Asignatura 2019-2020

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

8. – Post Mortem 27

Bibliography 28

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 actrivities and fragments are dividev in different folders, the netwok 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 charactersitics of the project are:

- **User Autentication:** with its own activity and resouces for the login interface, although the user is not persistant, and its hardcode; to access the user is "*wllop*" and the password is "*a123456**"

- **Browsing and Interaction:** achieve 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.

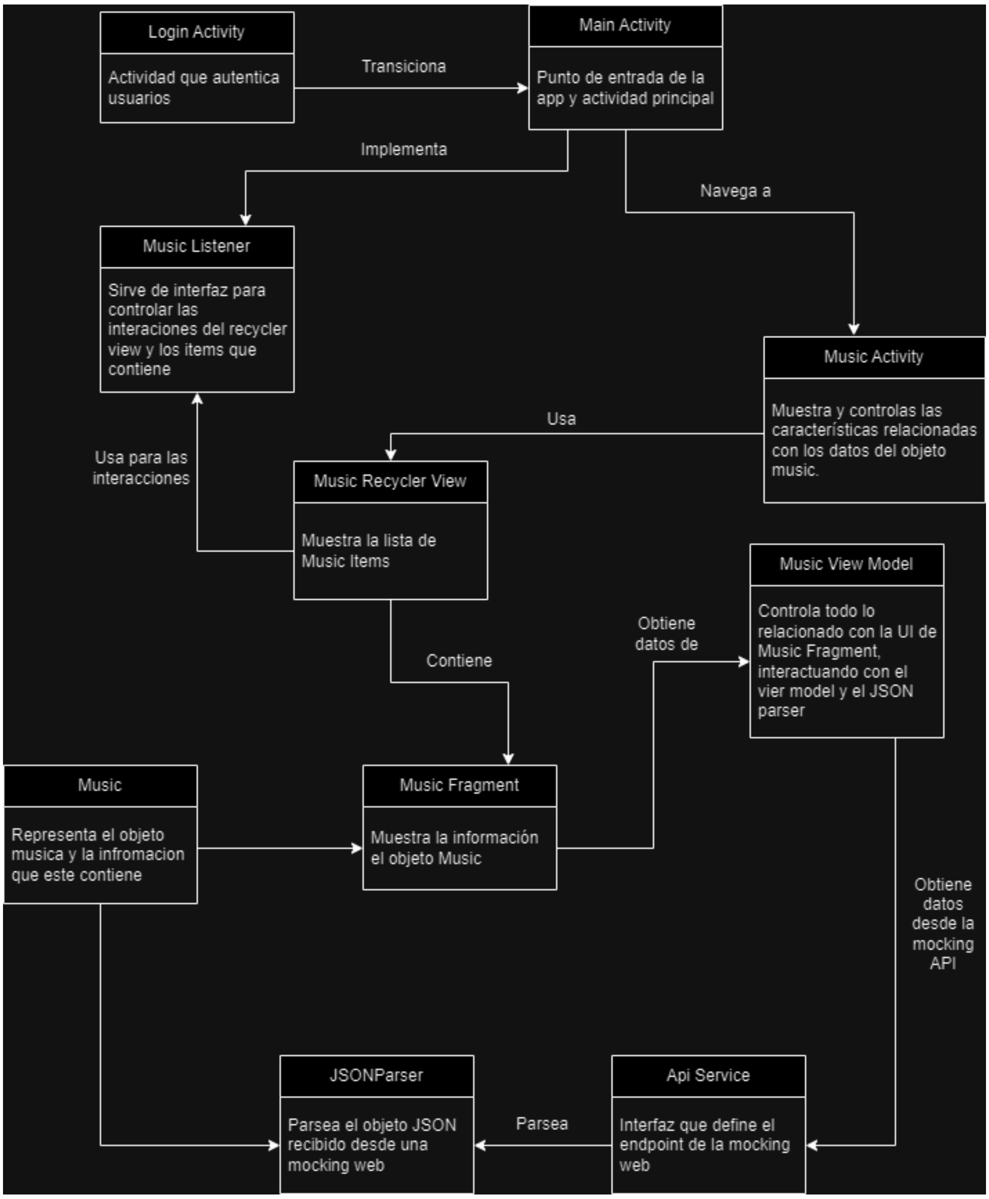
- **Firebase Crashlyctics:** to provide and test the application from possible crashes and exceptions, and to analyse the project.

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 extre dependencies using Gradle, for 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 Andoird application with a clear separation of classes and trasks 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 ensure the app is realiable and functional.

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



Login Activity
Actividad que autentica usuarios

Main Activity
Punto de entrada de la app y actividad principal

Music Listener
Sirve de interfaz para controlar las interacciones del recycler view y los items que contiene

Music Activity
Muestra y controlas las características relacionadas con los datos del objeto music.

Music Recycler View
Muestra la lista de Music Items

Music View Model
Controla todo lo relacionado con la UI de Music Fragment, interactuando con el vier model y el JSON parser

Music
Representa el objeto musica y la infromacion que este contiene

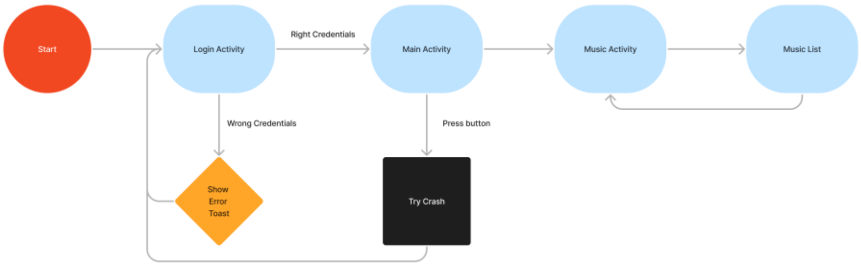
Music Fragment
Muestra la información el objeto Music

JSONParser
Parsea el objeto JSON recibido desde una mocking web

Api Service
Interfaz que define el endpoint de la mocking web

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 compuer was incompatible with Intel HAXM, since I posses an AMD processor. To solve this problem I have 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 almos from the beginning, fortunetly Kotlin is easy to learn and I rebembered some things that were key to develop the application, such as change the version SDKs, add dependencies, and how the activities interacts with each other.

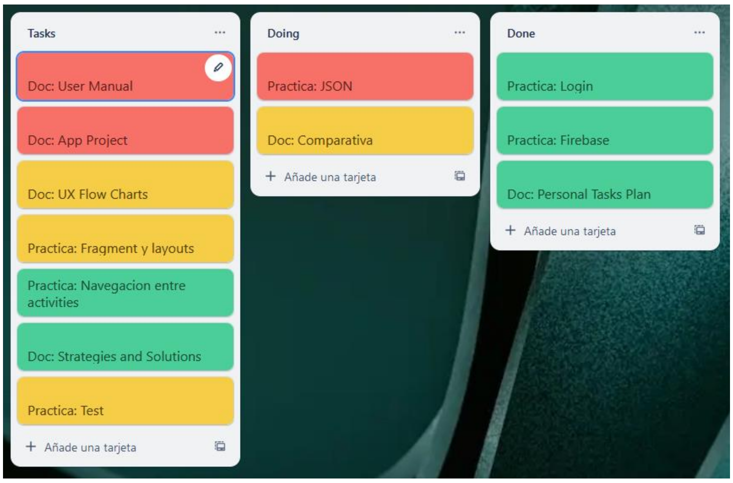
Related with this last one problem were the changes of the Android Studio’s API versions. The new changes in Gradle dependencies made hard to add new modules and dependencies to the new SDK, since the way they are include, as the same as the name of the dependencies had change from the last time I used Android Studio. Although, I finally manage 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 come from the JSON mocking and parser. The issue radicates in that I did not know what was the structure of a JSON list object, and the parser could not conver the data to be use 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 posses, this was divided into different tasks for a better managing and organazing. The tool used to create a personal tasks schema and follow the state of the stages is Trello.

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



Tasks
- Doc: User Manual
- Doc: App Project
- Doc: UX Flow Charts
- Practica: Fragment y layouts
- Practica: Navegacion entre activities
- Doc: Strategies and Solutions
- Practica: Test

Doing
- Practica: JSON
- Doc: Comparativa

Done
- Practica: Login
- Practica: Firebase
- Doc: Personal Tasks Plan

Here is a link:
https://trello.com/invite/b/OzHOlrmp/ATTIe79d069da7fafd0ccd0fa3d673ab210fCF8C14B2/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 synchonyzne the Gradle files, we can do this going into the build.gradke.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 ocurs the detail will be displayed in the “Build” window to resolve the current issues.

Once the project is builded, 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” button 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:

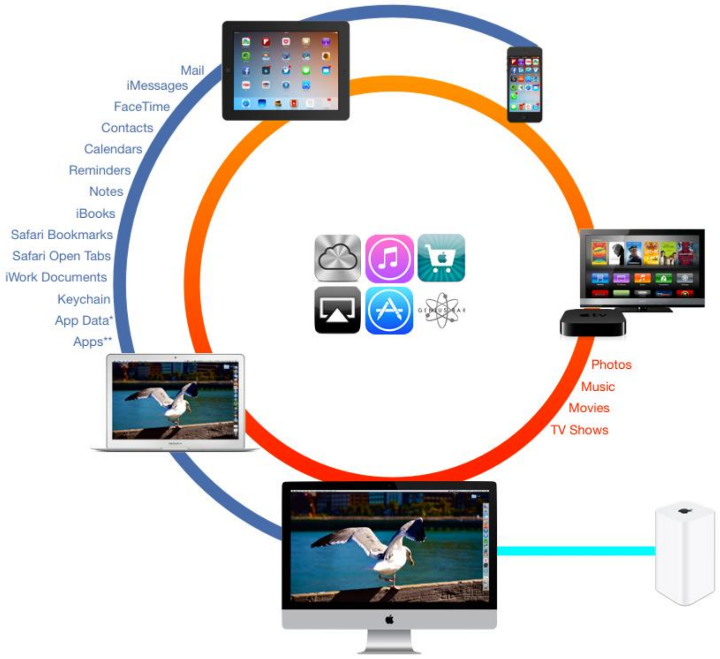
Xcode is the Integrated Development Enviroment created by Apple, to develop applications and software for Apple’s different operating sytems and devices such as, macOS, iOS, iPadOS, watchOS, tvOs, and the new visionOS. Xcode was first released in 2003, and its latest version was released in September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

Althoug Apple’s ecosystem is often described as a walled garden, a term that refers to the completly integration of Apple’s periphericals, 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 desgined 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 intutive 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 desing 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 powerfull and user-friendly.

