Setting Up Multiple Environments in React Native for iOS and Android

In React Native development, managing different configurations for multiple environments (e.g., development, staging, production) is a common practice. This post walks you through setting up a React Native project that targets multiple environments for both **iOS** and **Android** using the `react-native-config` package.

Prerequisites

Before we dive into the steps, ensure you have the following installed on your machine:

- **Node.js**
- **Yarn**
- **Xcode** (for iOS)
- **Android Studio** (for Android)

If you don't have these installed, you can follow the official documentation for each of them.

Step 1: Creating a New React Native Project

- First, create a new React Native project by running the following command:

```
npx @react-native-community/cli init MultiTarget --version 0.76.1
```

This project will be set up with React Native version 0.76.1 (stable at the time of implementation).

- Next, navigate to your project folder and start the React Native app:

```
yarn start
```

Step 2: Running the App on Android and iOS

- To run your app on **Android**, use the following command:

```
yarn android
```

- For **iOS**, navigate to the `ios` directory and install CocoaPods:

```
cd ios && pod install && cd ..
```

- Then, run the app on the iOS simulator:

```
yarn ios
```

- You should now see the default React Native landing page on your simulator/emulator.
- If you are facing any issues while installing you can check react native official documentation.

We'll be using the react-native-config module to expose config variables to your javascript code in React Native, supporting iOS, Android.

Step 3: Installing `react-native-config` for Managing Environment Variables

- We'll use the `react-native-config` module to manage environment-specific variables in our project. Install the module by running:

```
yarn add react-native-config
```

- After adding the module, run the following command to install the necessary pods:

```
cd ios && pod install && cd ..
```

Next Step is to create different environment files for each environment.
Here we are taking three environments: development, staging, and production

Step 4: Creating Environment Files

Create all three files in the root project directory as .env.development, .env.staging, and .env.production.

Inside `.env.development`, write ENV variable as:

```
ENV=development
```

Inside `.env.staging`, write ENV variable as:

```
ENV=staging
```

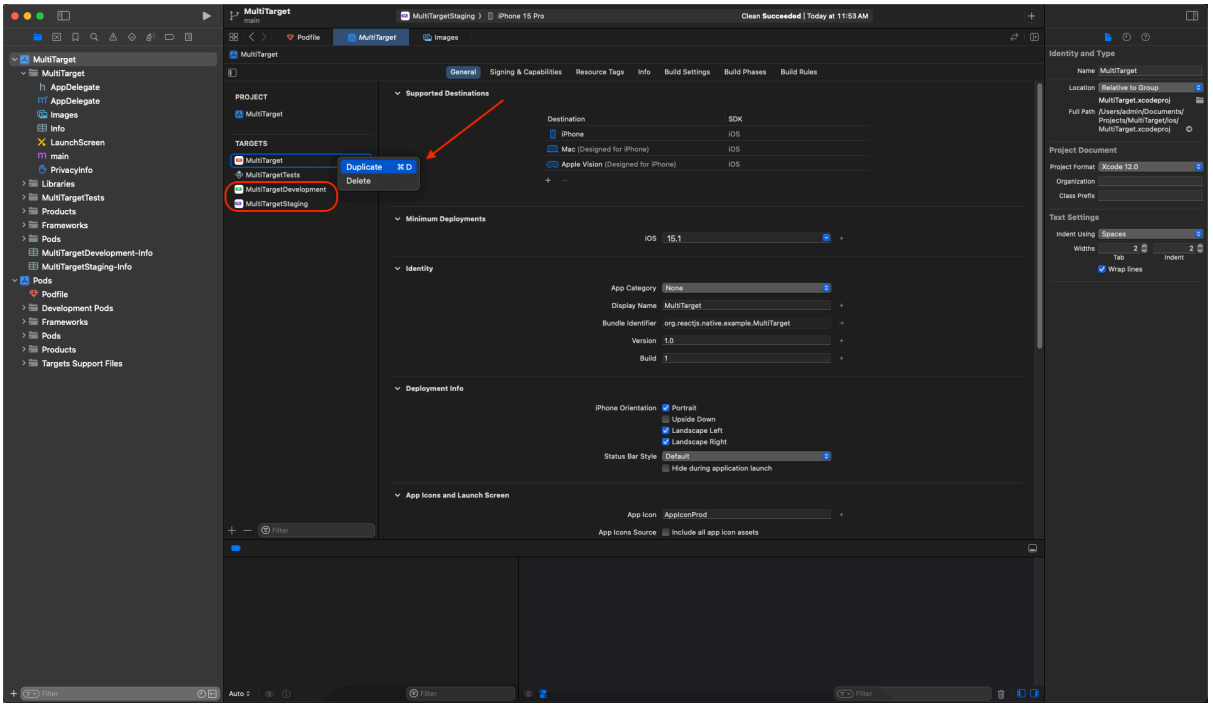
Inside `.env.production`, write ENV variable as:

