This file contains all the steps required to create build flavors for react-native.

(Note: This project was created using react native CLI, so if you have created project using EXPO, please refer to the package)

Pre-requisites:

Step-1: In the root folder of the project create file with the mentioned names below:

1. .env
2. .env.dev
3. .env.prod
4. .env.staging

The above names can be your desired name but suggested the above because of their meaning. The '.env' file is mandatory for the package we are about to use in the application as it implements the selected flavor key values inside the '.env' file which we will see in a bit.

Step-2: install the below mentioned package using either NPM or yarn. (the package manager you are using is preferred because of some package related issues when run from either of them.)

1. (yarn add react-native-config) or (npm install react-native-config)

Now for android device.

Step-1 open gradle file present in <main_project_folder>/android/app/build.gradle into the IDE and add below mentioned lines at the top of the file just after some applies as they are important.

```
project.ext.envConfigFiles = [
productiondebug: ".env.prod",
productionrelease: ".env.prod",
developmentrelease: ".env.dev",
developmentdebug: ".env.dev",
stagingrelease: ".env.qa",
stagingdebug: ".env.qa"
]
apply from: project(':react-native-config').projectDir.getPath() +
"/dotenv.gradle"
```

```
import com.android.build.OutputFile
```

Step-2: Then after this modify the android dictionary in the same gradle file.

```
android {
ndkVersion rootProject.ext.ndkVersion
buildToolsVersion rootProject.ext.buildToolsVersion
compileSdk rootProject.ext.compileSdkVersion
flavorDimensions "default"
productFlavors {
production {
applicationId "multipleflavours.production"
applicationIdSuffix "prod"
resValue "string", "build_config_package", "com.multipleflavours"
}
staging {
applicationId "multipleflavours.staging"
applicationIdSuffix "staging"

resValue "string", "build_config_package", "com.multipleflavours"
}
development {
applicationId "multipleflavours.development"
applicationIdSuffix "dev"

resValue "string", "build_config_package", "com.multipleflavours"

}
}
namespace "com.multipleflavours"
defaultConfig {
applicationId "com.multipleflavours"
minSdkVersion rootProject.ext.minSdkVersion
targetSdkVersion rootProject.ext.targetSdkVersion
versionCode 1
versionName "1.0"
}
signingConfigs {
debug {
storeFile file('debug.keystore')
storePassword 'android'
keyAlias 'androiddebugkey'
```

```
keyPassword 'android'
}
}
buildTypes {
debug {
signingConfig signingConfigs.debug
matchingFallbacks = ['debug', 'release']
}
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"
}}}
```

(Note: the above-mentioned text in ‘color’ are the modifications required for the build flavor to work so please go through the documentation carefully.)

Step-3: now open the <main_project_folder>/android/gradle.properties and change the flag of ‘newArchEnabled’ to false so that our build command can be run.

Step-4: now open the main react native project folder in IDE of your choice e.g. VS Code or any other and open the package.json file and in scripts section add the below commands to create a dev build on virtual device or real or create bundle or APK for android.

```
"android:staging": "react-native run-android --mode=stagingdebug
--appId multipleflavours.staging --appIdSuffix staging",
"android:staging-release": "react-native run-android --
mode=stagingrelease --appId multipleflavours.staging --
appIdSuffix staging",
"android:dev": "react-native run-android --mode=developmentdebug
--appId multipleflavours.development --appIdSuffix dev",
"android:dev-release": "react-native run-android --
mode=developmentrelease --appId multipleflavours.development --
appIdSuffix dev ",
"android:prod": "react-native run-android --mode=productiondebug
--appId multipleflavours.production --appIdSuffix prod",
```

```
"android:prod-release": "react-native run-android --
mode=productionrelease --appId multipleflavours.production --
appIdSuffix prod",
"ba": "react-native bundle --platform android --dev false --
entry-file index.js --bundle-output
android/app/src/main/assets/index.android.bundle --assets-dest
android/app/src/main/res",
"ra": "rm -rf android/app/src/main/res/drawable-hdpi && rm -rf
android/app/src/main/res/drawable-mdpi && rm -rf
android/app/src/main/res/drawable-xhdpi && rm -rf
android/app/src/main/res/drawable-xxhdpi && rm -rf
android/app/src/main/res/drawable-xxxhdpi && rm -rf
android/app/src/main/res/raw && npx jetify",
"androidProductionReleaseBundle": "yarn autoclean && cd android
&& ./gradlew bundleProductionRelease && killall -9 java &&
open ./android/app/build/outputs/bundle/production/release",
"androidDevelopmentDebugAPK": "yarn autoclean && cd android
&& ./gradlew assembleDevelopmentDebug && killall -9 java &&
open ./android/app/build/outputs/apk/development/debug",
"androidDevelopmentReleaseAPK": "yarn clean && cd android
&& ./gradlew assembleDevelopmentRelease && killall -9 java &&
open ./android/app/build/outputs/apk/development/release",
"androidProductionDebugAPK": "yarn clean && cd android
&& ./gradlew assembleProductionDebug && killall -9 java &&
open ./android/app/build/outputs/apk/production/debug",
"androidProductionReleaseAPK": "yarn clean && cd android
&& ./gradlew assembleProductionRelease && killall -9 java &&
open ./android/app/build/outputs/apk/production/release",
"androidStagingDebugAPK": "yarn clean && cd android && ./gradlew
assembleStagingDebug && killall -9 java &&
open ./android/app/build/outputs/apk/staging/debug",
"androidStagingReleaseAPK": "yarn clean && cd android
&& ./gradlew assembleStagingRelease && killall -9 java &&
open ./android/app/build/outputs/apk/staging/release"
```

