

Mobile Briefing

Course: Programación Mobile
Associated Module: App Development Frameworks
Teacher: Rubén Blanco
Year: 2023/2024
Author: Manuel Alcázar López



Index

1. - App Project.....	03
2. - UX Flow Chart	05
3. - Strategies and Solutions to the Development Problems.....	05
4. - Personal Tasks Plan.....	05
5. - User Manual	06
6. - Android vs Xcode.....	07
6.1 Android Studio	07
6.2 XCode	08
6.3 Comparison	10
6.4 Evaluation	14
6.5 Analysis	15
7. – Xcode Implementation Approach	15
Bibliography.....	



1. - App Project

The project is an app developed in Kotlin for Android, which shows a list of different songs and information about them, such as name, group, album and the image of the album. The project includes features like user login, browsing along the music list, changes between activities and app screens, and JSON parsing and web mocking to provide remote data access.

The structure of the project use the standard Android project layout, with the activities and fragments are divided in different folders, the network API classes and setup into another folder, the different UI elements and resources into their respective folders and all the classes related to the music and song, or the information that is displayed in the application is also in its own folder.

The main characteristics of the project are:

- **User Authentication:** with its own activity and resources for the login interface, although the user is not persistent, and its hardcoded; to access the user is "wllp" and the password is "a123456"
- **Browsing and Interaction:** achieved through the use of activities, fragments, view models, and recycler views; to display music and navigate through them.
- **Network mocking and JSON parsing:** using Retrofit dependency to handle network request, to fetch the music data for the application as a JSON Object, to be parsed into Kotlin code for the application.
- **Firebase Crashlytics:** to provide and test the application from possible crashes and exceptions, and to analyse the project.

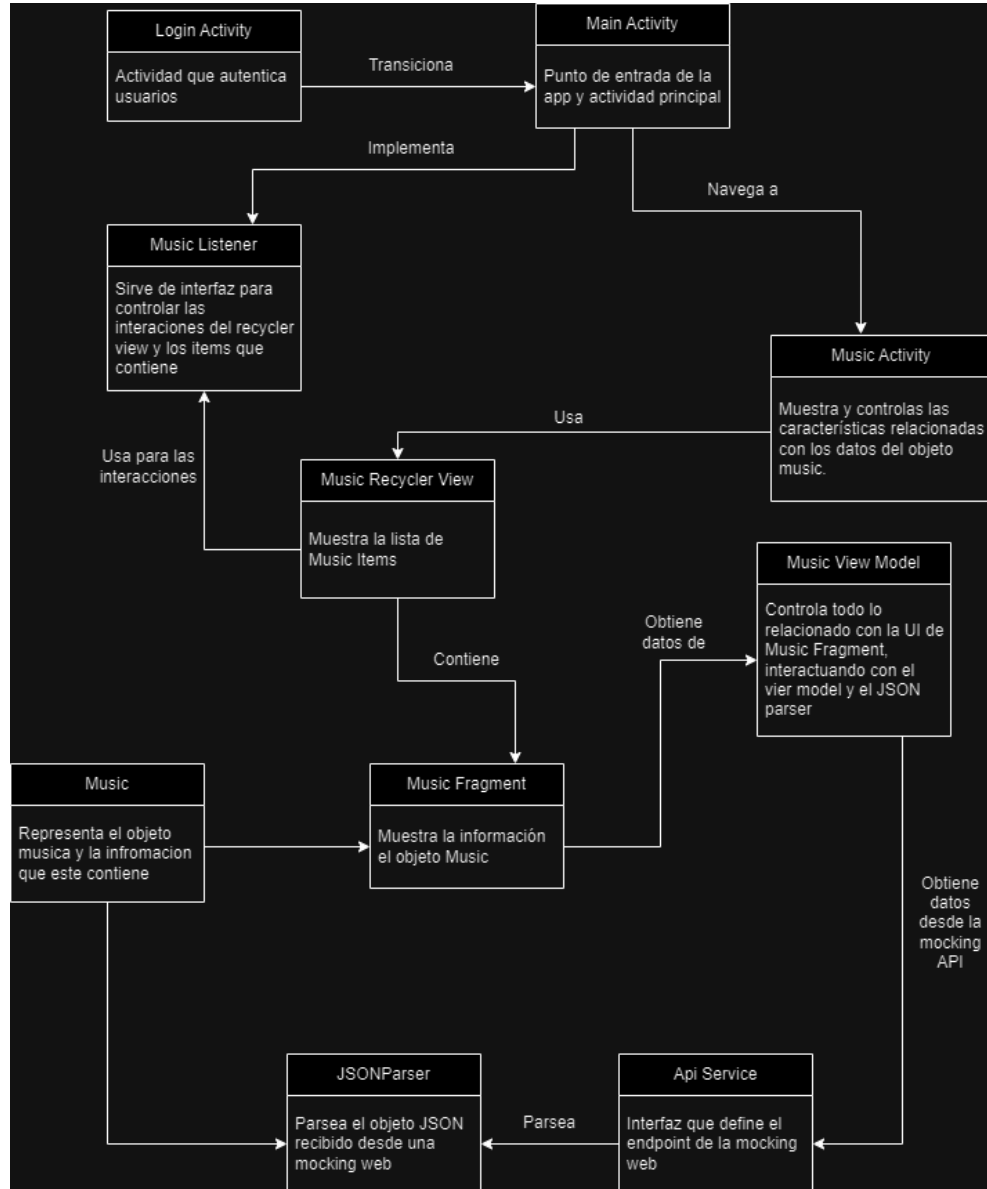
The project uses the ViewModel classes and the Activities and Fragments, which is a common development pattern in modern Android applications. This helps to separate the logic data from the UI logic and provides better handling of the project. Also the project includes extra dependencies using Gradle, for it is correctly functioning.

