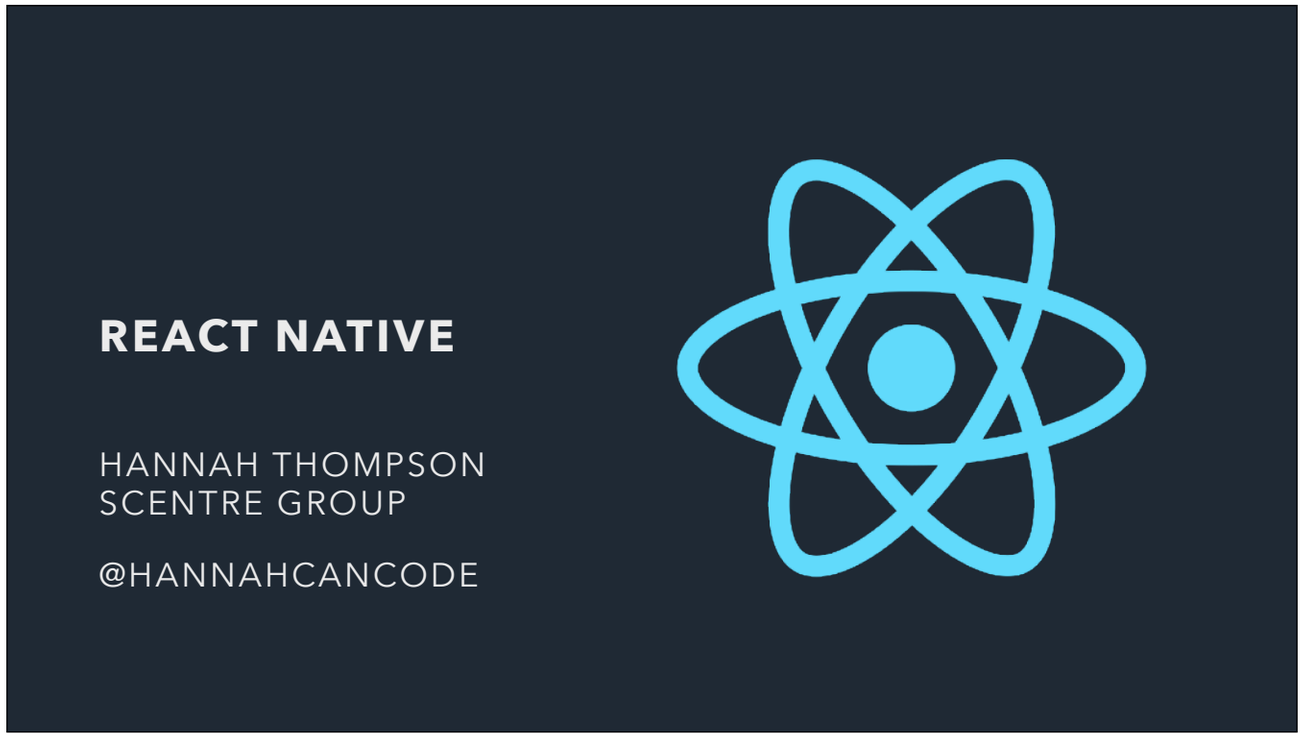


Used javascript?
Used React?

I’m going to be talking about React Native, which is a framework that allows you to make mobile apps, using what is essentially React. Afterwards you should be able to dive into the docs with more confidence and be able to make your own apps.

I’m going to be focussing mainly on iOS, because that’s where my experience has been, but a lot of what I say will also be applicable to Android.

Web dev not iOS dev
nb. license agnostic

Firstly, React Native is… more native than most javascript mobile frameworks. React Native uses Javascript Core, which is iOS uses for javascript in your mobile browsers. Javascript Core is accessible outside of the browser, however, and this is how React Native uses it.

Other frameworks like Ionic and Cordova use a native container with a WebView to render Javascript - so essentially you’re just writing a webpage in a native container.

So because we’re not in the browser, React Native access Native UI components. Once again, something like Ionic uses normally HTML and CSS for UI.

So why would you want to use React Native and javascript to develop a mobile app?

Well…

WHY?

ANDROID AND IOS

WHY?

EXISTING REACT APP

WHY?

HOT RELOADING WHOA

WHY?

YOU DON'T PLAN ON SUING FACEBOOK...
YET

REACT VS. REACT NATIVE

`div` → `View`

`p` → `Text`

GETTING SET UP

```
$ brew install node
```

Getting set up is easy on a mac, you just need homebrew installed.

First we'll install node, for running react-native, and watchman, which will watch and reload our code when things change. This allows us to do hot reloading so we don't have to rebuild our app every time we edit our code.

Next, using npm or yarn, we do a global install of the react native command line interface and use that to initialise a new react native app. Finally, we'll move into our app and get it running in the iOS simulator using the run-ios command.

GETTING SET UP

```
$ brew install node
$ brew install watchman
```

Getting set up is easy on a mac, you just need homebrew installed.

First we'll install node, for running react-native, and watchman, which will watch and reload our code when things change. This allows us to do hot reloading so we don't have to rebuild our app every time we edit our code.

Next, using npm or yarn, we do a global install of the react native command line interface and use that to initialise a new react native app. Finally, we'll move into our app and get it running in the iOS simulator using the run-ios command.

GETTING SET UP

```
$ brew install node
$ brew install watchman

$ npm install -g react-native-cli
```

Getting set up is easy on a mac, you just need homebrew installed.

First we'll install node, for running react-native, and watchman, which will watch and reload our code when things change. This allows us to do hot reloading so we don't have to rebuild our app every time we edit our code.

Next, using npm or yarn, we do a global install of the react native command line interface and use that to initialise a new react native app. Finally, we'll move into our app and get it running in the iOS simulator using the run-ios command.

GETTING SET UP

```
$ brew install node
$ brew install watchman

$ npm install -g react-native-cli

$ react-native init MyNewApp
```

Getting set up is easy on a mac, you just need homebrew installed.

First we'll install node, for running react-native, and watchman, which will watch and reload our code when things change. This allows us to do hot reloading so we don't have to rebuild our app every time we edit our code.

Next, using npm or yarn, we do a global install of the react native command line interface and use that to initialise a new react native app. Finally, we'll move into our app and get it running in the iOS simulator using the run-ios command.

GETTING SET UP

```
$ brew install node
$ brew install watchman

$ npm install -g react-native-cli

$ react-native init MyNewApp

$ cd MyNewApp
```

Getting set up is easy on a mac, you just need homebrew installed.

First we'll install node, for running react-native, and watchman, which will watch and reload our code when things change. This allows us to do hot reloading so we don't have to rebuild our app every time we edit our code.

Next, using npm or yarn, we do a global install of the react native command line interface and use that to initialise a new react native app. Finally, we'll move into our app and get it running in the iOS simulator using the run-ios command.

GETTING SET UP

```
$ brew install node
$ brew install watchman

$ npm install -g react-native-cli

$ react-native init MyNewApp

$ cd MyNewApp

$ react-native run-ios
```

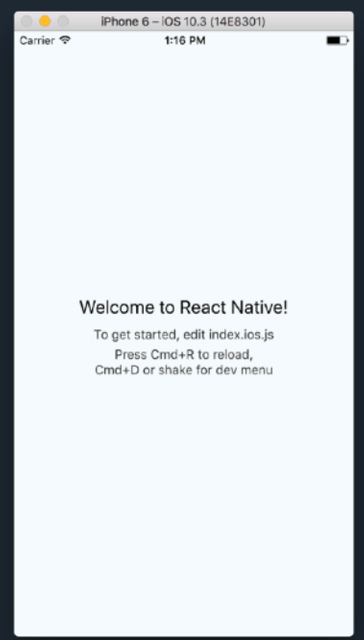
Getting set up is easy on a mac, you just need homebrew installed.

First we'll install node, for running react-native, and watchman, which will watch and reload our code when things change. This allows us to do hot reloading so we don't have to rebuild our app every time we edit our code.

Next, using npm or yarn, we do a global install of the react native command line interface and use that to initialise a new react native app. Finally, we'll move into our app and get it running in the iOS simulator using the run-ios command.

GETTING SET UP

```
$ brew install node
$ brew install watchman

$ npm install -g react-native-cli

$ react-native init MyNewApp

$ cd MyNewApp

$ react-native run-ios
```



Getting set up is easy on a mac, you just need homebrew installed.