Paste them as it is.
(Note: if you are using yarn then use the above commands with yarn but if using NPM then replace all the yarn commands with their npm alternatives.)

Step-5: to run the application on android with the specified env files we must use the above script names to be invoked in the terminal or shell. So, if using NPM use (npm run android:dev) to run dev server

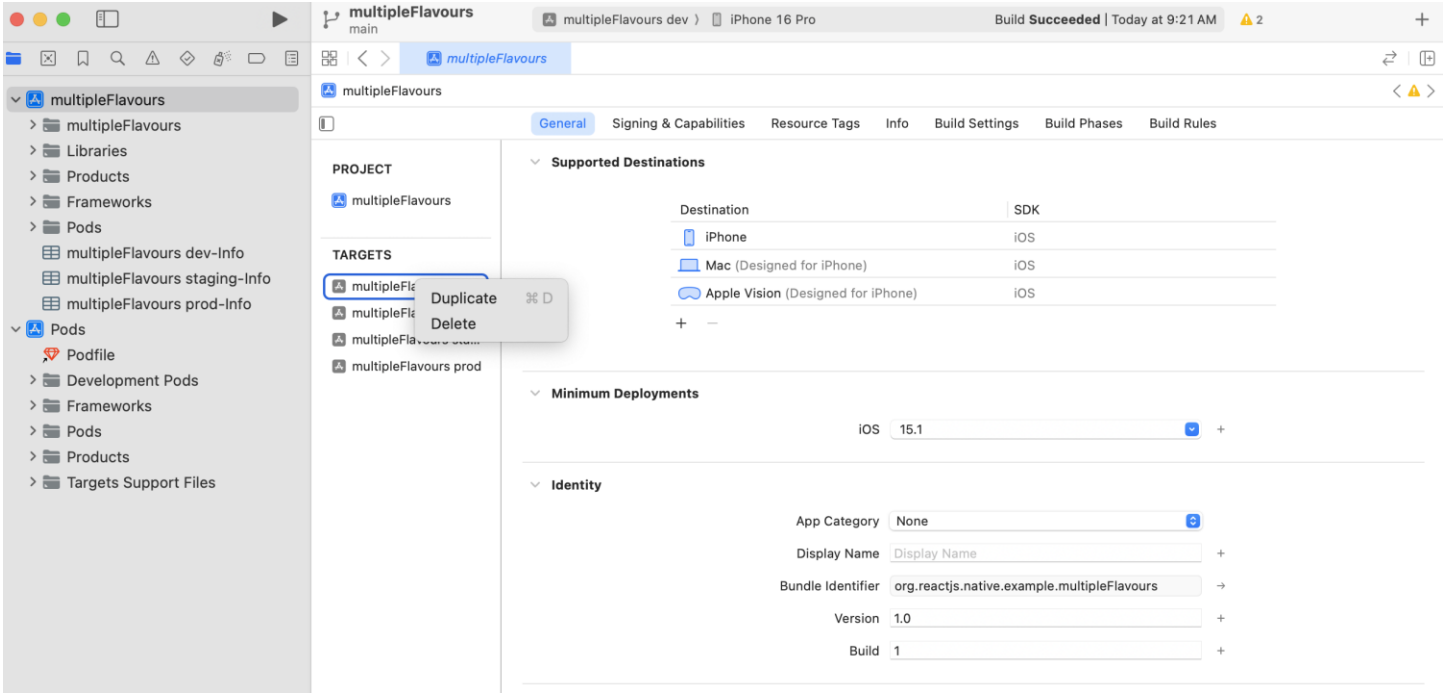
locally on a virtual or real device and so on, or if using yarn use (yarn android:dev) to do the same. We can only specify the keys used in the scripts section in package.json file to run our application. And if any modification is required then modify it only in package.json scripts section only or implement a new script key with value which will be executed in the terminal.