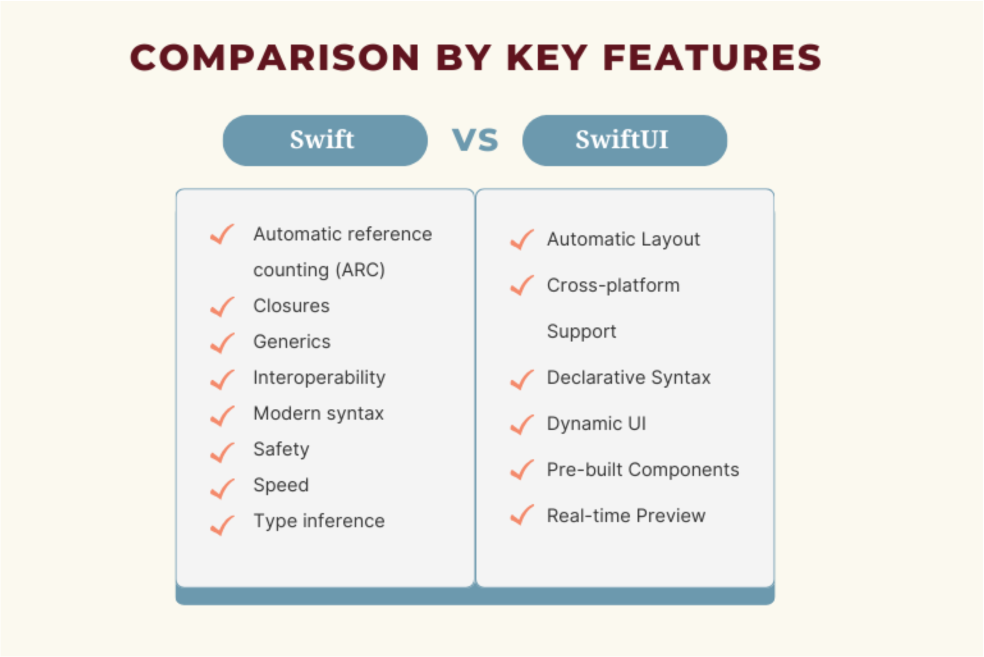
  It relies on the use view controllers to manage UI views; these can be from buttons to the screen view itself. It’s a mature and stable framework, with a wide range of UI components and extensive documentation and community support. Although, it’s hard to maintain when the applications become complex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

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

  But Swift posses a limited ecosystem compared to other languages, and has limited version compatibility due to the continous language updates and version and despite being easy to learn compared to other language 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 seamless with UIKit, AppKit, and Swift. It can also offer real time preview and make emphasis on accesibility and cross-platform compatibility.

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



COMPARISON BY KEY FEATURES

Swift	SwiftUI
✓ Automatic reference counting (ARC)	✓ Automatic Layout
✓ Closures	✓ Cross-platform Support
✓ Generics	✓ Declarative Syntax
✓ Interoperability	✓ Dynamic UI
✓ Modern syntax	✓ Pre-built Components
✓ Safety	✓ Real-time Preview
✓ Speed	
✓ Type inference	

- **Storyboards:** Is a visual representation of user interface, which displays the screens' content and the connection betweem 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 reprensentaion 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 posses' 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 writen 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 enviroment to give an idea about how certain thnigs 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 enviroment to get a better idea of how would things work.

6.3 Comparison:

Both Android Studio and Xcode posses the common features present in any IDE (Integrated Development Enviroment) 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 optimizaed 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 posses' 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 amoun of programming languages they support. Android Studio supports primarily Kotlin and Java, along with C++, but Xcode supports Switf, C, C++, Objective-C, Objective-C++, Java, AppleScript, Python, Ruby, Rez, 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 desing the user interface of the apps, allowing developers to quickly develop their applications. Xcode uses a combinations of drag-and-drop, tools and code, called “Interface Builder”, that allows more app development complexity in an easier way.

- **IDE Desing Framework:** Android Studio and Xcode have a very different philosphy about how to structure the code and navigation of the apps. Android use partition, the app is break into activities and fragments. The activity is the equivalent to one app screen, since the application can posses multiple screens, each of the activities uses fragments, to navigate between the activities. Instead, Xcode uses different types view controllers, which can control the entire screen or one part of it. These controllers can be managing in a several ways, through code, images, or XML files and more.

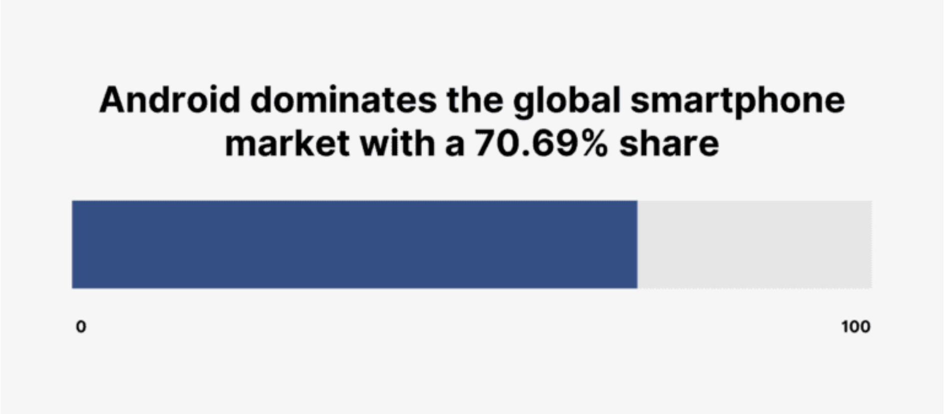
- **Availability:** both IDEs are proprietary software, but Android Studio is avaliable for Windows, macOS, and Linux, making it cross-platform. Xcode is only available for Apple devices. This makes a great difference, since Apple devices are expensive to obtain compared to the other devices, due this, Android Studio is a more available and flexbile option for developers.

- **Target Limitations:** there are different limitations related to the app distribution and emulation tools these IDEs provides. 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.

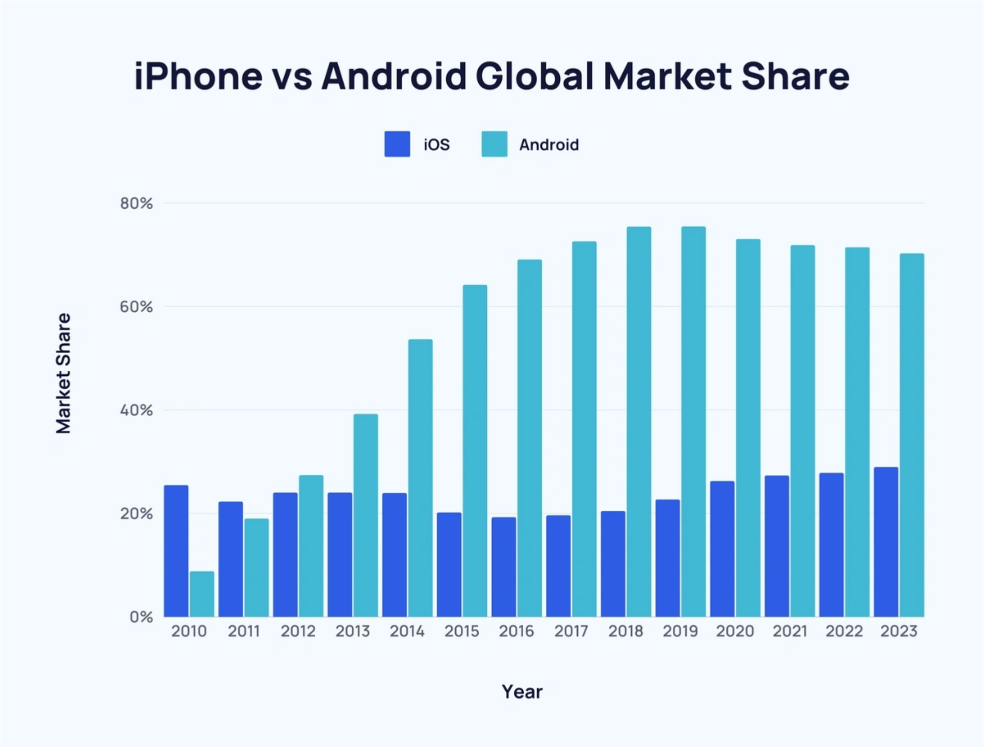
In conclusion both IDE offers their pros and cons. The flexibility and accessibility of Android Studio allows for more market share and wider app distribution, but since there are a lot of different devices with different features with Android as operating system, the developers may have issues while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