There is also an example test, which is typically used for testing the UI components and the interactions with the simulated or emulated devices, to ensure the app behaves as expected in a real-world environment.

Overall, the project demonstrates a well structured Android application with a clear separation of classes and tasks following the concerns of modern Android development practices. The structure is modular, making it easier to maintain and extend in a future, and the presence and use of test and Firebase indicates a focus on ensuring the app is reliable and functional.



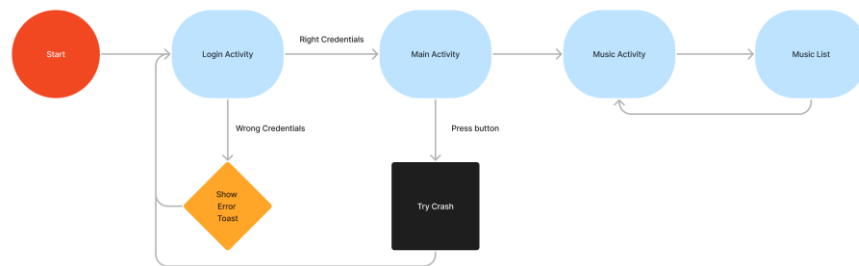
The project's classes have the following functionality and relationship:





2. - UX Flow Chart

The navigation through the application can be displayed like this:



3. - Strategies and Solutions to the Development Problems

There were several problems that have appeared during the development of the application. In first place, the CPU of my computer was incompatible with Intel HAXM, since I possess an AMD processor. To solve this problem I had to change the setup of the bios to be able to install the emulator accelerator. Even with the emulator accelerator, the IDE is in general slow and problematic. The UI can be overwhelming some times.

The main problem when developing the application is that I did lose the previous project I had made. And since there were a lot of time since I did not use the IDE and program in Kotlin, I basically had to learn everything almost from the beginning, fortunately Kotlin is easy to learn and I remembered some things that were key to develop the application, such as change the version SDKs, add dependencies, and how the activities interact with each other.

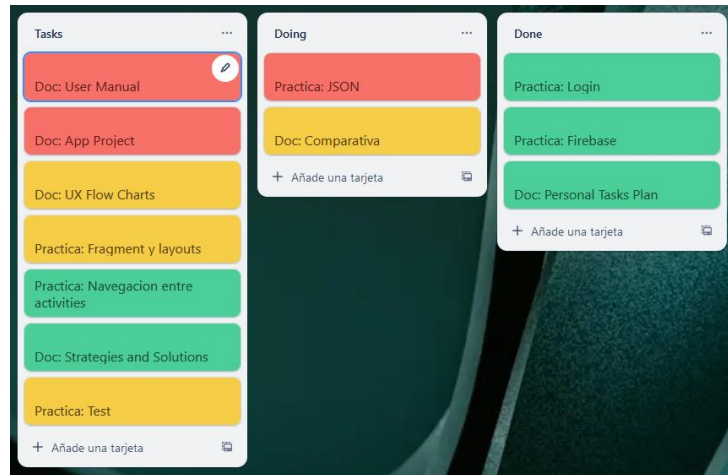
Related with this last one problem were the changes of the Android Studio's API versions. The new changes in Gradle dependencies made hard to add new modules and dependencies to the new SDK, since the way they are included, as the same as the name of the dependencies had changed from the last time I used Android Studio. Although, I finally managed to find the dependencies I was looking for, by changing the SDK version to a previous one, and then updating it again to change the dependencies to the new ones.

Finally, the main issue came from the JSON mocking and parser. The issue radiates in that I did not know what was the structure of a JSON list object, and the parser could not convert the data to be used by the application, but finally searching how this structure was, I was able to solve the problem, but it was obnoxious.

4. - Personal Tasks Plan

To accomplish the different stages the project possesses, this was divided into different tasks for a better managing and organizing. The tool used to create a personal tasks schema and follow the state of the stages is Trello.

The tasks are divided between the app tasks and the documentation tasks, and the color of the tasks reflects the amount of time it will take the task to be finished, being red the longest or hardest, and green the easiest.



Here is a link:

<https://trello.com/invite/b/OzHOIrmP/ATTle79d069da7fafd0ccd0fa3d673ab210fCF8C14B2/mobile>

5. - User Manual

To use the app in first place we need to meet certain system requirements to install Android Studio and open the project files. These requirements are:

- **Operating System:** Window, macOS, or Linux.
- **RAM:** minimum 8GB, recommended 16GB.
- **Storage:** minimum 10GB free space.
- **CPU compatible with Intel HAXM.**

Before we can open the project we need to setup Android Studio, we can find the installer from the official website, where we can download the installer files. Then we need to install Android Studio following the setup instructions of the installer.

