and many apps adopt a combination of models to maximise revenue potential.

1.4 Versions

Let's first briefly review the release history of each operating system:

Android:

- **Android 1.0**: Released in September 2008, it was the initial version of Android.
- **Android 2.0 Eclair**: Released in October 2009, it introduced expanded device support, an improved browser, Exchange support, and a more polished user interface.
- **Android 3.0 Honeycomb**: Released in February 2011, it was a tablet-focused update with a redesigned user interface, enhanced multitasking, improved keyboard, and support for multiple cameras.
- **Android 4.0 Ice Cream Sandwich**: Released in October 2011, it introduced a unified interface design for both smartphones and tablets.
- **Android 4.4 KitKat**: Released in October 2013, it focused on optimising performance and memory usage, particularly for devices with lower specifications.
- **Android 5.0 Lollipop**: Released in November 2014, it brought significant visual and user interface changes with the introduction of Material Design.
- **Android 6.0 Marshmallow**: Released in October 2015, it emphasised improved battery life, app permissions, and system-level enhancements.
- **Android 7.0 Nougat**: Released in August 2016, it introduced split-screen multitasking, direct reply to notifications, and other user interface improvements.
- **Android 8.0 Oreo**: Released in August 2017, it included features like picture-in-picture mode, notification channels, and autofill framework.
- **Android 9 Pie**: Released in August 2018, it focused on gesture-based navigation, adaptive battery, and app actions.
- **Android 10**: Released in September 2019, it introduced a system-wide dark mode, enhanced privacy features, and improved gesture navigation.

- **Android 11**: Released in September 2020, it emphasised conversation notifications, device controls, and media playback enhancements.
- **Android 12**: Released in October 2021, it introduced a refreshed design, improved privacy settings, and enhanced customization options.
- **Android 13**: Released in August 2022, it improved security settings, and introduced spatial audio.

iOS:

- **iOS 1.0:** Released in June 2007, it was the initial version of iOS, originally known as iPhone OS. It introduced the core features of the iPhone, including the touchscreen interface, phone functionality, and built-in apps like Safari and Mail.
- **iOS 2.0:** Released in July 2008, it brought support for the newly introduced App Store, enabling users to download third-party applications for the first time. It also introduced features like push email, contact search, and the ability to install apps directly on the device.
- **iOS 3:** Released in June 2009, it introduced various new features and improvements, including cut, copy, and paste functionality, MMS support, Spotlight search, and voice control.
- **iOS 4:** Released in June 2010, it introduced significant features like multitasking, allowing users to switch between apps, as well as the introduction of folders to organise apps. It also brought FaceTime for video calling, iBooks for reading eBooks, and support for the high-resolution Retina display.
- **iOS 5:** Released in October 2011, it introduced several notable features, including the revamped Notification Centre, iMessage for free messaging between iOS devices, iCloud for seamless device synchronisation, and Siri, the voice-controlled virtual assistant.
- **iOS 6:** Released in September 2012, it introduced Apple Maps as a replacement for Google Maps, Passbook for storing digital tickets and loyalty cards, and improvements to Siri, including support for additional languages and expanded capabilities.
- **iOS 7:** Released in September 2013, it marked a significant design overhaul with a flatter and more vibrant visual aesthetic. It introduced features such as Control Center for quick access to settings, AirDrop for easy file sharing, and improved multitasking capabilities.
- **iOS 8:** Released in September 2014, it brought several enhancements, including interactive notifications, the Health app for tracking health and fitness data, Apple Pay for mobile payments, and improved continuity features for seamless integration between iOS devices and Mac computers.
- **iOS 9:** Released in September 2015, it focused on performance improvements, enhanced multitasking on iPad with Split View and

Slide Over, proactive Siri suggestions, and the introduction of the News app for personalised news content.

- **iOS 10:** Released in September 2016, it introduced a redesigned Lock screen with richer notifications, an expanded Messages app with stickers and effects, the Home app for controlling smart home devices, and improvements to Siri's capabilities.
- **iOS 11:** Released in September 2017, it brought a redesigned Control Center, a Files app for file management, augmented reality (AR) capabilities through ARKit, and the ability to send money through iMessage using Apple Pay.
- **iOS 12:** Released in September 2018, it focused on performance improvements and stability. It introduced features like Screen Time for managing device usage, grouped notifications, improved augmented reality experiences, and Memoji, customizable avatars.
- **iOS 13:** Released in September 2019, it introduced a system-wide dark mode, improved privacy features such as Sign In with Apple, enhanced photo editing capabilities, and performance optimizations.
- **iOS 14:** Released in September 2020, it brought home screen widgets, an App Library for organising apps, App Clips for quick app access, and improved privacy features, including more detailed app tracking permissions.
- **iOS 15:** Released in September 2021, it introduced features like Focus mode for managing notifications, redesigned FaceTime with new video and audio enhancements, enhanced privacy controls, and updates to Maps, Weather, and Messages apps.
- **iOS 16:** Released in June 2022, it brought privacy enhancements, and Siri news.

The variation of Android and iOS versions can have a significant impact on app development like the Android's device diversity explained before. Let's analyse some key considerations for both operative systems:

Compatibility Challenges: With numerous Android devices running different OS versions, developers must ensure their apps work seamlessly across a wide range of configurations. This requires extensive testing and optimization, as well as handling device-specific quirks and hardware limitations.

Feature Utilisation: New OS versions introduce innovative features and APIs that developers can leverage to enhance their apps. However, they must carefully balance incorporating these features while maintaining backward compatibility with older OS versions to reach a wider audience.

User Adoption: Android users tend to have slower adoption rates for new OS versions compared to iOS users. Developers must consider the market share of different OS versions and determine the minimum version to support, optimising their development efforts accordingly.

Fragmented Ecosystem: Android's fragmented ecosystem, with multiple device manufacturers and custom UI overlays, adds complexity to app development. Developers must account for these variations to ensure consistent user experiences across devices and OS versions.

Testing and Maintenance: Supporting multiple OS versions requires comprehensive testing to identify and resolve compatibility issues. Maintaining backward compatibility adds to the development and maintenance workload, requiring ongoing updates and support.

Development Resources: App developers must allocate resources to handle OS version variations, which may include additional testing devices, development tools, and expertise in managing diverse platforms.

1.5 IDEs

The corresponding Android and iOS integrated development environments (IDEs) are Android Studio and Xcode. Let's compare them:

Platform Compatibility:

- Android Studio: Android Studio is specifically designed for developing Android applications. It provides extensive tools, libraries, and emulators tailored for Android development.
- Xcode: Xcode is the IDE for iOS, macOS, watchOS, and tvOS app development. It is the official toolset provided by Apple and is tightly integrated with the Apple ecosystem.