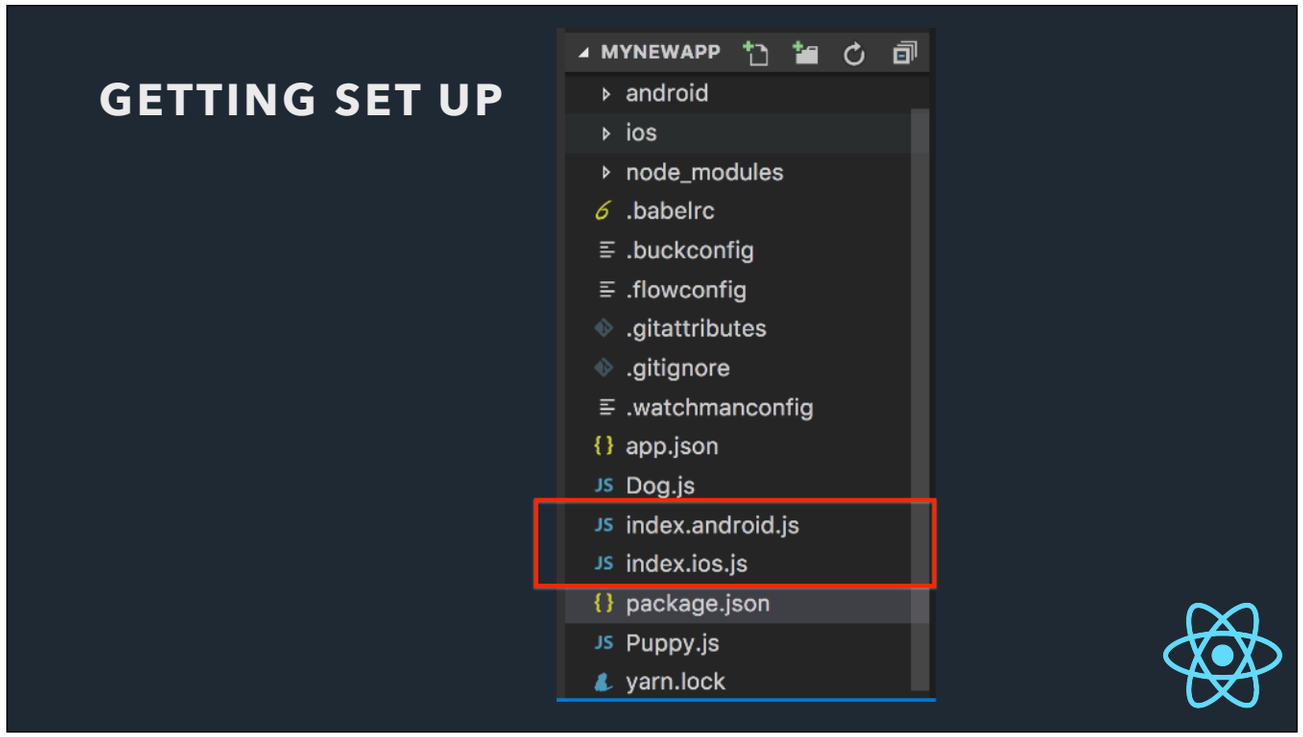
First we'll install node, for running react-native, and watchman, which will watch and reload our code when things change. This allows us to do hot reloading so we don't have to rebuild our app every time we edit our code.

Next, using npm or yarn, we do a global install of the react native command line interface and use that to initialise a new react native app. Finally, we'll move into our app and get it running in the iOS simulator using the run-ios command.

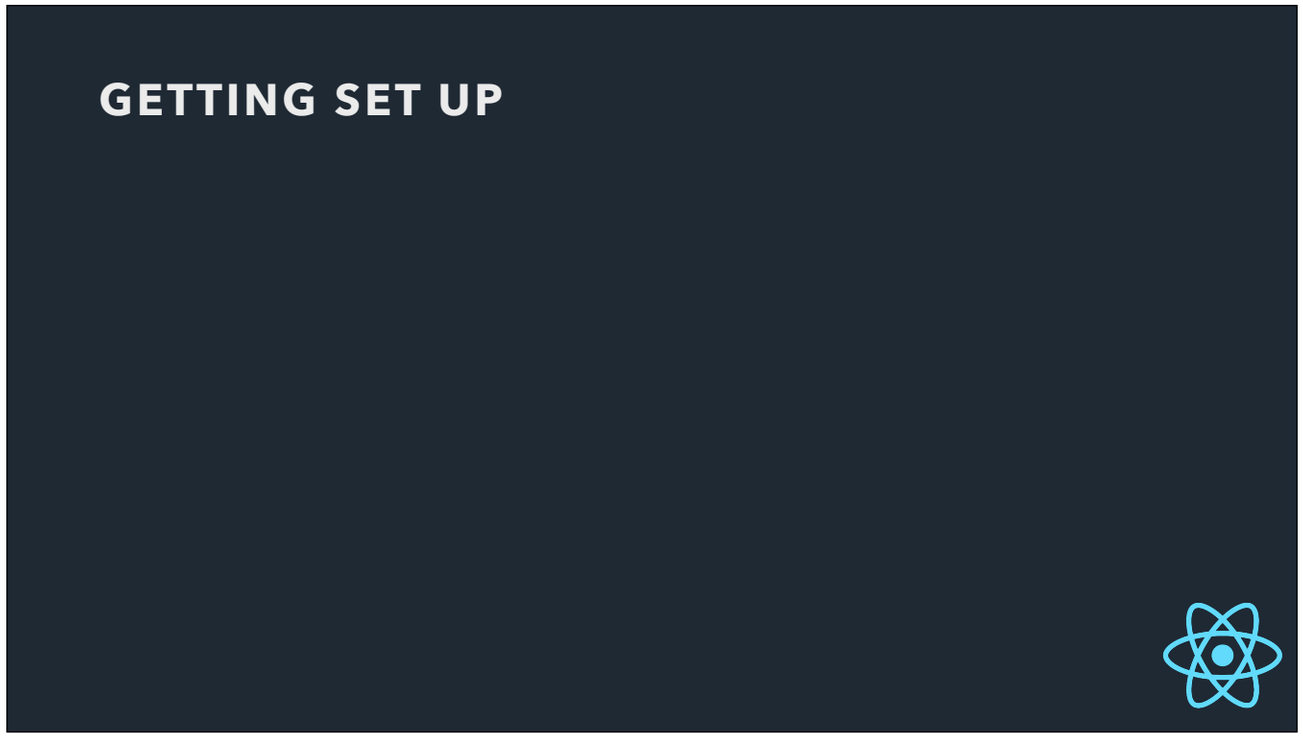
GETTING SET UP

CREATE-REACT-NATIVE-APP + EXPO

Now, if you want an easier way to check out React Native or you’re not quite ready to dive into XCode…



Back to our app, and you'll notice that there are two index files in the directory. This is where the entry points to our code are, one for android and one for iOS. Beyond that you can share components between these.



Now if you're familiar with node development, you'll recognise package.json. If you're not, package.json is just a json file that lists all the things npm needs to know to download your dependencies and run your packages.

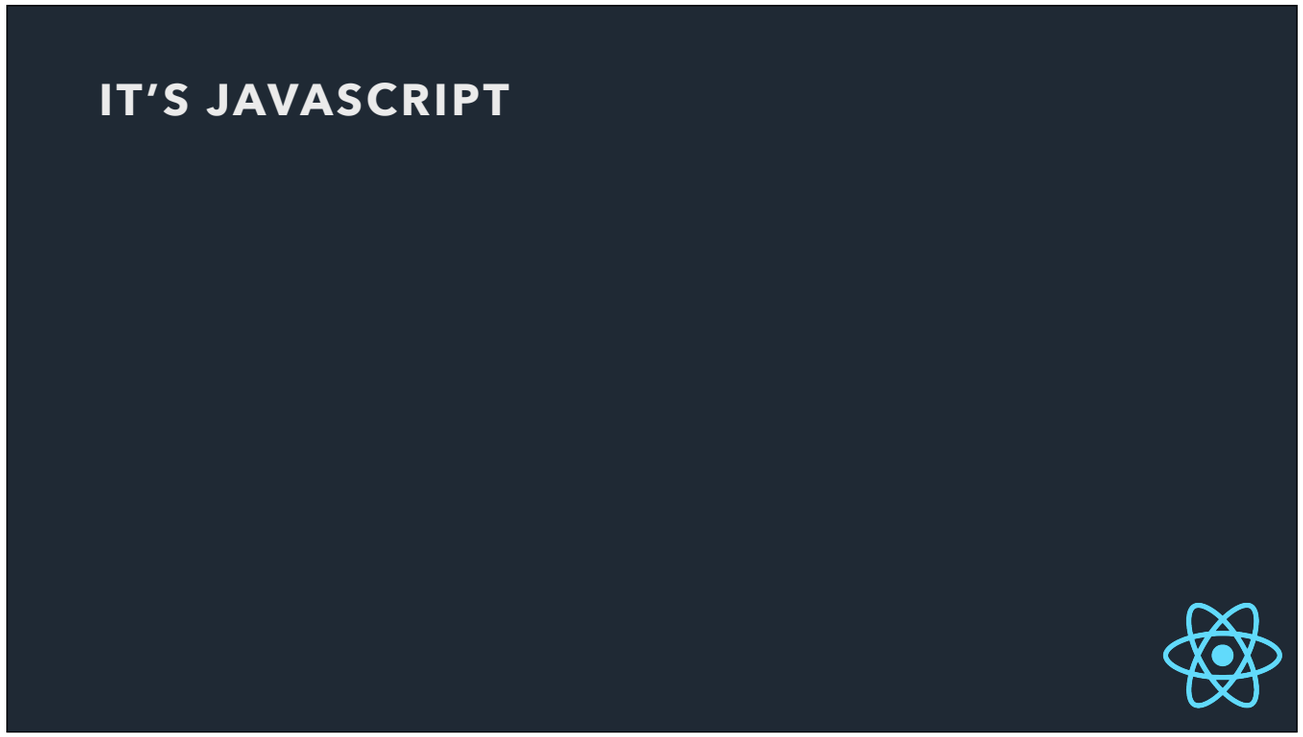
Taking a quick look inside package.json you can see it's bringing in some react, some react-native, and some babel stuff to translate our fancy javascript back to vanilla javascript. The great thing about React Native is that you have a community creating packages for native libraries, UI and logic. When you install them with npm they'll turn up here.