Once is finished, we can launch Android Studio and open the project. When the project is already open we may need to synchronizne the Gradle files, we can do this going into the build.gradle.kts file located inside the src folder of our project, and click in "Sync Now" button displayed on the top of the build.gradle.kts file.

After syncing, we can build the project by navigating to "Build > Make Project" or pressing "Ctrl+F9" on Windows / Linux or "Cmd+F9" on macOS. This will start building the application, ensuring that there are no errors, if any error occurs the detail will be displayed in the "Build" window to resolve the current issues.

Once the project is built, we just need to run it using an emulator, but before we can run the application in an emulator we need to setup one, although we can use a physical device too. To set up an emulator we need to move to the Android Virtual Device (AVD) in the "Tools" section up in the IDE's editor. If we want to connect instead a physical device we need to activate the "Developer Options" of our smart phone and then the "USB debugging", which is a process that may vary from one device to another.

After everything is setted up, we can run out application by clicking in the "Run" botton in the top side of the editor or press "Shift+F10". Android Studio will install the application of the selected device or emulator and will lauch it automatically.



Upon launching the application, the login screen will be displayed, the credentials for authenticate the users are hardcoded ("wllop" for the user and "a123456*" for the password), and there is no data persistence in the application. If we enter the wrong credentials a small toast will appear showing us that the credentials are incorrect, and therefore we have no logged in. If we ente the correct credentials and logged correctly we will navigate directly to the main screen.

From this screen we can access to the displayed elements and browse and navigate through them, we can also tap on them to view more details. We also have the chance to crash the application, although this is use only for testing the Firebase Charslytics.

6. - Android vs Xcode

6.1 Android Studio:

Android studio is the official IDE (Integrated Development Enviroment) of Google, for its operating system Android. It's based on IntelliJ IDEA, for Android development and incorporates code editing and developer tools. Android Studio supports different features and tools such as:

- **Android Emulator.**
- **Composite.**
- **Gradle-based build system.**
- **ProGuard integration.**
- **Code templates.**
- **GitHub integration.**
- **Support for Android Wear apps.**
- **Java, Kotlin, and C++ support.**
- **Debugging.**
- **User Interface Desing.**

Android studio was announced on May 2023, and released in December 2014, as a replacement for Eclipse Android Development Tools as the primary IDE for native Android application development.

6.2 Xcode:

Xcode is the Integrated Development Enviroment created by Apple, to develop applications and software for Apple's different operating sytems and devices such as, macOS, iOS, iPadOS, watchOS, tvOs, and the new visionOS. Xcode was first released in 2003, and its latest version was released in September 2023.

It posses a wide set of features for developers:

- **Wide variety of programming languages.**
- **Emulators and simulators of the different Apple devices.**
- **Reality composer, for 3D and AR experiences.**
- **Fat binary or universal binary to ease transitions between different devices architectures and CPUs.**
- **Custom Machine Learning.**
- **Debugging tools,**
- **Intuitive UI desing.**
- **Cloud services.**
- **Profiling tools.**

As mentioned before Xcode can develop applications for all Apple devices, such as iPhone, iPad, MacBook, Apple Watch, or Apple Vision, among others. This is called Apple ecosystem, which is a term used to describe the Apple's digitals service and products, and the applications running on them.



Although Apple's ecosystem is often described as a walled garden, a term that refers to the complete integration of Apple's peripherals, products and services, but with no compatibility to different devices such as Android or Windows, immersing the users into the ecosystems, and designed to keep the users from leaving.



Apple ecosystem offers different advantages to users within the ecosystem, related to ease of use, interconnectivity, security and workflow productivity:

- **Seamless integration and communication:** the Apple devices are designed to work together, and cohesive between them. They share the different applications and services in real-time, creating a feeling of natural integration.
- **User friendly interface:** Apple products are characterized for their intuitive and consistent design, its soft learning curve for new users, and its ease of use.
- **Robust security:** Apple takes proactive measures for its users' security and privacy, providing hardware-level encryption, strict App Store regulations, security updates, software design focused on protecting user data and advanced protection and security control and tools.

Because Xcode is old it offers a lot of different tools and ways to develop applications for Apple devices. We can develop application using:

- **UIKit:** is the foundation of iOS UI, which provides a variety of features to build applications. It's the oldest UI framework and it has been the main core until SwiftUI. It has added features over the time like auto layout, dynamic types, drag-and-drop functionality or collection views to make it more powerful and user-friendly.

It relies on the use view controllers to manage UI views; these can be from buttons to the screen view itself. It's a mature and stable framework, with a wide range of UI components and extensive documentation and community support. Although, it's hard to maintain when the applications become complex and lacks of built-in support for modern design patterns.