Step-6: Now move to the main android folder and create copies of main folder in the main directory and rename them according to the project flavors mentioned in the build.gradle to make it work and remove all the java folder inside them.

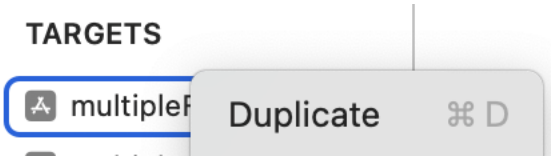
Step-7: Now locate the file present in the path /<Build_flavor>/res/values/strings.xml and inside that file of all the flavors rename the app_name string to desired application name to reflect changes when our application is build or deployed.

Setup For IOS :

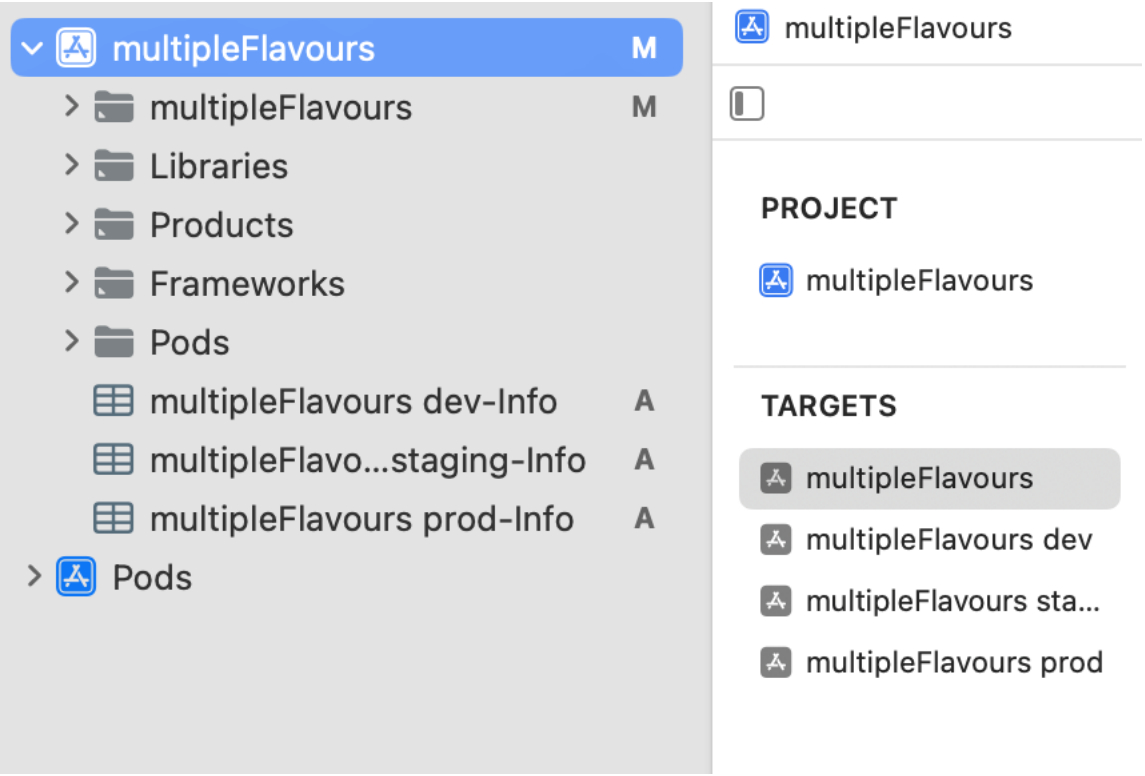
Step-1: navigate to iOS folder and run pod install.
Step-2: Now open .xcworkspace file in Xcode and click on the application tile shown in the below image.