GETTING SET UP

```
// package.json

{
  "name": "MyNewApp",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "start": "node node_modules/react-native/local-cli/cli.js start",
    "test": "jest"
  },
  "dependencies": {
    "babel-preset-react-native": "2.1.0",
    "react": "16.0.0-alpha.12",
    "react-native": "0.47.1"
  },
  "devDependencies": {
    "babel-jest": "20.0.3",
    "jest": "20.0.4",
    "react-test-renderer": "16.0.0-alpha.12"
  },
  "jest": {
    "preset": "react-native"
  }
}
```

Now if you're familiar with node development, you'll recognise package.json. If you're not, package.json is just a json file that lists all the things npm needs to know to download your dependencies and run your packages.

Taking a quick look inside package.json you can see it's bringing in some react, some react-native, and some babel stuff to translate our fancy javascript back to vanilla javascript. The great thing about React Native is that you have a community creating packages for native libraries, UI and logic. When you install them with npm they'll turn up here.



Amongst other things that react-native init has created for us, there's an index.ios.js file, which is a basic javascript class. This is our entry point into the app. If we look into the file, first We import some stuff from react and react native…

IT'S JAVASCRIPT

```
// index.ios.js
```

Amongst other things that react-native init has created for us, there's an index.ios.js file, which is a basic javascript class. This is our entry point into the app. If we look into the file, first We import some stuff from react and react native…

IT'S JAVASCRIPT

```
// index.ios.js

import React, { Component } from 'react'
```

Amongst other things that react-native init has created for us, there's an index.ios.js file, which is a basic javascript class. This is our entry point into the app. If we look into the file, first We import some stuff from react and react native…

IT'S JAVASCRIPT

```
// index.ios.js

import React, { Component } from 'react'

import {
  AppRegistry,
  StyleSheet,
  Text,
  View
} from 'react-native'
```

Amongst other things that react-native init has created for us, there's an index.ios.js file, which is a basic javascript class. This is our entry point into the app. If we look into the file, first We import some stuff from react and react native…

IT'S JAVASCRIPT

```
// index.ios.js

import React, { Component } from 'react'

import {
  AppRegistry,
  StyleSheet,
  Text,
  View
} from 'react-native'
```

Amongst other things that react-native init has created for us, there's an index.ios.js file, which is a basic javascript class. This is our entry point into the app. If we look into the file, first We import some stuff from react and react native…

We set up a class that is exported as MyNewApp,

Create a render method

And then in the render method we put all our UI.

If you’re familiar with React, you’ll notice that it’s really just React. So there’s not a huge jump!

IT'S JUST REACT

```
export default class MyNewApp extends Component {




}
```

We set up a class that is exported as MyNewApp,

Create a render method

And then in the render method we put all our UI.

If you're familiar with React, you'll notice that it's really just React. So there's not a huge jump!

IT'S JUST REACT

```
export default class MyNewApp extends Component {
  render() {




  }
}
```

We set up a class that is exported as MyNewApp,

Create a render method

And then in the render method we put all our UI.

If you're familiar with React, you'll notice that it's really just React. So there's not a huge jump!

IT'S JUST REACT

```
export default class MyNewApp extends Component {
  render() {
    return (

      // stuff to display goes here

    )
  }
}
```

We set up a class that is exported as MyNewApp,

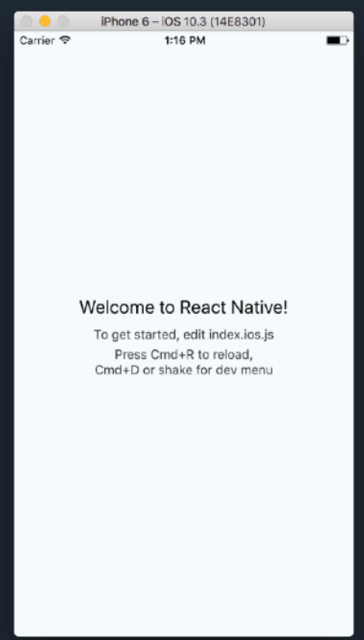
Create a render method

And then in the render method we put all our UI.

If you're familiar with React, you'll notice that it's really just React. So there's not a huge jump!

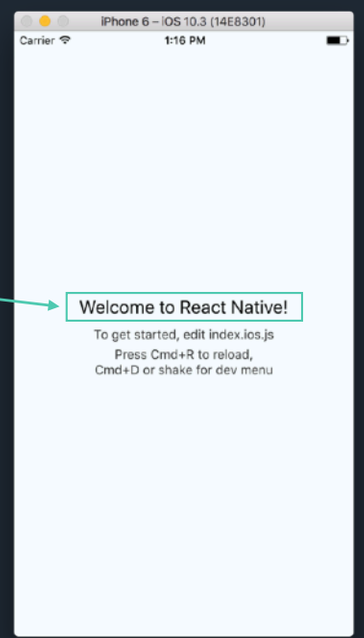
COMPONENTS

COMPONENTS

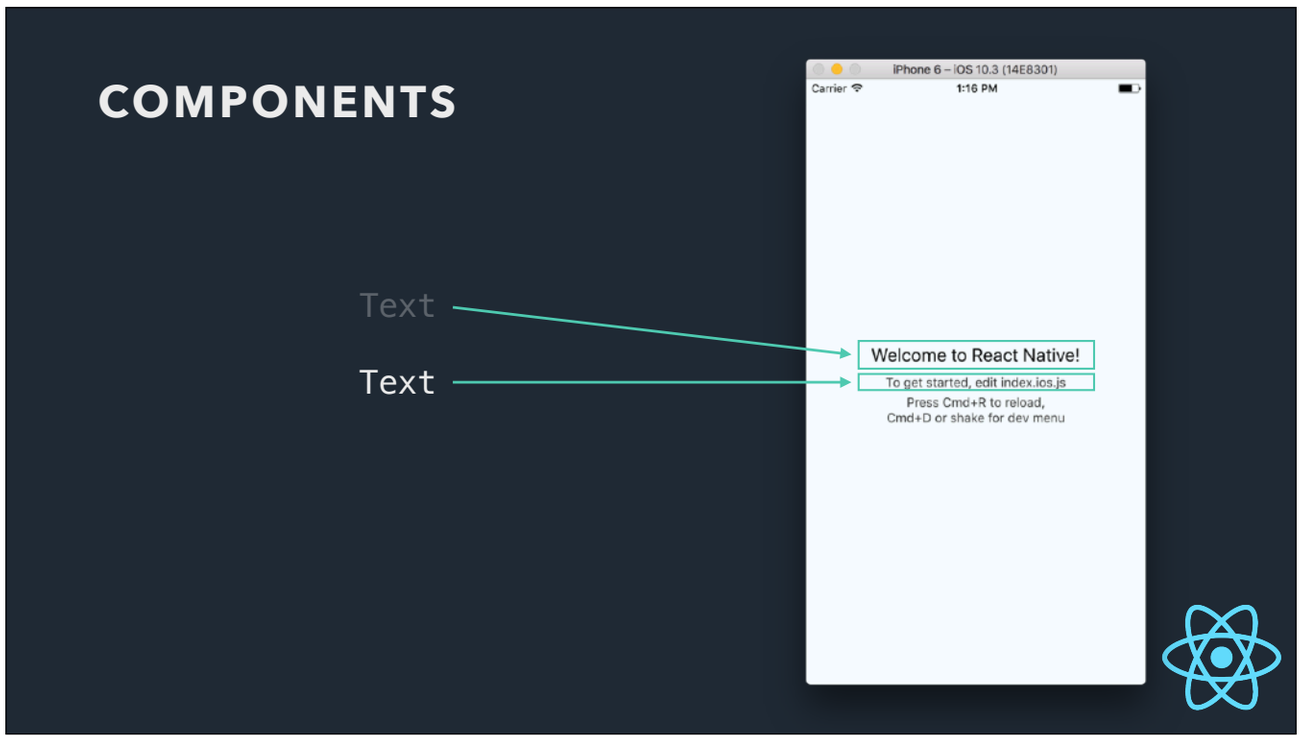

iPhone 6 – iOS 10.3 (14E8301)
Carrier
1:16 PM
Welcome to React Native!
To get started, edit index.ios.js
Press Cmd+R to reload,
Cmd+D or shake for dev menu