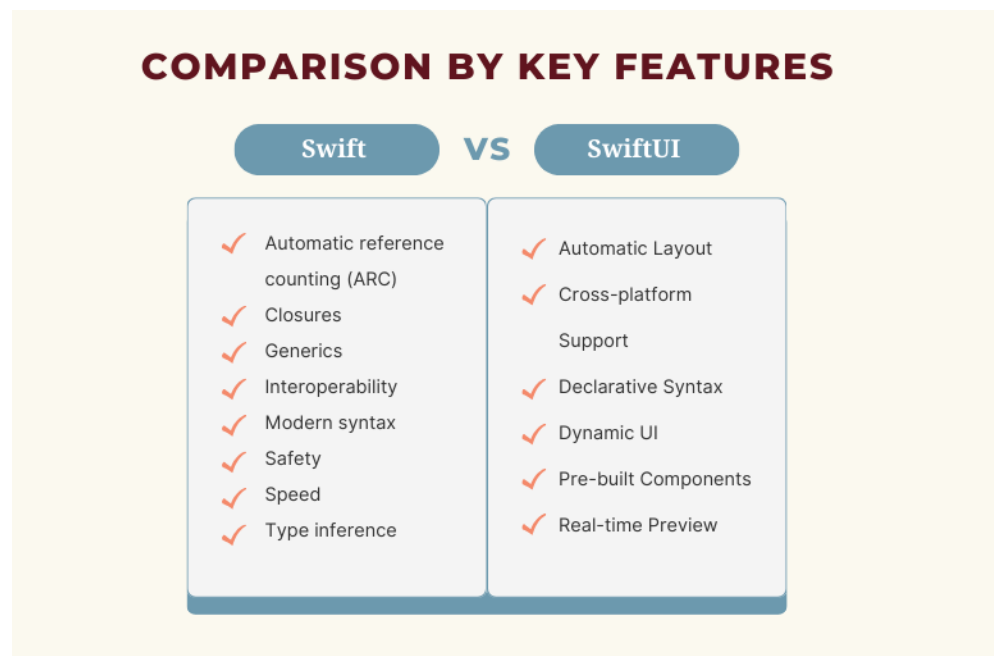


- **Swift:** is a modern, strong, high-level, high-performance programming language, characterized for its interoperability between C++ and Objective-C, its safety, is open source, and provides a fast and powerful runtime. It can work seamlessly across the different Apple's development frameworks, and possess a package manager tool for building, running, testing and packaging the different Swift library and executables.

But Swift possesses a limited ecosystem compared to other languages, and has limited version compatibility due to the continuous language updates and version and despite being easy to learn compared to other languages there is still a learning curve for new users.

- **SwiftUI:** is the newer framework created to replace UIKit, is mainly characterized for its declarative syntax that enables developers to create stunning applications. It can build spatial apps for new visionOS, it has interactive widgets, new chart types and interactivity, advanced animation control, and it also simplified the data flow, introduced a new MapKit API, and expanded watchOS API. It can work seamlessly with UIKit, AppKit, and Swift. It can also offer real-time preview and make emphasis on accessibility and cross-platform compatibility.

But since it is a relatively new framework, some of the features and functionalities are not as mature and comprehensive as those established in UIKit. It also does not possess backwards compatibility with older iOS versions, and despite being a versatile and cross-platform framework, some specific platforms still require UIKit and AppKit. In addition, SwiftUI is a new and growing ecosystem so third-party libraries and dependencies are not so extensive as those found in Swift and UIKit.



- **Storyboards:** Is a visual representation of user interface, which displays the screens' content and the connection between those screens. Storyboards allow managing different view controllers and seeing how they interact with each other, giving a complete look of the flow of the application. But it does not work well with big applications, it can't reuse views and are hard to debug.



- **Xib:** while storyboards are a representation of different screens or view controllers Xibs are the visual representation of a single view controller or screen. Xibs are the single representation of a view. Xibs can reuse views and code, they provide visual content and layout of a view, and allow more customization than storyboards. Xibs are having different disadvantages, also the idea is to isolate the different views of the applications they are still hard to debug, and for each views is necessary to create a new xib and class separately.
- **Dependencies:** Apple's framework possesses different package managers and third party dependencies and libraries for Swift, UIKit, SwiftUI, and AppKit to develop applications. The main package managers or dependency managers are:
 - **Swift Package Manager:** to manage the source code distribution, to make easy to share it and reuse others' code. It can directly compile and link Swift packages, dependencies, versions, and supports flexible distribution and collaboration models.
 - **CocoaPods:** a dependency manager written in Ruby for Swift and Objective-C, with more than 100 thousand libraries. CocoaPods manages app dependencies integrations and it can be accessed from any source code repository.
- **Simulators and Emulators:** a simulator mimics the behavior of a device copying the real world into a virtual environment to give an idea about how certain things would work. While an emulator duplicates the hardware and software features of real device to get a full imitation of the real device in a virtual environment to get a better idea of how would things work.

6.3 Comparison:

Both Android Studio and Xcode possess the common features present in any IDE (Integrated Development Environment) since both are used for developing applications for different platforms:

- **Debug Tools.**
- **Profiler Tools.**
- **Memory Analysis and Tracking Tools.**
- **Code Editing**
- **Device Emulators and Simulators.**
- **User Interface.**
- **API Documentation.**
- **Templates.**
- **Pre-built components.**

All of these development tools are high quality and helps developers to save time and effort. It also provides ease of use and better user experience when developing complex apps, that with all the tools provided can be profiled and optimized for specific platforms and allows developers to deliver bug-free applications.