Step-3: Now you will see a targets section in the main window and now select the tile which shows the project name when created and right click on it and duplicate the target like did in the below image.



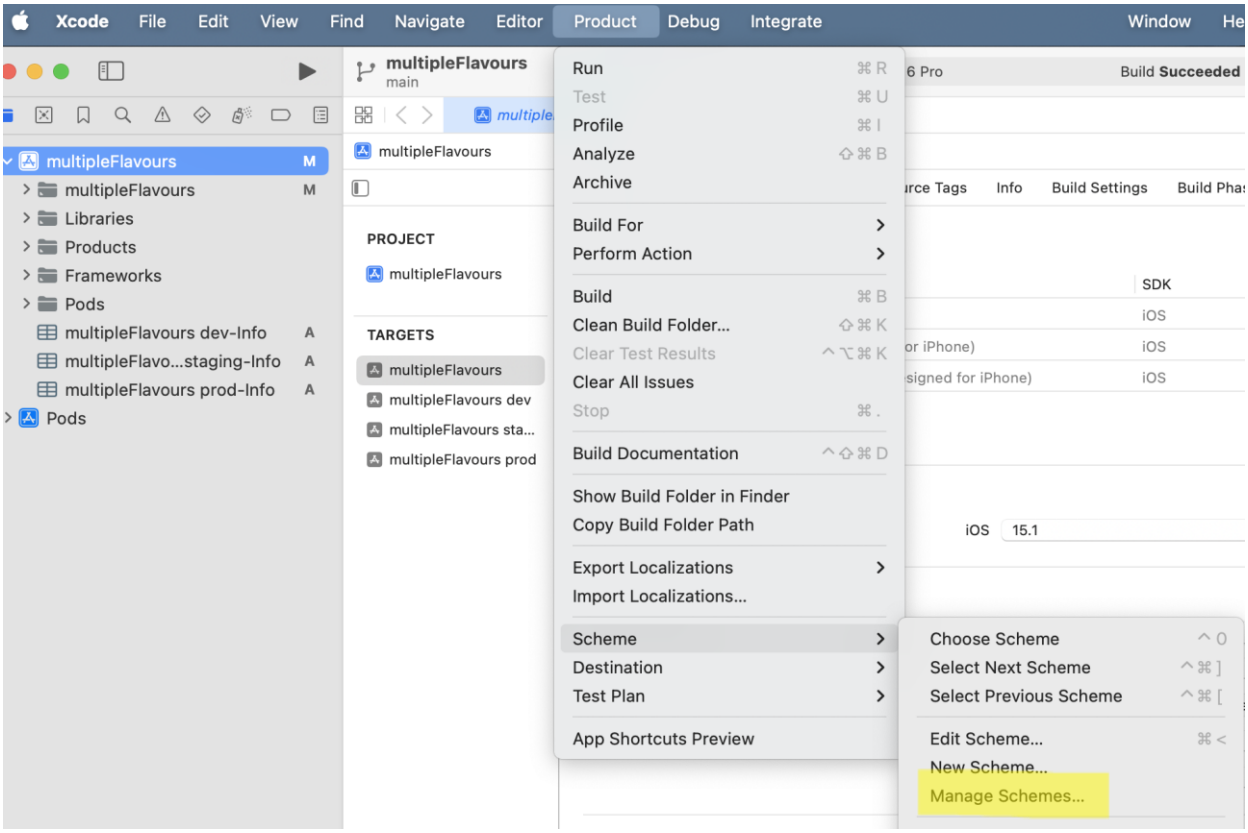
Step-4: After creating <application_name> dev, staging, and production target from the main <application_name> tile. We will see the .plist file created for them automatically by their names when we create their duplicates.



