

De Android Compose a Compose Multiplataforma: lo que no te cuentan

Alejandro Gómez | Robotics Engineer | Sr Android Engineer

Alejandro Gómez

Control Engineer | Robotics Engineer

(@aldajo92) | Github | Linkedin



Compose Multiplatform vs Kotlin Multiplatform

Get started

- ▶ Kotlin Multiplatform overview
- ▼ Create an app with shared logic and native UI
 - Get started with Kotlin Multiplatform – tutorial
 - 1. Set up an environment
 - 2. Create your first cross-platform app
 - 3. Update UI
 - 4. Add dependencies
 - 5. Share more logic
 - 6. Wrap up your project
- ▼ Create an app with shared logic and UI
 - Get started with Compose Multiplatform – tutorial
 - 1. Set up an environment
 - 2. Create your multiplatform project
 - 3. Explore composable code
 - 4. Modify the project
 - 5. Create your own application
- Make your app multiplatform
- ▶ Develop with Kotlin Multiplatform

Get started with Kotlin Multiplatform

Share a UI with Compose Multiplatform or keep it native? Start from scratch or migrate from an existing Android app? Do it your way!



Share logic and keep UI native

Create your first Kotlin Multiplatform app with a native UI for Android (Jetpack Compose UI) and iOS (SwiftUI)



Share both logic and UI

Create your first Kotlin Multiplatform app with a shared UI across Android, iOS, and desktop using Compose Multiplatform

Migration

Live Demo by Gustavo Gómez

Gustavo Gómez

Electrical Engineer (Student)

(@GGomezMorales) | [Github](#) | [Linkedin](#)



Live Demo: Android Compose

https://github.com/GGomezMorales/AndroidKt_Ice_Cream_Shop

Live Demo by Juan Gómez

Juan Gómez

Control Engineer

(@jumagova) | [Github](#) | [Linkedin](#)



Basic migration to Compose Multiplatform

<https://github.com/Jumagova/IceCreamShop-KMM>

When should I use Compose Multiplatform (or KMM)?

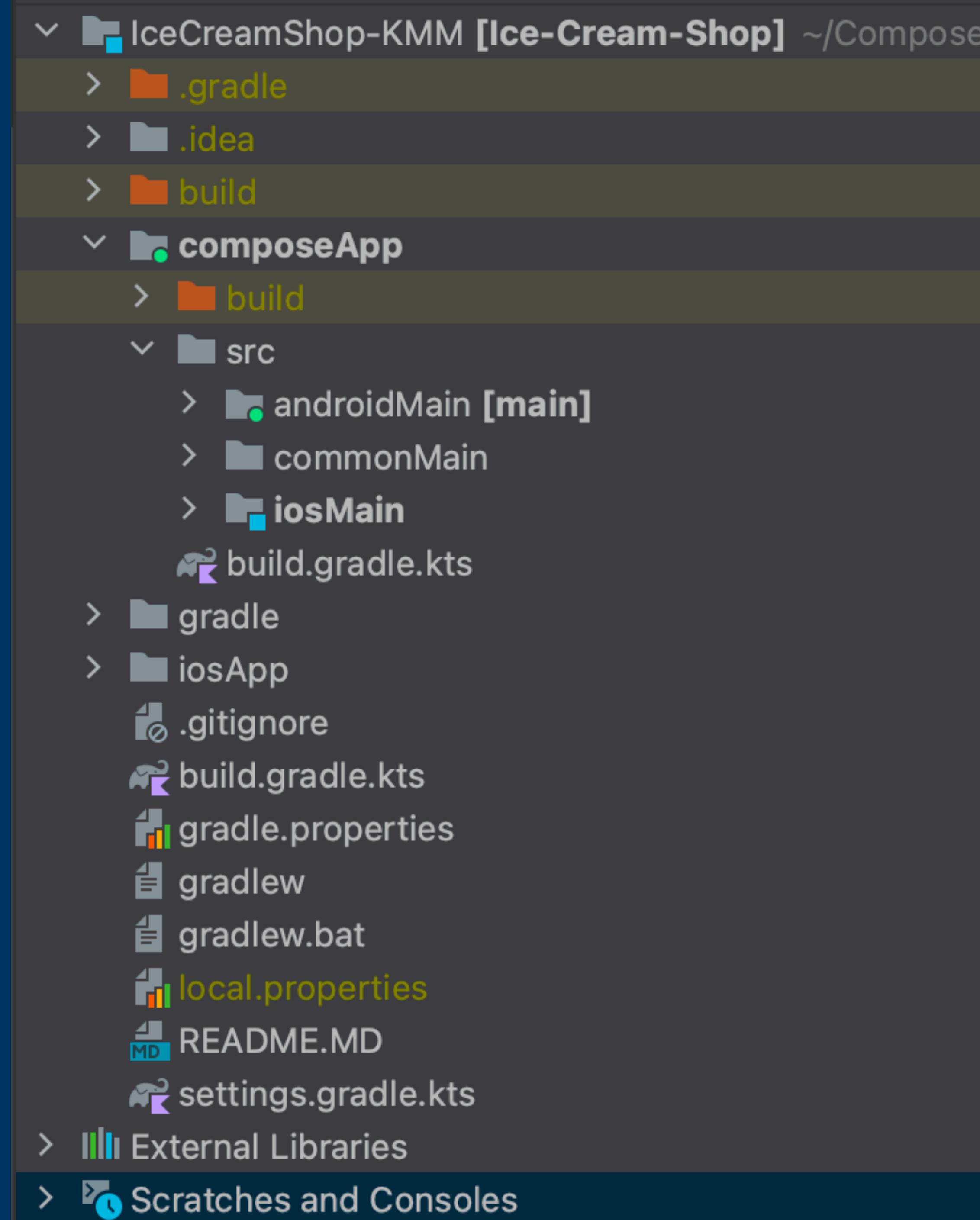
**There is two ways to start working with
Compose Multiplatform:**