COMPONENTS

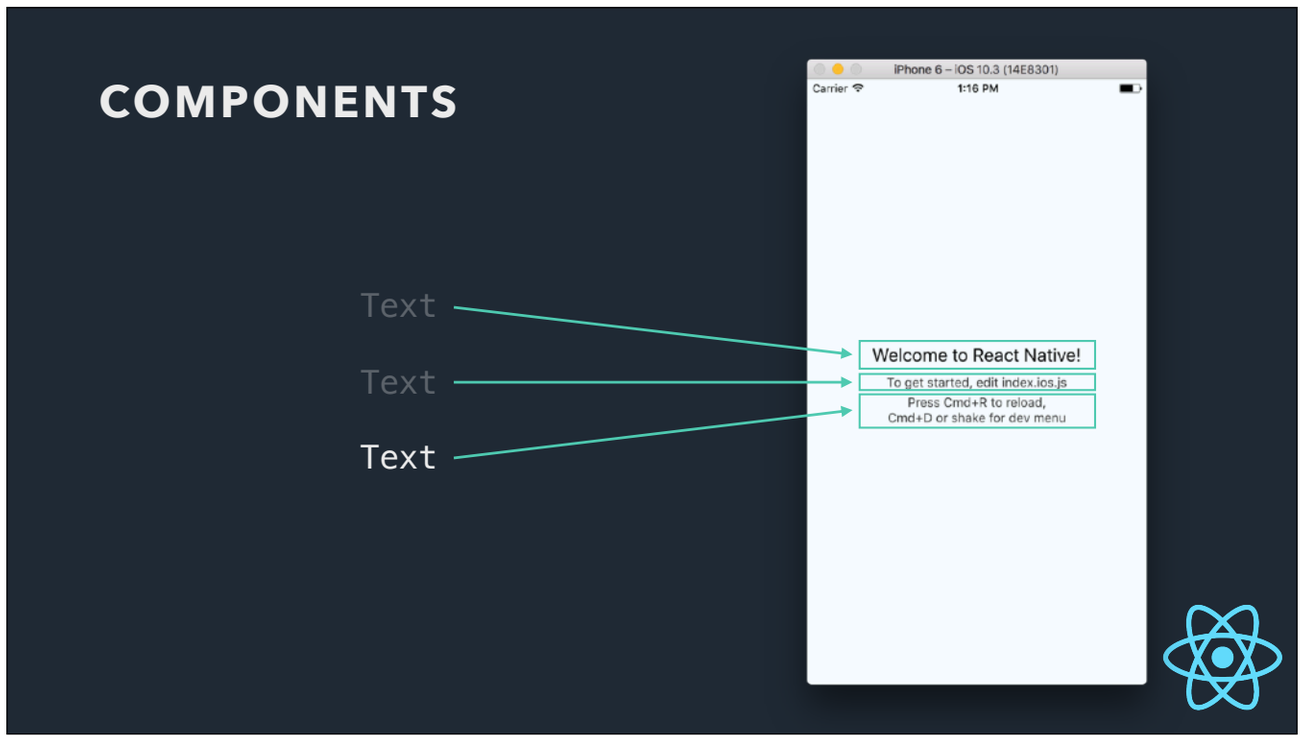
Text



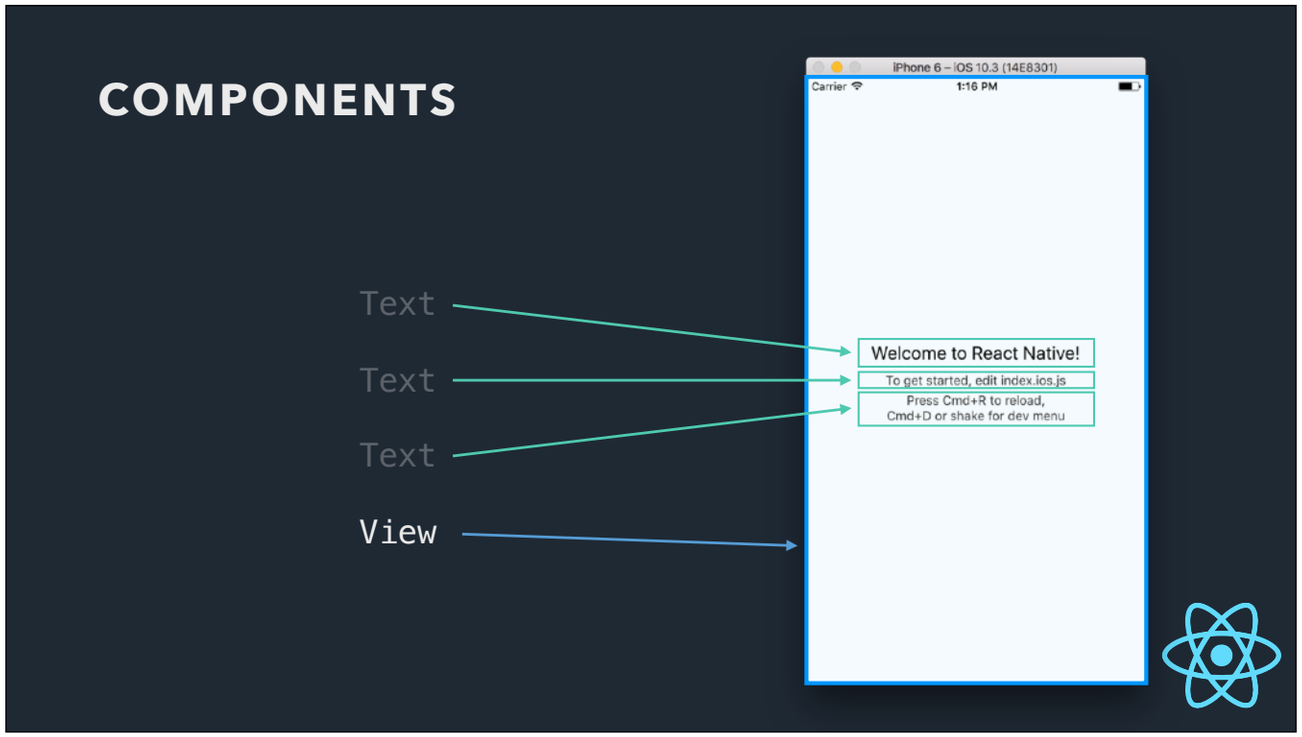
COMPONENTS

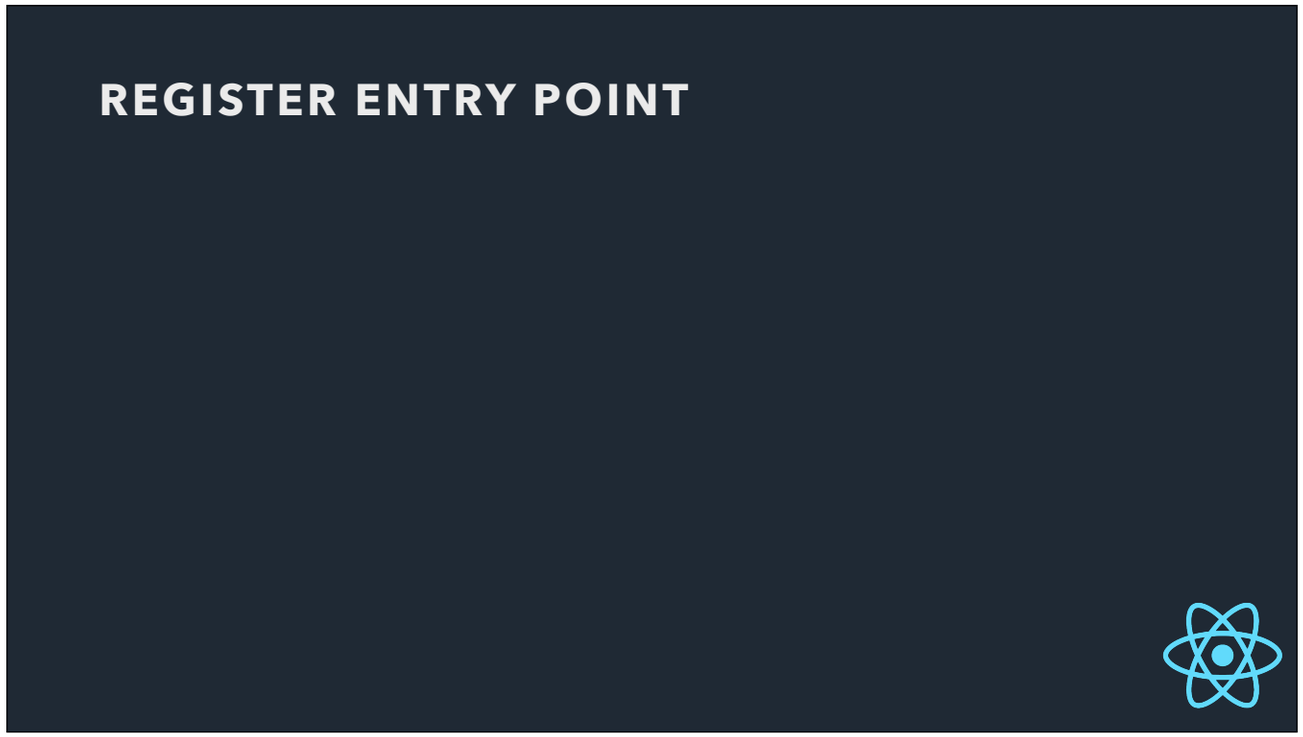

Text
Text
iPhone 6 – iOS 10.3 (14E8301)
Carrier
1:16 PM
Welcome to React Native!
To get started, edit index.ios.js
Press Cmd+R to reload,
Cmd+D or shake for dev menu

COMPONENTS


Text
Text
Text
iPhone 6 – iOS 10.3 (14E8301)
Carrier
1:16 PM
Welcome to React Native!
To get started, edit index.ios.js
Press Cmd+R to reload,
Cmd+D or shake for dev menu

COMPONENTS


Text
Text
Text
View
iPhone 6 – iOS 10.3 (14E8301)
Carrier
1:16 PM
Welcome to React Native!
To get started, edit index.ios.js
Press Cmd+R to reload,
Cmd+D or shake for dev menu



Finally, we register the entry point component of our app, so the react-native package knows where to build from.

REGISTER ENTRY POINT

```
AppRegistry.registerComponent(
  'MyNewApp', () => MyNewApp
)
```

Finally, we register the entry point component of our app, so the react-native package knows where to build from.