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

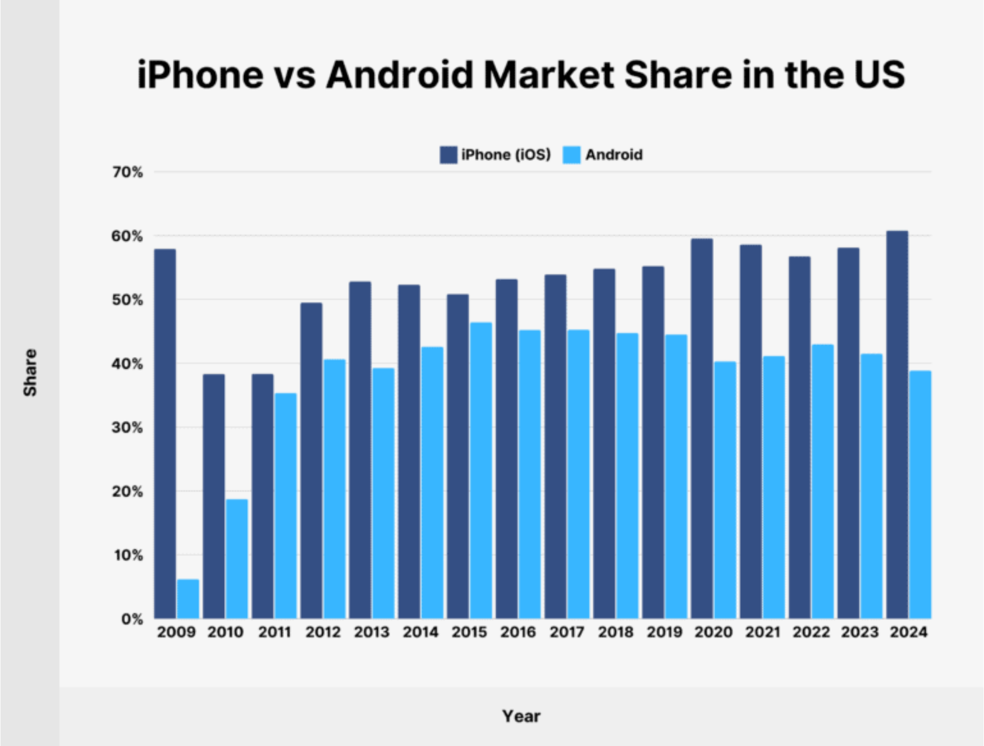
Finally, is also important, to talk about the different maket shares both operating systems posses. According to the lastest data of early 2024, Android has 70,69% of the total smartphone global market dominating over the 28,58% of the global market share iOS has.



Since the release of the first iOS in 2009, Apple were more popular than Android until 2011, although by that time both systems were overshdowed by SymbianOS and Blackberry that controlled 35,49% of the mobile market share in 2009. By 2010 SymbianOS and Blackberry falled 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.



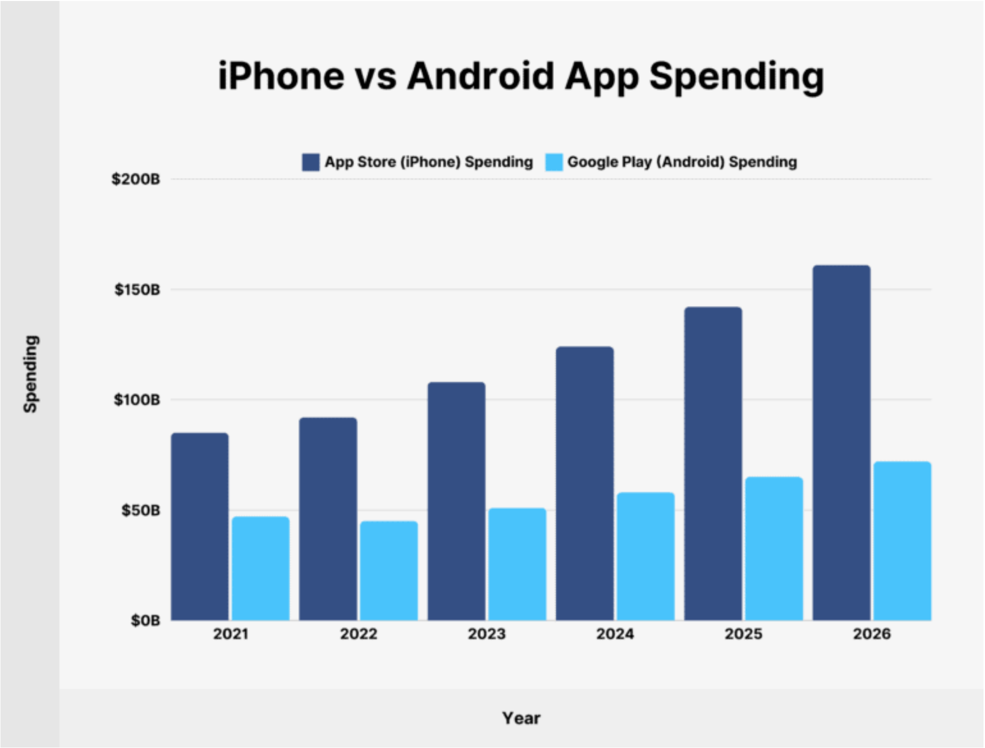
Althoug Android dominates the operating system mobile market share worldwide, iOS leads the sales in certain countries such as Japan (68,86%), Canada (60,31%), Australia (59,63%), United Kingdom (52,29%), and the most importat United States (60,77%), where iOS has lead the market since its release in 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, againts the $47.9 billion in Android. iOS is the winner when it comes to revenue, which seems stranges because both devices posses 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 spends $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 availabity 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 ad managing for Android.

These dependencies also use different frameworks and API's. Althogh the application is scalable in the long term, the different updates of the Android Studio's API, makes the project unease to maintain in the long term, since the changes from the version of Android Studio are so different, some dependencies and code parts needs to be change from one version to another, which is an issue for developers once the application have 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 obnoxius. 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 aploximetly implementation based on the current project develop on Android.

Due to the hard availability of Apple's devices, this approach will be done using AI promts, to provide visual support alongside the documentation to provide a better example of a possible implementation in Xcode.

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 actrivity 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.

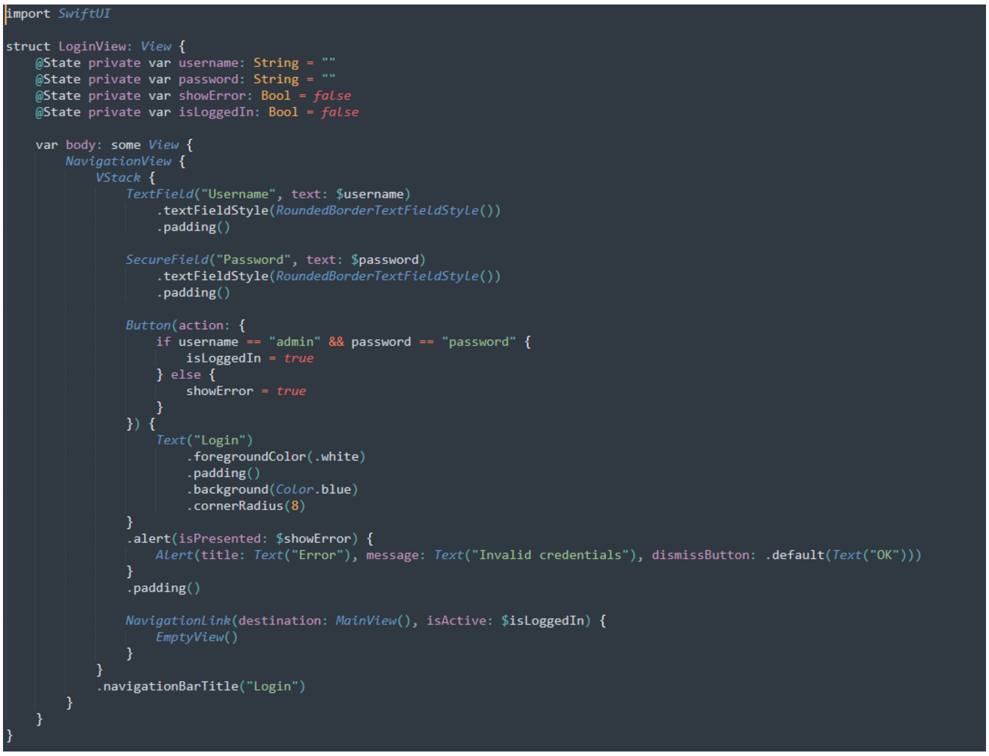
In this case, we create a first view controller called LoginView that will contain antoher view controller called ”body”. Inside the body view we can add a VStack to order the elements stored inside in a vertical layout.

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 hardoced values we have to check if the user is right or worng. 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 NavigationLink. If the password is wrong we will show a toast message that displays the given values are wrong.

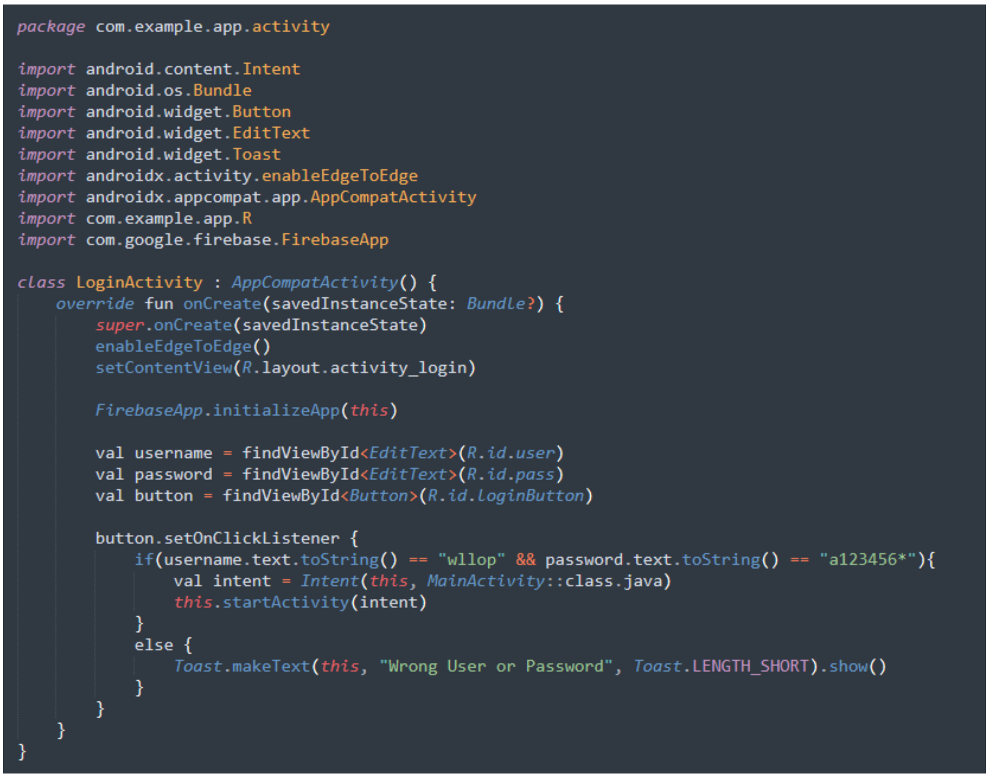
To display a toast message we also need to create a new view controller. Inside the toast view we can modify its body, to display a message to the user and even add animations of displaying and disappearing, with a given duration, using onAppear, and Text for both display the message and set the duration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