- From Android Studio: Bad idea
- Template wizard: God idea

The screenshot shows a dark-themed web application titled "Compose Multiplatform Wizard". At the top, there's a navigation bar with icons for file operations, a shield logo, and links to Apple, Bing, Google, and Yahoo. Below the header is a toolbar with various icons. The main title "Compose Multiplatform Wizard" is displayed prominently. On the left, there's a large blue hexagonal icon. The central part of the page contains input fields for "Project name" (set to "Multiplatform App") and "Project ID" (set to "org.company.app"). Below these are four platform selection buttons: ANDROID (checked), DESKTOP (checked), IOS (checked), and BROWSER (checked). The main content area displays several library cards:

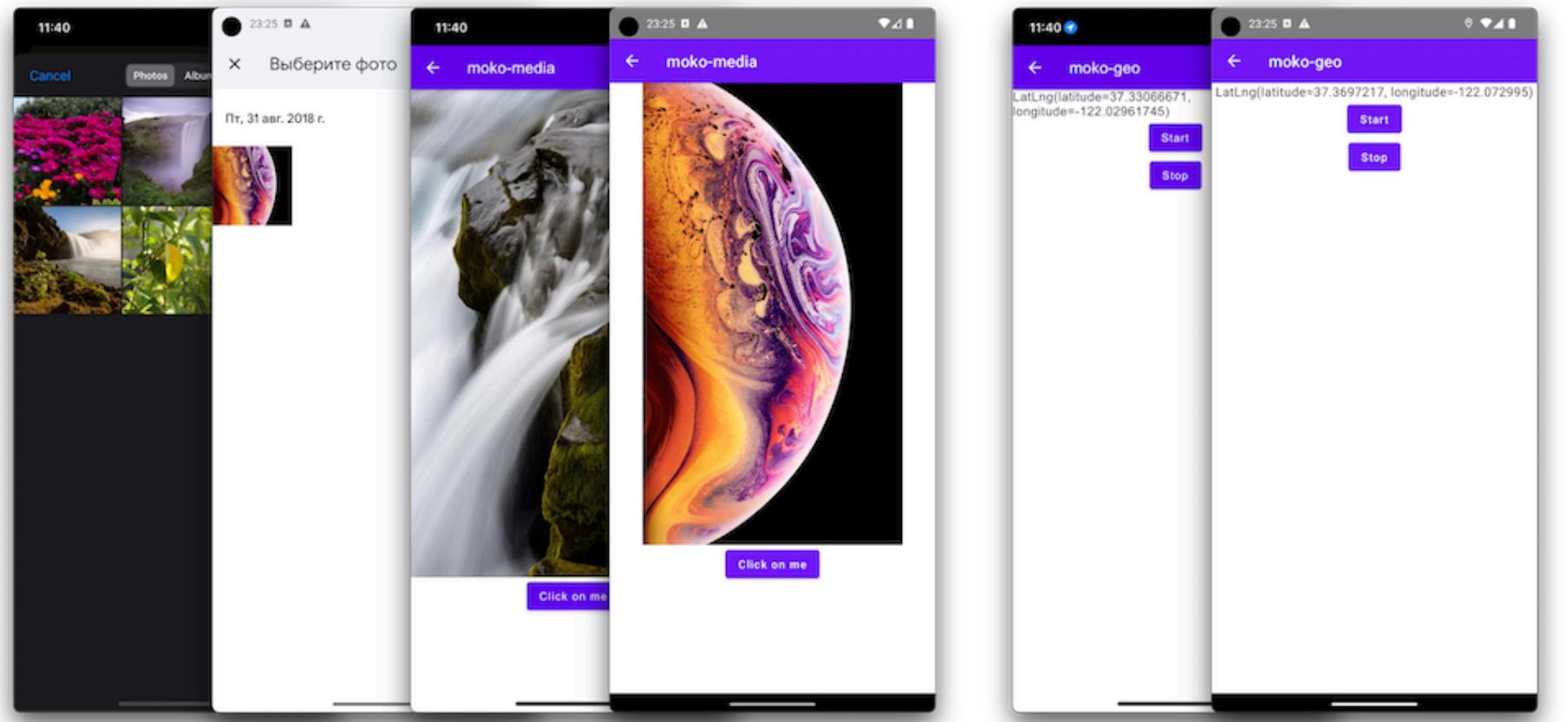
- Voyager** (checked): A pragmatic navigation library for Compose. Buttons: OTHER, INFO.
- Compose ImageLoader** (checked): Compose Image library for Kotlin Multiplatform. Button: INFO.
- Napier** (checked): Napier is a logger library for Kotlin Multiplatform. Buttons: OTHER, INFO.
- Build Config** (checked): A plugin for generating BuildConstants. Buttons: OTHER, INFO.
- Kotlinx Coroutines** (checked): Library support for Kotlin coroutines with multiplatform support. Button: INFO.
- Moko MVVM** (unchecked): This is a Kotlin Multiplatform library that provides architecture components of Model-View-ViewModel for UI applications. Button: INFO.