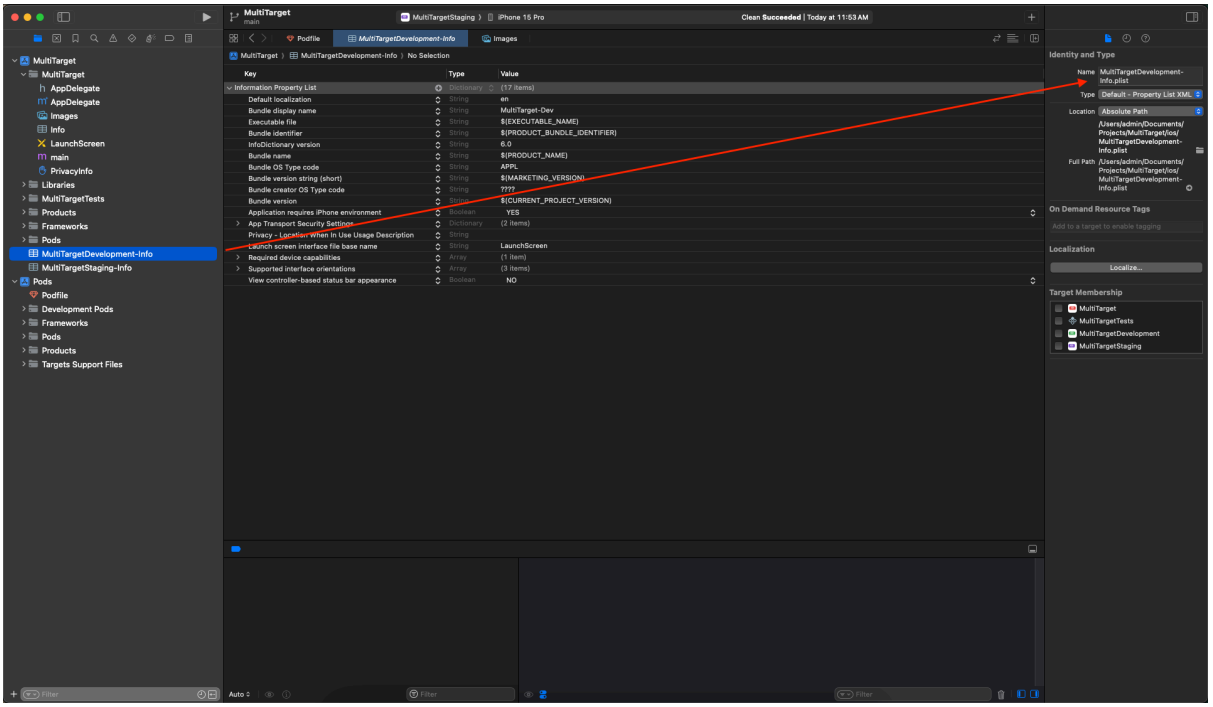
```
ENV=production
```

Step 5: iOS-Specific Configuration

5.1: Setting Up Multiple Targets in Xcode

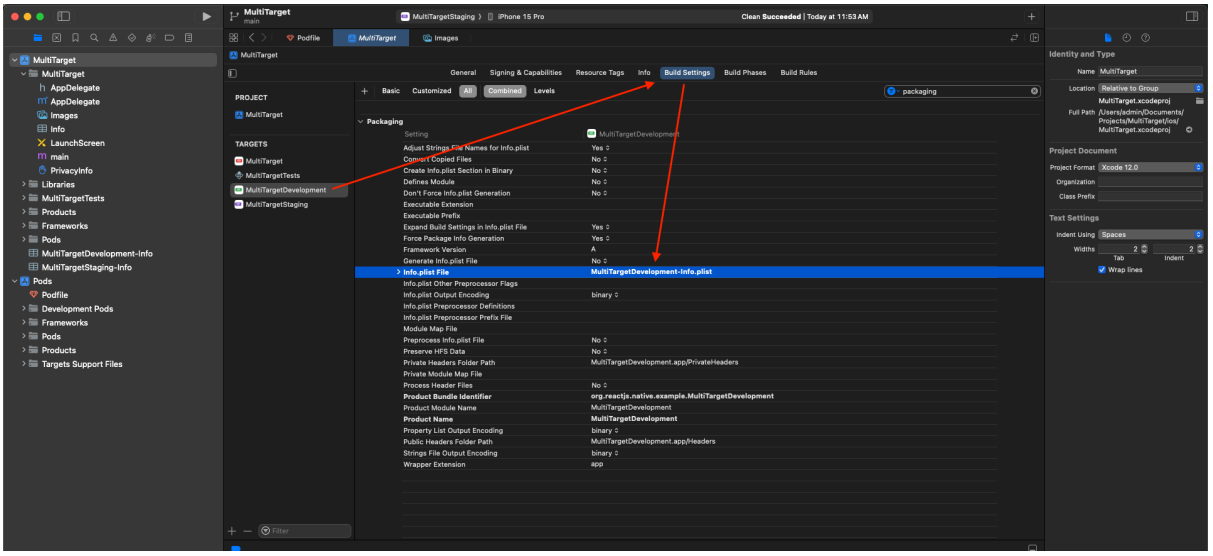
1. Open the `.xcodeproj` file in Xcode.
2. In the **General** tab, create two duplicate targets. One will be for **production**, and the other will be for **development** and **staging**.
3. Rename the targets as `MultiTargetDevelopment` and `MultiTargetStaging`.
4. Rename the corresponding `.plist` files as `MultiTargetDevelopment-Info.plist` and `MultiTargetStaging-Info.plist`.

