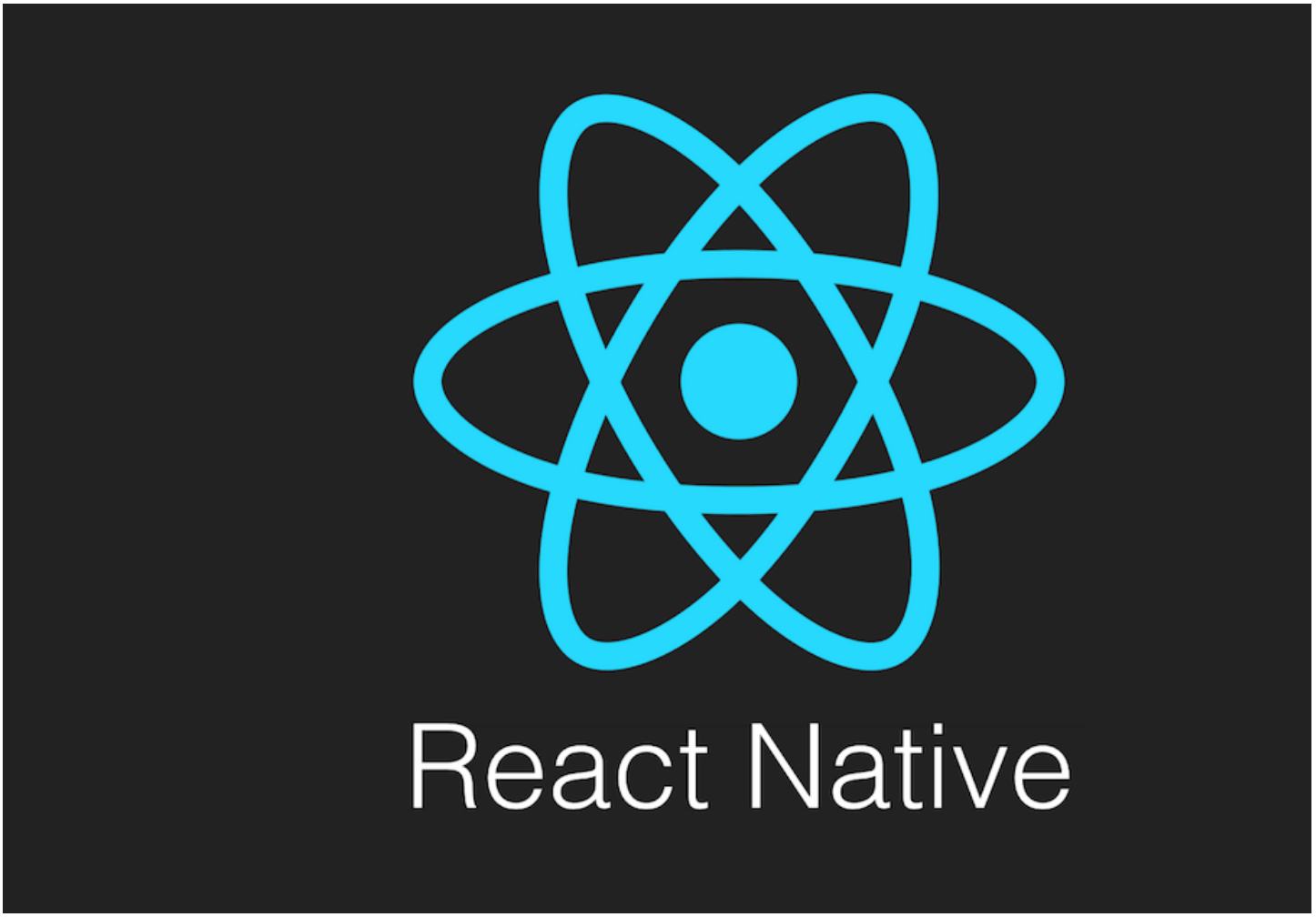
AppCoda

Introduction to React Native: Buildin iOS Apps with JavaScript

april 28, 2015 by joyce echessa • 17 comments

A few weeks ago Facebook open sourced React Native which is a framework that lets you build native iOS and Android (at the moment Android support is still under development) applications with JavaScript.

