De Android Compose a Compose Multiplataforma:

lo que no te cuentan

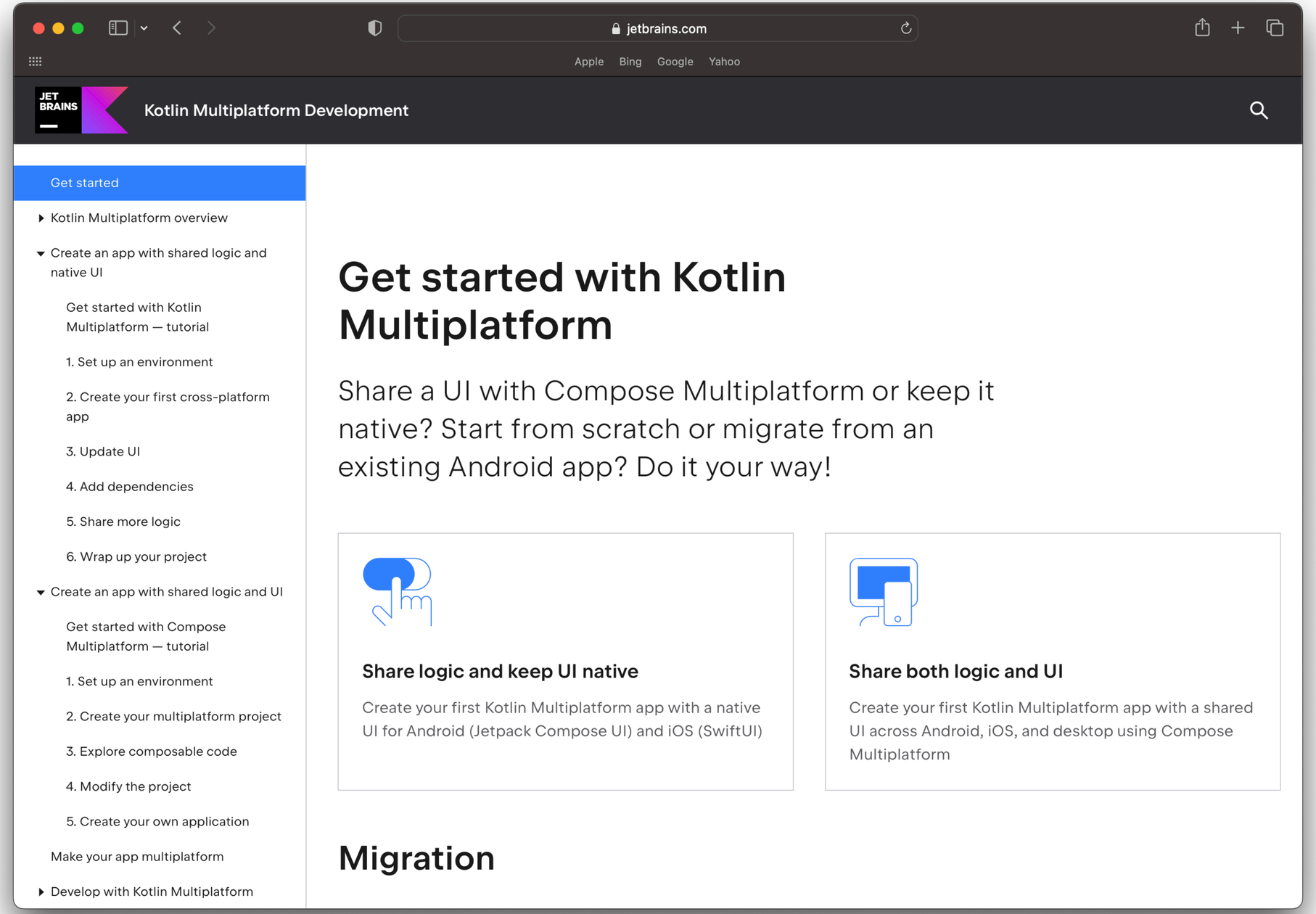
Alejandro Gómez | Robotics Engineer | Sr Android Engineer