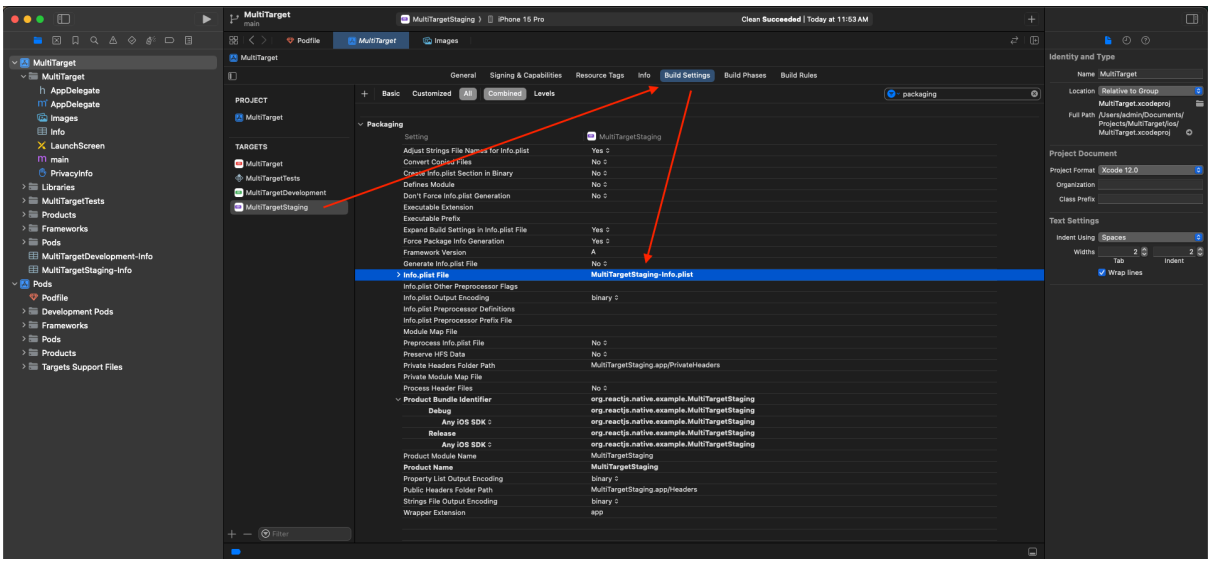




5.2: Assigning `.plist` Files to the Targets

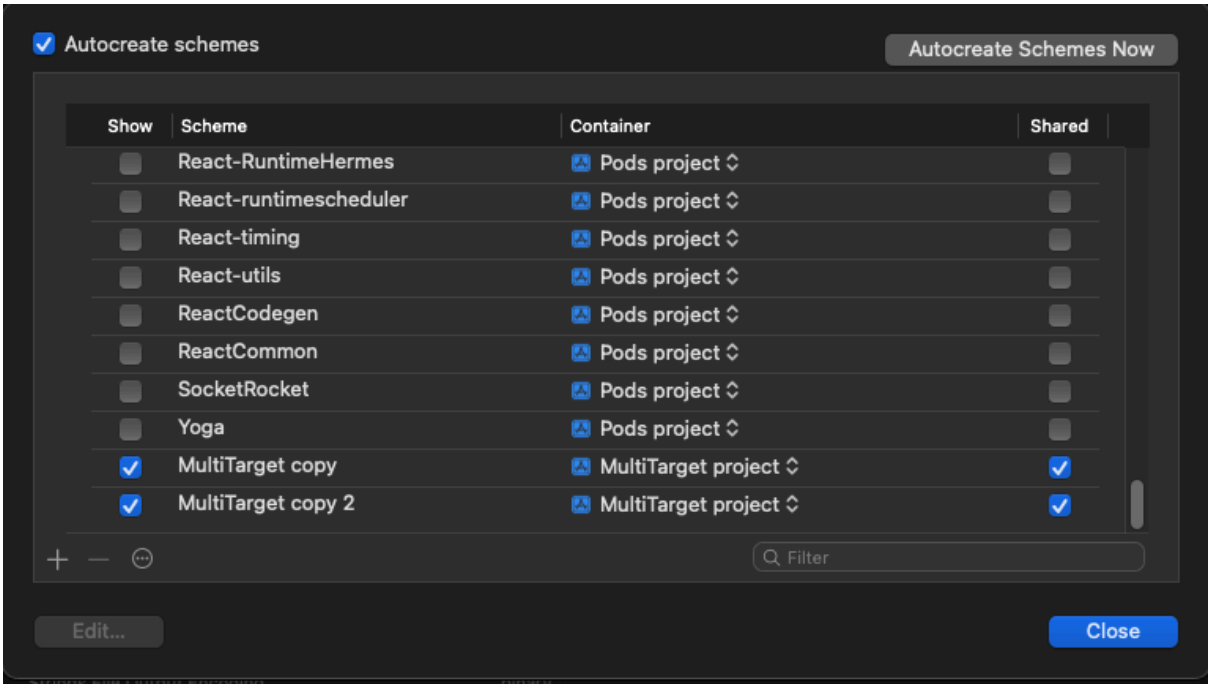
- For each target (development, staging, and production), assign the correct `.plist` file in the **Build Settings** tab under **Packaging**.