Language Support:

- Android Studio: Android Studio primarily supports the Java programming language. However, it also offers support for Kotlin, a modern and officially supported language for Android development.
- Xcode: Xcode supports Swift, the primary programming language for iOS, macOS, watchOS, and tvOS development. Additionally, it also supports Objective-C, which is widely used in legacy iOS projects.

User Interface Design:

- Android Studio: Android Studio provides a layout editor that allows visual design of user interfaces using XML and drag-and-drop

functionality. It supports the Android XML-based layout system and provides a WYSIWYG editor for creating UI components.
- Xcode: Xcode includes Interface Builder, a visual editor for designing user interfaces using a storyboard or XIB files. It allows you to visually arrange UI components, set constraints, and define interactions.

Emulator and Simulator:

- Android Studio: Android Studio includes the Android Emulator, which allows developers to test their apps on various virtual Android devices with different configurations and versions of Android.
- Xcode: Xcode provides the iOS Simulator, which allows developers to test their apps on virtual iOS devices with different screen sizes and iOS versions.

Publishing and Distribution:

- Android Studio: Android Studio integrates with Google Play Store, allowing developers to easily publish and distribute their apps to Android users worldwide.
- Xcode: Xcode provides tools for submitting apps to the App Store, Apple's official marketplace for iOS, macOS, watchOS, and tvOS apps.

Developer Tools and Resources:

- Android Studio: Android Studio offers a wide range of developer tools and resources, including debugging tools, performance profiling, code templates, and the Android Developer documentation.
- Xcode: Xcode provides various tools like debugging, performance analysis, and Interface Builder. It also includes Apple's developer resources, such as documentation, WWDC videos, and the Apple Developer website.
-

1.6 Programming languages

Although there is more than one programming language available in each of the IDEs, I am going to focus on Swift and Kotlin as they are the languages I have used in this subject.

Swift:

- Swift is the primary programming language for iOS, macOS, watchOS, and tvOS app development in Xcode.
- Swift is a modern, safe, and expressive language developed by Apple.
- It offers features like type inference, optionals, generics, closures, and pattern matching, making code concise and readable.
- Swift has a strong focus on safety, with built-in memory management and protection against common programming errors.
- It is designed to work seamlessly with Apple's frameworks, providing access to a wide range of APIs and technologies.
- Swift has a growing community and regular updates, introducing new features and improvements.

Kotlin:

- Kotlin is a modern programming language developed by JetBrains, primarily used for Android app development in Android Studio.
- Kotlin is fully interoperable with Java, which means developers can use existing Java libraries and frameworks alongside Kotlin code.
- It offers features such as null safety, extension functions, coroutines, and improved syntax, enhancing developer productivity and code quality.
- Kotlin focuses on eliminating boilerplate code and providing concise and expressive syntax.
- It has excellent compatibility with the Java Virtual Machine (JVM) and Android Runtime (ART), making it seamless to adopt in existing projects.
- Kotlin has gained popularity due to its enhanced safety features, improved development experience, and strong community support.

Comparing Swift and Kotlin:

Syntax and Readability: Both Swift and Kotlin aim for readability and conciseness, employing modern language features to make code more expressive and maintainable.
Safety Features: Both languages prioritise safety, with built-in mechanisms to prevent common programming errors, such as null safety and type inference.
Interoperability: Swift is designed to work seamlessly with Apple's frameworks, while Kotlin maintains strong interoperability with Java and existing Android codebases.

Community and Ecosystem: Swift has a growing community and strong support from Apple, while Kotlin has gained traction within the Android development community and benefits from JetBrains' tools and resources.

2.- Android App Project

2.1 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.

2.2 Development

2.2.1 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.

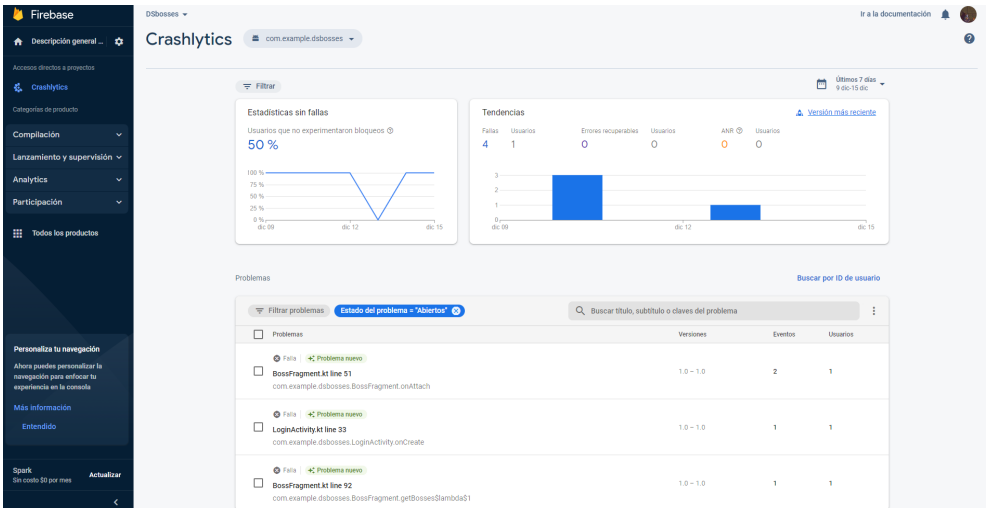


Fig.01 Crashlytics panel

2.2.2 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.

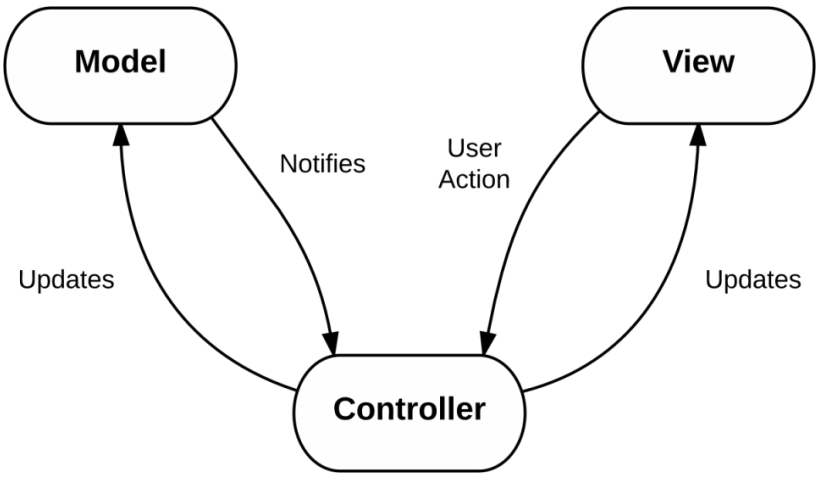


Fig.02 MVC Scheme [3]

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.

