Ca-Phun Ung Follow

Jan 14, 2019 · 5 min read · Listen

React Native: How to Load SPA or Static HTML Site inside WebView

React Native WebView allows you to load a publicly accessible resource with the `uri` property. It works just as a usual website inside an in-app browser. However, it's not so cut-and-dry when you want to serve it locally as a SPA or static HTML site.

Defining the Problem

from CRA.

The criteria of a working solution is:

- a site served as `./path/web.html` should be no different to one served as `http://mysite.com/path/web.html` .
- url parameters should work if passed to the local site, as it should to any typical site.
- all static resources such as HTML, JS, CSS, images, and fonts must be stored in the app–without external dependencies–to support offline mode.
- must work the same for both Android and iOS.
- assets must be served from a single location for all platforms to avoid unnecessary duplication.

As I explored the various solutions to embed a fully working Local Site within a WebView, I realised there is no solution out there that satisfied the above criteria, so I set out to devise one.

Devising the Solution

Many hours of googling and experimentation lead me to a few critical finds, which form the basis of the entire solution:

1. Any folder named with a `.bundle` extension is seen has a package in Xcode. The contents of the package could be changed any way you want, without affecting the complicated file references Xcode uses to track assets.
2. In Xcode you can add folder groups that are linked by reference. i.e. Assets do not have to reside within the `ios/MyReactNativeApp` folder.
3. Android Studio allows you to add multiple Asset folders, and the folder could even be relative to the `android/app` folder.

If you prefer to read code and figure it out yourself, I made a working example on github:

caphun/react-native-webview-example

An example RN App with an embedded static HTML site - caphun/react-native-webview-example