REGISTER ENTRY POINT

```
AppRegistry.registerComponent(
  'MyNewApp', () => MyNewApp
)
```

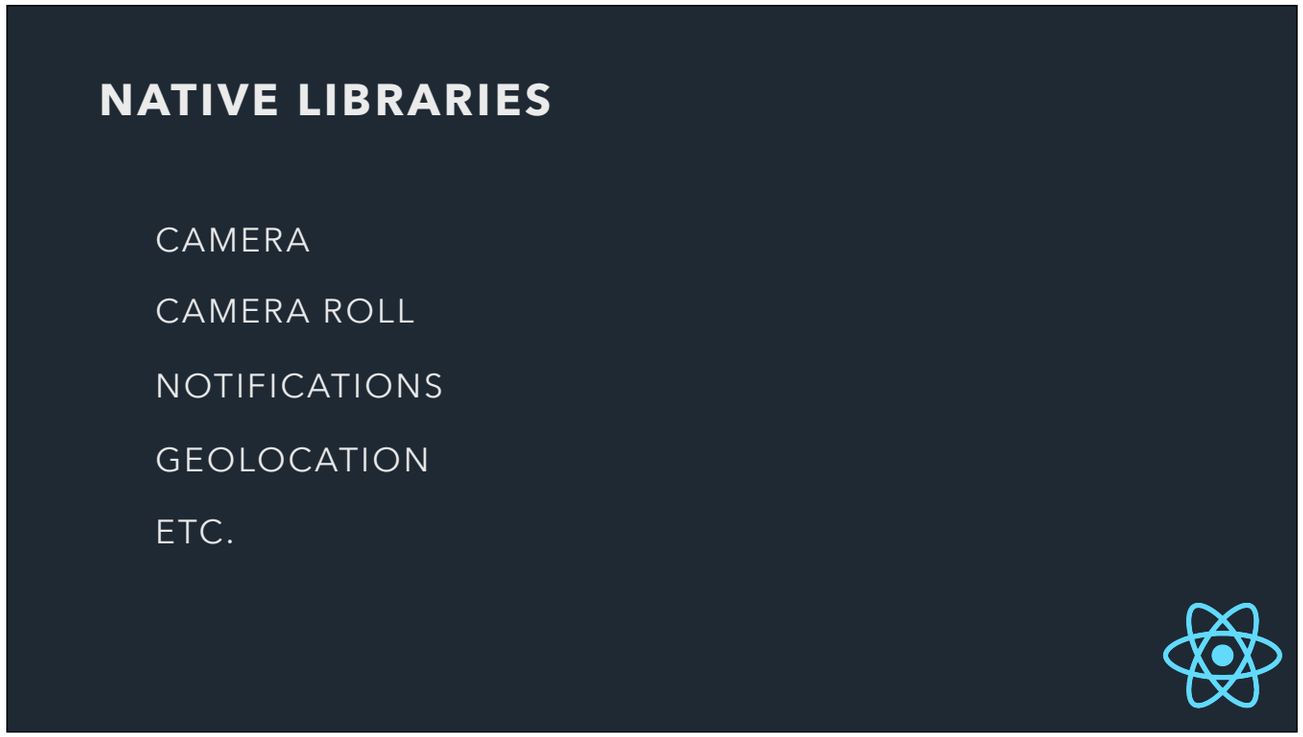
```
 */

#import "AppDelegate.h"

#import <React/RCTBundleURLProvider.h>
#import <React/RCTRootView.h>

@implementation AppDelegate

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(
    launchOptions
{
  NSURL *jsCodeLocation;

  jsCodeLocation = [[RCTBundleURLProvider sharedSettings] jsBundleURLForBundleRo
      fallbackResource:nil];
```

Finally, we register the entry point component of our app, so the react-native package knows where to build from.

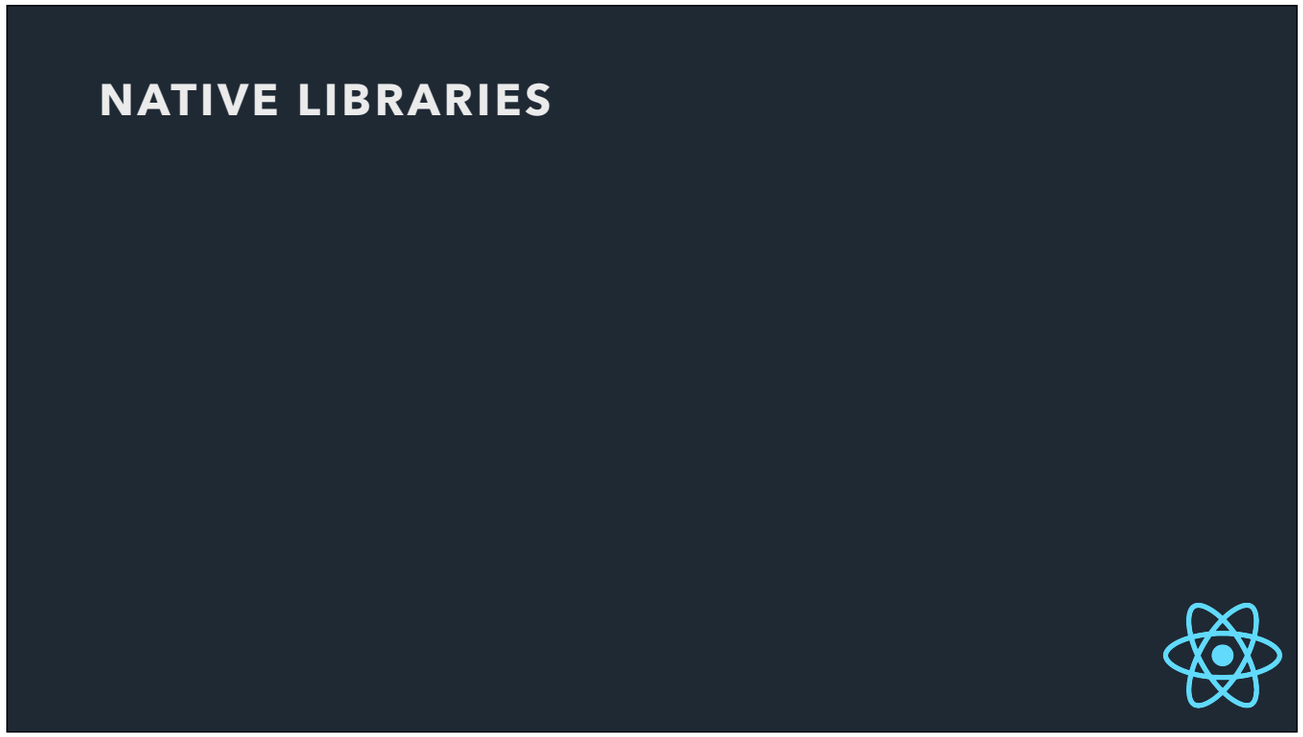
XCODE

```
Hannah@Hannahs-MacBook MyNewApp $ ls
__tests__                index.android.js        node_modules
android                  index.ios.js            package.json
app.json                 ios                     yarn.lock
Hannah@Hannahs-MacBook MyNewApp $ cd ios/
Hannah@Hannahs-MacBook ios $ ls
MyNewApp                 MyNewApp-tvOSTests      MyNewAppTests
MyNewApp-tvOS            MyNewApp.xcodeproj      build
Hannah@Hannahs-MacBook ios $
```

If we navigate into our app, and then into the ios folder, we find an xcodeproj folder.



If we navigate into our app, and then into the ios folder, we find an xcodeproj folder.



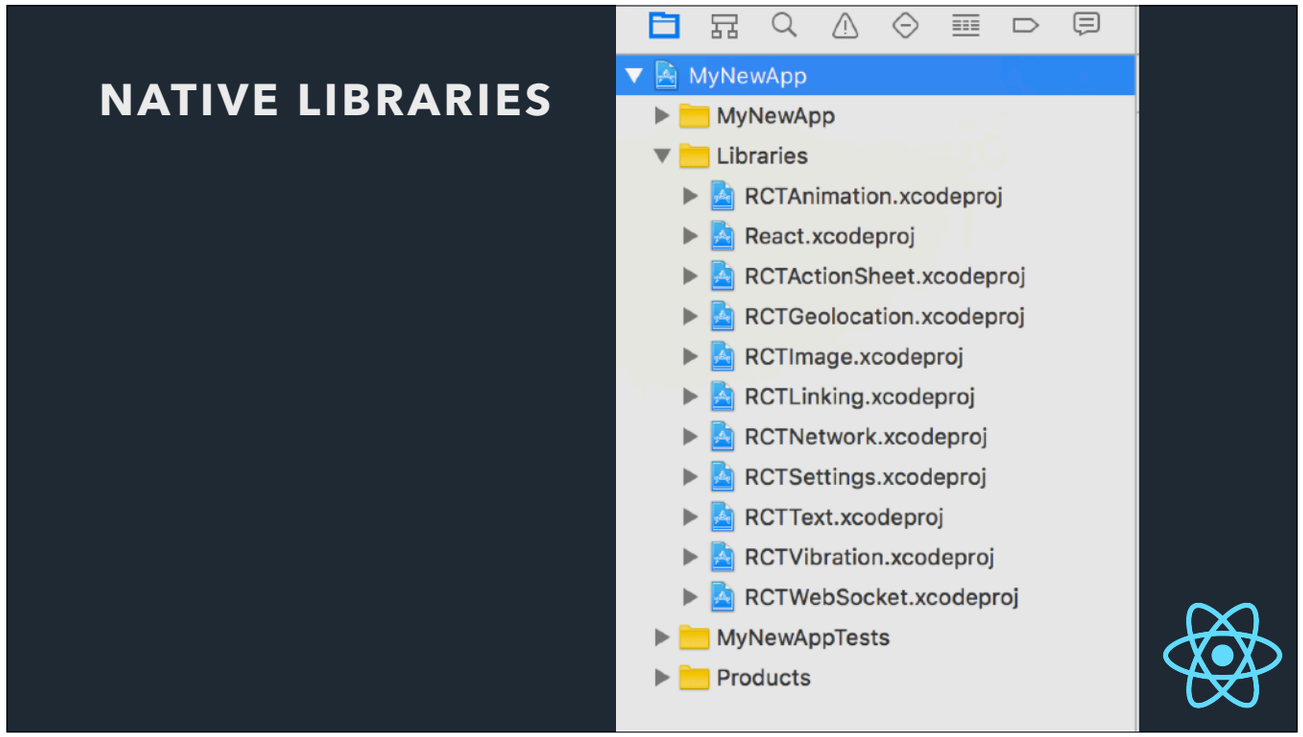
If we navigate into our app, and then into the ios folder, we find an xcodeproj folder.