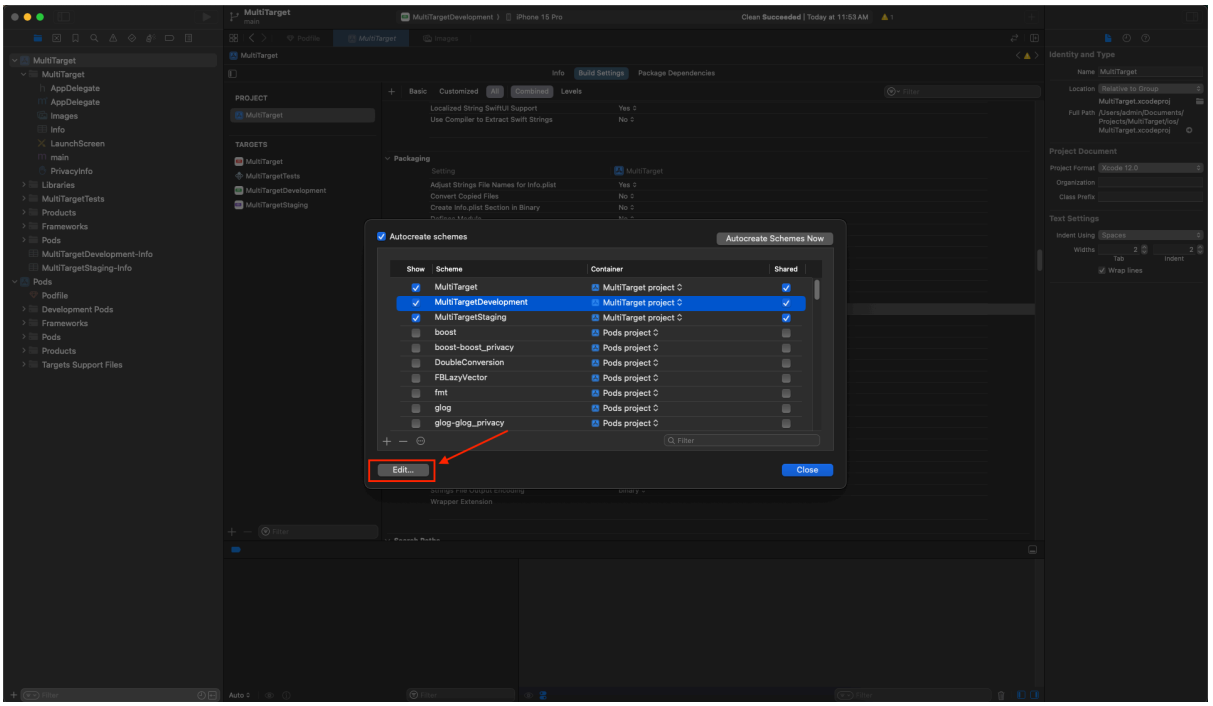


5.3: Creating and Managing Schemes

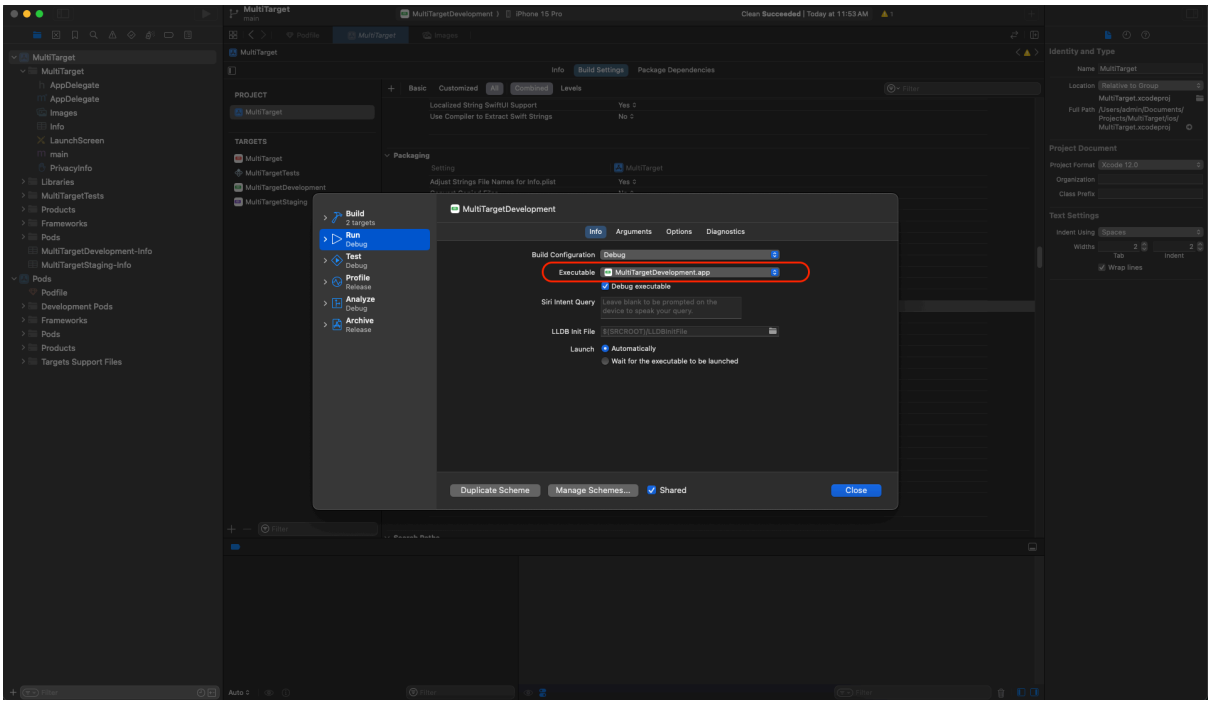
- Now we will be creating schemes for our apps. `Go to Product -> Scheme -> Manage Schemes...`
- You will see 2 copies of Target with `MultiTarget Copy` and `MultiTarget Copy 2` names.



- Change this to `MultiTargetDevelopment` and `MultiTargetStaging`.
- Next step is to Go to `Edit` for each scheme.



- Ensure `multiplatformDevelopment.app` is selected in Executable as:

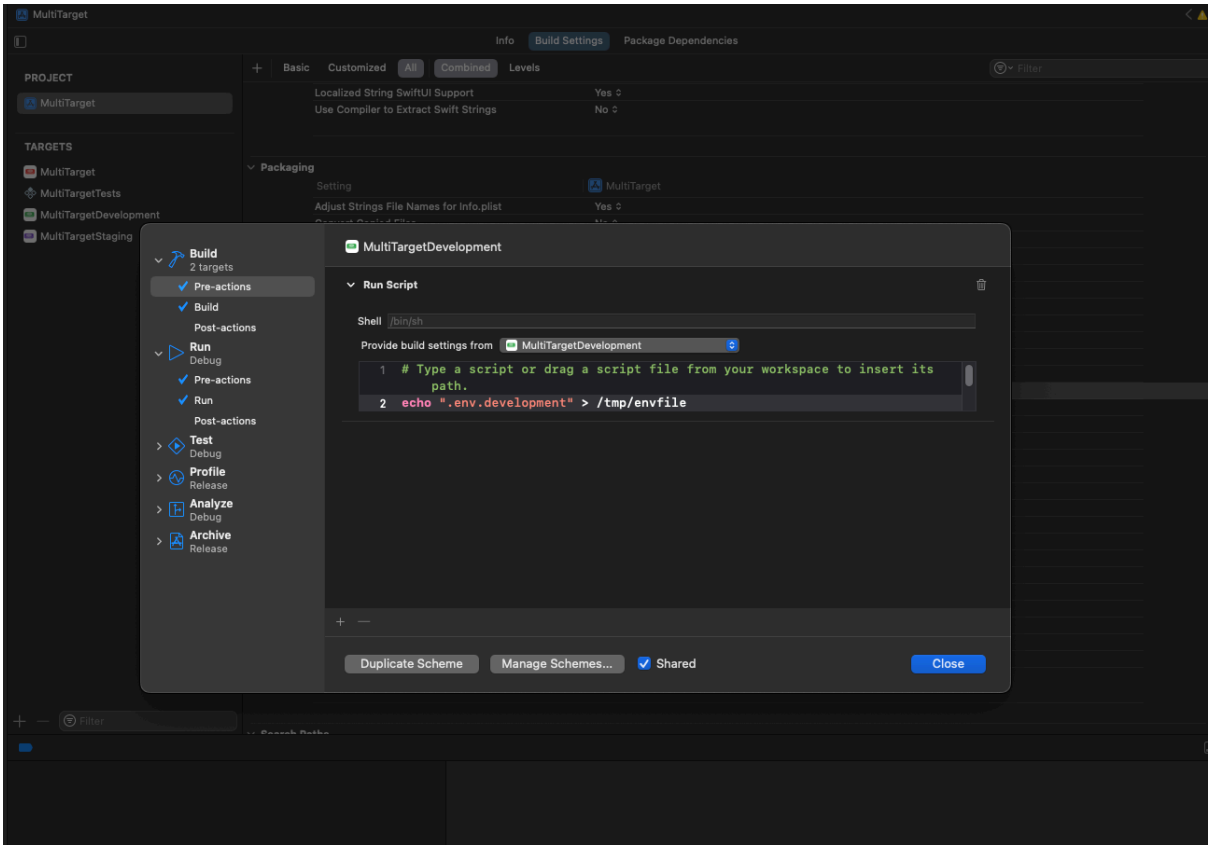


5.4: Adding a Pre-Action Script for Environment Variables

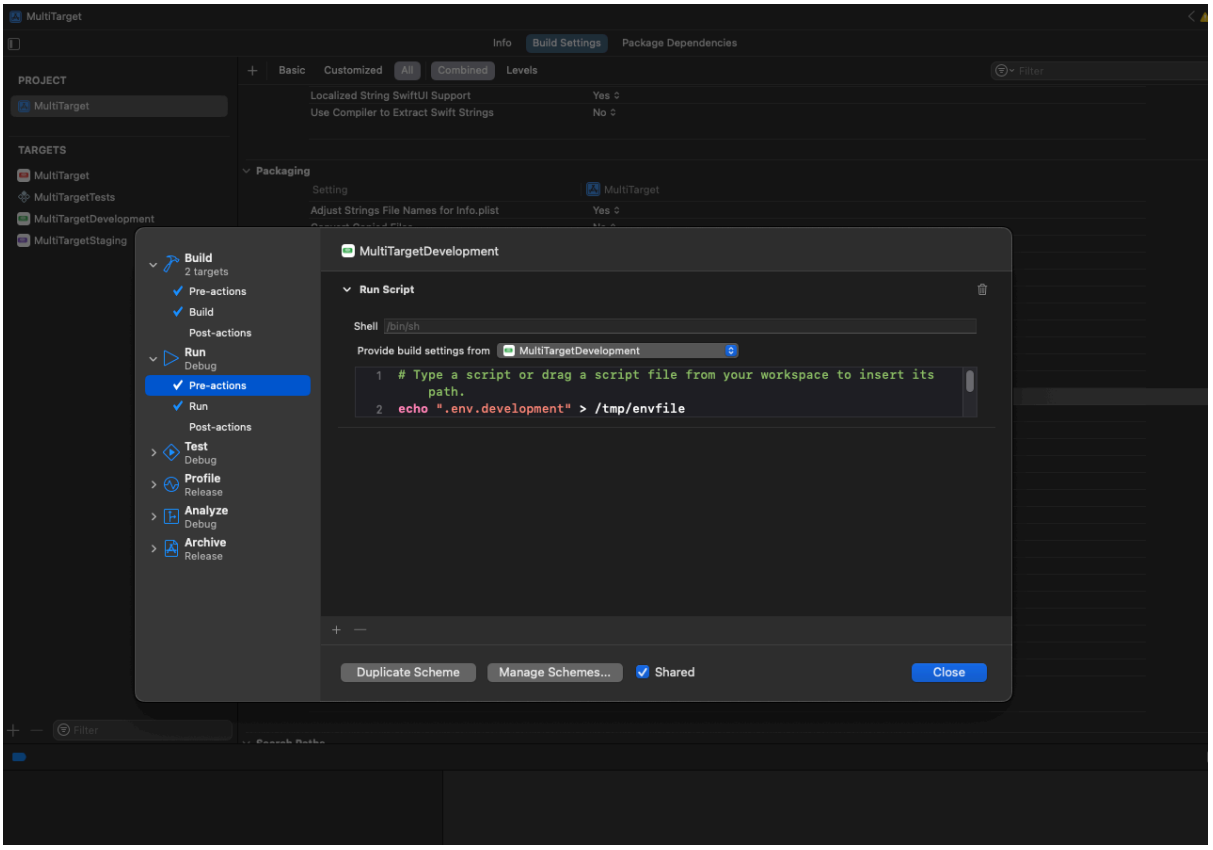
- Now we need to add a script for ensuring we have the correct `env` file for each target.
- Under **Build -> Pre-actions**, add a new **Run Script Action.** If no Run Script present you can create one.
- Click on “+” icon and then `select "New Run Script Action"`.

1. Add the following script for the **development** scheme:

```
echo ".env.development" > /tmp/envfile
```



- Next step is `Go to Run -> pre-actions`. Add same script for development.

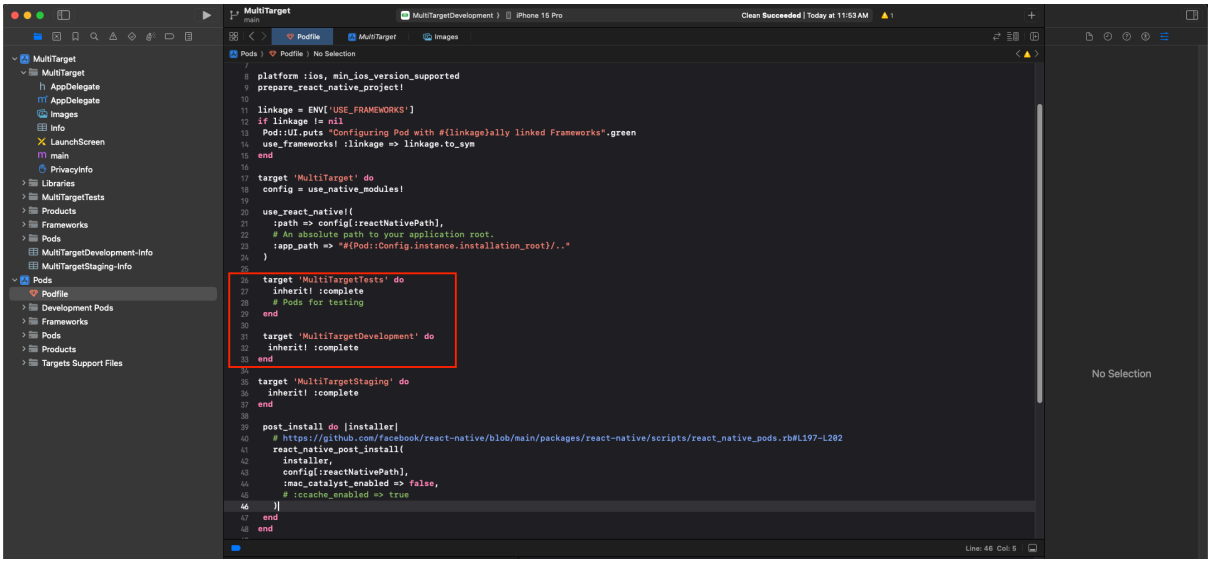


- Repeat this for **staging** and **production**, using their respective environment files:
- Script will be
    - Staging
      `echo ".env.staging" > /tmp/envfile`
    - Production
      `echo ".env.production" > /tmp/envfile`

5.5: Configuring the `Podfile`

Next step is to add targets in the podfile. Go to podfile and add this targets as shown:

```
target 'MultiTargetDevelopment' do
  inherit! :complete
end

target 'MultiTargetStaging' do
  inherit! :complete
end
```