But since both IDEs are targeting very different platform, their common tools provides different features and the IDEs also possess uncommon features and characteristics such as:

- **Platform:** as mentioned before they target different platforms, Android Studio is primarily used for developing applications for Android, while Xcode is focuses in Apple's devices such as iOS or macOS.



- **Programming Language:** one of the most significant differences between Android Studio and Xcode is the amount of programming languages they support. Android Studio supports primarily Kotlin and Java, along with C++, but Xcode supports Swift, C, C++, Objective-C, Objective-C++, Java, AppleScript, Python, Ruby, REXX, Ada, Go, Perl, D and C#, and other languages like GNU Pascal and Free Pascal.
- **User Interface:** both have different UI that provides different user experience. Android Studio uses the "LayoutEditor" a drag-and-drop layout editor for visually designing the user interface of the apps, allowing developers to quickly develop their applications. Xcode uses a combination of drag-and-drop, tools and code, called "Interface Builder", that allows more app development complexity in an easier way.
- **IDE Design Framework:** Android Studio and Xcode have a very different philosophy about how to structure the code and navigation of the apps. Android uses partition, the app is broken into activities and fragments. The activity is the equivalent to one app screen, since the application can possess multiple screens, each of the activities uses fragments, to navigate between the activities. Instead, Xcode uses different types of view controllers, which can control the entire screen or one part of it. These controllers can be managed in several ways, through code, images, or XML files and more.
- **Availability:** both IDEs are proprietary software, but Android Studio is available for Windows, macOS, and Linux, making it cross-platform. Xcode is only available for Apple devices. This makes a great difference, since Apple devices are expensive to obtain compared to the other devices, due to this, Android Studio is a more available and flexible option for developers.
- **Target Limitations:** there are different limitations related to the app distribution and emulation tools these IDEs provide. Xcode's emulators are limited to Apple devices, while Android Studio allows testing the apps in various Android devices with different operating systems and hardware profiles. Also, Android allows to distribute the apps through different channels, from Google Play to any other third-party app store, Xcode, instead, only allows distribution through the Apple's App Store.

In conclusion, both IDEs offer their pros and cons. The flexibility and accessibility of Android Studio allows for more market share and wider app distribution, but since there are a lot of different devices with different features with Android as the operating system, the developers may have issues while developing and distributing their apps. In Xcode, although it has more limited market share, and only one channel of app distribution, since it is focused on Apple devices, the development and optimization of the applications is easier.

Another important difference between Android and Xcode relies on security, Android applications are not encrypted, but obfuscated, and the operating system is also more vulnerable to cybersecurity threats, compared to iOS or Xcode applications which are encrypted.

Finally, it is also important to talk about the different market shares of both operating systems. According to the latest data of early 2024, Android has 70.69% of the total smartphone global market, dominating over the 28.58% of the global market share of iOS.