We've seen frameworks like Titanium and PhoneGap which offer developers an option of building mobile applications using web technologies. This is an advantage as they enable developers to use one set of skills for web and mobile development. Not only that, but the same code base could be used with little modification for multiple platforms – what became known as "Write once, run everywhere". However these frameworks fall shor comes to the performance of the apps that are built with them, so as much as they offer so attractive features, it has always been preferred to build native applications.



React Native is different from those types of frameworks. While a framework like PhoneGa wrapping web content in a WebView resulting in UI elements that don't quite have a native them, React native uses JavaScript components backed by native iOS or Android componer app you build is fully native.

React Native is not a "Write once, run everywhere" frameworks, as Facebook's Tom Occhin the video linked to at the end of the article. As you will see in the tutorial, you build the UI components that are specific to the platform, so you can't take the same code and run it in What React Native enables you to do is learn one set of skills and use it to build for multipl as Occhino goes further to say it is a "Learn once, write anywhere" framework. This tutoria introduce you to React Native development by going through the process of building a sim with the framework.

Getting Started

First, we'll go through the process of installing React Native to your development machine.

Before we get started I should mention this: you can grab the React Native framework cod Github. There are some example projects in there that you can run and learn from namely

game), **Movies** (A movie browser app), **SampleApp** (a blank React Native app), **TicTacToe** (th and **UIExplorer** (an app that shows examples of all the react native components that you us your apps like ListView, TabBar, MapView, Slider, e.t.c.). These are great for learning how c elements are built using React Native especially the UIExplorer app which has just about ev element you might need. However, some of the apps are buggy, I have had them crash on r times after trying to take certain actions. Still they are great to learn from and you can also documentation for further details.

Now for the installation. React native uses Node.js to build the JavaScript code. If you alrea installed on your computer, you can skip the next few steps, otherwise, proceed.

We'll install Node.js using Homebrew This isn't the only way to install Node but I find that I is great as a package manager. With it you can easily install the latest as well as specific ver package, have different versions of a package, select which version to use, update and unin packages, e.t.c. To install Homebrew, go to their website and follwow the instruction at the page. I'd rather not copy and paste the download link here incase it ever changes.

After Homebrew is installed, install Node.js by pasting the following in a Terminal window.

```
brew install node
```

Next install watchman.

```
brew install watchman
```

Watchman is a file watcher from Facebook. React Native uses it to detect code changes so rebuild when they occur.

Next install the React Native CLI tool with the following.

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

npm is the Node Package Manager. You can think of it as RubyGems for Ruby, CocoaPods f Gradle/Maven for Java e.t.c. Basically it enables you to easily download and manage any dependancies your project needs.