Alejandro Gómez

Control Engineer | Robotics Engineer

(@aldajo92) | Github | Linkedin

Compose Multiplatform vs Kotlin Multiplatform



https://www.jetbrains.com/help/kotlin-multiplatform-dev/get-started.html

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

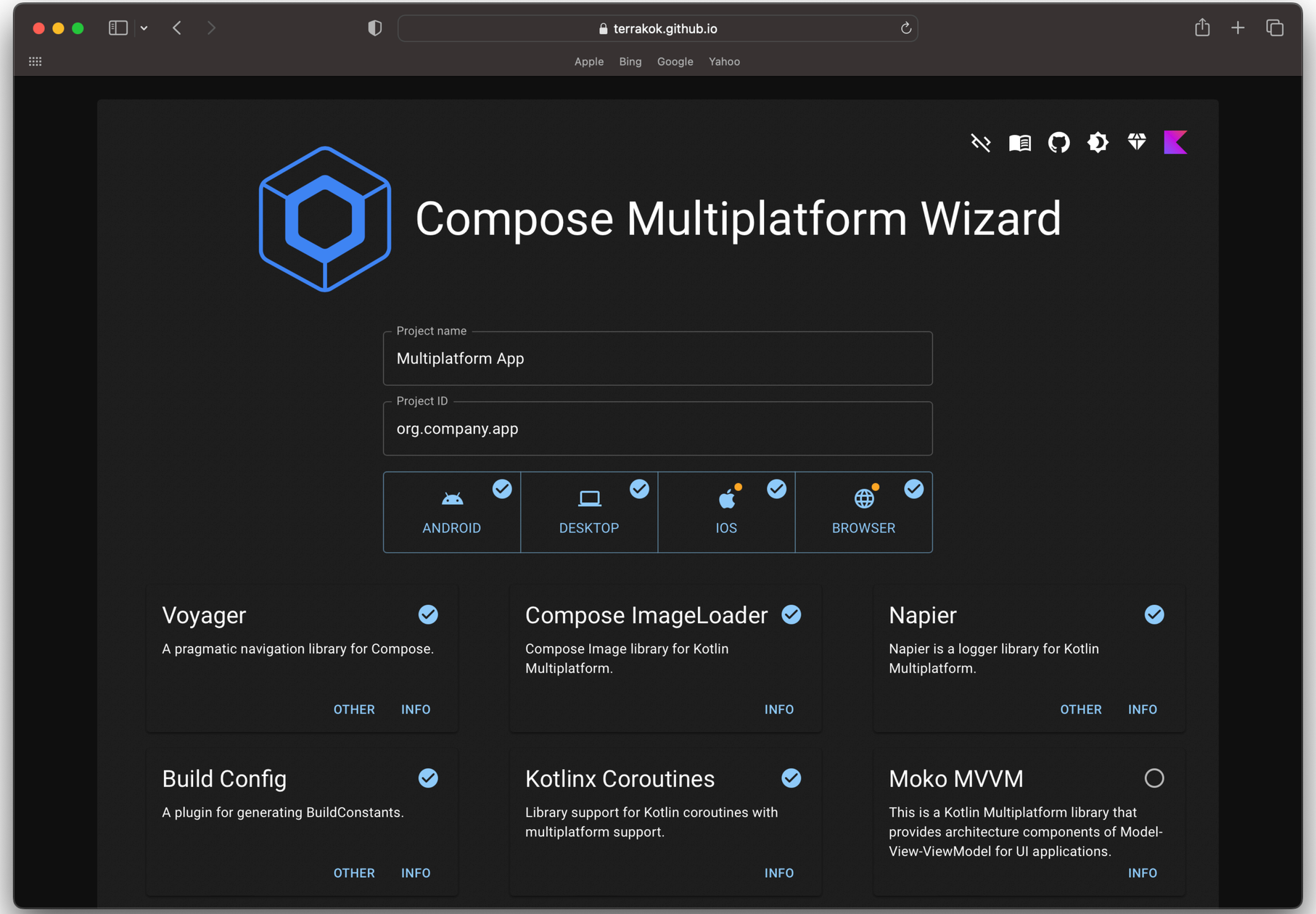
https://github.com/Jumagova/IceCreamShop-KMM

When should I use Compose Multiplatform (or KMM)?

There is two ways to start working with Compose Multiplatform:

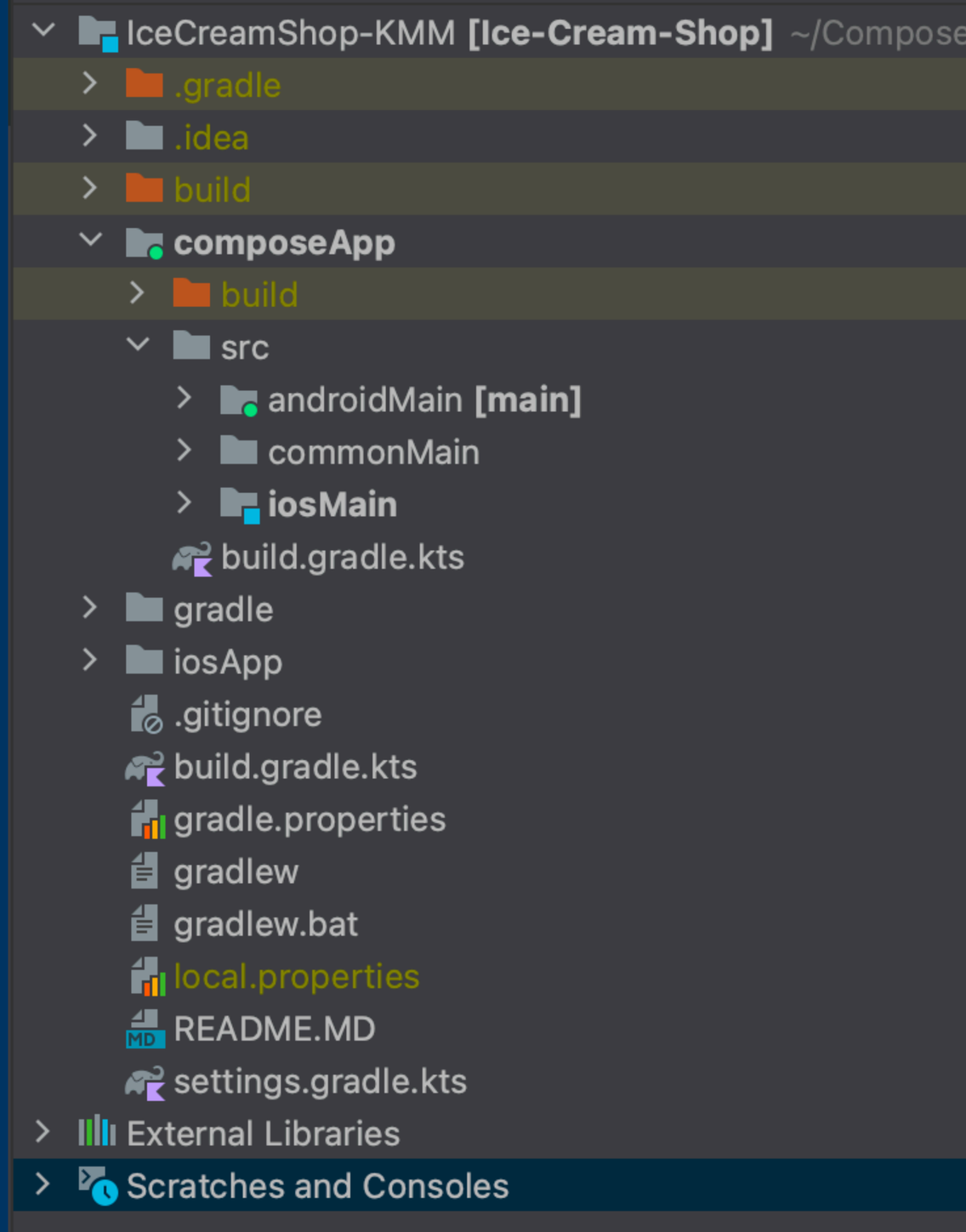
- From Android Studio: Bad idea
- Template wizard: God idea

https://terrakok.github.io/Compose-Multiplatform-Wizard/

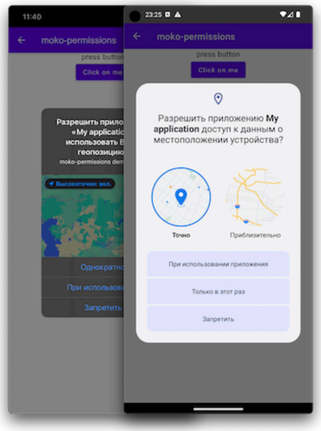


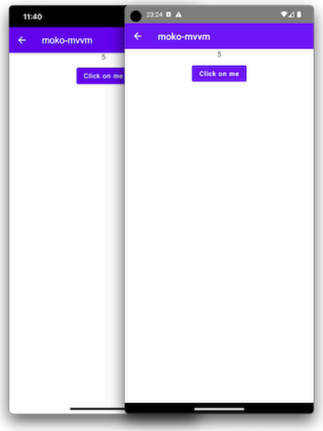
https://terrakok.github.io/Compose-Multiplatform-Wizard/