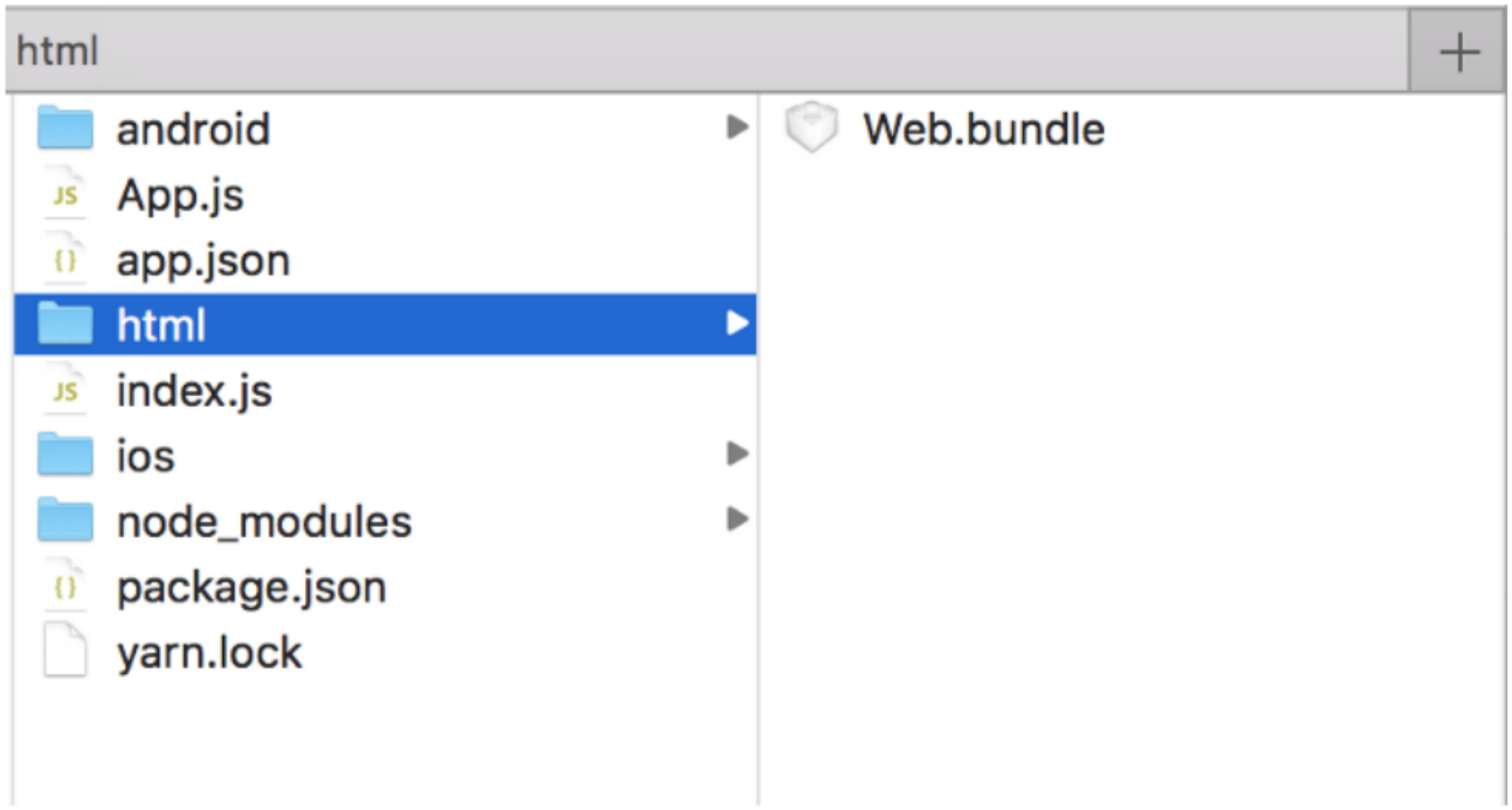
github.com

If you don't mind all the waffling, then please read on, as I finally get down to the nitty gritty of the solution, and take you through a step by step...

Step 1 — Setup Web Assets Folder

Create a `html` folder in your RN project root and add a folder named `Web.bundle` .

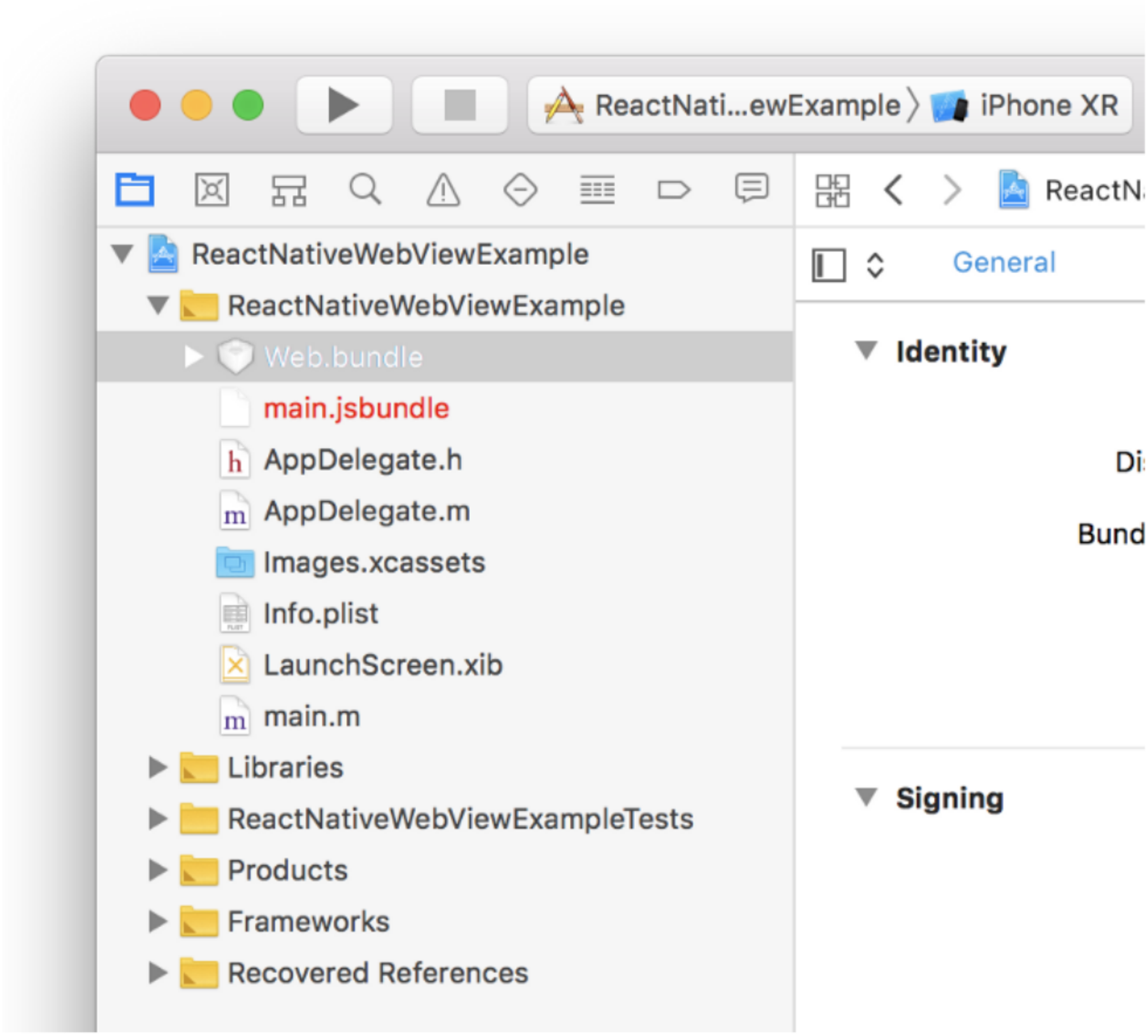
It should look like this:



`path/to/project/html/Web.bundle`

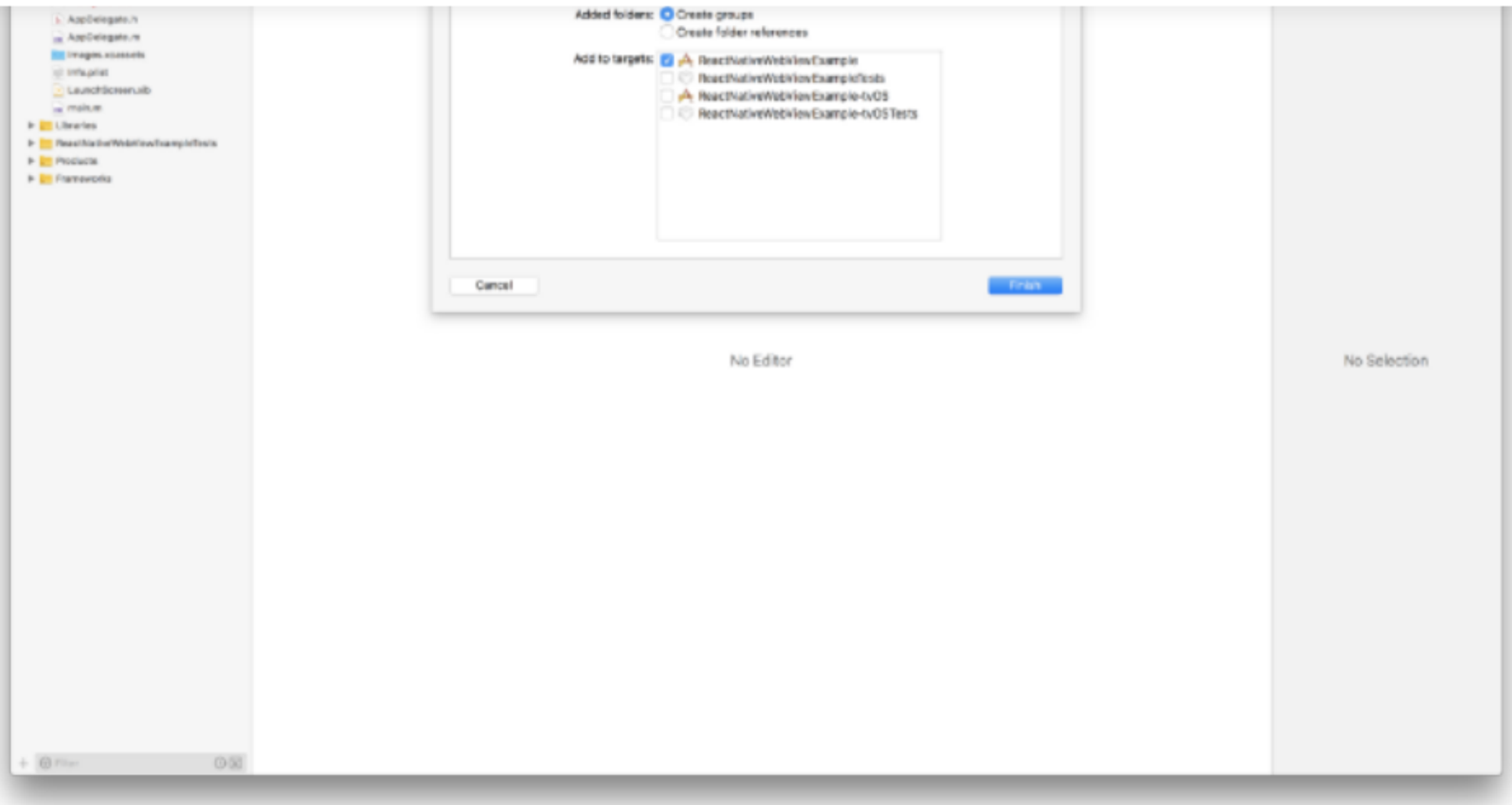
Next, you need to setup the Android and iOS projects so that they recognise the