Basic project structure





No R Files: Moko do the job



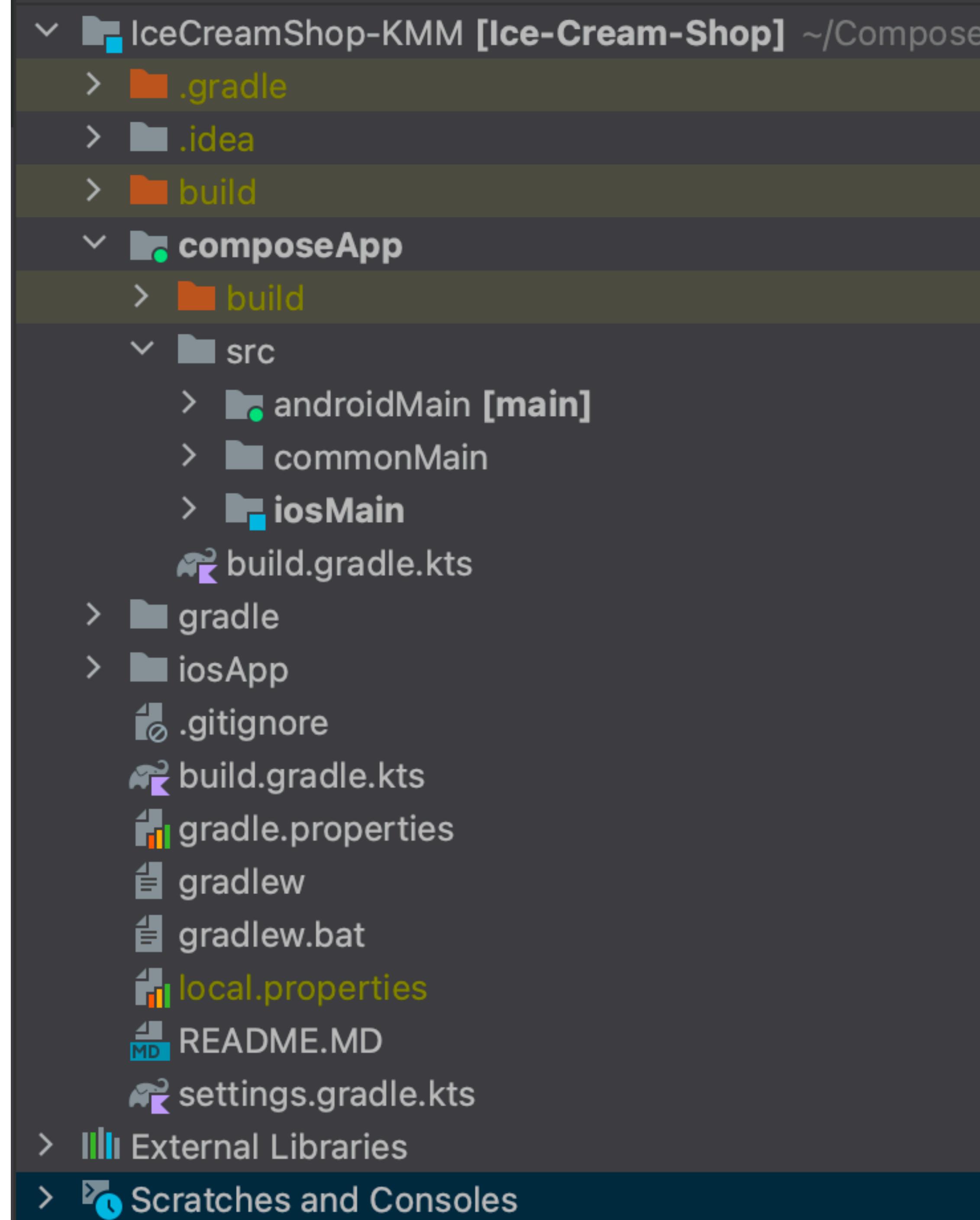
iOS executable

When you use `executable` kotlin target you should add custom build phase to xcode, after kotlin compilation:

```
"$SRCROOT/../gradlew" -p "$SRCROOT/.." :shared:copyResourcesDebugExecutableIosSimulatorArm64
-Pmoko.resources.BUILT_PRODUCTS_DIR=$BUILT_PRODUCTS_DIR \
-Pmoko.resources.CONTENT_FOLDER_PATH=$CONTENTS_FOLDER_PATH
```

Caution with:

- modules:
 - shared
 - android
 - iOS
- actual vs expect (lets see the code)



Gradle configurations have been slightly modified, given that projects now come with Gradle Kotlin DSL (gradle.kts) by default.

Dependencies

Basic to start

- Clean Arquitecture is the smart way for any migration
- Compose Multiplatform + KMM can save your job
- Networking: KTOR
- Caution with Android context
- Gradle (groovy -> kotlin)
- Custom UI -> Wrapper
- Limitaciones en HW

iOS side

Caution with:

- **info.plist**
- **swift concepts**

Swift UI

```
var body: some View {
    VStack(alignment: .leading) {
        Text("All Coins")
            .font(.headline)
            .padding()
        
        HStack{
            Text("Coins")
            Spacer()
            Text("Price")
        }
        .font(.caption)
        .foregroundColor(.gray)
        .padding(.horizontal)
        
        ScrollView{
            VStack{
                ForEach(viewModel.coins){ coin in
                    CoinRowView(coin:coin)
                }
            }
        }
        .padding(.horizontal)
    }
}
```



Swift UI: View Model

```
import SwiftUI
class CryptoViewModel: ObservableObject {
```

Create a ViewModel class: This class should inherit from **ObservableObject** protocol.

Swift UI: View Model

```
import SwiftUI
class CryptoViewModel: ObservableObject {
    @Published var coins = [Coin]()
    @Published var topMovingCoins = [Coin]()
}
```

Declare properties to hold the data you want to expose to the View. Use **@Published** to make them observable, meaning changes will be automatically reflected in the View.

Swift UI: View Model

```
import SwiftUI
class HomeViewModel: ObservableObject {
    @Published var coins = [Coin]()
    @Published var topMovingCoins = [Coin]()

    init() {
        fetchCoinData()
    }

    func fetchCoinData() {

    }
}
```

The **init** block marks the starting point of the ViewModel's lifecycle. Then create functions for any business logic related to the data.