NATIVE LIBRARIES

```
$ npm install <library> --save
$ react-native link
```

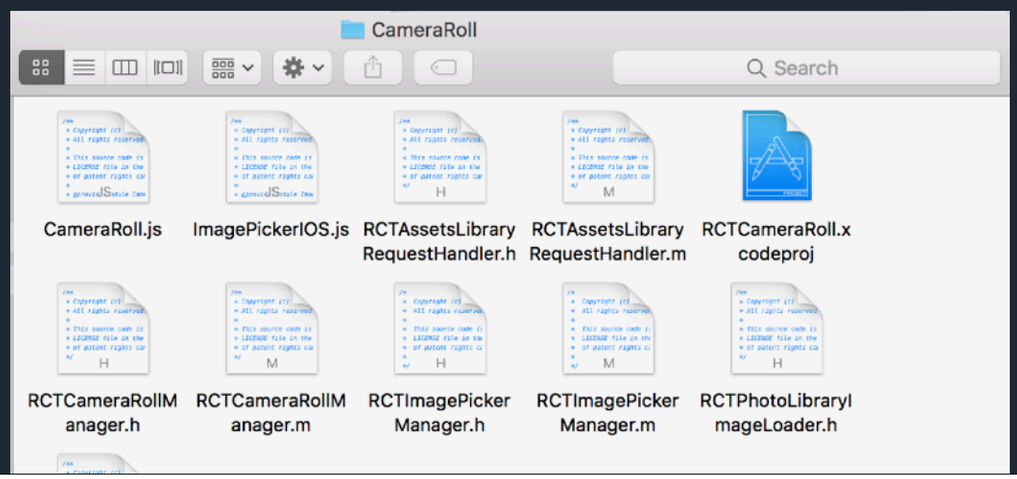
If we navigate into our app, and then into the ios folder, we find an xcodeproj folder.

If we navigate into our app, and then into the ios folder, we find an xcodeproj folder.



If we navigate into our app, and then into the ios folder, we find an xcodeproj folder.

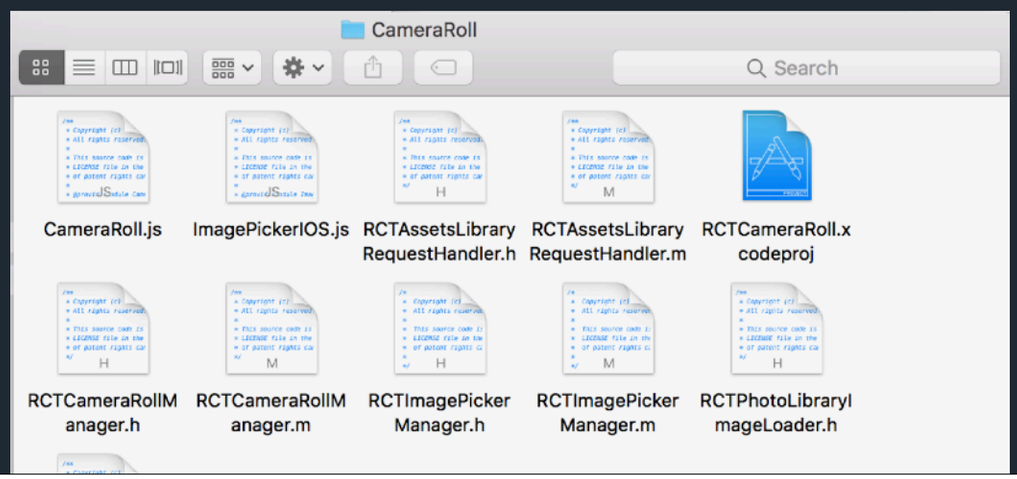
NATIVE LIBRARIES



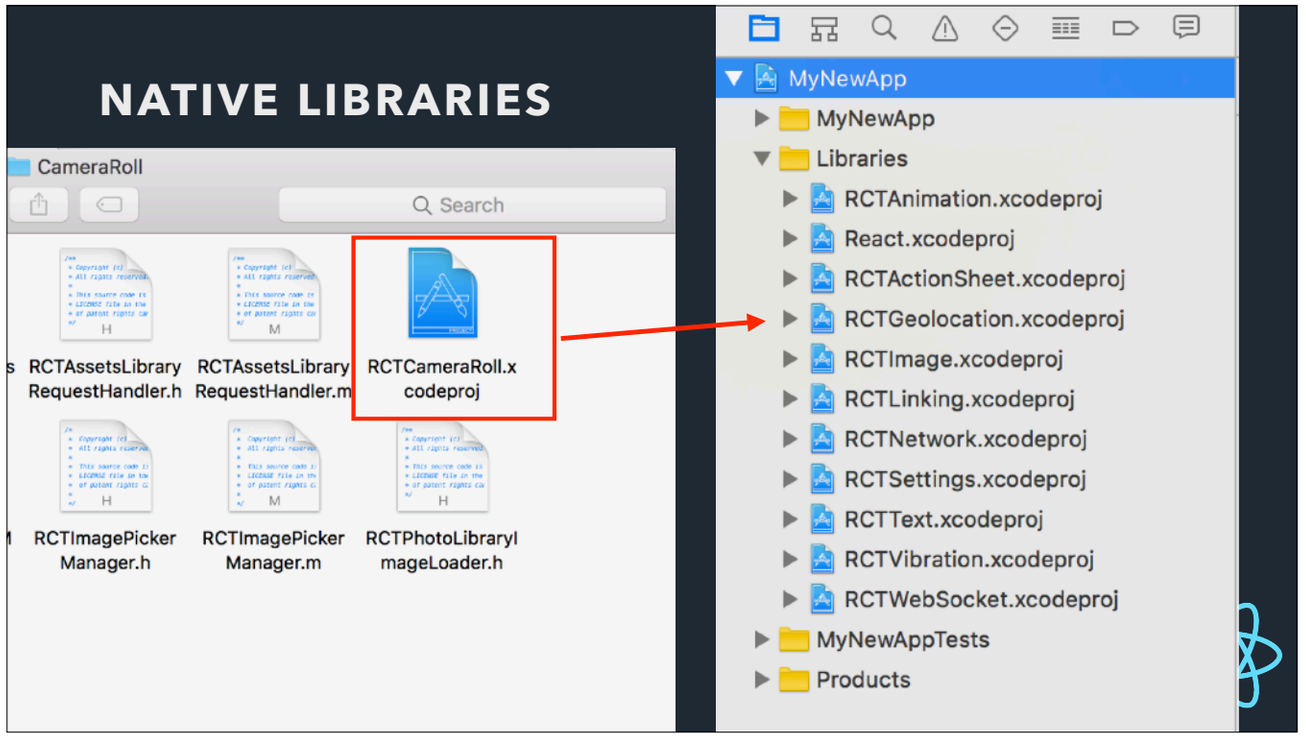
If we navigate into our app, and then into the ios folder, we find an xcodeproj folder.

NATIVE LIBRARIES

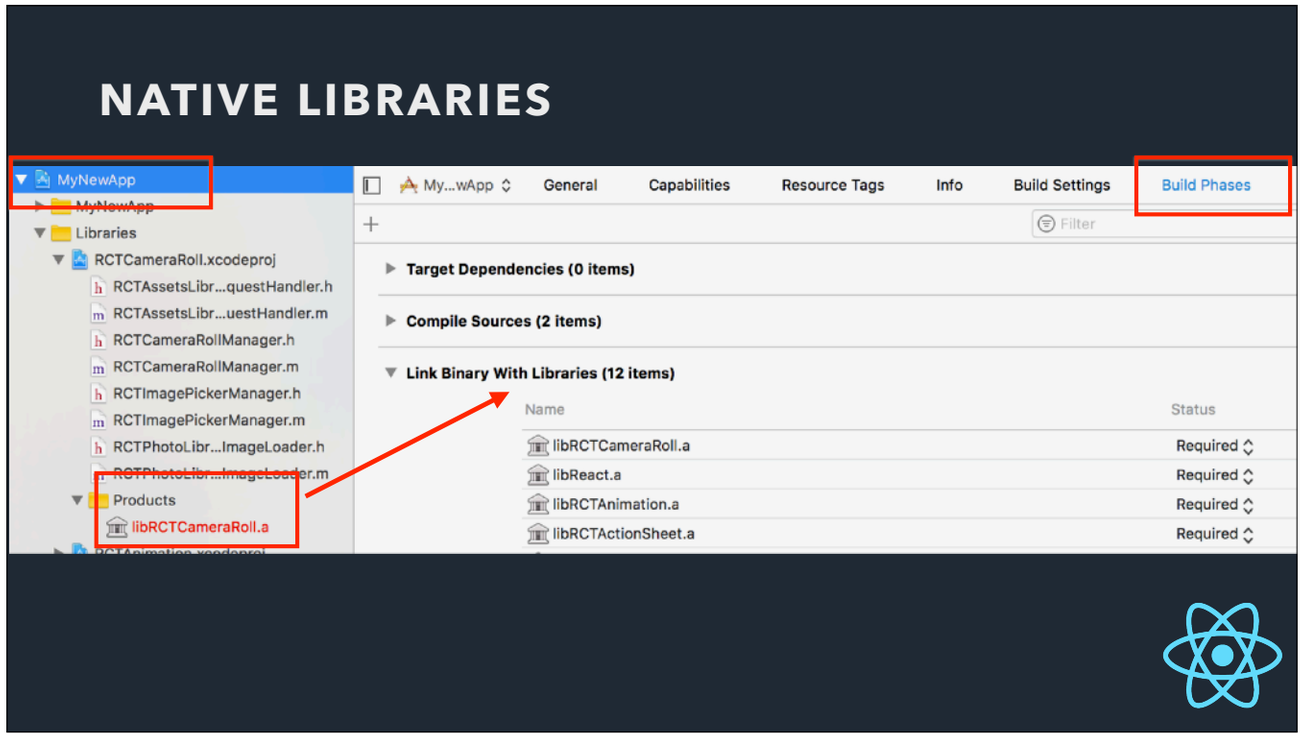
`node_modules/react_native/Libraries/CameraRoll`



If we navigate into our app, and then into the ios folder, we find an xcodeproj folder.