This is the code provided by ChatGPT, when uploading LoginActivity.kt, and providing this prompt: “*Make an implementation of this file into Xcode using SwiftUI”*



```
import SwiftUI

struct LoginView: View {
    @State private var username: String = ""
    @State private var password: String = ""
    @State private var showError: Bool = false
    @State private var isLoggedIn: Bool = false

    var body: some View {
        NavigationView {
            VStack {
                TextField("Username", text: $username)
                    .textFieldStyle(RoundedBorderTextFieldStyle())
                    .padding()

                SecureField("Password", text: $password)
                    .textFieldStyle(RoundedBorderTextFieldStyle())
                    .padding()

                Button(action: {
                    if username == "admin" && password == "password" {
                        isLoggedIn = true
                    } else {
                        showError = true
                    }
                }) {
                    Text("Login")
                        .foregroundColor(.white)
                        .padding()
                        .background(Color.blue)
                        .cornerRadius(8)
                }
                .alert(isPresented: $showError) {
                    Alert(title: Text("Error"), message: Text("Invalid credentials"), dismissButton: .default(Text("OK")))
                }
                .padding()

                NavigationLink(destination: MainView(), isActive: $isLoggedIn) {
                    EmptyView()
                }
            }
            .navigationBarTitle("Login")
        }
    }
}
```

And here is the code in Kotlin.



```kotlin
package com.example.app.activity

import android.content.Intent
import android.os.Bundle
import android.widget.Button
import android.widget.EditText
import android.widget.Toast
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import com.example.app.R
import com.google.firebase.FirebaseApp

class LoginActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_login)

        FirebaseApp.initializeApp(this)

        val username = findViewById<EditText>(R.id.user)
        val password = findViewById<EditText>(R.id.pass)
        val button = findViewById<Button>(R.id.loginButton)

        button.setOnClickListener {
            if(username.text.toString() == "wllop" && password.text.toString() == "a123456*"){
                val intent = Intent(this, MainActivity::class.java)
                this.startActivity(intent)
            }
            else {
                Toast.makeText(this, "Wrong User or Password", Toast.LENGTH_SHORT).show()
            }
        }
    }
}
```

As summary:

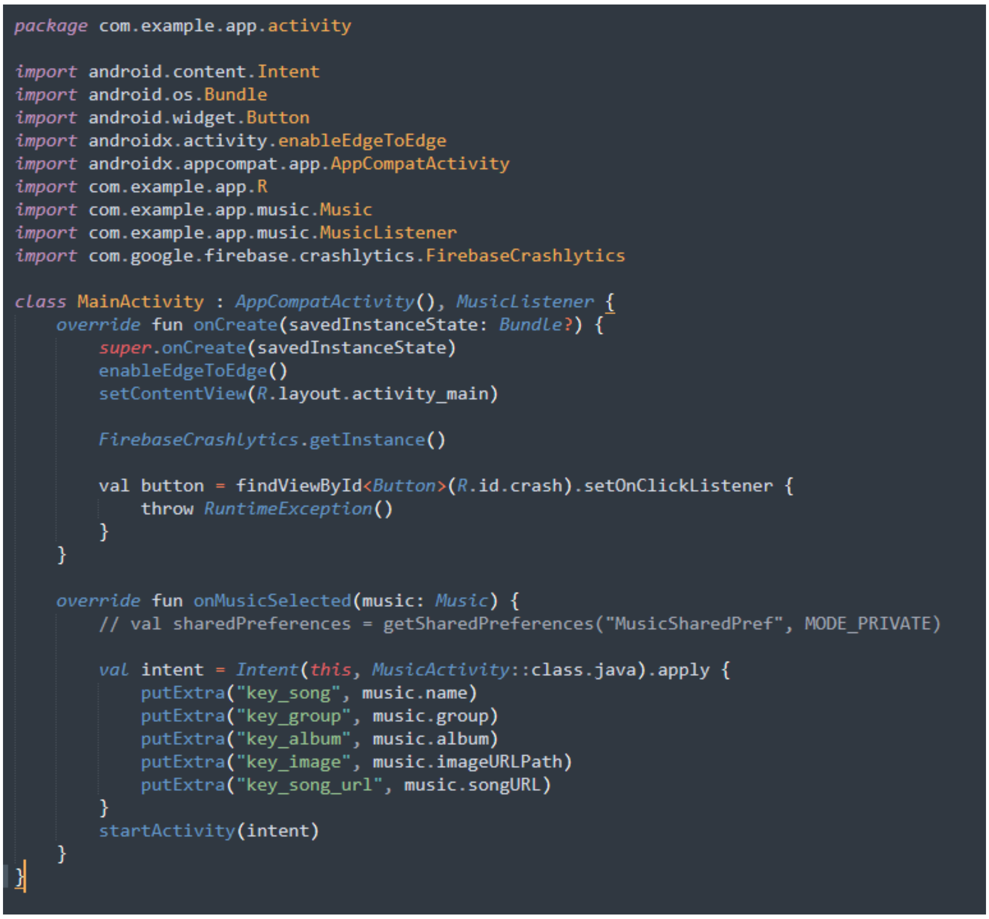
- **The LoginView struct represents the screen.**
- **The NavigationView provides navigation capabilities.**
- **TextField and SecureField are used for input.**
- **Button checks the credentials.**
- **If succeded, NavigationLink navigates to the MainView, otherwise an alert is shown**.

3. Main Activity:

For now on instead of explaining the implementation step by step, we will compare the result given by the AI to those in our project to get conclusions about a possible implementation.

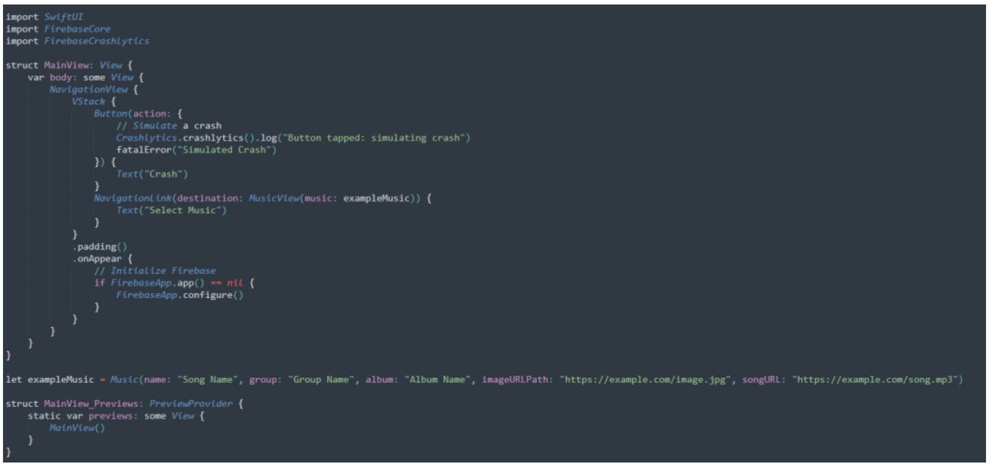
The key fetures of our MainActivity is to display through an interface listener the list of Music data class we have request from the network. It also has a button to crash the application to test Firebase Crashlyctics. When selecting the different RecycleViews of the shown Music list we also pass information about the selected Music data to the MusicActivity, and then we navigate to the MusicActivity.

Here is the code in Kotlin



```
package com.example.app.activity

import android.content.Intent
import android.os.Bundle
import android.widget.Button
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import com.example.app.R
import com.example.app.music.Music
import com.example.app.music.MusicListener
import com.google.firebase.crashlytics.FirebaseCrashlytics

class MainActivity : AppCompatActivity(), MusicListener {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_main)

        FirebaseCrashlytics.getInstance()

        val button = findViewById<Button>(R.id.crash).setOnClickListener {
            throw RuntimeException()
        }
    }

    override fun onMusicSelected(music: Music) {
        // val sharedPreferences = getSharedPreferences("MusicSharedPref", MODE_PRIVATE)

        val intent = Intent(this, MusicActivity::class.java).apply {
            putExtra("key_song", music.name)
            putExtra("key_group", music.group)
            putExtra("key_album", music.album)
            putExtra("key_image", music.imageURLPath)
            putExtra("key_song_url", music.songURL)
        }
        startActivity(intent)
    }
}
```

And this is the example code provided by ChatGPT when uploading the file and with the next prompt: *“Translate this code into Xcode using SwiftUI”*



```
import SwiftUI
import FirebaseCore
import FirebaseCrashlytics

struct MainView: View {
    var body: some View {
        NavigationView {
            VStack {
                Button(action: {
                    // Simulate a crash
                    Crashlytics.crashlytics().log("Button tapped: simulating crash")
                    fatalError("Simulated Crash")
                }) {
                    Text("Crash")
                }
                NavigationLink(destination: MusicView(music: exampleMusic)) {
                    Text("Select Music")
                }
            }
            .padding()
            .onAppear {
                // Initialize Firebase
                if FirebaseApp.app() == nil {
                    FirebaseApp.configure()
                }
            }
        }
    }
}

let exampleMusic = Music(name: "Song Name", group: "Group Name", album: "Album Name", imageURLPath: "https://example.com/image.jpg", songURL: "https://example.com/song.mp3")

struct MainView_Previews: PreviewProvider {
    static var previews: some View {
        MainView()
    }
}
```

Althoug the image is hard to see, we can appreciate that the code includes all the different charcteristics of our MainActivity.kt. In first place we have a NavigationView to enable navigation between screens, used as a replacement for the interface we were using on our Kotling code, and are able to display and navigate the list of stored Music data.