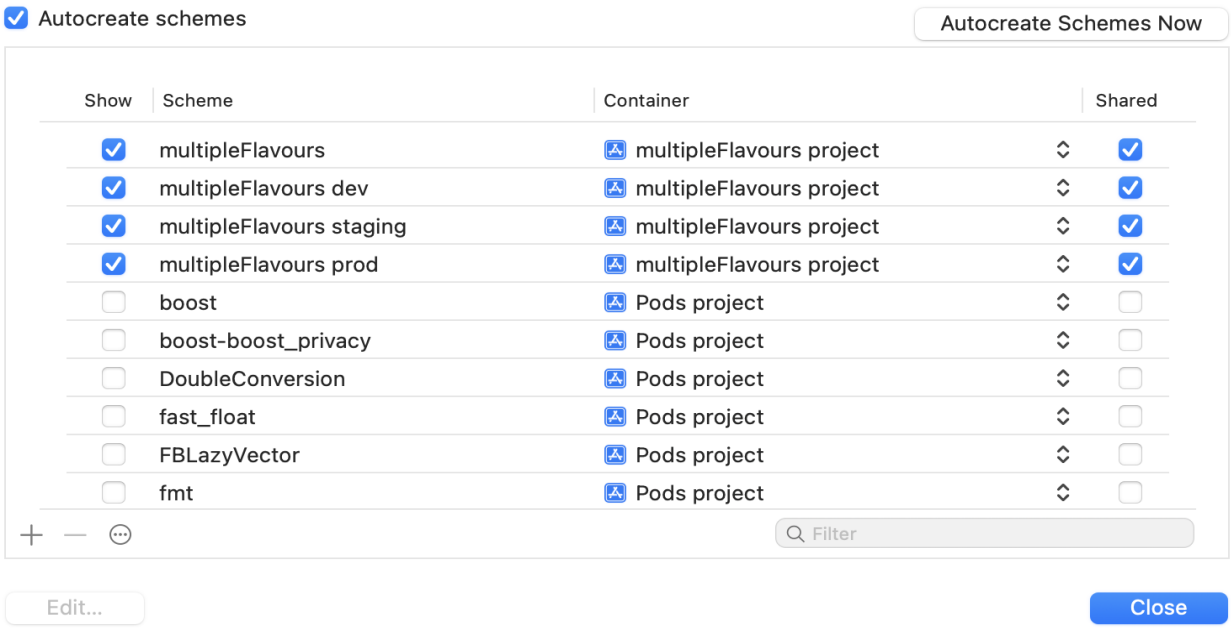
(**Note:** don't forget to rename the duplicates and the .plist files according to their purpose like I did in the above image as this will help to distinguish between the multiple .plist files in the projects.)

Step-5: Now tap on each newly created target and select "Build Settings" and search for "info.plist File" and changes the existing with the renamed .plist file from the above step so that it can get the changes related to build schemes and carefully rename them because if they does not match from the file system will result in build.

Step-6: Now we must create a scheme for each target, and we can do this by managing the scheme in the Xcode. Refer the below image.