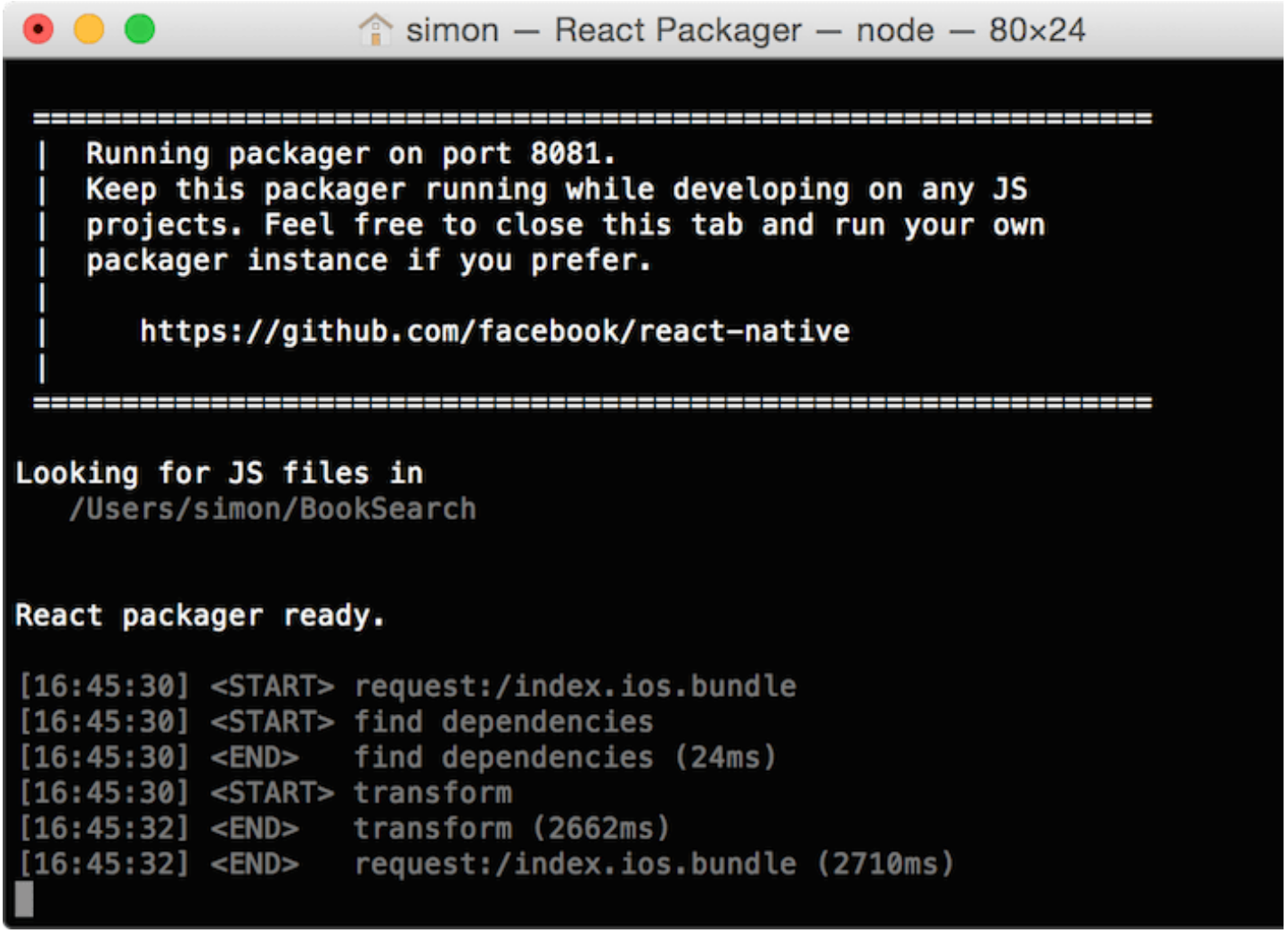
In Terminal, navigate to the folder you'd like to save your project to and then run the follov

```
react-native init BookSearch
```