Navigator and expand the first folder named after your project name. Have the Finder side-by-side, then drag-in the `Web.bundle` folder. It should look like this at the end:



Web.bundle Group in Xcode

When you drop the folder, a dialog will appear. Make sure “Copy items if needed” is **unchecked.**



To setup Android — open `path/to/project/android/app/build.gradle` file in your editor and add this line inside the android clause like so:

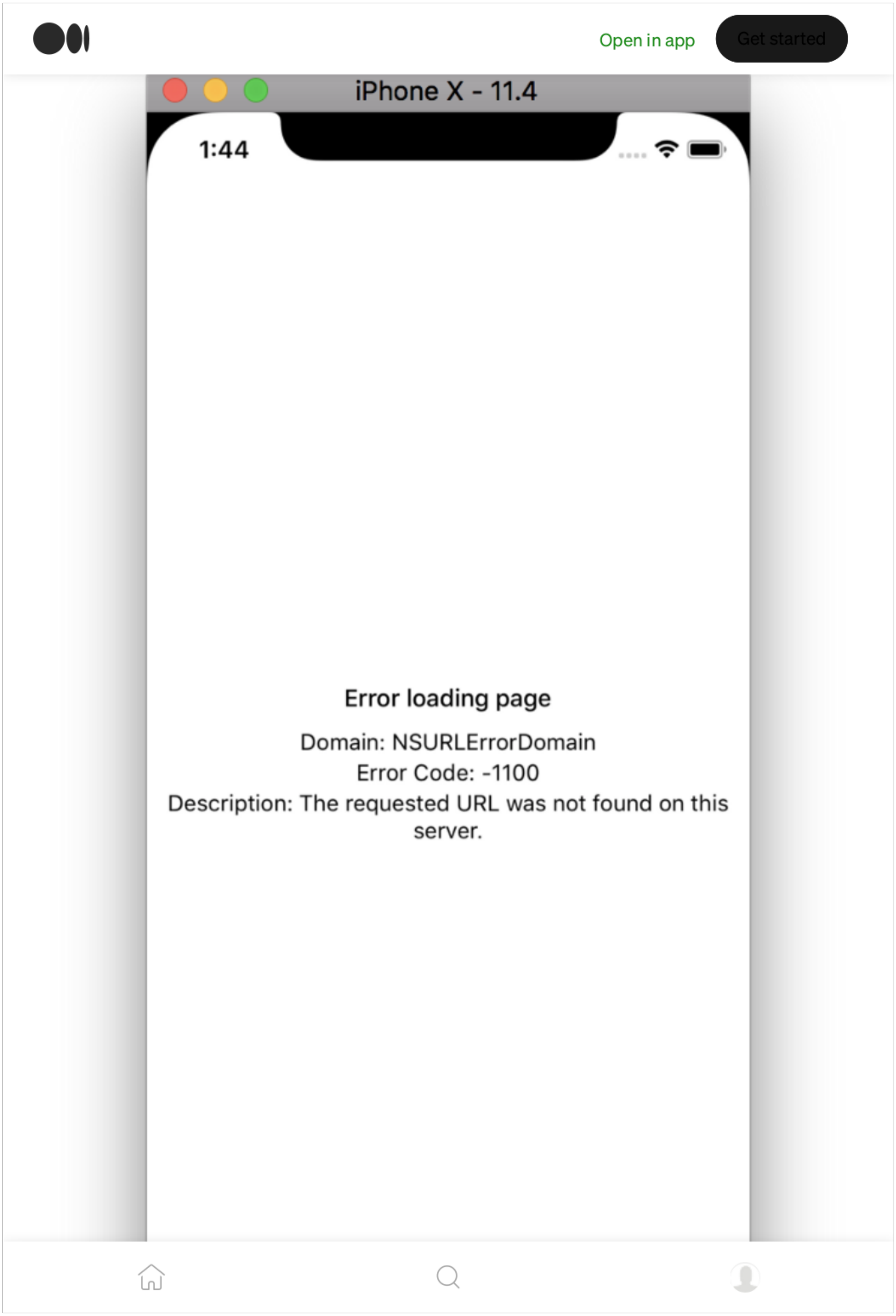
```
android {
  ...
  sourceSets {
    main { assets.srcDirs = ['src/main/assets', '../../html'] }
  }
}
```