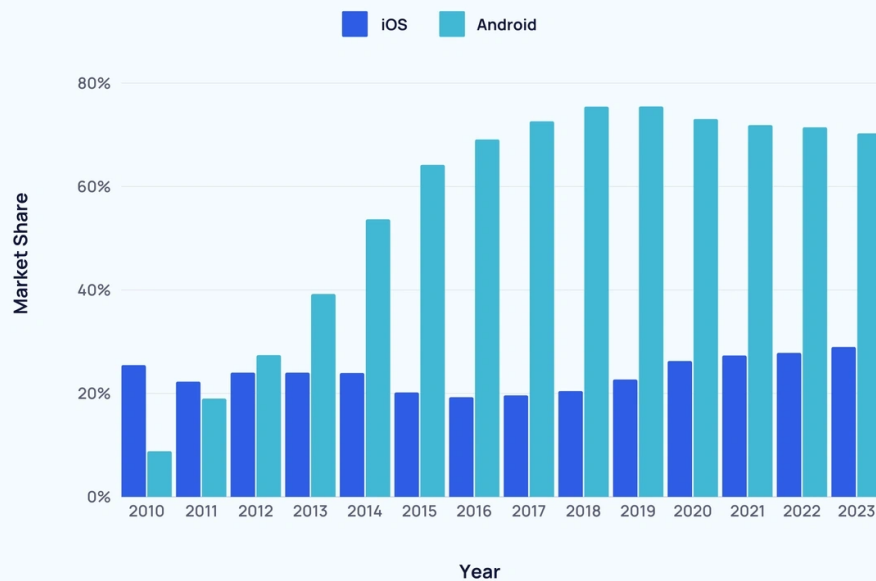


Android dominates the global smartphone market with a 70.69% share

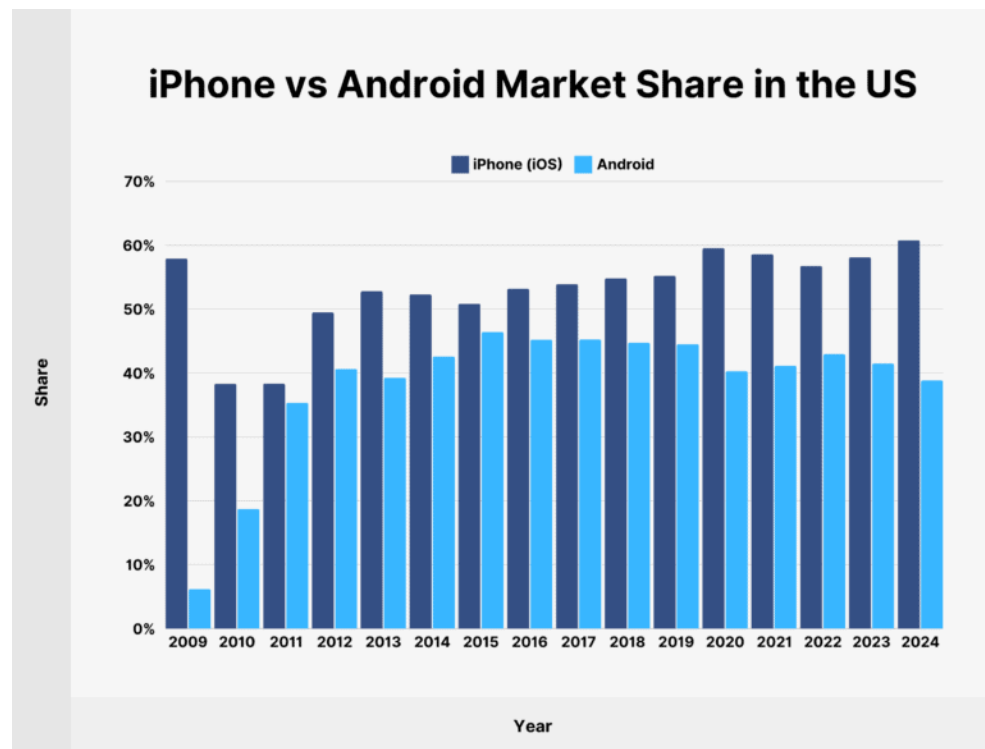


Since the release of the first iOS in 2009, Apple were more popular than Android until 2011, although by that time both systems were overshadowed by SymbianOS and Blackberry that controlled 35,49% of the mobile market share in 2009. By 2010 SymbianOS and Blackberry fell out in favor of iOS and Android, that became the two giants in the mobile space. In the year 2015 Android accounted half of the mobile market share with a 53,65% and continued its dominance until today.

iPhone vs Android Global Market Share



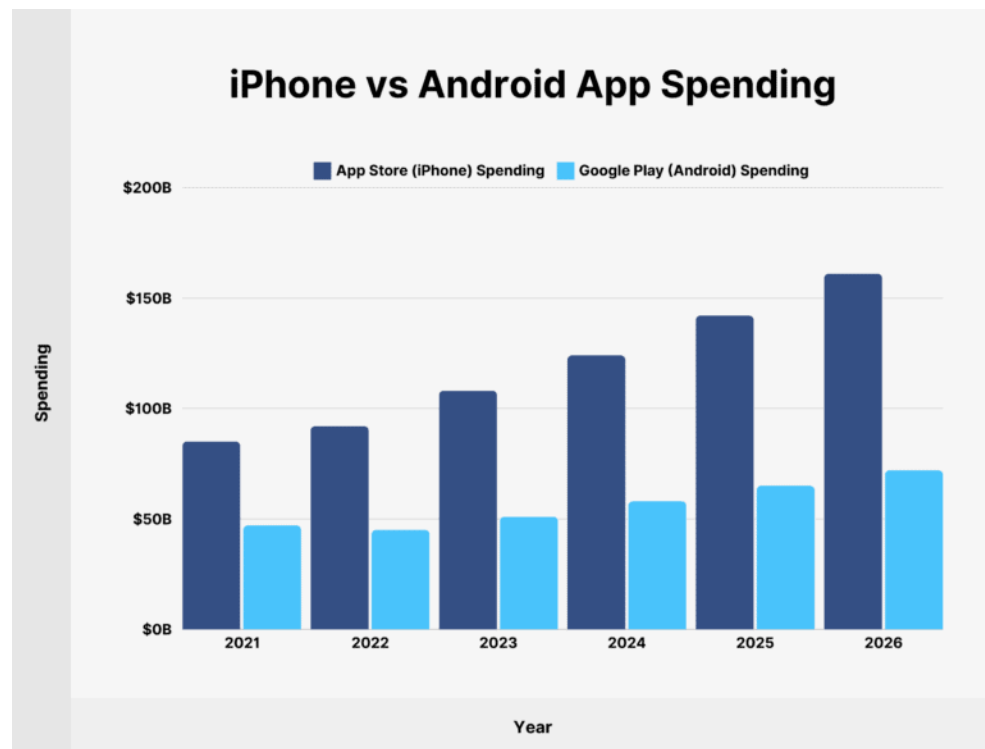
Although Android dominates the operating system mobile market share worldwide, iOS leads the sales in certain countries such as Japan (68,86%), Canada (60,31%), Australia (59,63%), United Kingdom (52,29%), and the most important United States (60,77%), where iOS has lead the market since its release in 2009.