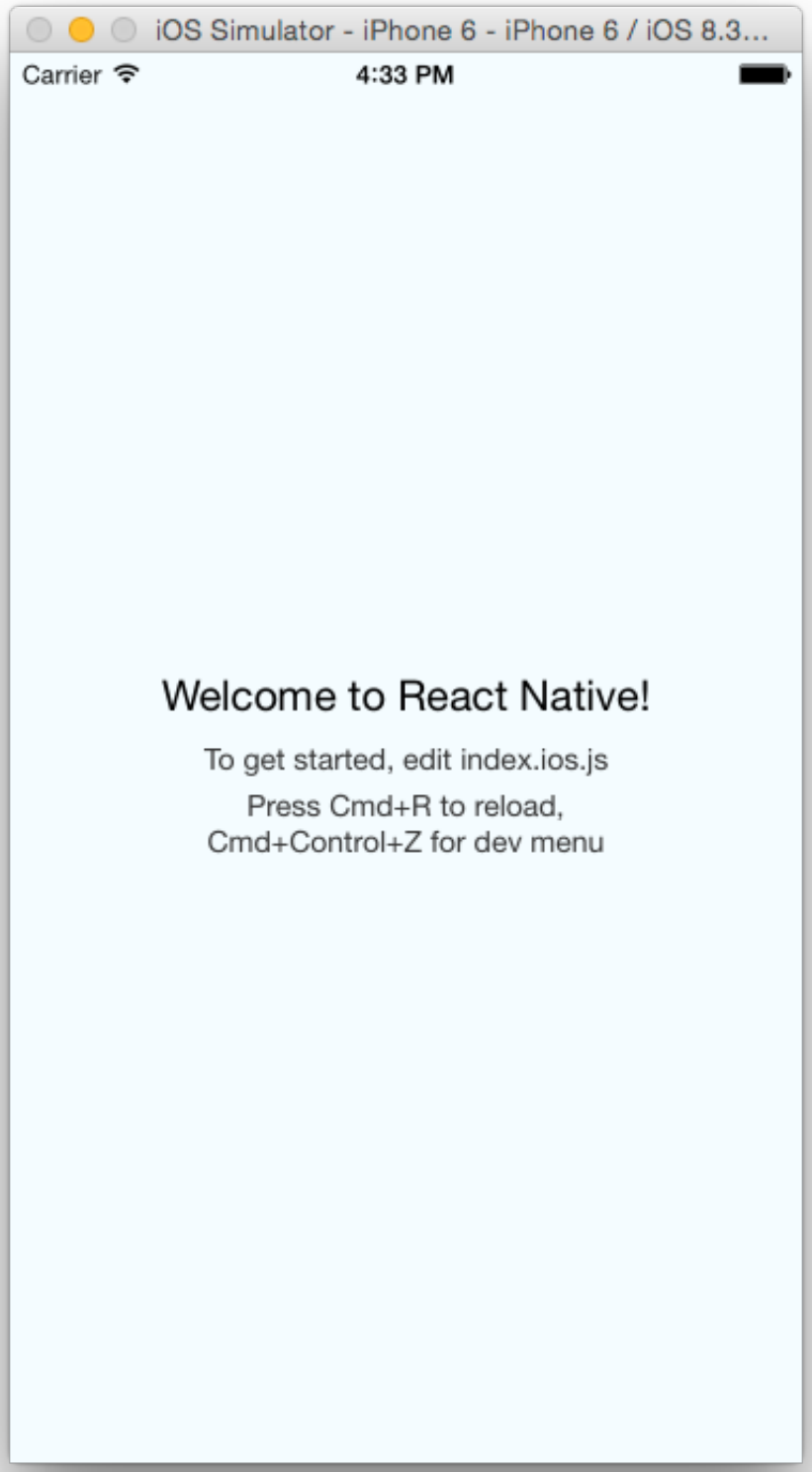
The above uses the CLI tool to construct a React Native project that is ready to build and r
When the process is done, you will get a message on the Terminal window to open *BookSearch.xcodeproj* in Xcode and run the app as usual. Do this and the simulator will sta your app running. A terminal window will also be opened. When a React Native app launch the JavaScript application from the following URL.

```
1 http://localhost:8081/index.ios.bundle
```