2.2.3 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.

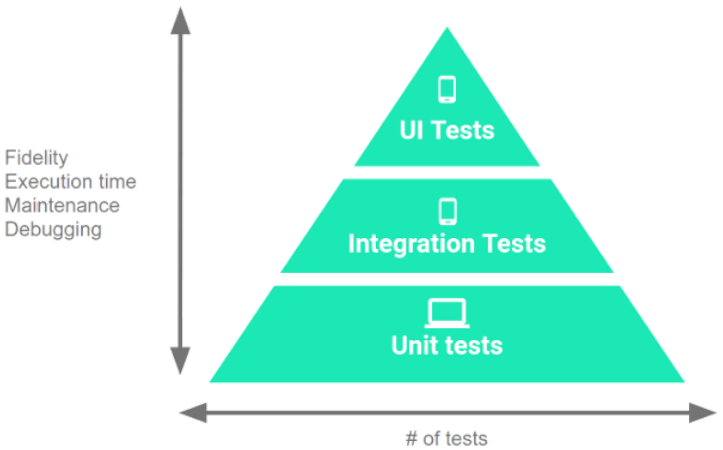


Fig.03 Tests Pyramid [4]

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.

2.3 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:

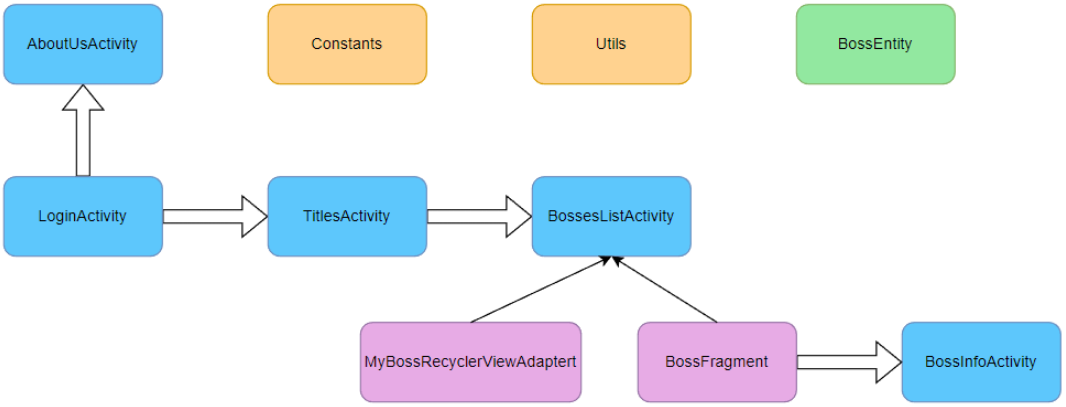


Fig.05 Class Diagram

2.4 UI/UX Flow Chart

2.4.1 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:

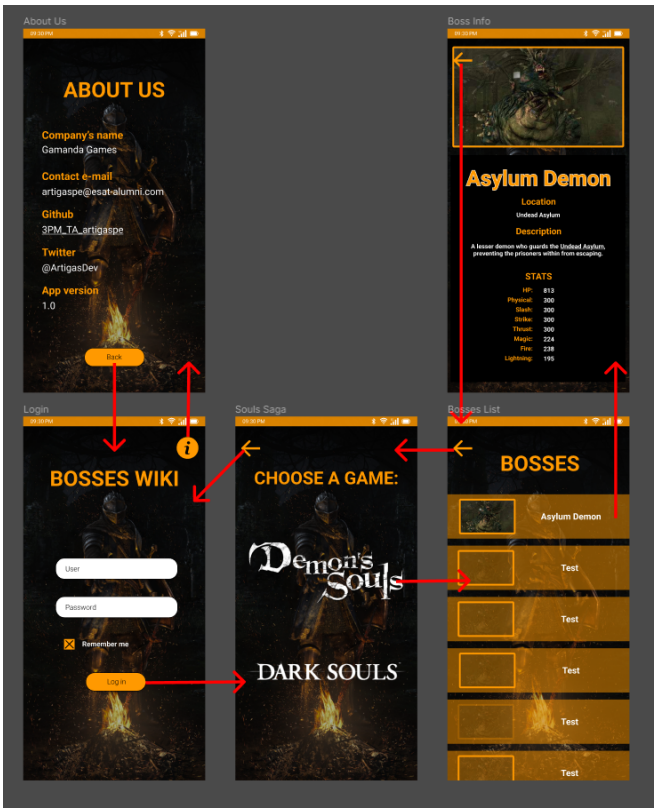


Fig.06 Figma design and flow

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

2.4.2 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

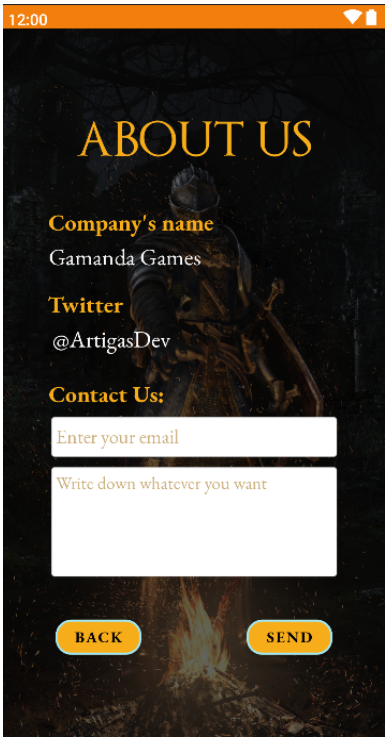


Fig.11 AboutUsActivity

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

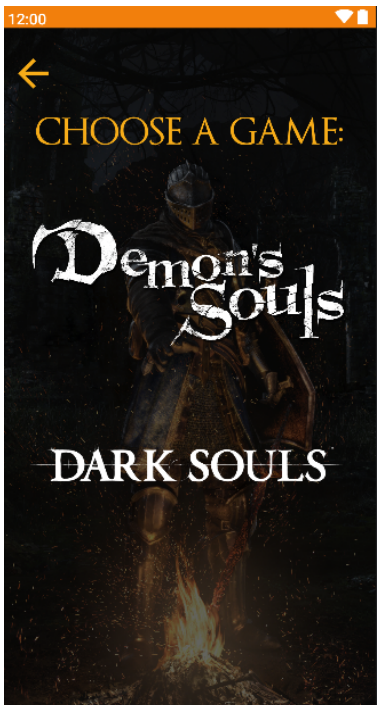


Fig.12 TitlesActivity

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

2.4.3 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:

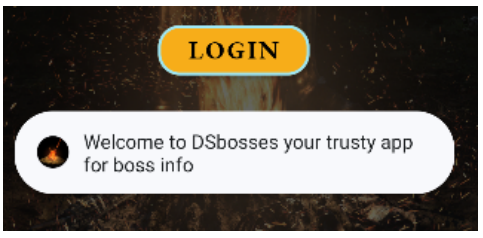


Fig.15 Welcome Toast

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

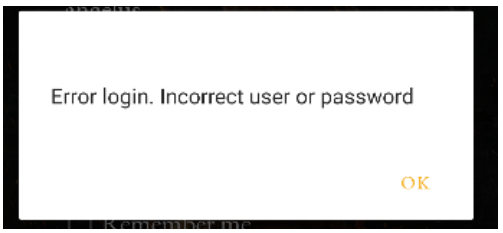


Fig.16 Error login AlertDialog

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:

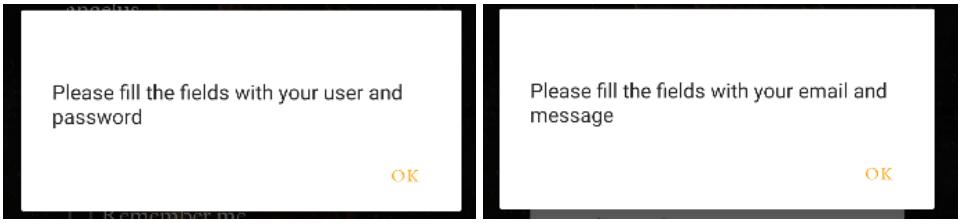


Fig.17 Empty fields AlertDialog

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

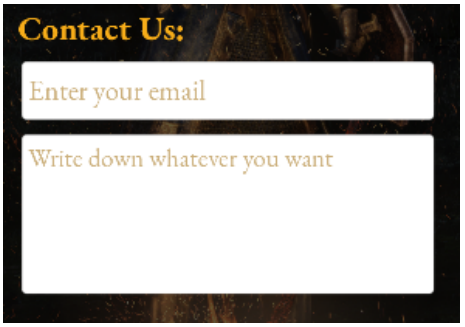


Fig.18 EditTexts hint

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.

2.5 Strategies and Solutions to the Development Problems

2.5.1 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.

2.5.2 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.

2.6 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

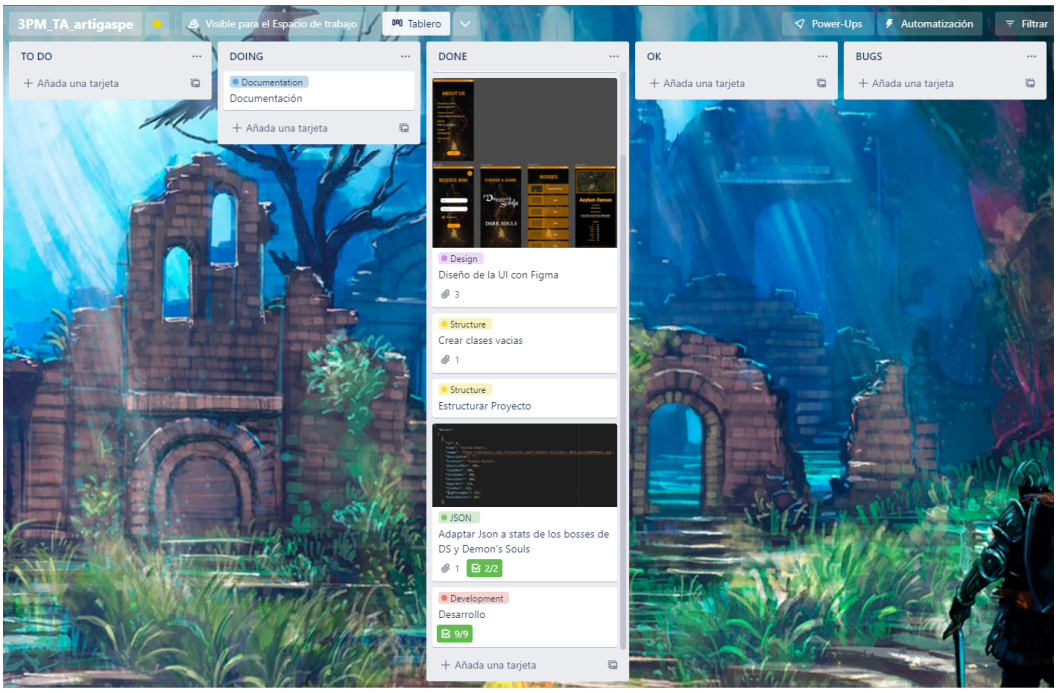


Fig.20 Trello Board

2.7 Short User Manual and Demo Examples

2.7.1 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]**

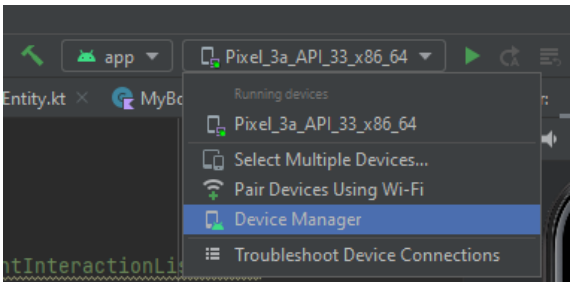


Fig.21 Device Manager

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

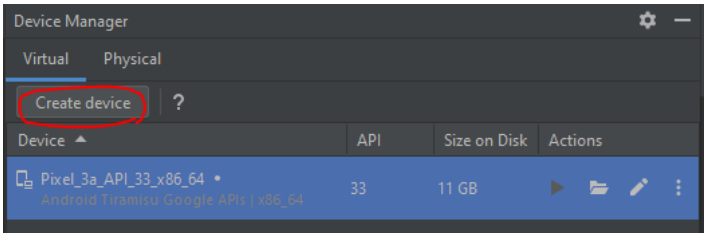


Fig.22 Create Device

Choose the device that best suits your needs:

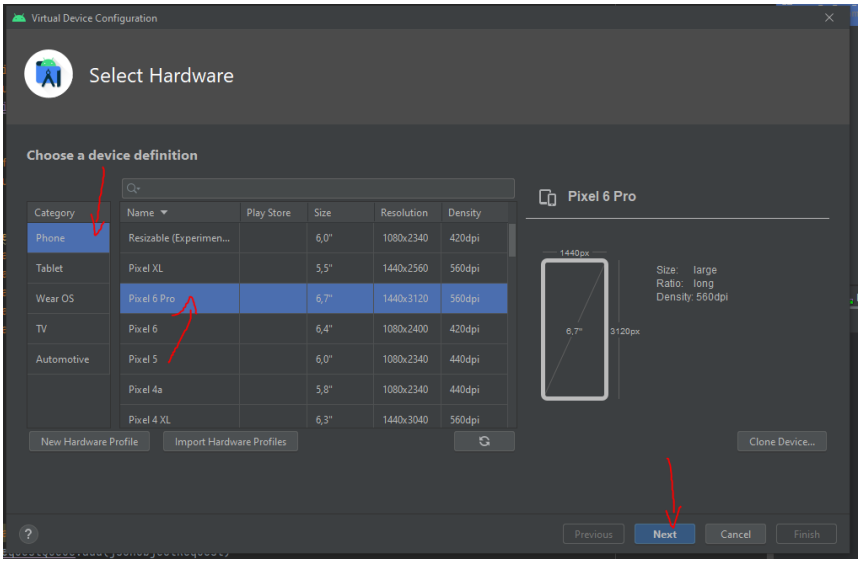


Fig.23 Choose device

Choose the API level desired for the virtual device:

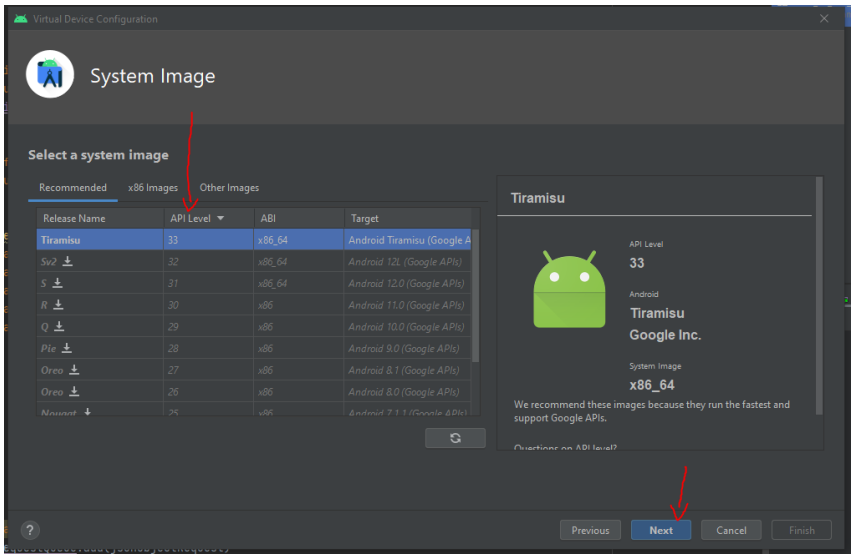


Fig.24 Choose API

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

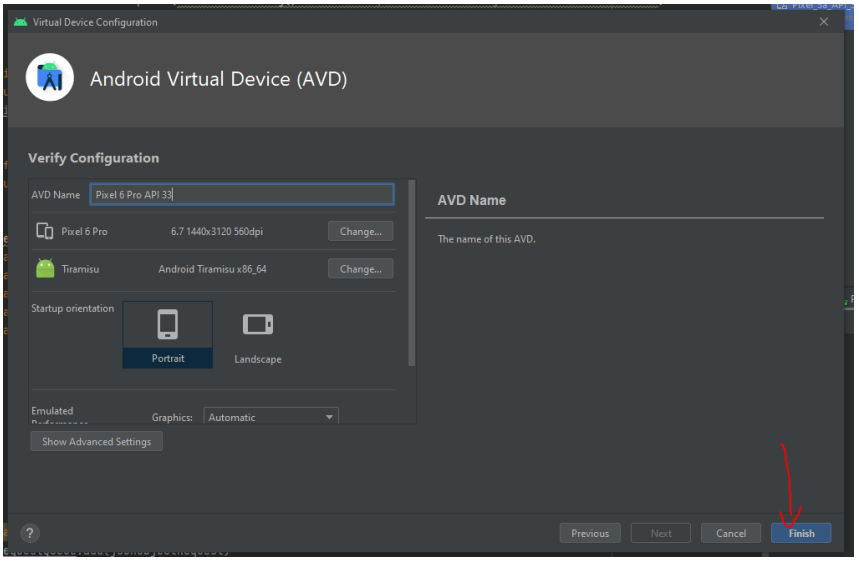


Fig.25 Creation of AVD Finished

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.