We can also see that includes a VStack to create a vertical layout and inside this, there is a button to crash the application for Firebase Crashlytics, which we can see is initialize later inside the ViewController.Finally, an example variable is used to pass information to the MusicActivity as we do in our project.

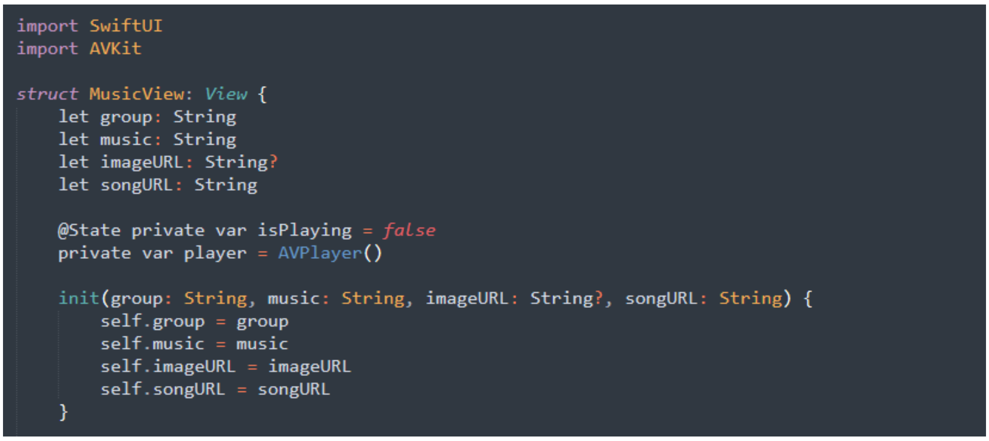
In conclusion this could be a way to implement the MainActivity to Xcode and SwiftUI, although the way it pass values to the next screen could be wrong since it is a hard-coded variable. In resume:

- **The NavigationView inside the view maybe used as a listener interface.**
- **Firebase is correctly initialized and tested.**
- **The variable exampleMusic is used as a parameter for the NavigationLink**

4. Music Activity:

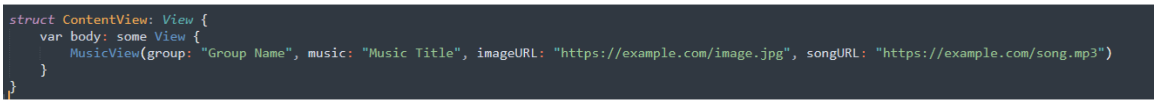
The MusicActivity displays the information of the given item stored in the music list. It receives the information from the Intent incoming from the MainActivity. There is also a try to include a button to play music from this activity from a URL but its not included in the final version.

The implementation given by AI with the prompt: "*Implement this code into Xcode with SwiftUI*"

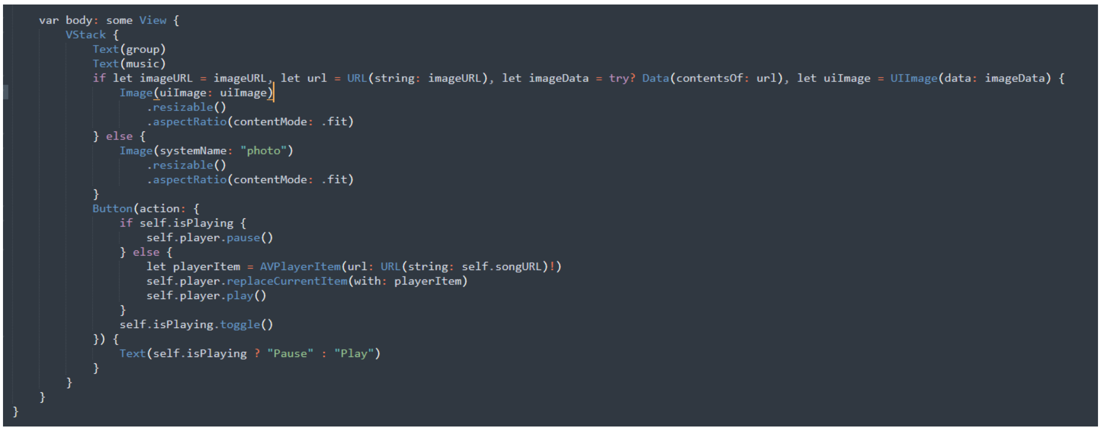


```
import SwiftUI
import AVKit

struct MusicView: View {
    let group: String
    let music: String
    let imageURL: String?
    let songURL: String

    @State private var isPlaying = false
    private var player = AVPlayer()

    init(group: String, music: String, imageURL: String?, songURL: String) {
        self.group = group
        self.music = music
        self.imageURL = imageURL
        self.songURL = songURL
    }
```

In this part of the code the values receive from another screen are initialized into the MusicView variables.



```
struct ContentView: View {
    var body: some View {
        MusicView(group: "Group Name", music: "Music Title", imageURL: "https://example.com/image.jpg", songURL: "https://example.com/song.mp3")
    }
}
```



```
    var body: some View {
        VStack {
            Text(group)
            Text(music)
            if let imageURL = imageURL, let url = URL(string: imageURL), let imageData = try? Data(contentsOf: url), let uiImage = UIImage(data: imageData) {
                Image(uiImage: uiImage)
                    .resizable()
                    .aspectRatio(contentMode: .fit)
            } else {
                Image(systemName: "photo")
                    .resizable()
                    .aspectRatio(contentMode: .fit)
            }
            Button(action: {
                if self.isPlaying {
                    self.player.pause()
                } else {
                    let playerItem = AVPlayerItem(url: URL(string: self.songURL)!)
                    self.player.replaceCurrentItem(with: playerItem)
                    self.player.play()
                }
                self.isPlaying.toggle()
            }) {
                Text(self.isPlaying ? "Pause" : "Play")
            }
        }
    }
}
```

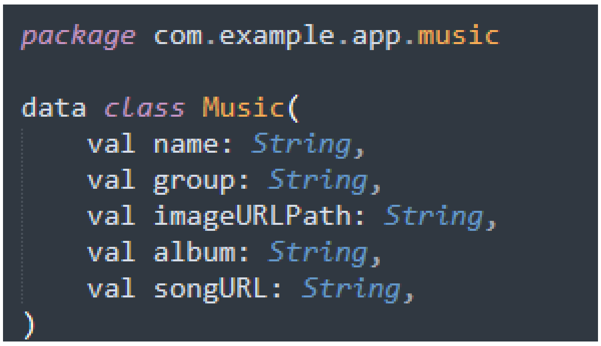
In this part the View displays the information of the selected item. The example also includes the commented code section of the MusicActivity that tries to play music form the activity.

This could be a potential implementation or at least work as a template to adjust later during the development, since it’s clear that some elements may require specific beahviors or error handling. And also because the AI does not have access to the full project files, and the approach may have some mistakes.

5. Music Framework:

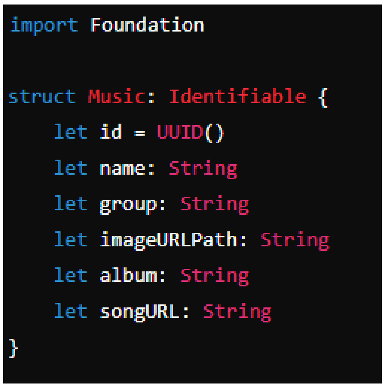
This step includes different files about how the applications makes use of the data gathered from the JSON, how this data is displayed and organized, how this data is reuse using RecyclingViews.

The first step is to create a data class similar to the one we have in our project which is:



```
package com.example.app.music

data class Music(
    val name: String,
    val group: String,
    val imageURLPath: String,
    val album: String,
    val songURL: String,
)
```

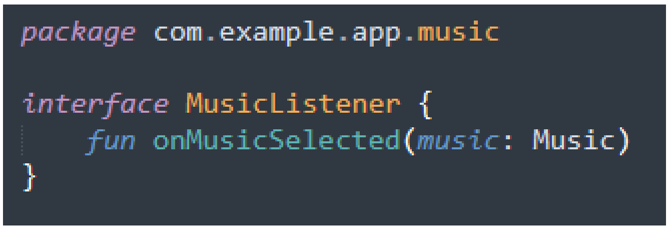
We asked ChatGPT to make a similar implementation by uploading Music.kt file and asking: *“Implement this code in Xcode using SwiftUI”.*



```
import Foundation

struct Music: Identifiable {
    let id = UUID()
    let name: String
    let group: String
    let imageURLPath: String
    let album: String
    let songURL: String
}
```

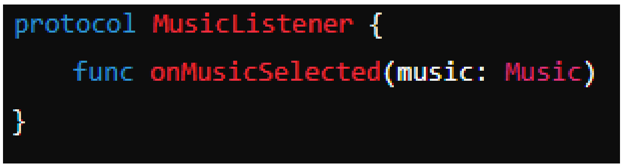
We can see that the code is pretty similar but with the main different that the data class has an inheritance from Identifiable, a protocol Xcode uses to identify instyances of a data in a collection, and it also requires a unique identifier, which we can see it has also been included in the example.

The following step is to implement the interface used for calling the selected data from out list in our MainActivity to pass it to the MusicActivity.