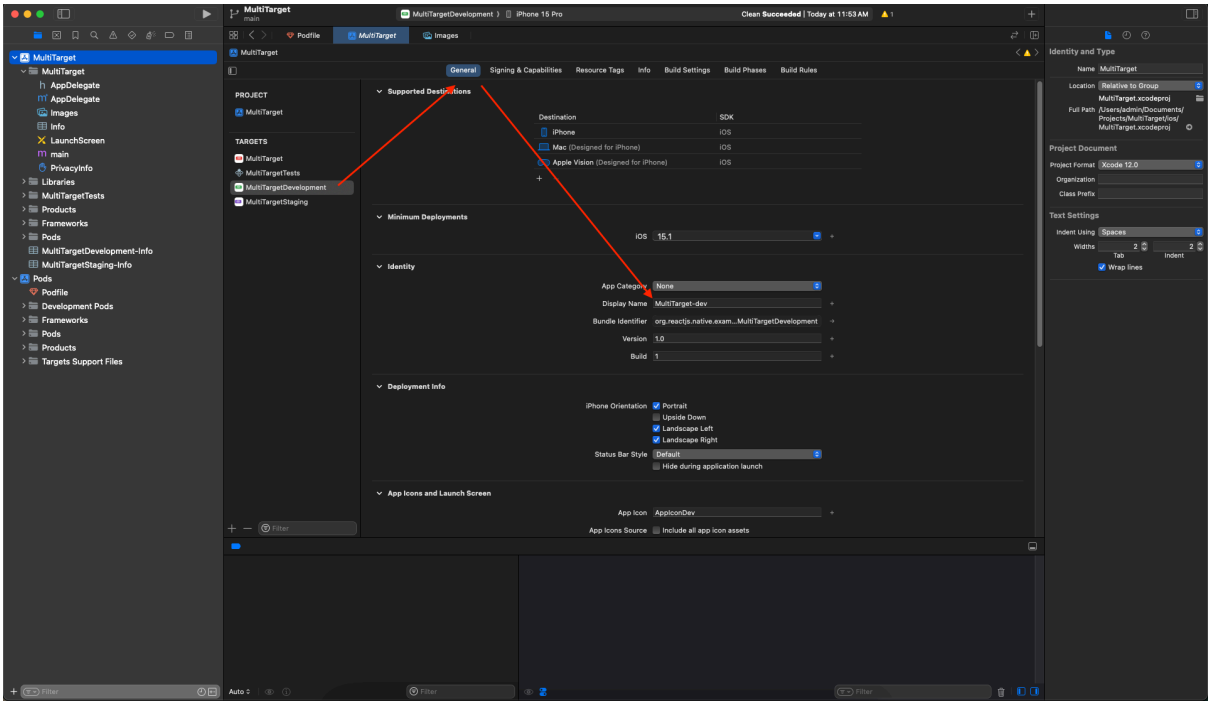


```
platform :ios, min_ios_version_supported
prepare_react_native_project!

linkage = ENV['USE_FRAMEWORKS']
if linkage != nil
 Pod::UI.puts "Configuring Pod with #{linkage}ally linked Frameworks".green
 use_frameworks! :linkage => linkage.to_sym
end

target 'MultiTarget' do
 config = use_native_modules!

 use_react_native!(
   :path => config[:reactNativePath],
   # An absolute path to your application root.
   :app_path => "#{Pod::Config.instance.installation_root}/.."
 )

 target 'MultiTargetTests' do
   inherit! :complete
   # Pods for testing
 end

 target 'MultiTargetDevelopment' do
  inherit! :complete
 end

target 'MultiTargetStaging' do
  inherit! :complete
end

 post_install do |installer|
   # https://github.com/facebook/react-native/blob/main/packages/react-native/scripts/react_native_pods.rb#L197-L202
   react_native_post_install(
     installer,
     config[:reactNativePath],
     :mac_catalyst_enabled => false,
     # :ccache_enabled => true
   )
 end
end
```

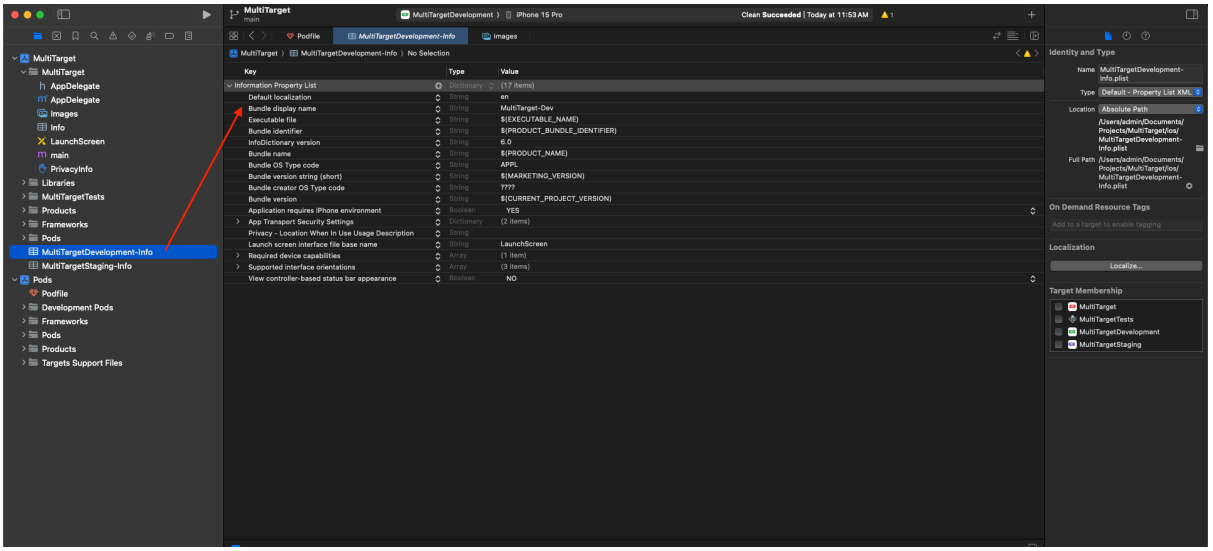
- Run `pod install` from ios directory.

5.6: Naming each Environment and assigning App Icons

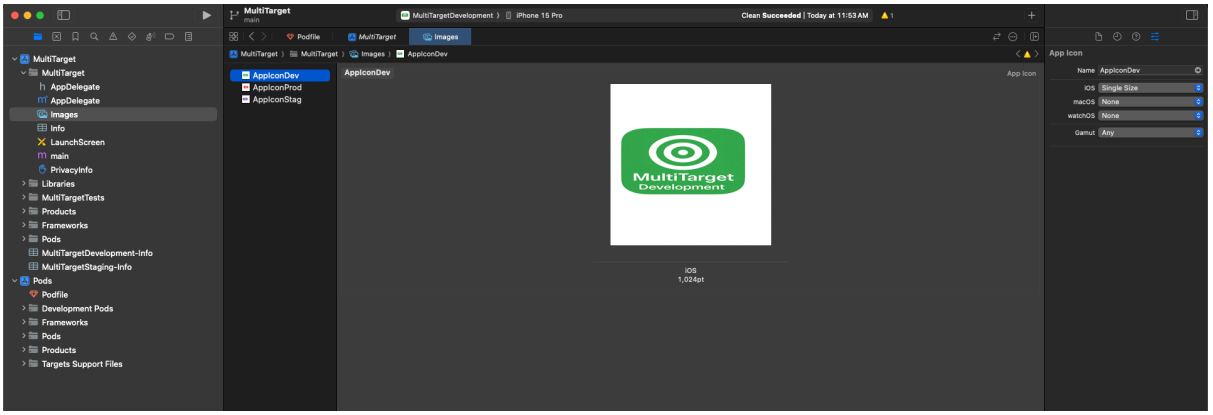
- Go to each target and change the display name to your specific need. Here I am attaching the process of doing so and double check the name from plist for one target. The same goes for all others.