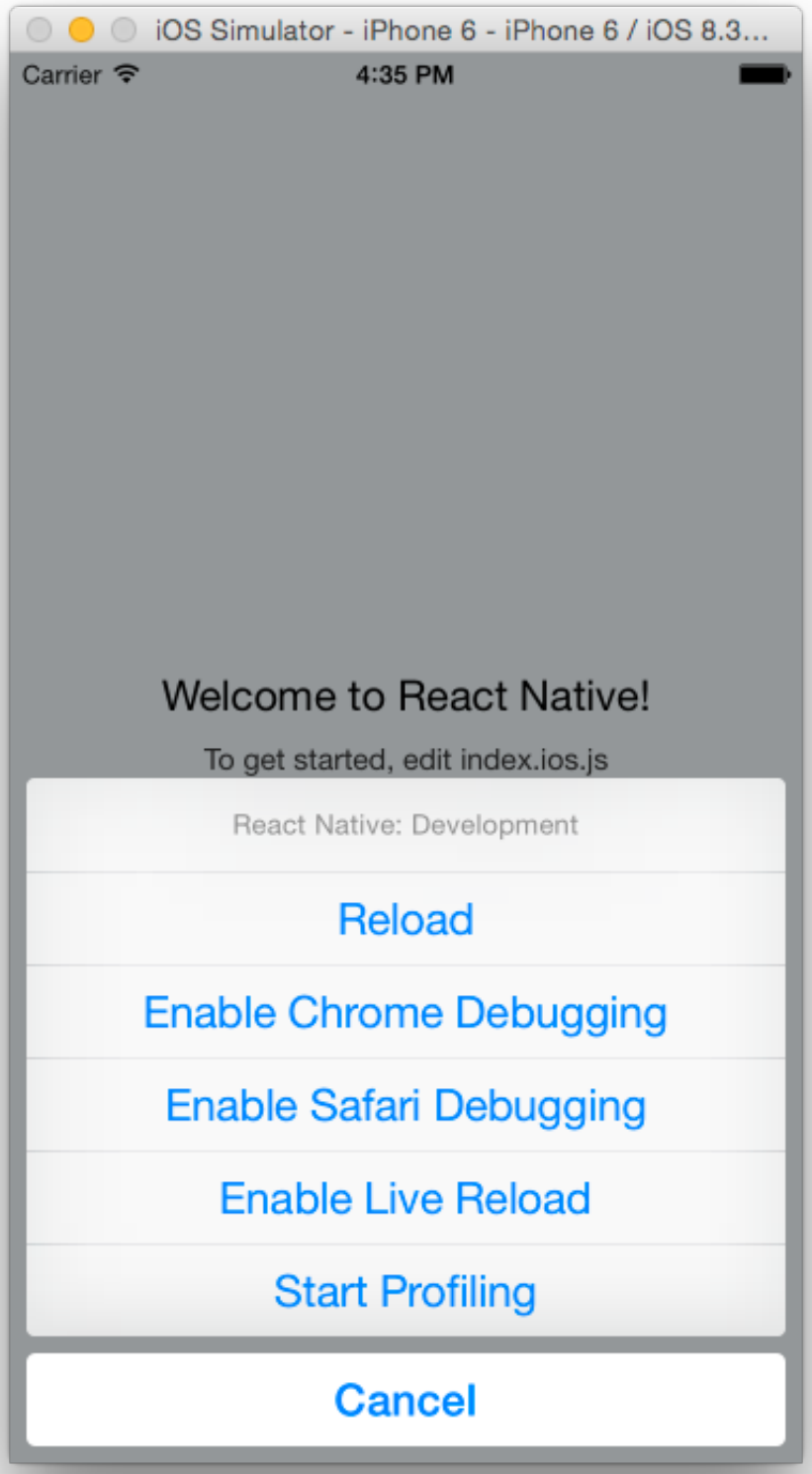
The Terminal is opened to start the React Packager and a server to handle the above reque React Packager is responsible for reading and building the JSX (we'll look at this later) and J code.



On running the app, you should see the following in the simulator. If you want to run on a there are a few steps you should take to do so.



The welcome screen gives some crucial instructions which you should note: to edit the app should edit the *index.ios.js* file that was generated when you created the project, if you ma to the JavaScript code, reload the app with Command-R to see the changes and if you want options, use Command-Control-Z to open the developer menu which offers such options a live reloading and browser debugging.



At any point while doing the tutorial should you encounter a red screen on the simulator, c
error message on the simulator. This will let you know whether the problem is with your c
the server. I have encountered a problem with the server connection several times whereb
gotten the error message “Could not connect to the server” on the simulator and on check
Terminal got a “Process terminated” error message. When this happens, close the Termina
stop the app in Xcode and run it again. For other error that are due to syntax errors in cod

network request timeouts (if your app is fetching data from the internet), a simple reloadin fixing the issue should do.

If you are pressing Command-R on your keyboard and nothing is happening, the hardware might not be connected to the simulator. To connect it select the option on the simulator going to Hardware > Keyboard > Connect Hardware Keyboard.

If you've done the above and it still won't reload, then you might require a restart of your c I've encountered this once whereby it was working fine and then it stopped working. Resta computer fixed it.

We'll now start building our app. Open the *index.ios.js* file. I recommend using an IDE suita development. You can still use Xcode but you'll find that it's not very suitable for this as it v much when it comes to code formatting, autocompletion or syntax error highlighting. For JavaScript IDEs you can read through this post and make your decision. I used RubyMine, b any IDE that has support for JavaScript will do. If you can get one that supports JSX that wi better.

When you open the *index.ios.js* file you will see the code that builds the UI you saw on the app. You will see the following code blocks.

```
1 'use strict';
```

The above enables Strict Mode, which adds improved error handling to the React Native Ja code.

```
1 var React = require('react-native');
```

The above loads the react-native module and assigns it to the variable React. You have to l external modules into your file before you can call any functions in the modules. Think of i importing libraries in Swift and Objective-C.

```
1 var {
2   AppRegistry,
3   StyleSheet,
4   Text,
5   View,
6 } = React;
```