Alternatively, you could create the assets folder in Android Studio – When "Change Folder Location" checkbox appears, click it and input `../../html` .

Step 2 — Add Launcher and Site Files

When pointing the Source URI of a WebView to local file resource, there is a major gotcha moment! If you point to `{uri: '...index.html'}` the resource loads without issue but once you pass a parameter,

e.g. `{uri: '...index.html?platform=ios'}` , RN WebView will show error like this:


Open in app
Get started
iPhone X - 11.4
1:44
Error loading page
Domain: NSURLErrorDomain
Error Code: -1100
Description: The requested URL was not found on this server.

To overcome this issue, add a `loader.html` file to `Web.bundle/loader.html` with a single link tag:

```
<a id="progress-bar" href="#"></a>
```

This will be used in the next Step, more about that later!

Next, add the actual static local site files to `Web.bundle/site/` or any folders and files you wish. You should end up with a structure similar to this:

In Step 1 we setup the `Web.bundle` so that in iOS you could reference the launcher as `Web.bundle/loader.html` and Android as `file:///android_asset/Web.bundle/loader.html`, see below:

```
const sourceUri = (
  Platform.OS === 'android'
    ? 'file:///android_asset/'
    : ''
) + 'Web.bundle/loader.html';
```

Now, here's the BIG reveal! Remember in Step 2 you added the launcher file? Well, that's so we could inject some javascript on initial launch. Yes, it's a hack! And sorry if this feels like trickery, but a necessary evil in order to pass in launch parameters (which are so very important :), see below:

```
const params = 'platform='+Platform.OS;

const injectedJS = `
  if (!window.location.search) {
    var link = document.getElementById('progress-bar');
    link.href = './site/index.html?${params}';
    link.click();
  }
`;
```

You may notice in the above snippet I have a `!window.location.search` condition. That's to avoid the issue below which occurs only in iOS (as of writing react-native-webview is version 3.1.2):

And finally, for best results the WebView should be used with properties specified like so:

```
return <WebView
  injectedJavaScript={injectedJS}
  source={{ uri: sourceUri }}
  javaScriptEnabled={true}
  originWhitelist={['*']}
  allowFileAccess={true}
/>;
```

Still here? There's nothing more to say other than, as mentioned earlier I have a working example on github, clone it, play with it, break it and give feedback!

caphun/react-native-webview-example

An example RN App with an embedded static HTML site - caphun/react-native-webview-ex

465 12

github.com

Further Notes

- As of RN `0.56.0` the `injectedJS` hack is not required for Android. Source URI's to local resources could be directly linked with URL parameters.
- You can avoid the `injectedJS` hack by using `postMessage` interface instead. However, you need to setup WebView bridging to facilitate communication between Native and HTML.