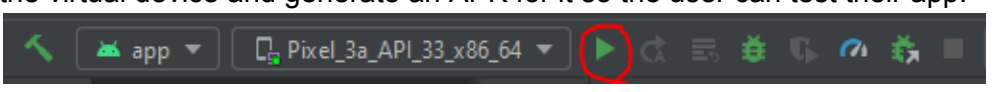


Fig.26 Run button

2.7.2 APK Debug Build

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

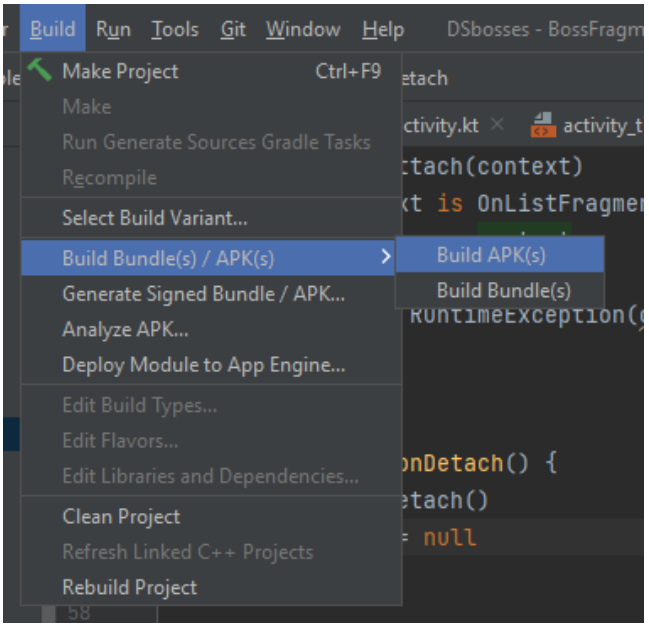


Fig.27 Build button

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

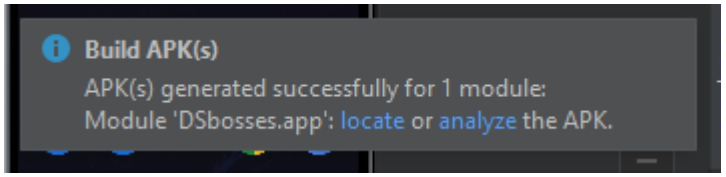


Fig.28 Build location message

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

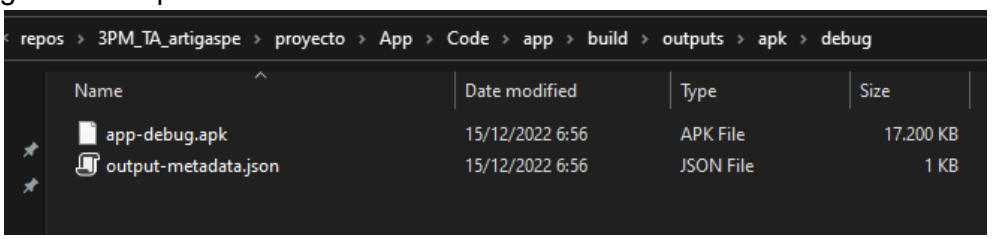


Fig.29 Build path

2.7.3 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:

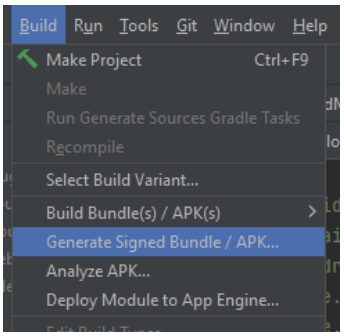


Fig.30 Signed generation

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

3.- iOS App Implementation

3.1 Introduction

In the previous section we have explained the features and structure that have been followed to develop an application in Android Studio from scratch. Then, in the following sections we will explain how the same application could have been developed, with the same features, but in the Xcode environment.

3.2 Activities & Life Cycles

The component corresponding to the Android Studio Activity is ViewController from Xcode.

Activities are an essential component in Android app development, representing a single screen with a user interface. They play a crucial role in handling user interactions, displaying UI elements, and managing the flow of the app. Each Activity has its own life cycle, consisting of various callback methods that are called at different stages of the Activity's existence.

The typical life cycle of an Activity includes methods such as onCreate(), onStart(), onResume(), onPause(), onStop(), and onDestroy().
These methods allow developers to perform necessary tasks during different stages, such as initialising UI elements, handling data persistence, and managing resources.

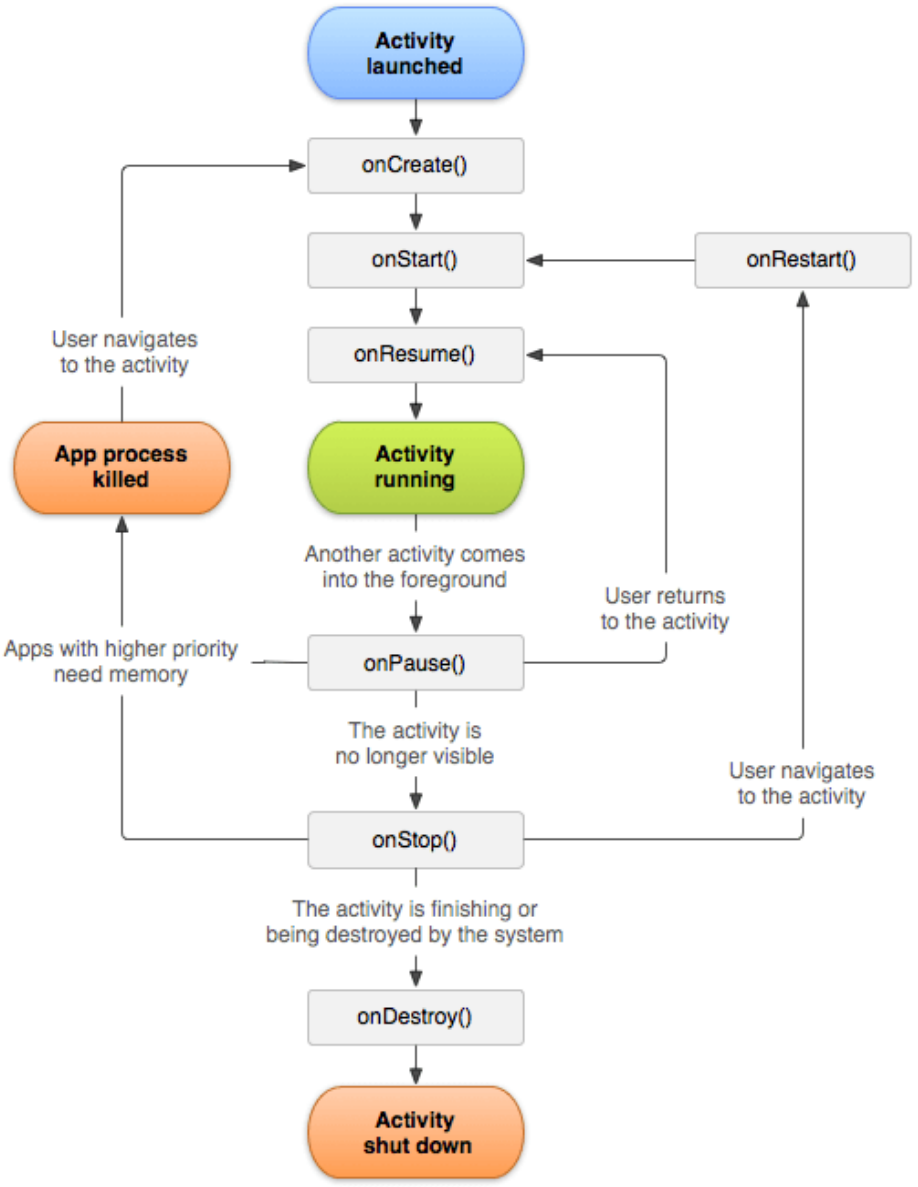


Fig.32 Activity Life Cycle

ViewControllers serve a very similar purpose in iOS app development, managing the visual presentation and user interactions of a single screen. They also control the logic behind the views and respond to events triggered by user interactions or system events.

ViewControllers also have a life cycle, which includes different methods for handling various stages of the ViewController's existence.
The life cycle of a ViewController includes methods like viewDidLoad(), viewWillAppear(), viewDidAppear(), viewWillDisappear(), and viewDidDisappear().