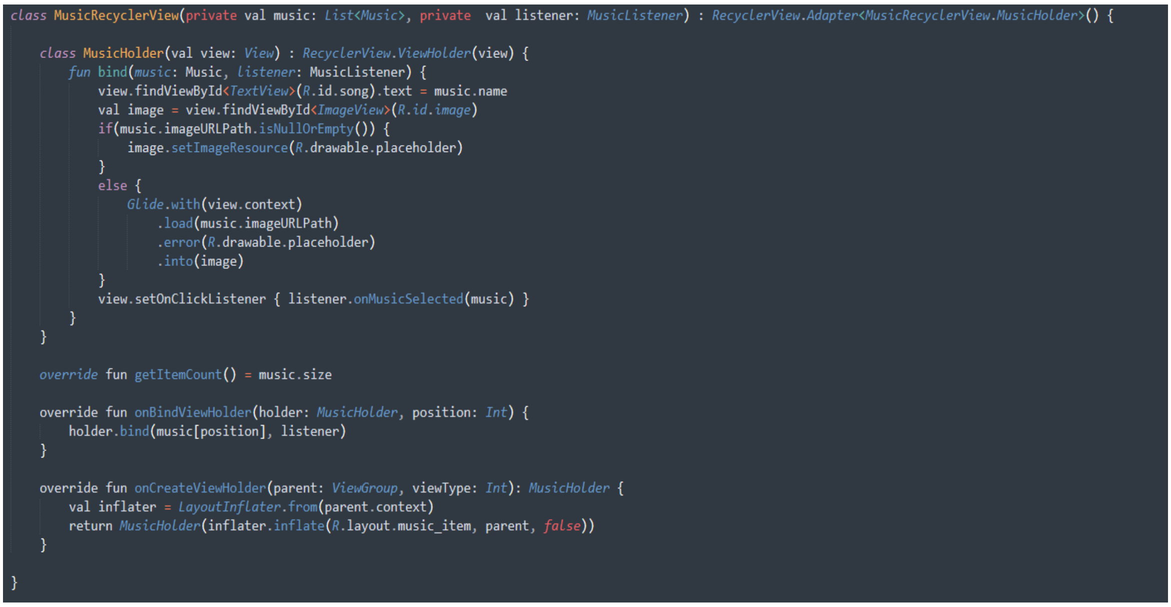
```
package com.example.app.music

interface MusicListener {
    fun onMusicSelected(music: Music)
}
```

The implementation for interfaces in Xcode is done using Protocols, here is an example.



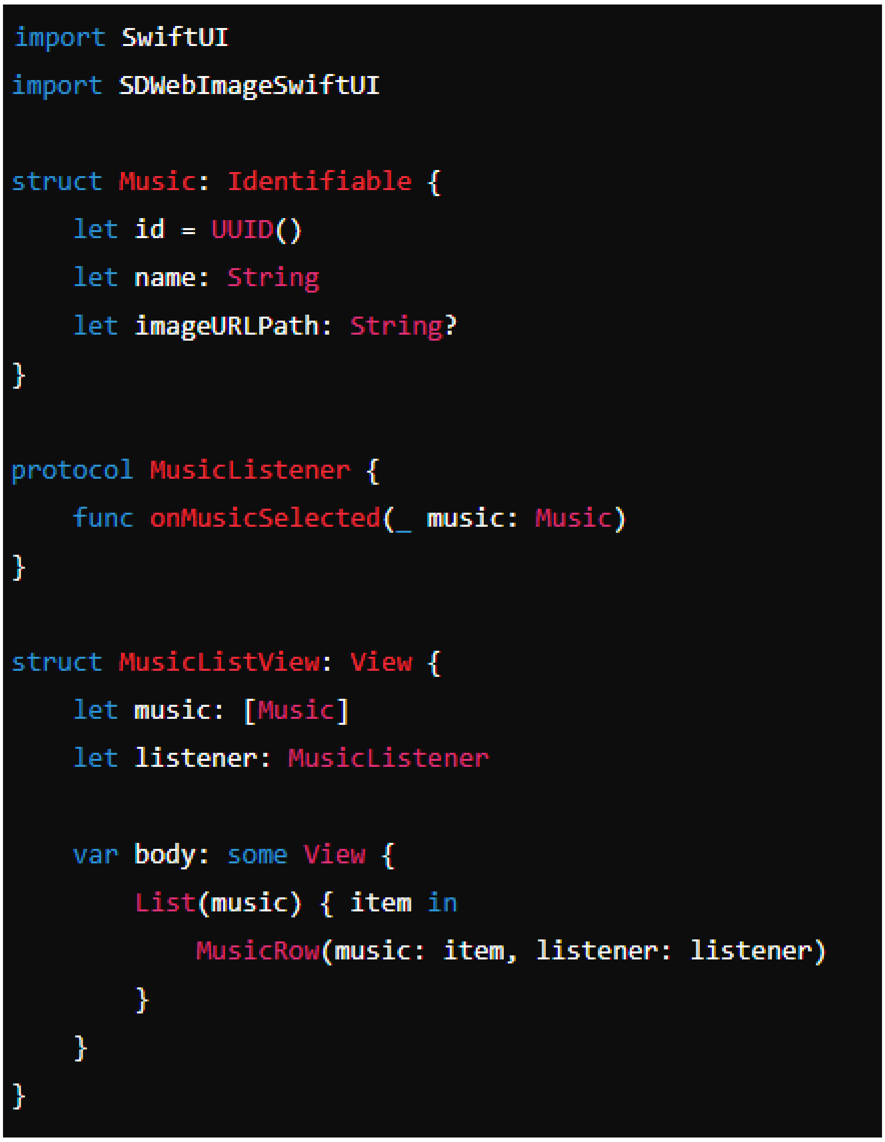
```
protocol MusicListener {
    func onMusicSelected(music: Music)
}
```

The next step is to implement our MusicRecyclerView, this class implements a MusicHolder class inside that is binded to MusicListener present in our MainActivity and it's used to display the information of our Music data class and recycle the music_item.xlm. Its main task is to display the information in a vertical layout list and navigate to the MusicActivity once is pressed from out MainActivity.



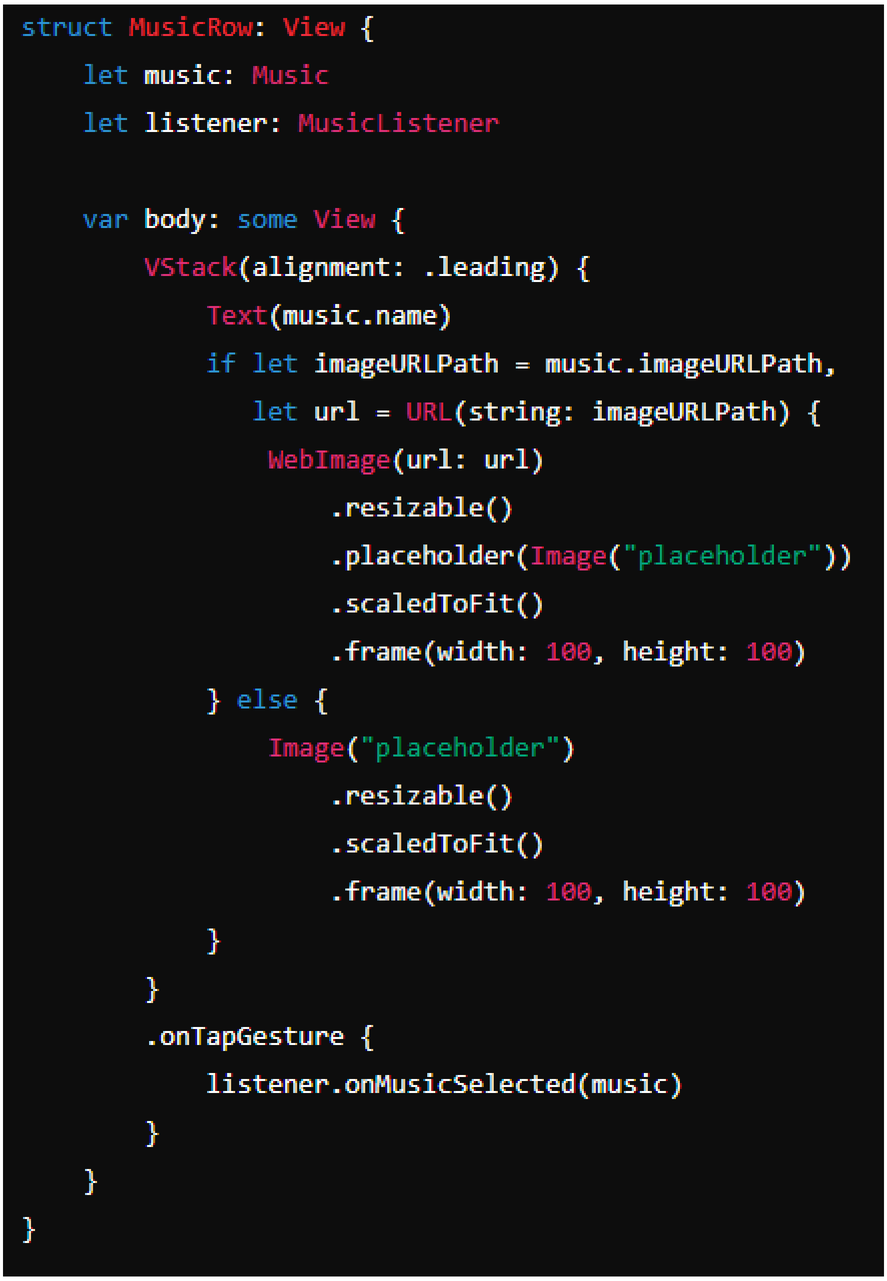
```
class MusicRecyclerView(private val music: List<Music>, private  val listener: MusicListener) : RecyclerView.Adapter<MusicRecyclerView.MusicHolder>() {

    class MusicHolder(val view: View) : RecyclerView.ViewHolder(view) {
        fun bind(music: Music, listener: MusicListener) {
            view.findViewById<TextView>(R.id.song).text = music.name
            val image = view.findViewById<ImageView>(R.id.image)
            if(music.imageURLPath.isNullOrEmpty()) {
                image.setImageResource(R.drawable.placeholder)
            }
            else {
                Glide.with(view.context)
                    .load(music.imageURLPath)
                    .error(R.drawable.placeholder)
                    .into(image)
            }
            view.setOnClickListener { listener.onMusicSelected(music) }
        }
    }

    override fun getItemCount() = music.size

    override fun onBindViewHolder(holder: MusicHolder, position: Int) {
        holder.bind(music[position], listener)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MusicHolder {
        val inflater = LayoutInflater.from(parent.context)
        return MusicHolder(inflater.inflate(R.layout.music_item, parent, false))
    }

}
```

Here is a look of our code. Now let's see an implementation using ChatGPT with the following prompt: "*Implement an example of this code using SwiftUI*"



```
import SwiftUI
import SDWebImageSwiftUI

struct Music: Identifiable {
    let id = UUID()
    let name: String
    let imageURLPath: String?
}

protocol MusicListener {
    func onMusicSelected(_ music: Music)
}

struct MusicListView: View {
    let music: [Music]
    let listener: MusicListener

    var body: some View {
        List(music) { item in
            MusicRow(music: item, listener: listener)
        }
    }
}
```

When implementing the code, ChatGPT has filled the gaps in the example code, since it does not posses the full project files. These files are dummy implementation of the already seen clases, needed for the example. The most destacable element is the View that contains the music list.