If we navigate into our app, and then into the ios folder, we find an xcodeproj folder.



If we navigate into our app, and then into the ios folder, we find an xcodeproj folder.

If we navigate into our app, and then into the ios folder, we find an xcodeproj folder.

API

```
import { CameraRoll } from 'react-native'


CameraRoll.getPhotos(params)
  .then((data) => dealWithIt(data))
```

If we navigate into our app, and then into the ios folder, we find an xcodeproj folder.

Inside the render method we use JSX, which looks like this. JSX allows us to use XML type markup within javascript.

Here is what the View component that we saw in the simulator looks like in JSX.

Inside the View component we build in the Text components.

We can use Javascript inside the JSX by escaping with the curly braces. Here we're accessing a javascript object called styles, and passing it into the style parameter of the component.

JSX

```
<View style={styles.container}>




</View>
```

Inside the render method we use JSX, which looks like this. JSX allows us to use XML type markup within javascript.

Here is what the View component that we saw in the simulator looks like in JSX.

Inside the View component we build in the Text components.

We can use Javascript inside the JSX by escaping with the curly braces. Here we're accessing a javascript object called styles, and passing it into the style parameter of the component.

JSX

```
<View style={styles.container}>
  <Text style={styles.content}>
    Welcome to React Native!
  </Text>


</View>
```

Inside the render method we use JSX, which looks like this. JSX allows us to use XML type markup within javascript.

Here is what the View component that we saw in the simulator looks like in JSX.

Inside the View component we build in the Text components.

We can use Javascript inside the JSX by escaping with the curly braces. Here we're accessing a javascript object called styles, and passing it into the style parameter of the component.

JSX

```
<View style={styles.container}>
  <Text style={styles.content}>
    Welcome to React Native!
  </Text>
  <Text style={styles.content}>
    Everything is a component
  </Text>
</View>
```

Inside the render method we use JSX, which looks like this. JSX allows us to use XML type markup within javascript.

Here is what the View component that we saw in the simulator looks like in JSX.

Inside the View component we build in the Text components.

We can use Javascript inside the JSX by escaping with the curly braces. Here we're accessing a javascript object called styles, and passing it into the style parameter of the component.

JSX

```
<View style={styles.container}>
  <Text style={styles.content}>
    Welcome to React Native!
  </Text>
  <Text style={styles.content}>
    Everything is a component
  </Text>
</View>
```

Inside the render method we use JSX, which looks like this. JSX allows us to use XML type markup within javascript.

Here is what the View component that we saw in the simulator looks like in JSX.

Inside the View component we build in the Text components.

We can use Javascript inside the JSX by escaping with the curly braces. Here we're accessing a javascript object called styles, and passing it into the style parameter of the component.

JSX

```
<View style={styles.container}>
  <Text style={styles.content}>
    Welcome to React Native!
  </Text>
  <Text style={styles.content}>
    Everything is a component
  </Text>
</View>
```

Inside the render method we use JSX, which looks like this. JSX allows us to use XML type markup within javascript.

Here is what the View component that we saw in the simulator looks like in JSX.

Inside the View component we build in the Text components.

We can use Javascript inside the JSX by escaping with the curly braces. Here we're accessing a javascript object called styles, and passing it into the style parameter of the component.

In React Native you have no choice but to write your styles within your javascript, because your styles ARE javascript. React Native styles are css LIKE, but aren't exactly the same as CSS. We use the StyleSheet class to make a new stylesheet object, and assign it to a constant that can then be given to our components. We then add in some style classes within our style object. These can be assigned to components wherever we import our stylesheet.

CamelCase
Most components have a style prop to feed your styles down to.

STYLES

```
const styles = StyleSheet.create({
```

In React Native you have no choice but to write your styles within your javascript, because your styles ARE javascript. React Native styles are css LIKE, but aren’t exactly the same as CSS. We use the StyleSheet class to make a new stylesheet object, and assign it to a constant that can then be given to our components. We then add in some style classes within our style object. These can be assigned to components wherever we import our stylesheet.

CamelCase
Most components have a style prop to feed your styles down to.

STYLES

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
```

In React Native you have no choice but to write your styles within your javascript, because your styles ARE javascript. React Native styles are css LIKE, but aren't exactly the same as CSS. We use the StyleSheet class to make a new stylesheet object, and assign it to a constant that can then be given to our components. We then add in some style classes within our style object. These can be assigned to components wherever we import our stylesheet.

CamelCase
Most components have a style prop to feed your styles down to.

STYLES

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  content: {
    …
  }
```

In React Native you have no choice but to write your styles within your javascript, because your styles ARE javascript. React Native styles are css LIKE, but aren't exactly the same as CSS. We use the StyleSheet class to make a new stylesheet object, and assign it to a constant that can then be given to our components. We then add in some style classes within our style object. These can be assigned to components wherever we import our stylesheet.

CamelCase
Most components have a style prop to feed your styles down to.

STYLES

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  content: {
    …
  }
})
```

In React Native you have no choice but to write your styles within your javascript, because your styles ARE javascript. React Native styles are css LIKE, but aren't exactly the same as CSS. We use the StyleSheet class to make a new stylesheet object, and assign it to a constant that can then be given to our components. We then add in some style classes within our style object. These can be assigned to components wherever we import our stylesheet.

CamelCase
Most components have a style prop to feed your styles down to.

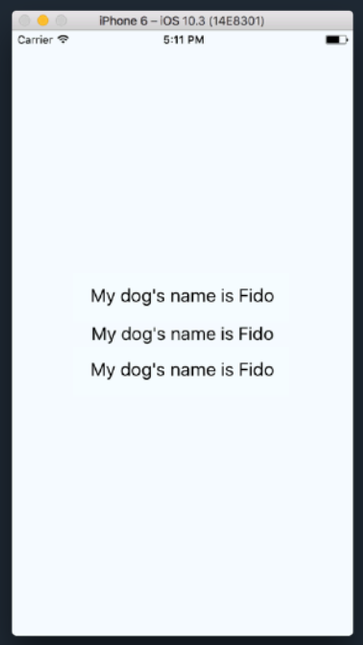
FLEXBOX