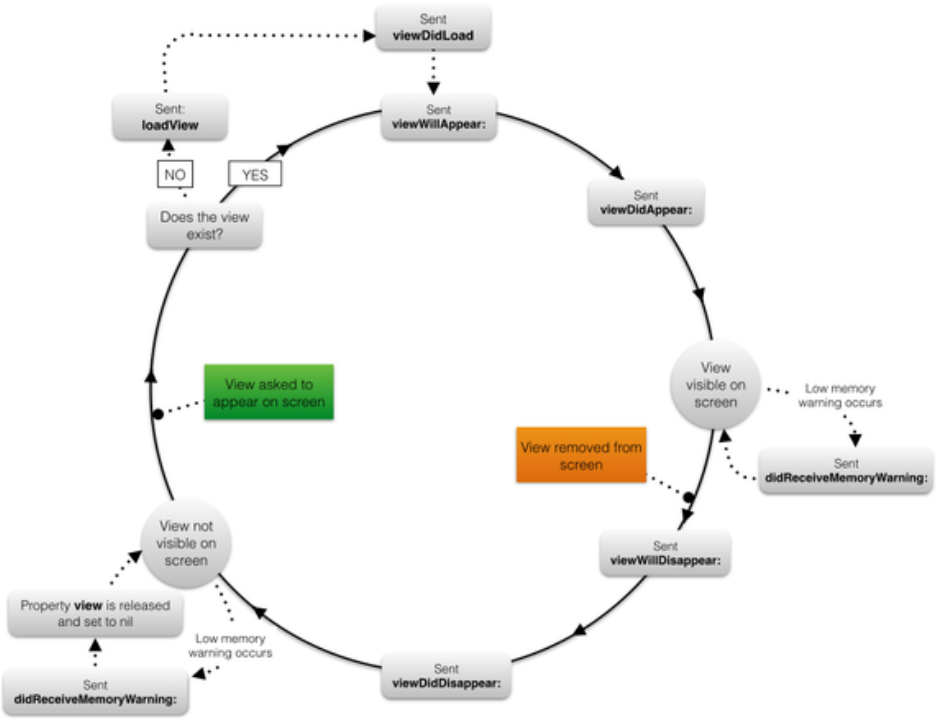


Fig.33 ViewController Life Cycle

3.3 In-App Navigation

The navigation of the application developed in Android was done through Intents. Intents are objects that facilitate communication between different components of an Android app and are particularly useful for starting new activities or passing data between them.

The navigation in Xcode would be achieved using Storyboards. These Storyboards provide a visual interface for designing app screens (ViewControllers) and their interconnections. The connections are made with Segues, which are visual connectors that define the navigation flow and associated transitions. Xcode also includes the Navigation Controller, which manages the navigation stack and provides a built-in navigation bar for easy backward navigation.

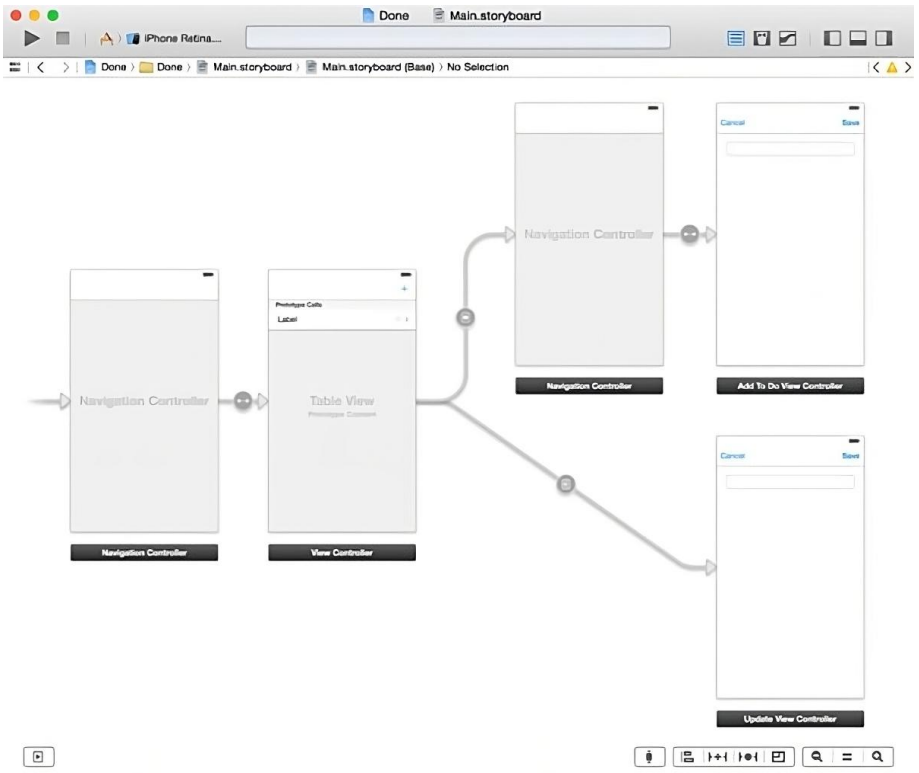


Fig.34 Storyboard

In addition to using segues, developers can navigate programmatically by instantiating and presenting ViewControllers manually, utilising methods like pushViewController() and presentViewController().

3.4 Data Persistence

For the application developed in Android Studio we made use of data persistence to preserve, for example, the user name logged into the application for a future session. This data persistence was achieved through the use of SharedPreferences.

In the corresponding implementation for Xcode we will have to make use of the so-called UserDefaults.

Let's analyse the following example implementation:

```swift
@IBOutlet weak var textInputText: UITextField!
@IBOutlet weak var saveButton: UIButton!

override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view.
    let email = UserDefaults.standard.string(forKey: Keys.email.rawValue)
    self.textInputText.text = email
}


@IBAction func saveEmail(_ sender: Any) {
    let email = self.textInputText.text
    UserDefaults.standard.set(email, forKey: Keys.email.rawValue)
    //print(self.textInputText.text)
}
```

Fig.35 Example UserDefaults

UITextField and UIButton would be the corresponding EditText and Button in Android Studio.

IBOutlet would be similar to "findViewById" from Android Studio. IBOutlet is a special keyword used in Xcode, specifically in the Interface Builder, to establish a connection between a UI element defined in a storyboard and the corresponding code in a View Controller class.

IBAction would be equivalent to "setOnClickListener" from Android Studio, and its use to associate a function to a UI element of the storyboard.

These two features are much more convenient to use in Xcode because the creation of these IBOutlet and IBAction is automatic by simply dragging the visual element of our ViewController to its associated code class.