The above is a destructuring assignment which enables you to assign multiple object prope

single variable. This makes them scoped references in the file. The above is optional, but if out, then every time you use a component in your code, you would have to use its full quali for example 'React.AppRegistry' instead of 'AppRegistry' or 'React.StyleSheet' instead of 'Sty e.t.c.

```
var BookSearch = React.createClass({
  render: function() {
    return (
      <View style={styles.container}>
        <Text style={styles.welcome}>
          Welcome to React Native!
        </Text>
        <Text style={styles.instructions}>
          To get started, edit index.ios.js
        </Text>
        <Text style={styles.instructions}>
          Press Cmd+R to reload,{'\n'}
          Cmd+Control+Z for dev menu
        </Text>
      </View>
    );
  }
});
```

The above creates a class that has only one function *render()*. Whatever is defined in rende will be output to the screen. The above uses JSX (JavaScript syntax extension) to construct UI. If you've used XML (or even HTML) before, then JSX will look familiar to you. It has the balanced use of opening and closing tags and use of attributes to set values on tags. You do use JSX with React Native, you can just use plain JavaScript, but JSX is recommended for its in defining tree structures. If you have a lot of code for your UI, it will be much easier to re the large JSX tree structure.

```
var styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF',
  },
  welcome: {
    fontSize: 20,
    textAlign: 'center',
    margin: 10,
  },
  instructions: {
    textAlign: 'center',
    color: '#333333',
    marginBottom: 5,
```

```
  },
});
```

Above are the styles that are applied to the view's content. If you've done web development CSS (Cascading Style Sheets) before then this should be familiar. React Native uses CSS to s app's UI. If you look at the JSX code you will see where each style is used, for example *style {styles.container}* sets the style defined for *container* for the outer view that encloses the c components.

```
AppRegistry.registerComponent('BookSearch', () => BookSearch);
```

The above defines the entry point to the application. This is where the JavaScript code star executing.

That's the basic structure of a React Native UI. Every view we define will follow the same ba structure.

In the tutorial, we'll create an app that lets you browse through books and see details abou like the author, title and a short description of the book. You will also be able to search for the book title and/or author. Below is what the app will look like. We'll use the Google boo the data.

Adding a Tab Bar

The app will have a tab bar with two items on it – Featured and Search. We'll add this first.

While you can have all your code in the *index.ios.js* file, this is not recommended as things get messy as the app's code grows. We'll create our classes in different files, for better man

Create two JavaScript files in the root directory of your project (the same location as the in file). Name the files Search.js and Featured.js. Open Featured.js and add the following code.

```
'use strict';

var React = require('react-native');

var {
    StyleSheet,
    View,
```

```
    Text,
    Component
  } = React;

var styles = StyleSheet.create({
    description: {
        fontSize: 20,
        backgroundColor: 'white'
    },
    container: {
        flex: 1,
        justifyContent: 'center',
        alignItems: 'center'
    }
});

class Featured extends Component {
    render() {
        return (
        <View style={styles.container}>
            <Text style={styles.description}>
              Featured Tab
            </Text>
        </View>
        );
    }
}

module.exports = Featured;
```

You should be familiar with the above code; it's very similar to the code we looked at earlie the Strict Mode, load the react-native module, create the view styles and render UI output *render()* function. The last line of the code exports the Featured class thus making it availal by other files. Notice we declare the class and function a little differently from the example *index.ios.js*. JavaScript has different ways to declare classes and functions. Feel free to cho you prefer. For the rest of the tutorial, we'll use the style we used above.

In the stylesheet definition, we see basic CSS properties. We set the font size and backgrou for the text and center content in the outer view. But you might not be familiar with the *fle* This is flexbox, a recent addition to the CSS specification. *flex: 1* here makes the element m container to take up the rest of the space on screen not occupied by sibling elements, othe will just take up enough space to fit its content. We'll look more into flex later on. To learn Flexbox styling, you can read this guide.

In Search.js add the following.

```
'use strict';
```

```
2
3  var React = require('react-native');
4
5  var {
6      StyleSheet,
7      View,
8      Text,
9      Component
10    } = React;
11
12 var styles = StyleSheet.create({
13     description: {
14         fontSize: 20,
15         backgroundColor: 'white'
16     },
17     container: {
18         flex: 1,
19         justifyContent: 'center',
20         alignItems: 'center'
21     }
22 });
23
24 class Search extends Component {
25     render() {
26         return (
27         <View style={styles.container}>
28             <Text style={styles.description}>
29               Search Tab
30             </Text>
31         </View>
32         );
33     }
34 }
35
36 module.exports = Search;
```

The above is similar to the code in Featured.js except for the text in the Text component.