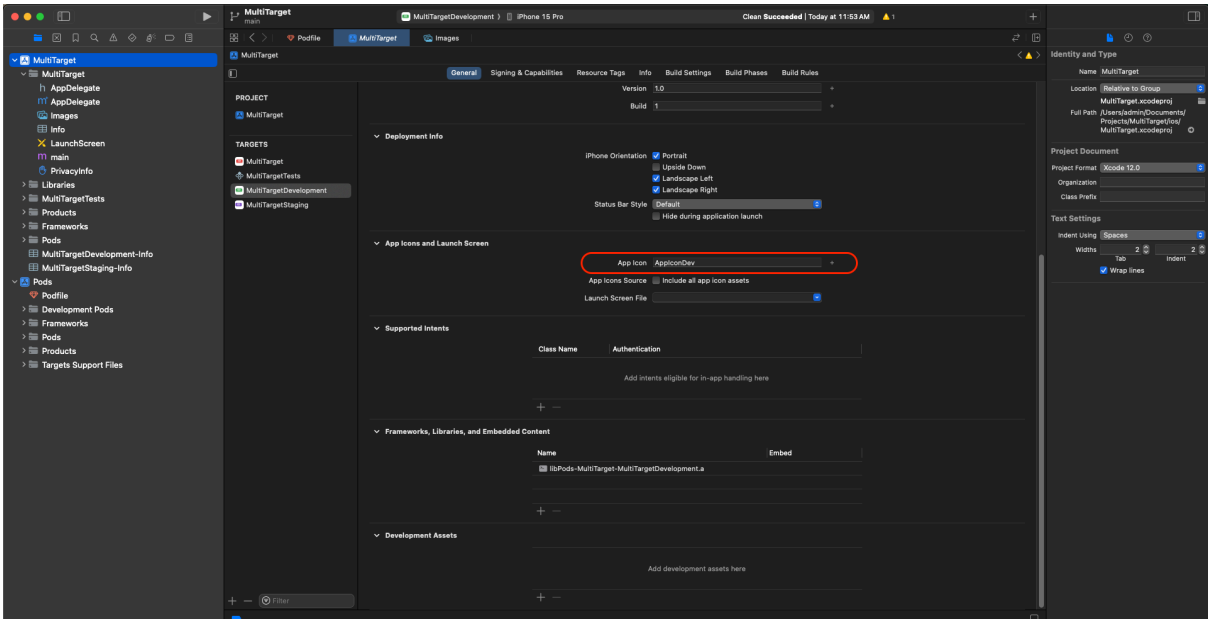
- Double check the `Bundle Display Name` under the plist file.



- Here we have also set different App icons for different Targets.
- Add App Icons in the Assets folder.



- Set appropriate icon name in App Icons and Launch Screen section in General Tab of Appropriate Target.



- Repeat above steps for adding icons for Production and Staging Targets.

Step 6: Android-Specific Configuration

6.1: Setting Up Environment Variables

- In your `android/app/build.gradle`, add the following code to apply the environment files:

```
project.ext.envConfigFiles = [
       production: ".env.production",
       staging: ".env.staging",
       development: ".env.development"
]
apply from: project(':react-native-config').projectDir.getPath() +
"/dotenv.gradle"
```

6.2: Defining Product Flavors

- Now add `productFlavours` in android block:

```
flavorDimensions "default"
productFlavors {
   production {
       minSdkVersion rootProject.ext.minSdkVersion
       applicationId "com.multitarget"
       targetSdkVersion rootProject.ext.targetSdkVersion
       buildConfigField "String", "ENV_FILE",
'"${project.ext.envConfigFiles.production}"'
   }
   staging {
       minSdkVersion rootProject.ext.minSdkVersion
       applicationId "com.multitarget"
       targetSdkVersion rootProject.ext.targetSdkVersion
       buildConfigField "String", "ENV_FILE",
'"${project.ext.envConfigFiles.staging}"'
   }
   development {
       minSdkVersion rootProject.ext.minSdkVersion
       applicationId "com.multitarget"
       targetSdkVersion rootProject.ext.targetSdkVersion
       buildConfigField "String", "ENV_FILE",
'"${project.ext.envConfigFiles.development}"'
   }
}
```

6.3: Configuring `buildTypes`

- In the `build.gradle` file, ensure the `defaultConfig` and `matchingFallbacks` are correctly set:

```
resValue "string", "build_config_package", "com.multitarget"
```

- Additionally, add `matchingFallbacks` to the `debug` block under `buildTypes`:

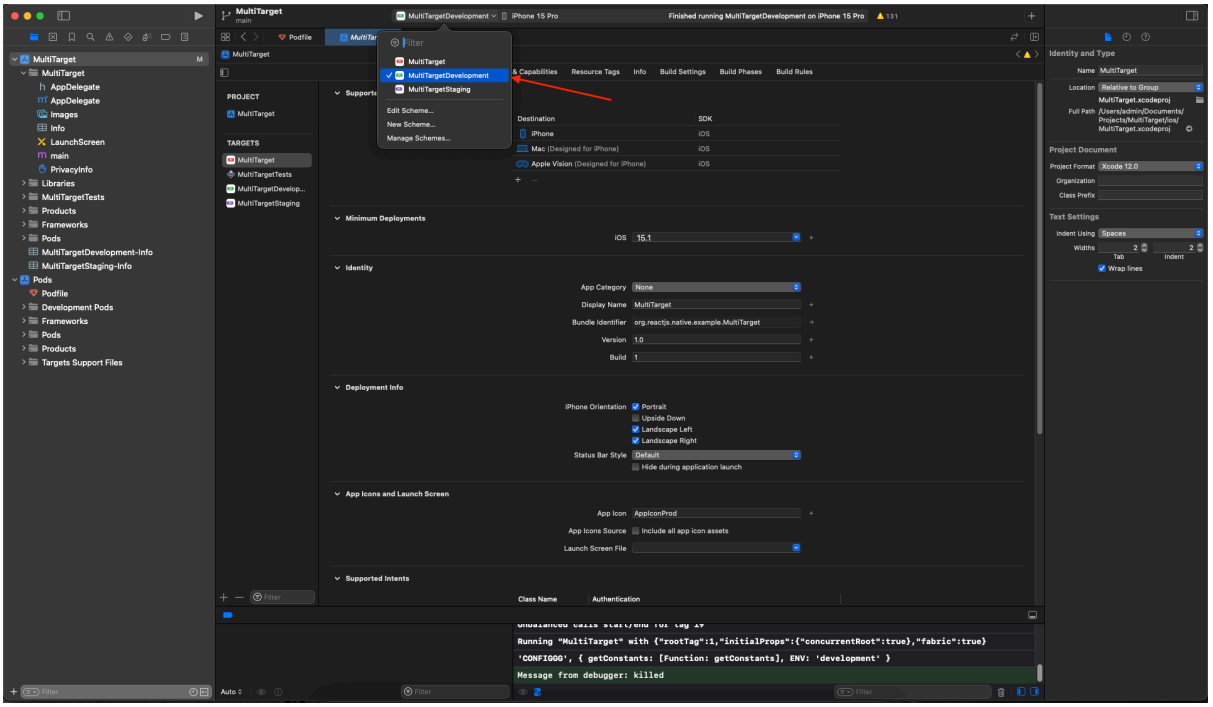
```
matchingFallbacks = ['debug', 'release']
```

Step 7: Running the Apps

7.1: Running iOS

7.1.1: Running from Xcode.app

- For iOS, you can select the appropriate scheme in Xcode and run the app directly. Choose between **development**, **staging**, or **production** and click the play button to launch the app.