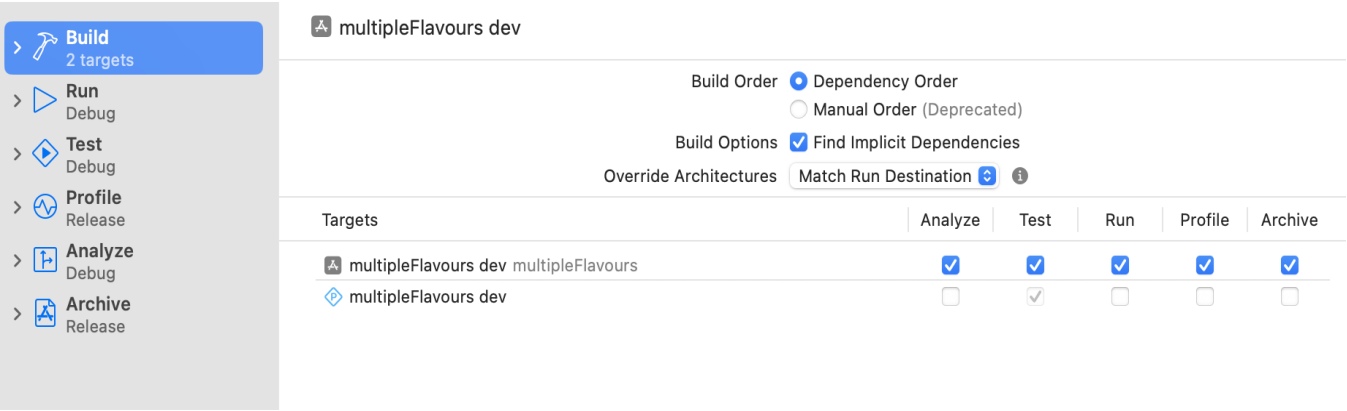


Step-7: Now when we tap on manage schemes this will open a new window showing all the existing schemes. Make sure that the shared checkbox is selected and autocreate schemes are selected as well. There we will see the schemes that were created when we duplicated the target files. So what we will do is only tick mark the duplicates schemes and we will see '-' button at bottom left corner of the window and make sure only the duplicates are selected check boxes and on tap of '-' button delete those schemes and then tap on the '+' button on the right of '-' button on the same window and this will show us the prefilled schemes and their names according to the targets we have created in the step-4 as seen in the below image.

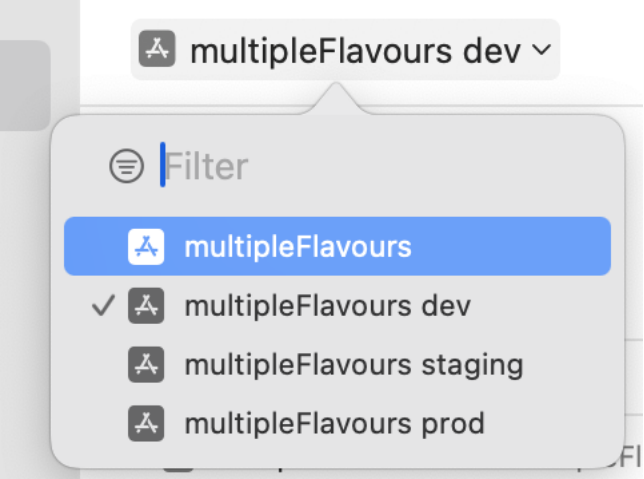


Step-8: Now we must edit each scheme manually as we must add a custom script which will run before the build is created so that our env file fields are inserted in the application itself.

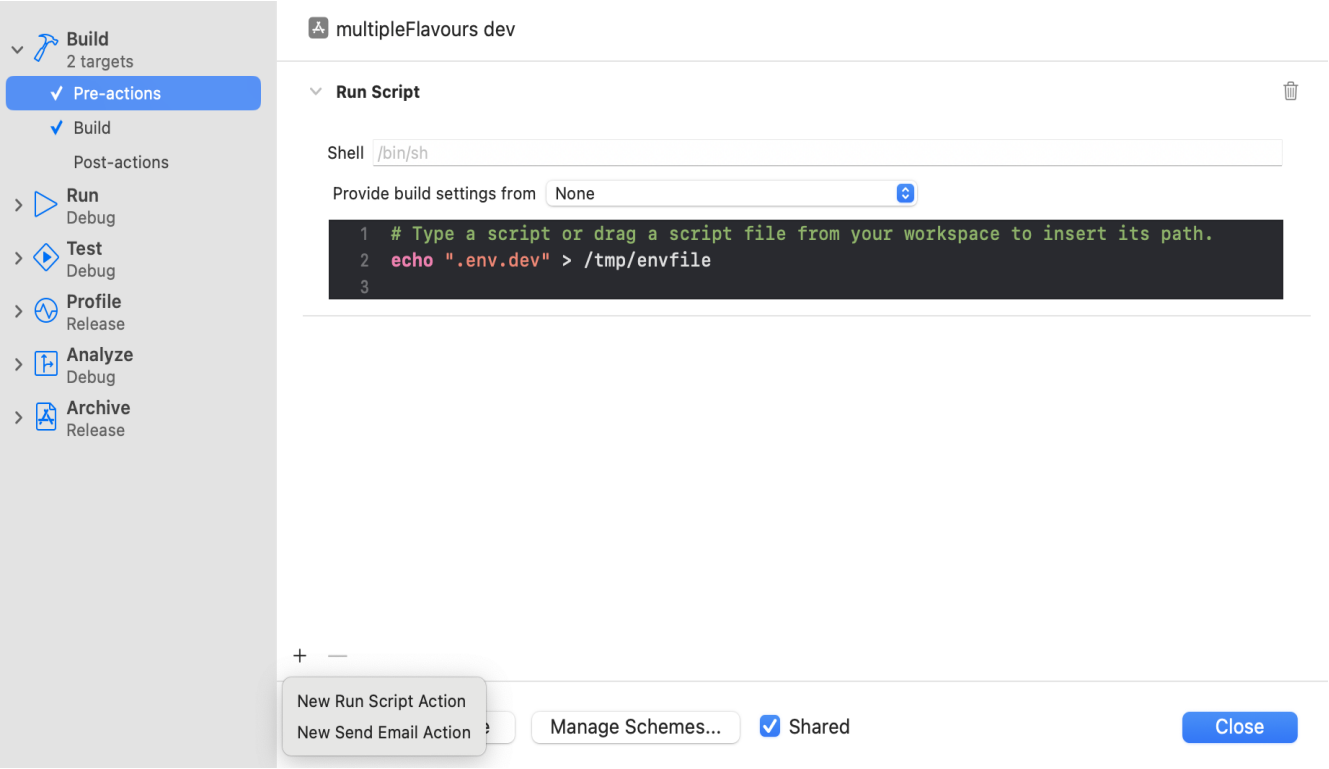
Step-9: Now repeat step-5 but select the ‘Edit scheme’ button which will open a new window showcasing our current selected scheme like in the below image.