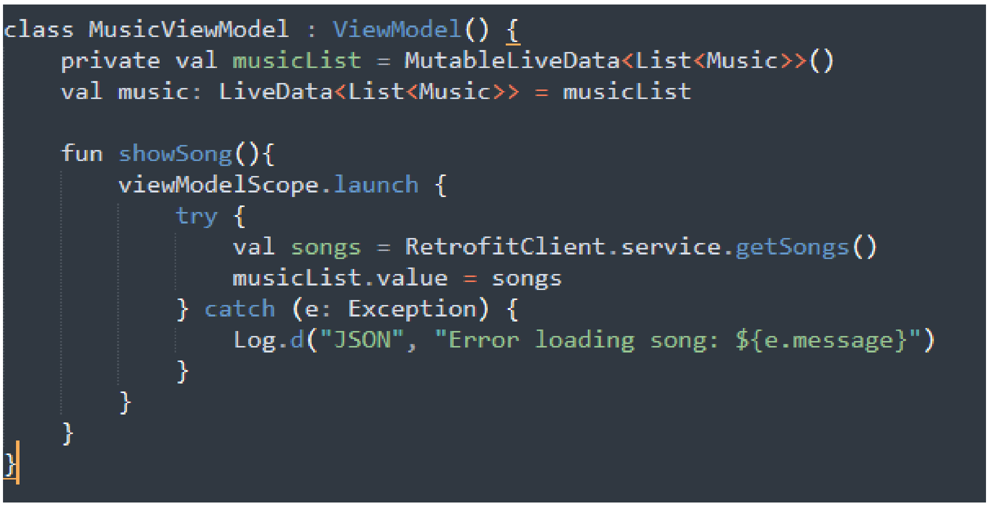


```swift
struct MusicRow: View {
    let music: Music
    let listener: MusicListener

    var body: some View {
        VStack(alignment: .leading) {
            Text(music.name)
            if let imageURLPath = music.imageURLPath,
               let url = URL(string: imageURLPath) {
                WebImage(url: url)
                    .resizable()
                    .placeholder(Image("placeholder"))
                    .scaledToFit()
                    .frame(width: 100, height: 100)
            } else {
                Image("placeholder")
                    .resizable()
                    .scaledToFit()
                    .frame(width: 100, height: 100)
            }
        }
        .onTapGesture {
            listener.onMusicSelected(music)
        }
    }
}
```

And this is the possible implementation for our MusicRecycleView, we can see that it use a vertical layout, and inside are allocated all the elements necessary for display the information of our data class.

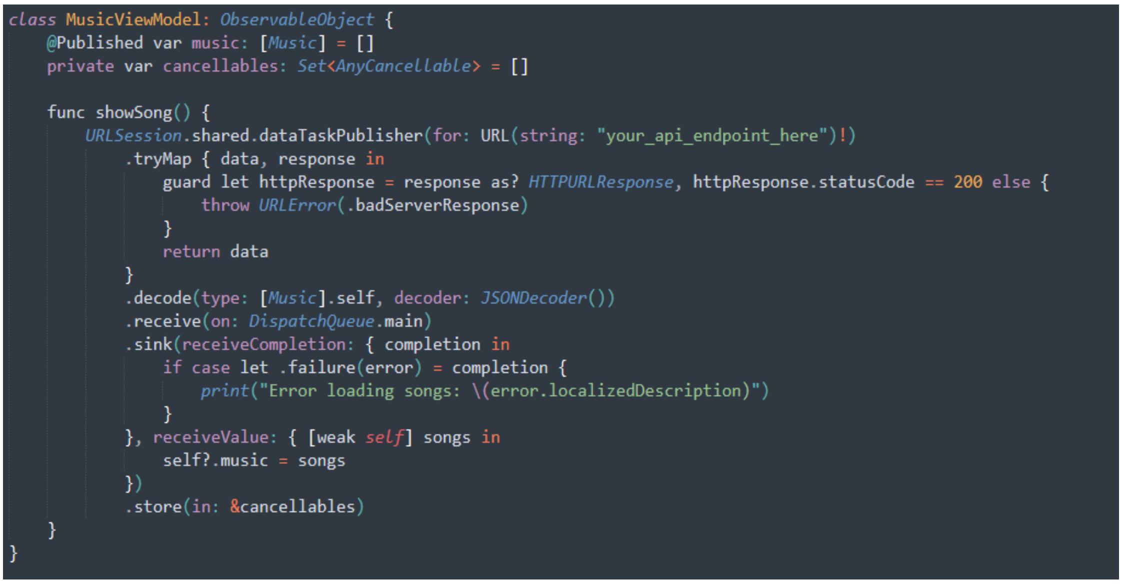
Comparing this code to our Kotlin code, we can appreciate that the MusicHolder class has dispared and the bind function inside it now corresponds to the body of the MusicView. This could be because the all the different classes use in our Kotlin code are included in the same SwiftUI file. So this approach will not be suitable, although it could help to get an approximately idea of how to make a similar implementation.

Finally, the MusicViewModel which posses the music list fetchet from the network endpoint and exposing it as a live data object. It also includes a exception handler to know when a issue occurs and why. The Kotlin code looks like this:



```
class MusicViewModel : ViewModel() {
    private val musicList = MutableLiveData<List<Music>>()
    val music: LiveData<List<Music>> = musicList

    fun showSong(){
        viewModelScope.launch {
            try {
                val songs = RetrofitClient.service.getSongs()
                musicList.value = songs
            } catch (e: Exception) {
                Log.d("JSON", "Error loading song: ${e.message}")
            }
        }
    }
}
```

And the implementation created by ChatGPT with this prompt: “*Implement this code in SwiftUI*” is:

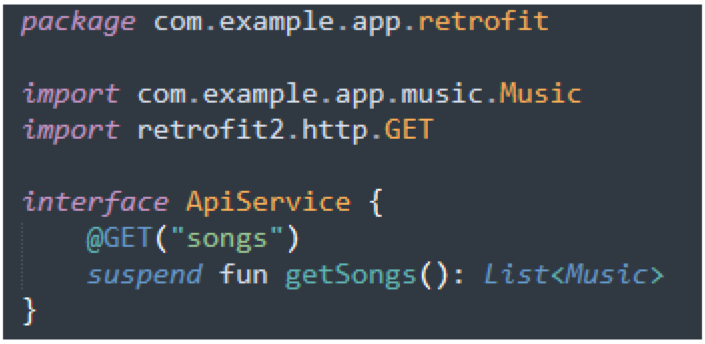


```
class MusicViewModel: ObservableObject {
    @Published var music: [Music] = []
    private var cancellables: Set<AnyCancellable> = []

    func showSong() {
        URLSession.shared.dataTaskPublisher(for: URL(string: "your_api_endpoint_here")!)
            .tryMap { data, response in
                guard let httpResponse = response as? HTTPURLResponse, httpResponse.statusCode == 200 else {
                    throw URLError(.badServerResponse)
                }
                return data
            }
            .decode(type: [Music].self, decoder: JSONDecoder())
            .receive(on: DispatchQueue.main)
            .sink(receiveCompletion: { completion in
                if case let .failure(error) = completion {
                    print("Error loading songs: \(error.localizedDescription)")
                }
            }, receiveValue: { [weak self] songs in
                self?.music = songs
            })
            .store(in: &cancellables)
    }
}
```

Here we can see how the JSON parser is implemented using the JSON encoder and decoder Xcode has by default. We can also see that the MusicViewModel for this implementation inherits from ObservableObject, a protocol of SwiftUI to handle asynchronous events and data streams. Since as we can appreciate this example already includes a way to fetch information form a network endpoint and decode that information into a usable data. This example is very helpful since it can allow us to create a similar functioning but with less code and clases, which helps the scalability of the project.

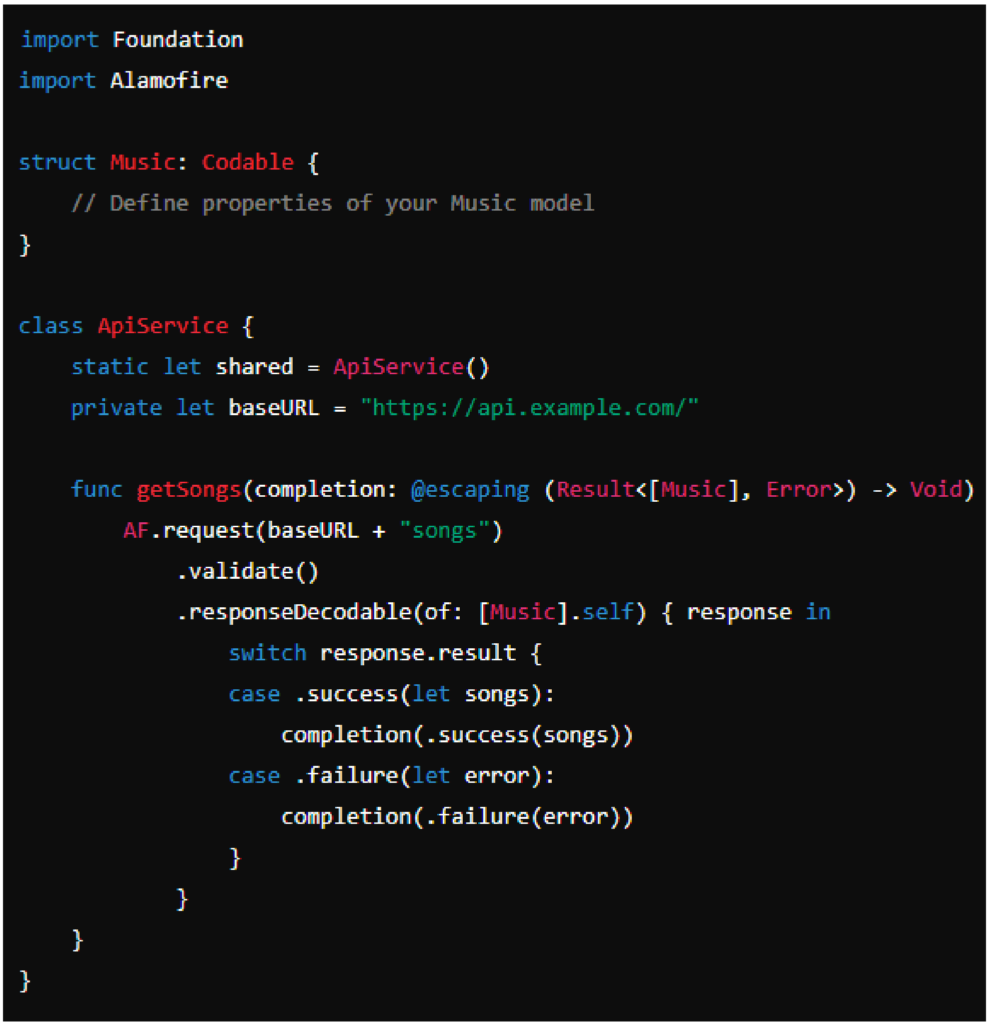
6. Mocking API and JSON Parser:

Our file APIService.kt provides an interface for Retrofit, a third-party dependency for fetching a list of song form an endpoint in the network. Our Kotlin file looks like this:



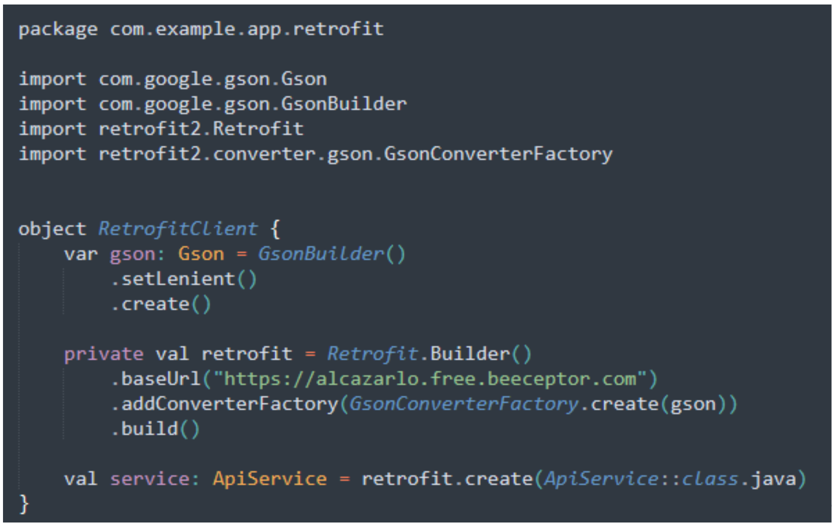
```
package com.example.app.retrofit

import com.example.app.music.Music
import retrofit2.http.GET

interface ApiService {
    @GET("songs")
    suspend fun getSongs(): List<Music>
}
```

And the code provided for ChatGPT after upload the file and input the following prompt: “*Make an implementation for Xcode of this code”*



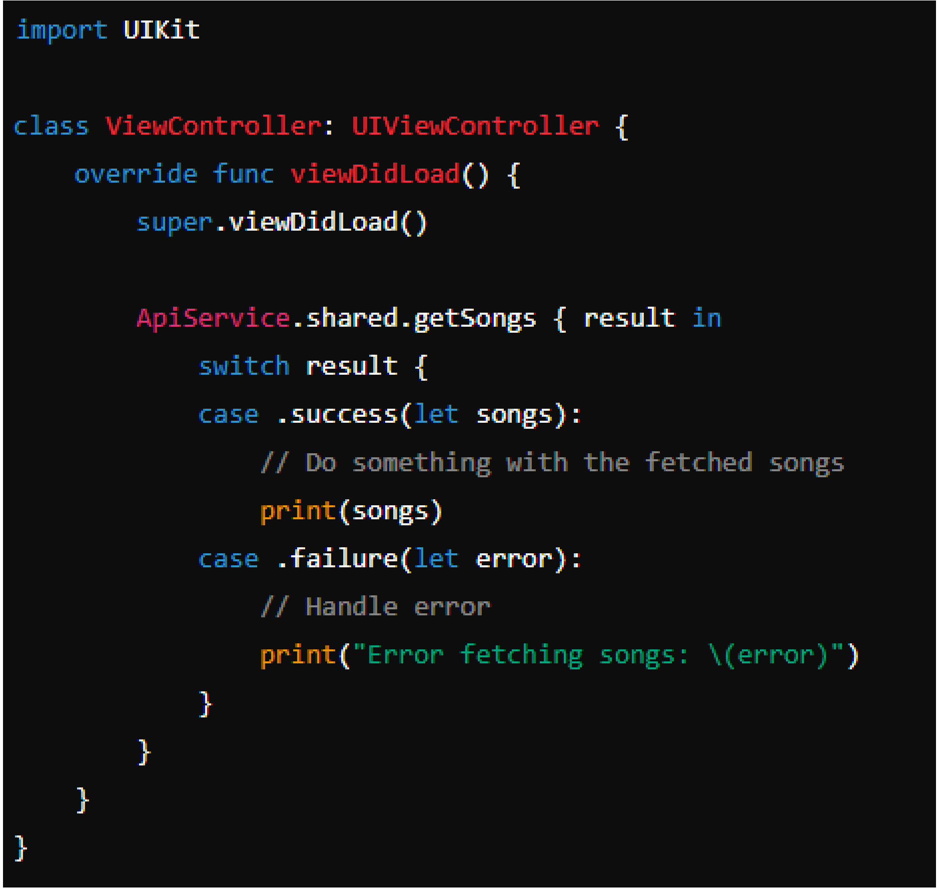
```
import Foundation
import Alamofire

struct Music: Codable {
    // Define properties of your Music model
}

class ApiService {
    static let shared = ApiService()
    private let baseURL = "https://api.example.com/"

    func getSongs(completion: @escaping (Result<[Music], Error>) -> Void)
        AF.request(baseURL + "songs")
            .validate()
            .responseDecodable(of: [Music].self) { response in
                switch response.result {
                case .success(let songs):
                    completion(.success(songs))
                case .failure(let error):
                    completion(.failure(error))
                }
            }
    }
}
```

We can see that the AI has included a dependency similar to Retrofit called Alamofire, and then has make a straightforward implementation, similar to the code we have in out project.



```
package com.example.app.retrofit

import com.google.gson.Gson
import com.google.gson.GsonBuilder
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory


object RetrofitClient {
    var gson: Gson = GsonBuilder()
        .setLenient()
        .create()

    private val retrofit = Retrofit.Builder()
        .baseUrl("https://alcazarlo.free.beeceptor.com")
        .addConverterFactory(GsonConverterFactory.create(gson))
        .build()

    val service: ApiService = retrofit.create(ApiService::class.java)
}
```

The following step is to implement the JSON parser that will fetch and load into a list the JSON receive from an endpoint.



```
import UIKit

class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()

        ApiService.shared.getSongs { result in
            switch result {
            case .success(let songs):
                // Do something with the fetched songs
                print(songs)
            case .failure(let error):
                // Handle error
                print("Error fetching songs: \(error)")
            }
        }
    }
}
```

This is the code provided by ChatGPT after asked to implement our code: “*Implement this code into Xcode too”*

The main difference is that Android uses an object to implement an instance of the RetrofitClient, and Xcode uses a ViewController, and also it does not store the values into a list, this could be because the AI does not posses all the information but we can use the provided code as a template and easily change the prints for our music list.

8. – Post Mortem

In first place, about the development of the project, in this second occasion, because the first version of the application was lost, I had encountered several issues that were quite hard and obnoxious to fix. Because my processor is AMD is not compatible with Intel HAXM, a virtualizarer booster used by Android Studio to enhance the performance and speed of the simulators used in the IDE. To overcome this problem I had to make some adjustements in the bios of my CPU, basically enabling the virtualization of AMD, called AMD-V, enabling in the bios the SVM mode.

Added to this issue, happen for unknown reason that any time I open Android Studio I had to do the same process to go to the bios to enable the SVM mode, which was quite bothersome and annoying. An issue that I had no encountered the first time I develop this project.

On second place I had to rebember everything from the beginning, plus of that I had no knowledge about iOS and Xcode, so when it comes to develop a possible implemetation for Xcode, I had encountered several troubles trying to understand how it works. But my teacher helped with this issue giving me some guidelines to follow, from which im gratefull.

When it comes to compare Android Studio IDE against Xcode IDE, for me personally there is no chance for Android Studio, Xcode is clearly focused to application developlment and for make the life easier for developers while developing. While Android Studio seems to try to undermine your project, its slower, there is a lot of nonsense change from one version to another, which is troublesome for the developers since they have to change the code of the project from version to version, even in a small and unimportant ways like changing the name to the dependencies, making the project unable to compile. Simulators and emulators are also slower; there is no seamless integration between the different code languages and frameworks, and many details that makes the development with Android digusting.

Finally, comment that personally I never like mobile programming, or the smartphones. But I found the course very entertaining and interesting, at least the part related to iOS, because Android Stduio have just reinforced the idea to not develop for Android. Although the idea to develop my own music player application has come to my mind more than one time. Also I think that have knowledge about mobile programming nowadays is very usefull; since is a growing market, and develop skills is always a good idea.

Also I would like to thank my teacher for the help and the patience, since due to some personal circumstances I have not been present in most of his clases and I would like to come to all the iOS classes since I really liked Xcode and developing for iOS seems a good idea.

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

ESAT
INNOVATION SCHOOL

ESCUELA
SUPERIOR DE
ARTE Y
TECNOLOGÍA