In index.ios.js delete everything and paste in the following.

```
1  'use strict';
2
3  var React = require('react-native');
4  var Featured = require('./Featured');
5  var Search = require('./Search');
6
7  var {
8      AppRegistry,
9      TabBarIOS,
10     Component
11   } = React;
```

```
12
13 class BookSearch extends Component {
14
15     constructor(props) {
16         super(props);
17         this.state = {
18             selectedTab: 'featured'
19         };
20     }
21
22     render() {
23         return (
24             <TabBarIOS selectedTab={this.state.selectedTab}>
25                 <TabBarIOS.Item
26                     selected={this.state.selectedTab === 'featured'}
27                     icon={{uri:'featured'}}
28                     onPress={() => {
29                         this.setState({
30                             selectedTab: 'featured'
31                         });
32                     }}>
33                     <Featured/>
34                 </TabBarIOS.Item>
35                 <TabBarIOS.Item
36                     selected={this.state.selectedTab === 'search'}
37                     icon={{uri:'search'}}
38                     onPress={() => {
39                         this.setState({
40                             selectedTab: 'search'
41                         });
42                     }}>
43                     <Search/>
44                 </TabBarIOS.Item>
45             </TabBarIOS>
46         );
47     }
48 }
49
50 AppRegistry.registerComponent('BookSearch', () => BookSearch);
```

Here we require the two modules we exported in the files we created and assign them to v
Inside the class, we specify a constructor which we use to set a state for the class. Compor
state variable which we use here. We create a property named *selectedTab* and set its valu
'featured'. We'll use 'featured' to determine which tab should be active. We've set the defau
Featured tab.

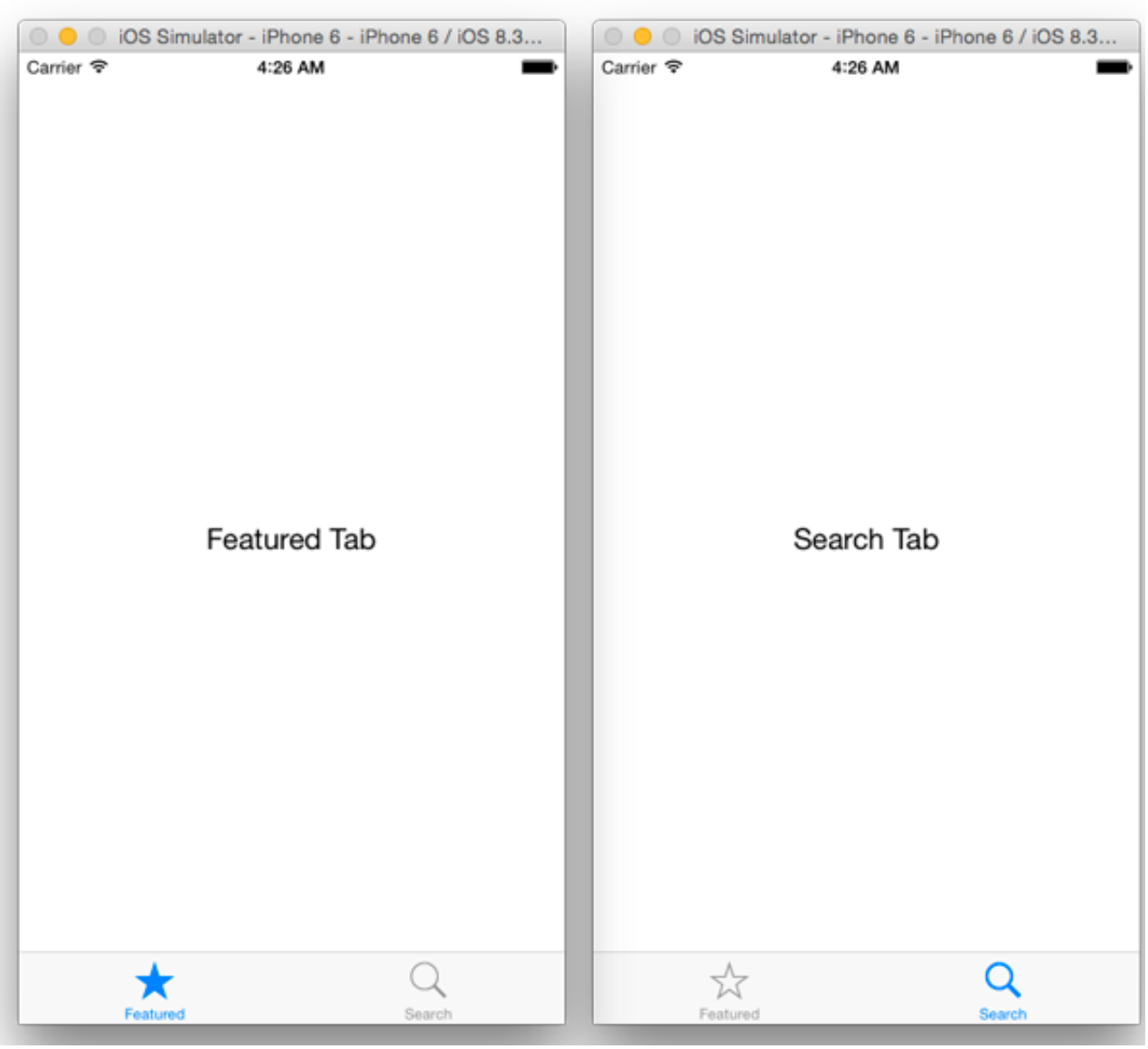
In the *render()* function we use the TabBarIOS component to create a tab bar. Remember t
components you use to the destructuring assignment otherwise use its full qualified name
React.TabBarIOS.