7.1.2: Running iOS App with Terminal

Development

```
yarn ios --scheme "MultiTargetDevelopment" --simulator=\"iPhone 15 Pro\"
--mode Debug
```

Production

```
yarn ios --scheme "MultiTarget" --simulator=\"iPhone 15 Pro\" --mode
Debug
```

Development

```
yarn ios --scheme "MultiTargetStaging" --simulator=\"iPhone 15 Pro\"
--mode Debug
```

7.2: Running Android

In this section, we will explore how to run your React Native app for **Android** with different environments: **Development**, **Staging**, and **Production**.

There are two approaches to handle multiple environments in Android:

- **First Approach (Using Product Flavours)** – Separate apps for each environment (Development, Staging, Production).
- **Second Approach (Using Build Types)** – Same app, different environments.

7.2.1: First Approach: Using Product Flavours

- Product Flavours allow you to create separate targets for each environment. In this case, you will have **three different apps**: one for Development, one for Staging, and one for Production.

Running the App for Different Environments

- To run the app, use the following commands. Open the terminal in the root project directory and run the command for the corresponding environment.

- Development
    - Debug Build

```
cd android && ENVFILE=.env.development ./gradlew
assembleDevelopmentDebug && cd .. && adb install
android/app/build/outputs/apk/development/debug/app-development-de
bug.apk && adb shell am start -n com.multitarget/.MainActivity
```

    - Release Build

```
cd android && ENVFILE=.env.development ./gradlew
assembleDevelopmentRelease && cd .. && adb install
android/app/build/outputs/apk/development/release/app-development-
release.apk && adb shell am start -n com.multitarget/.MainActivity
```

- Staging
    - Debug Build

```
cd android && ENVFILE=.env.staging ./gradlew assembleStagingDebug
&& cd .. && adb install
android/app/build/outputs/apk/staging/debug/app-staging-debug.apk
&& adb shell am start -n com.multitarget/.MainActivity
```

    - Release Build

```
cd android && ENVFILE=.env.staging ./gradlew
assembleStagingRelease && cd .. && adb install
android/app/build/outputs/apk/staging/release/app-staging-release.
apk && adb shell am start -n com.multitarget/.MainActivity
```

- Production
    - Debug Build

```
cd android && ENVFILE=.env.production ./gradlew
assembleProductionDebug && cd .. && adb install
android/app/build/outputs/apk/production/debug/app-production-debu
g.apk && adb shell am start -n com.multitarget/.MainActivity
```

    - Release Build

```
cd android && ENVFILE=.env.production ./gradlew
assembleProductionRelease && cd .. && adb install
android/app/build/outputs/apk/production/release/app-production-re
lease.apk && adb shell am start -n com.multitarget/.MainActivity
```

7.2.2: Second Approach: Using Build Types

Instead of creating multiple product flavours for each environment, you can use **Build Types**. This allows you to use the same app but with different environment configurations.

In this approach, the app will not be built as separate targets, but rather different environments will be managed using `buildConfigFields` for each build type.

Configuration in `build.gradle`

In your `android/app/build.gradle` file, update the `buildTypes` block with `buildConfigFields` for each environment (Development, Staging, and Production).

```
buildTypes {
    debug {
        signingConfig signingConfigs.debug
        matchingFallbacks = ['debug', 'release']
        buildConfigField "String", "ENV_FILE", '"env.development"'
        buildConfigField "String", "API_URL", '"https://dev-api.example.com"'
        buildConfigField "String", "ENV", '"development"'
    }
    release {
        // Caution! In production, you need to generate your own keystore file.
        // see https://reactnative.dev/docs/signed-apk-android.
        signingConfig signingConfigs.debug
        minifyEnabled enableProguardInReleaseBuilds
        proguardFiles getDefaultProguardFile("proguard-android.txt"),
"proguard-rules.pro"
        buildConfigField "String", "ENV_FILE", '"env.production"'
        buildConfigField "String", "API_URL", '"https://prod-api.example.com"'
        buildConfigField "String", "ENV", '"production"'
    }
    staging {
        signingConfig signingConfigs.debug
        matchingFallbacks = ['debug', 'release']
        buildConfigField "String", "ENV_FILE", '"env.staging"'
        buildConfigField "String", "API_URL", '"https://staging-api.example.com"'
        buildConfigField "String", "ENV", '"staging"'
    }
}
```

Running the App with Different Environments in single app

Now, use the following commands to build the app for **Development**, **Staging**, or **Production** environments. The app will not be a separate target but will switch between environments based on the build configuration.

Debug

```
cd android && ENVFILE=.env.development ./gradlew assembleDebug &&
cd .. && adb install
android/app/build/outputs/apk/debug/app-debug.apk && adb shell am
start -n com.multitarget/.MainActivity
```

Production

```
cd android && ENVFILE=.env.production ./gradlew assembleRelease &&
cd .. && adb install
android/app/build/outputs/apk/release/app-release.apk && adb shell
am start -n com.multitarget/.MainActivity
```

Staging

```
cd android && ENVFILE=.env.staging ./gradlew assembleStaging && cd
.. && adb install
android/app/build/outputs/apk/staging/app-staging.apk && adb shell
am start -n com.multitarget/.MainActivity
```

That’s it!

🎉 **Congrats!** You're all set with your React Native project, now rocking multiple environments for both **iOS** and **Android**! 🚀 Thanks to the powerful **react-native-config** package, managing different setups for **Development**, **Staging**, and **Production** has never been easier. 💡 Whether you're testing new features or getting ready for deployment, switching between environments is a breeze! 🌟

✨ Don’t forget to add those little touches—custom **app icons**, stunning **splash screens**, and unique **bundle identifiers** for each environment, tailored for both platforms. 🖼️📱

Happy coding, and may your app soar to new heights! 🚀🎉

GitHub Repository