Some demographic studies have shown interesting result about the difference between iOS users and Android users:

- iOS user tend to be higher earners than Android users. The popularity of both operating systems can be divided according to the country wealth and per capita revenue. iOS is more popular in richer countries while Android is more popular in the poorest countries.
- iOS users are also more active with their smartphones than Android users, and they feel more confident about their security and privacy than Android users.
- Worldwide, younger generations, such as Millenials and Gen Z, are more likely to iOS than older generations which prefer Android.

These data can help us to understand that, despite the difference between the market share globally between Android and iOS, being Android the most dominant mobile operating system. Still iOS generates more revenue per year than Android according to in-app purchases.



In 2021 consumers spent in the App Store \$85.1 billion in purchases, against the \$47.9 billion in Android. iOS is the winner when it comes to revenue, which seems strange because both devices possess similar free vs paid apps. iOS' apps are 95% free and 5% paid. Android's apps are 97% free and 3% paid. And on average iOS users spend \$12.77 per app against the \$6.19 of Android users. This could suggest that iOS users are more willing to spend money in apps than Android users, and if we take into account the previous demographics study we can all conclude that since iOS users are higher earners will also have more money to spend in app purchases.

6.4 Evaluation:

The environment used for the development is Android Studio for its availability and ease of use. The application is developed according to the Android Studio's framework, using its main characteristic of code partition using activities and fragments. The project uses extra modules and dependencies such as:

- **Firebase Crashlytics:** mainly used to test possible crashes in the application.
- **Retrofit:** for JSON parsing and web mocking to load data into a list.
- **Glide:** for image load and managing for Android.

These dependencies also use different frameworks and API's. Although the application is scalable in the long term, the different updates of the Android Studio's API, makes the project uneasy to maintain in the long term, since the changes from the version of Android Studio are so different, some dependencies and code parts need to be changed from one version to another, which is an issue for developers once the application has become complex and bigger.



6.5 Analysis:

The using of Android Studio has been a trouble since the beginning. My computer has an AMD processor that is not compatible with Intel HAXM, used for accelerating android device emulators, which has make the development process slow. Also, Android Studio consumes a lot of resources, and has the same as before, the lack of RAM, has also make the development slow.

Also, Android Studio's user interface can be overwhelming if do not have the knowledge. Overall, the development with Android Studio has been slow and obnoxious. This project has consumed more time than expected, because the code is neither complex nor large.

7. – Xcode Implementation Approach

As we saw before Xcode uses different tools from Android to develop applications, but the logic of the application is likely the same. Since we are unable to make a direct implementation in Xcode for iOS, due to the hard obtainability of Apple devices, will make an aproximetly implementation based on the current project develop on Android.

1. Setup Project:

In first place we need to setup the different necessary dependencies, libraries and assets used to develop the application.

- **Placeholder Image:** in Android we had a placeholder image stored in the folder drawables, used as a placeholder when an image from the mocking API is not found or wrong. In Xcode we can store images in the folder Assets.xcassets by simple drag-and-drop the image into the folder, similar as we do in Android.
- **Firebase:** to add Firebase to out project we need first to register our project into Firebase, and then we need to configure Firebase and download its SDK to make use of it into our application. Once finished, we just need simply to initialize Firebase into our app. These steps are similar than those made in Android.
- **JSON Parser:** Swift posses by default in its standard library a JSON encoder and decoder to parse JSON object coming from a mocking API, so no extra libraries or dependencies are needed for this step.
- **Mock API:** for the HTTPS request handling we need to use third party libraries like Mockingjay that can be download from Swift Package Manager or CocoaPods

Once the project is setted up we can start implementing the logic of the application in the different classes.

2. Login Activity:

To implement a similar login activity from our Android project into Xcode using SwiftUI, we need tyo understand first that in SwiftUI everything are view controllers, then we can use the declarative synthax of SwiftUI to change the body of our view controller with additional view controlers or with the different options these posses, to add all the style, logic and desing, to get our desired approach.

In this case, we create a first view controller called LoginView that will contain antoher view controller called "body". Inside the body view we can add a VStack to order the elements stored inside in a vertical layout.



Inside the VStack we now add a text to display a “Login” text for the users and two Fields one TextField for user and another for password, this last one as a SecureField to provide security to the user’s password, and get user’s input and compare against the hardcoded values we have to check if the user is right or wrong. We can also add the different styles and desing for these views using the different options we have such as padding, font size, background, margins, size of the views and much more.

We need also to add a button, which will check once clicked, if the input from the user and the hardcoded user and password are correct, if they are correct, will move to the next screen using a NavLink. If the password is wrong we will show a toast message that displays the given values are wrong.

To display a toast message we also need to create a new view controller. Inside the toast view we can modify its body, to display a message to the user and even add animations of displaying and disappearing, with a given duration, using onAppear, and Text for both display the message and set the duration of the toast once displayed using the DispatchQueue in Xcode. Or we can also add an Alert to the button to display an alert message instead of creating our own toast.

3. Main Activity:

In this view we need to add a button inside the view’s body, to throw an exception that will be caught by Firebase Crashlytics. We can throw an exception in Xcode using fatalError inside the button’s action scope.