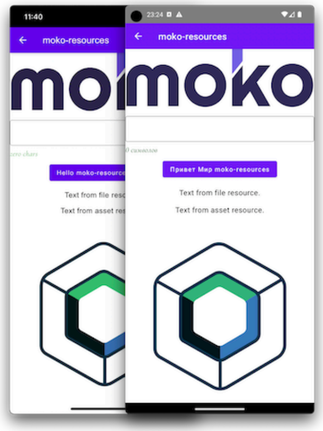
Basic project structure

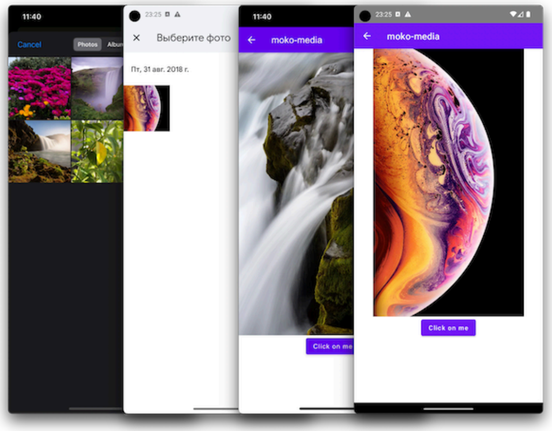


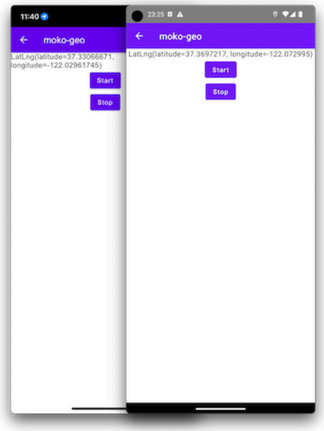
No R Files:
Moko do the job











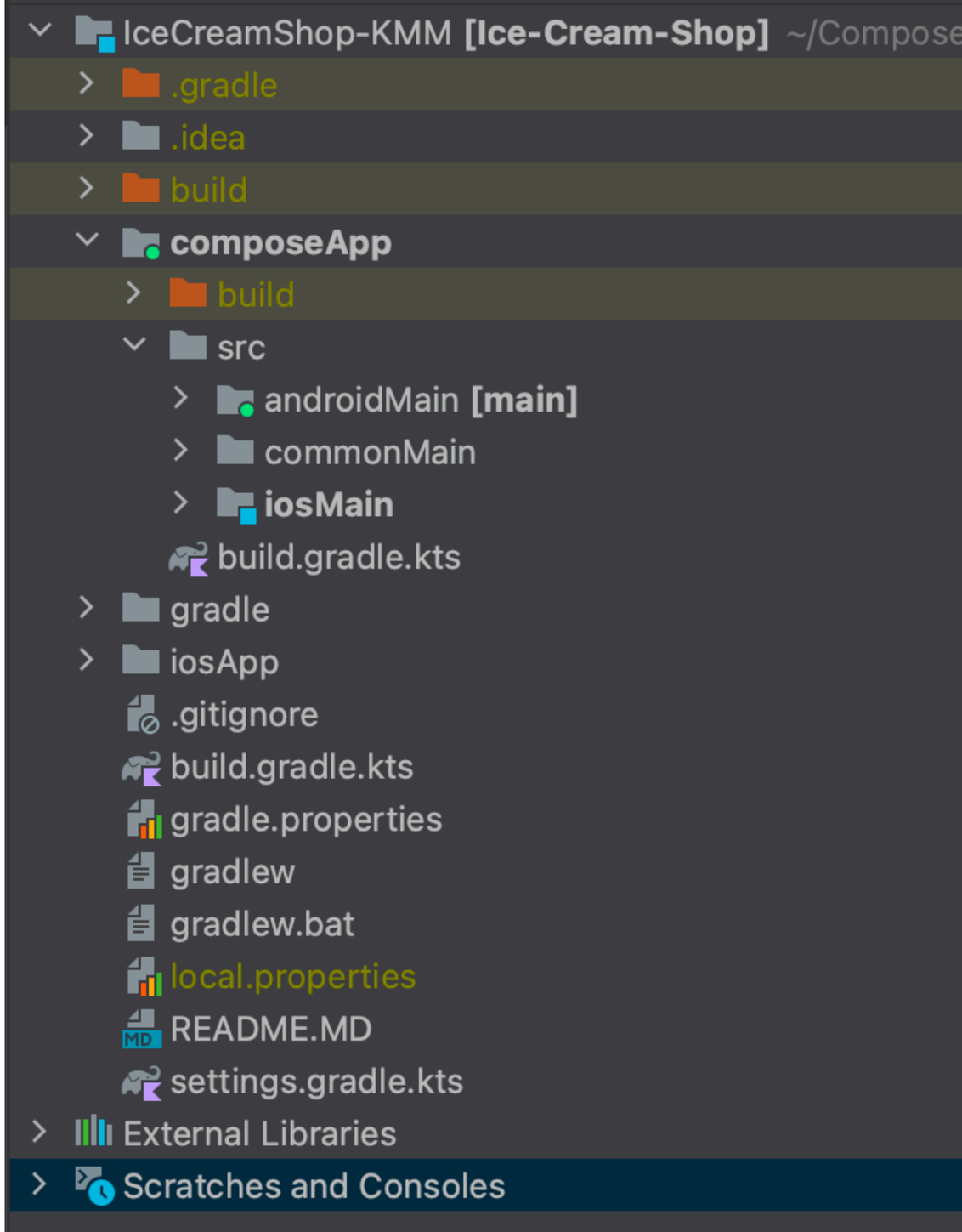
iOS executable

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=$BUILT_PRODUCTS_DIR \