```
render() {
  return (
    <View>
      <Dog />
    </View>
  )
}
```

Now we can import Dog.js into our parent component, and call it using JSX. We can call it as many times as we like.

FLEXBOX

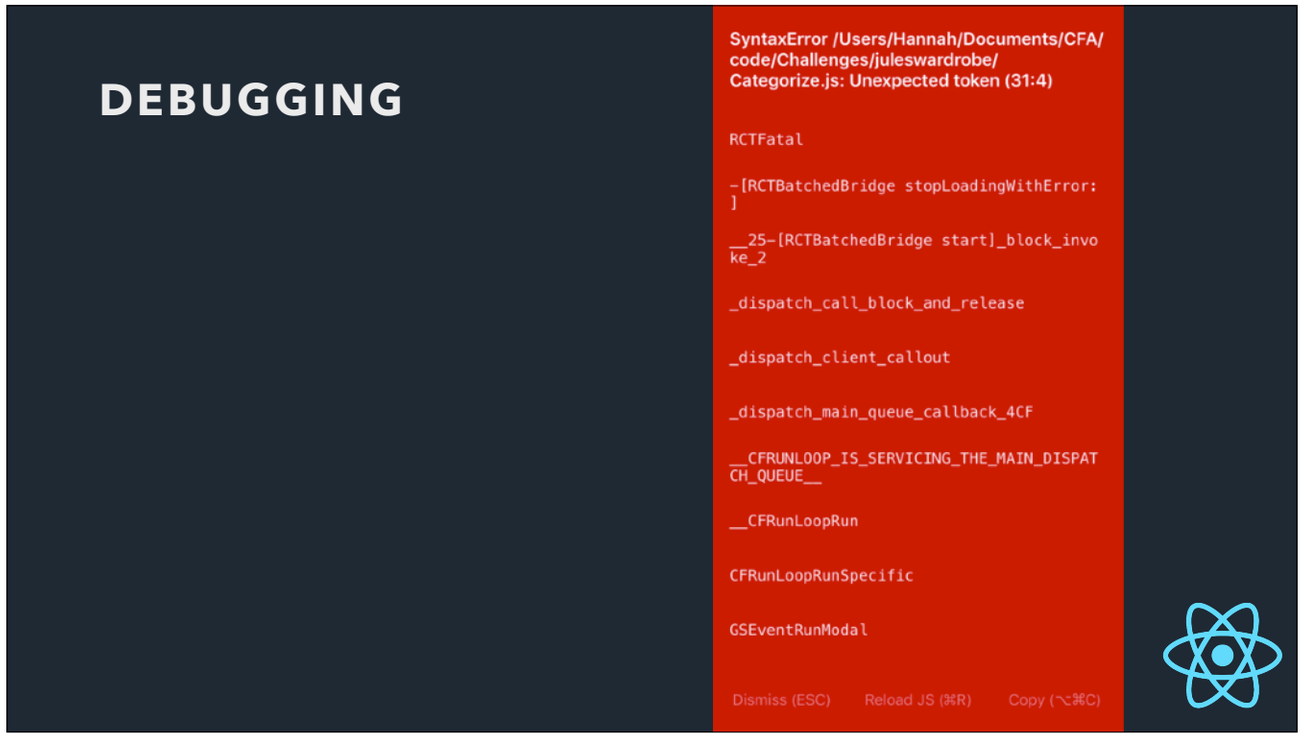
```
render() {
  return (
    <View>
      <Dog />
      <Dog />
      <Dog />
    </View>
  )
}
```



Now we can import Dog.js into our parent component, and call it using JSX. We can call it as many times as we like.

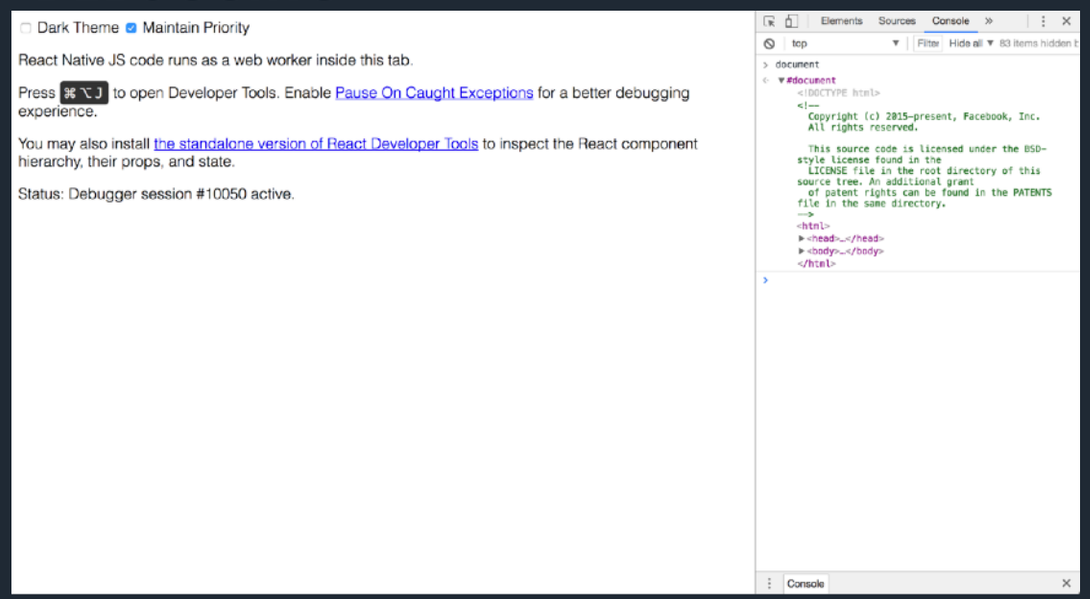
DEBUGGING

If, after all my great tips, you still find yourself confronted with the RED SCREEN OF DEATH, you can debug in the browser using Dev Tools to access the console.



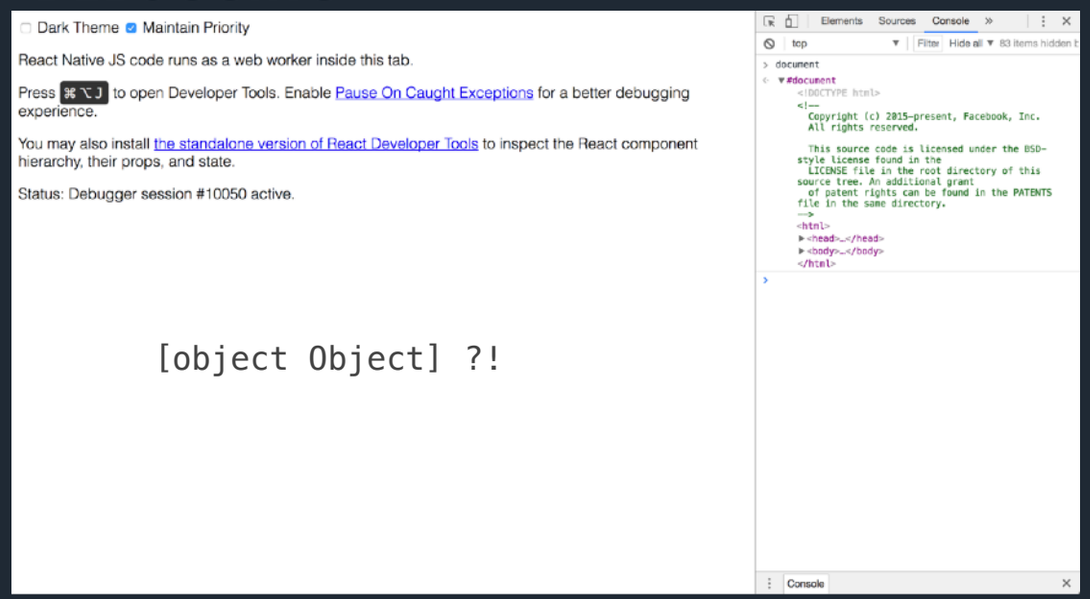
If, after all my great tips, you still find yourself confronted with the RED SCREEN OF DEATH, you can debug in the browser using Dev Tools to access the console.

DEBUGGING



If, after all my great tips, you still find yourself confronted with the RED SCREEN OF DEATH, you can debug in the browser using Dev Tools to access the console.

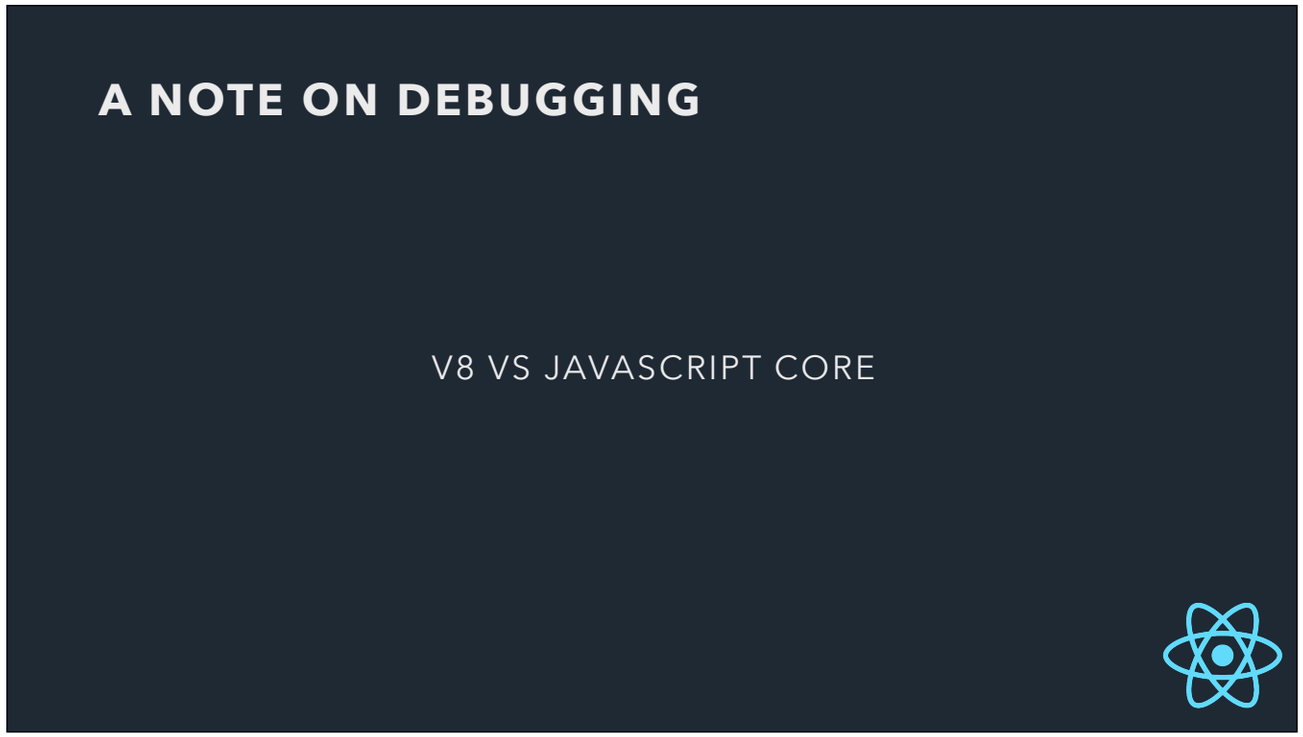
DEBUGGING



If, after all my great tips, you still find yourself confronted with the RED SCREEN OF DEATH, you can debug in the browser using Dev Tools to access the console.

A NOTE ON DEBUGGING

If, after all my great tips, you still find yourself confronted with the RED SCREEN OF DEATH, you can debug in the browser using Dev Tools to access the console.



If, after all my great tips, you still find yourself confronted with the RED SCREEN OF DEATH, you can debug in the browser using Dev Tools to access the console.

THANKS!

QUESTIONS?

🤔