4. Music Activity:

5. Music Framework:

6. Mocking API and JSON Parser:



Bibliography

CONTRIBUTOR, TechTarget. What is Android Studio?: Definition from TechTarget. *Mobile Computing* [online]. 23 January 2023. [Accessed 18 May 2024]. Available from: <https://www.techtarget.com/searchmobilecomputing/definition/Android-Studio>

Android Studio. *Wikipedia* [online]. 16 May 2024. [Accessed 18 May 2024]. Available from: https://en.wikipedia.org/wiki/Android_Studio

What is Xcode: Features, installation, uses, advantages and limitations. *BrowserStack* [online]. 25 April 2023. [Accessed 18 May 2024]. Available from: <https://www.browserstack.com/guide/what-is-xcode>

Xcode. *Wikipedia* [online]. 14 May 2024. [Accessed 18 May 2024]. Available from: <https://en.wikipedia.org/wiki/Xcode>

5 major differences between IOS and Android App Development. *EGO Creative Innovations - we design apps that your users love* [online]. [Accessed 18 May 2024]. Available from: <https://www.ego-cms.com/post/5-major-differences-between-ios-and-android-app-development>

Android Studio vs. Xcode – Pros and cons of each application development platform. *iubenda* [online]. [Accessed 18 May 2024]. Available from: <https://www.iubenda.com/en/help/127491-android-studio-vs-xcode-pros-and-cons-of-each-application-development-platform>

Android Studio vs xcode: What are the differences? *StackShare* [online]. [Accessed 18 May 2024]. Available from: <https://stackshare.io/stackups/android-studio-vs-xcode>

iPhone vs. Android User & Revenue Statistics (2024). *Backlinko* [online]. 13 March 2024. [Accessed 27 May 2024]. Available from: <https://backlinko.com/iphone-vs-android-statistics>

HOWARTH, Josh. iPhone vs Android User Stats (2024 data). *Exploding Topics* [online]. 6 December 2023. [Accessed 27 May 2024]. Available from: <https://explodingtopics.com/blog/iphone-android-users>

PAWAR, Ritu. The power of Apple Ecosystem: How it transforms your organisation. *iPlanet APP Digital* [online]. 10 October 2023. [Accessed 27 May 2024]. Available from: <https://iplanet.one/blogs/enterprise/the-power-of-apple-ecosystem-how-it-transforms-your-organisation>

ELALJ, Simo and SIMO ELALJ. The apple ecosystem: Is it really that good? *RefurbMe* [online]. 16 May 2024. [Accessed 27 May 2024]. Available from: <https://www.refurb.me/blog/apple-ecosystem-everything-you-need-to-know>

Apple Ecosystem. *Wikipedia* [online]. 21 April 2024. [Accessed 27 May 2024]. Available from: https://en.wikipedia.org/wiki/Apple_ecosystem

UIKit. *Apple Developer Documentation* [online]. [Accessed 28 May 2024]. Available from: <https://developer.apple.com/documentation/uikit>

Swiftui vs UIKit: The best choice for IOS in 2024. *Sendbird* [online]. [Accessed 28 May 2024]. Available from: <https://sendbird.com/developer/tutorials/swiftui-vs-uikit>



INC., Apple. Swift. *Apple Developer* [online]. [Accessed 28 May 2024]. Available from: <https://developer.apple.com/swift/>

Swiftui. *Apple Developer Documentation* [online]. [Accessed 28 May 2024]. Available from: <https://developer.apple.com/documentation/swiftui/>

INC., Apple. SwiftUI overview - xcode - apple developer. *Overview - Xcode - Apple Developer* [online]. [Accessed 28 May 2024]. Available from: <https://developer.apple.com/xcode/swiftui/>

Cocoa application competencies for IOS. *Storyboard* [online]. 6 April 2018. [Accessed 28 May 2024]. Available from: <https://developer.apple.com/library/archive/documentation/General/Conceptual/Devpedia-CocoaApp/Storyboard.html>

Locking views in storyboard and Xib Files. *Apple Developer Documentation* [online]. [Accessed 28 May 2024]. Available from: <https://developer.apple.com/documentation/xcode/locking-views-in-storyboard-and-xib-files>

ALVAREZ, Pedro. Storyboards X xibs x viewcode: Which one is better? *Medium* [online]. 6 May 2022. [Accessed 28 May 2024]. Available from: <https://pedroalvarez-29395.medium.com/storyboards-x-xibs-x-viewcode-which-one-is-better-6e0c7a24b867>

APPLE. Swift-package-manager/documentation at main · Apple/Swift-package-manager. *GitHub* [online]. [Accessed 28 May 2024]. Available from: <https://github.com/apple/swift-package-manager/tree/main/Documentation>

TEAM, CocoaPods Dev. CocoaPods.org. *CocoaPods.org* [online]. [Accessed 28 May 2024]. Available from: <https://cocoapods.org/>

JAIN, Jigna. Emulation vs. simulation- Android, IOS. *LinkedIn* [online]. 2 January 2022. [Accessed 30 May 2024]. Available from: <https://www.linkedin.com/pulse/emulation-vs-simulation-android-ios-jigna-jain>