The example code shown would make that after pressing a button the email written in a certain TextField is saved as a Key of our UserDefaults.

Once this button has been pressed the next time this ViewController is loaded (func viewDidLoad()), the TextField will show the email that is saved as Key in UserDefaults.

3.5 FragmentList

In one of the activities of the application developed in Android Studio a Fragment List was used. To achieve a similar structure with the views in

Xcode I would make use of an UITableView as a container and fill it with UITableViewCells.
UITableView is a view class provided by the UIKit framework in iOS development. It represents a scrollable list of items arranged in a single column. UITableView can display a large number of rows efficiently by reusing cells as they scroll off-screen. This element follows the delegate and data source pattern, where a delegate is responsible for handling user interactions, and a data source provides the content and layout information for the table.

UITableViewCell is a view class used as a reusable container for content in a UITableView. It represents an individual row or item within the table. Each UITableViewCell can contain multiple subviews, such as labels, images, or custom views.

UITableView reuses UITableViewCell instances as the user scrolls, improving performance and memory efficiency. Together they allow developers to create dynamic and interactive lists or tables within their iOS apps.

3.6 Unit Tests

As mentioned in the Android Studio implementation, the creation and use of tests is fundamental to the development process of any application. We will now look at what tools Xcode provides us with, and how we can perform a unit test.

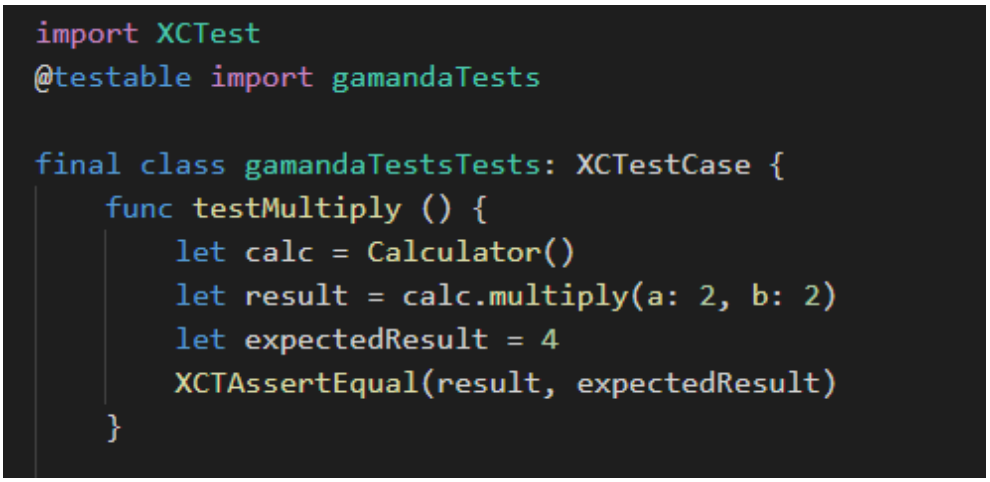
The primary tool for writing and executing unit tests in Xcode is the XCTest framework, which is built into Xcode and supports both Swift and Objective-C.

To create a unit test we must follow these steps:

1. **Create a Test Target:** In Xcode's Project Navigator, right-click on your project or target and select "New Target." Choose "Unit Test Bundle" under the "Test" category and give it a name.

2. **Write a Test Case Class:** Create a new Swift or Objective-C file and subclass it from XCTestCase. This class will contain your individual test methods.

3. **Define Test Methods:** Inside your test case class, create individual test methods by prefixing them with the word "test." These methods will define the specific tests you want to run.



```
import XCTest
@testable import gamandaTests

final class gamandaTestsTests: XCTestCase {
    func testMultiply () {
        let calc = Calculator()
        let result = calc.multiply(a: 2, b: 2)
        let expectedResult = 4
        XCTAssertEqual(result, expectedResult)
    }
```

Fig.36 Example Test

4. **Add Assertions:** Use assertions provided by XCTest, such as XCTAssertEqual, XCTAssertTrue, or XCTAssertFalse, to validate the expected behaviour of your code.

5. **Run the Tests:** Build your project and go to the Test Navigator in Xcode (Cmd + 6). Click on the Run button next to your test target to execute the tests.

6. **View Test Results:** Xcode will display the test results in the Test Navigator, indicating whether each test passed or failed. You can also view detailed logs and diagnostics for failed tests.

3.7 Additional Features

GSON:
On Android, in order to parse the JSON with the list of Bosses used to fill the fragmentList, we make use of GSON. To simulate this on Xcode we can use Codable.

Glide:
On Android, an external library called Glide has been used to load images from a URL. In Xcode we can achieve the same functionality by adding a dependency with KingFisher.

Volley:
On Android, in order to access an external API made with Beeceptor, Volley has been used. In Xcode we can achieve the same functionality by using Alamofire.

4.- Critical Conclusion

I really enjoyed having the opportunity to compare what it would be like to develop the same application for the Android and iOS markets.

The experience with Android Studio has been somewhat bitter. It is true that in my personal case I had already had to develop some applications using this IDE, so I was more used to it than the rest of my colleagues, but I would still say that going back to Android Studio has been a negative experience. The programming and coding part using Kotlin seems comfortable and intuitive to me, but the layout composition, either through xml or using the "graphical" part that Android Studio provides, seems exaggeratedly clumsy and old-fashioned to me.

In addition to this, during the months of development in Android Studio I have had more than one problem with Gradle, which has given me a feeling of slowness and very little stability, because with some minor error it can make it very difficult to reopen a project.

On the other hand, iOS development has been a pleasant surprise for me. I am totally in love with Xcode. On the programming and coding side, I would say that the experience with Swift has been quite similar to that with Kotlin, I've liked it a lot. However, as far as the visual part of Xcode is concerned, I found it a real marvel. The whole organisation of the Storyboard, the composition of the ViewController, and the ease of connecting content from the visual part to the code classes, not only seemed very intuitive to me, but they also give a feeling that they have been developed with the aim of making their use simple and satisfactory for the developer.

It is true that having to rotate each class with my colleagues in order to use Xcode due to the lack of Macs has slowed down my learning of the IDE, but I would still say that my experience with Xcode has been much sweeter than with Android Studio.

Finally, I am sure that this experience with both IDEs has been very useful to expand my knowledge about mobile development, which is a very important and in-demand market.

5.- Bibliography

https://chat.openai.com/ and https://bing.com/chat have been used for the purpose of seeking information and support the drafting of content for several of the **sections 1** and **3**.

0.
Global Market Share Held by Mobile. Online. 7 April 2023. [Accessed 31 May 2023]. Available from: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/

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