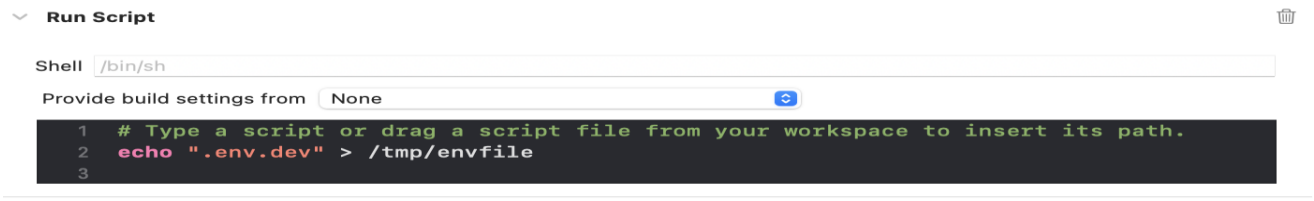
Now as we can see in the below image, we can change the selected scheme from the dropdown when we tap on the current selected scheme and change to the scheme we can to edit.



Step-10: Now select the build tile in the left pane and expand using the dropdown button and select pre-actions menu which will open a new panel in center. Now we can see there are ‘+’ and ‘-’ button in the bottom left corner of the UI. Tap on the ‘+’ and it will show us two options as we select new run script action option.



Step-11: Now in the Run Script section add the command in the field. This will run at the time of building. Eg: `echo ".env.dev" > /tmp/envfile`



Step-12: we must repeat step-6 to 10 for each scheme to make it work. And keep in mind to change the '.env.<flavor>' according to the scheme like I did in the above image. These files are located on the main directory, and each scheme has its own .env file and specify each one during this process.

Now after saving all the details. We have to modify the Podfile as well to reflect these changes.

Step-13: open Podfile in VSCode editor and do the below changes.

```
target 'multipleFlavours' do

abstract_target 'multipleFlavoursCommonPods' do
```

And add below lines just after the above changes.

```
target 'multipleFlavours dev' do
end

target 'multipleFlavours staging' do
end

target 'multipleFlavours prod' do
end
```

Make sure to keep the target names as the schemes to make it work or else it will break.

Step-14: Now for testing create a build using shortcut command of "command + R" in Xcode and make sure everything works fine. If does not work, then run the pod install in iOS folder and then try again.

Step-15: Now we must specify the command for react-native which when executed will auto select the schemes for use and auto build the application and run in simulator or real device. (For real devices make sure the developer account and developer team is selected to make it install on real devices.)

Step-15: Now add the commands below in the scripts section of the package.json file.

```
"ios": "react-native run-ios -i",

"ios:dev": "cd ios && pod install && cd .. && react-native run-ios --scheme \"multipleFlavours dev\"",

"ios:staging": "cd ios && pod install && cd .. && react-native run-ios --scheme \"multipleFlavours staging\"",

"ios:prod": "cd ios && pod install && cd .. && react-native run-ios --scheme \"multipleFlavours prod\""
```

(Note: To create a release build use Xcode to create archive and release on test flight from there.)