    -Pmoko.resources.CONTENTS_FOLDER_PATH=$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

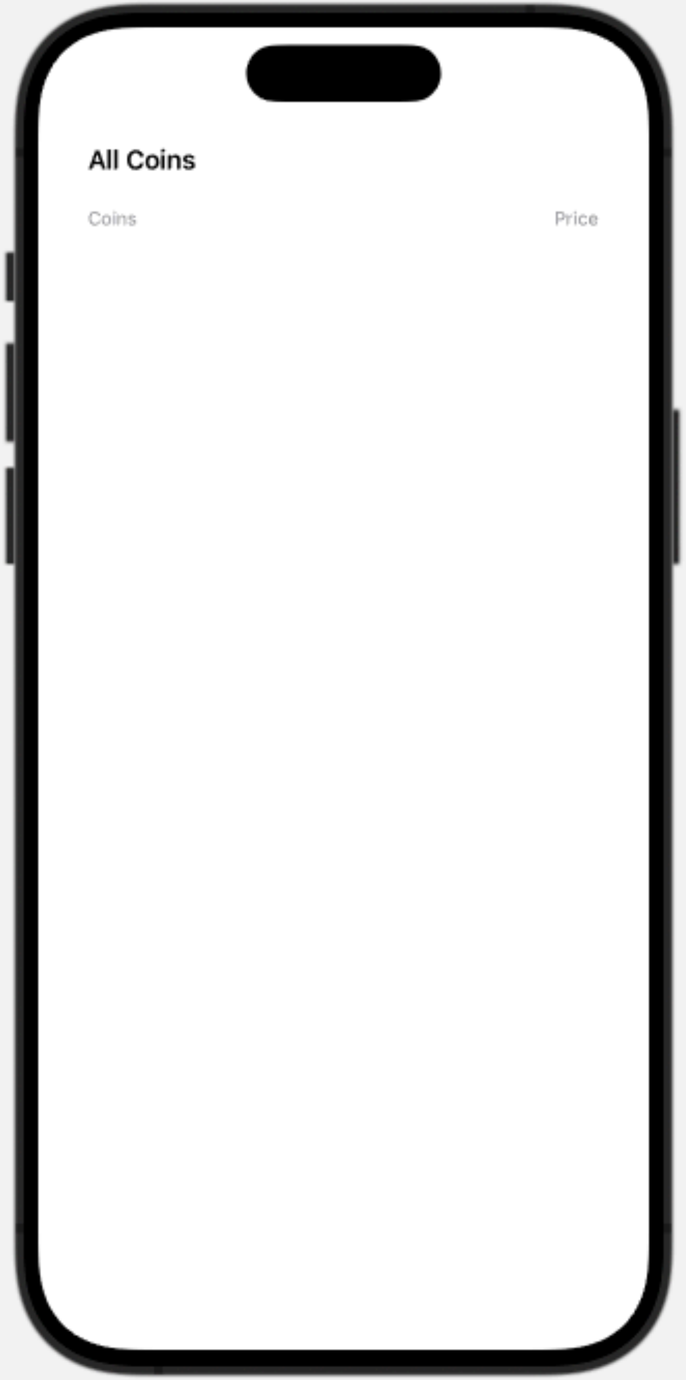
Caution with:
- info.plist
- swift concepts

Swift UI

```swift
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.