We create two tab bar items. For each item we set its selected state and define a function t called when the item is pressed. For the Featured tab, *selected* is set to true if the *selected* we defined earlier has a value of 'featured' otherwise it will be set to false. Same things for tab except we check if *selectedTab* is equal to 'search'. Whichever item has *selected* set to the active tab. We use system icons for the tab bar items.

Notice we use our custom components in tags just like any other component e.g. . Since w the corresponding module and assigned it to a variable, you can use the variable to bring ir component to the file. This results in the code that is in the *render()* function of the compo to be included as if it was part of the file. By the way, I use the same name for the variables respective class names but it's not a requirement, you can use any name you prefer.

When a tab bar item is pressed, the callback function defined in the component's *onPress* a called. The function sets a value for the *selectedTab* property which will eventually determ active tab.

Bring up the simulator and press Command-R to reload the app. You should see the follow



Adding a Navigation Bar

Next we'll add a navigation bar to the app. Add two more files to the project. These will be t
views in the navigation stack of their respective tabs. Name the files BookList.js and Search

In BookList.js add the following code.

```
'use strict';

var React = require('react-native');

var {
    StyleSheet,
    View,
    Component
  } = React;

var styles = StyleSheet.create({

});
```

```
14
15 class BookList extends Component {
16     render() {
17         return (
18             <View>
19         </View>
20         );
21     }
22 }
23
24 module.exports = BookList;
```

In SearchBooks.js add the following.

```
1  'use strict';
2
3  var React = require('react-native');
4
5  var {
6      StyleSheet,
7      View,
8      Component
9     } = React;
10
11 var styles = StyleSheet.create({
12
13 });
14
15 class SearchBooks extends Component {
16     render() {
17         return (
18             <View>
19         </View>
20         );
21     }
22 }
23
24 module.exports = SearchBooks;
```

In both files, we've created a module with a blank view and exported the module.

Modify Featured.js as shown.

```
1  'use strict';
2
3  var React = require('react-native');
4  var BookList = require('./BookList');
5
6  var {
7      StyleSheet,
```

```
    NavigatorIOS,
    Component
   } = React;

var styles = StyleSheet.create({
    container: {
        flex: 1
    }
});

class Featured extends Component {
    render() {
        return (
            <NavigatorIOS
                style={styles.container}
                initialRoute={{
            title: 'Featured Books',
            component: BookList
            }}/>
        );
    }
}

module.exports = Featured;
```

The above uses the NavigatorIOS component to construct a navigation controller. We set i route to the BookList component (which will be its root view) and set a title that will appea navigation bar.

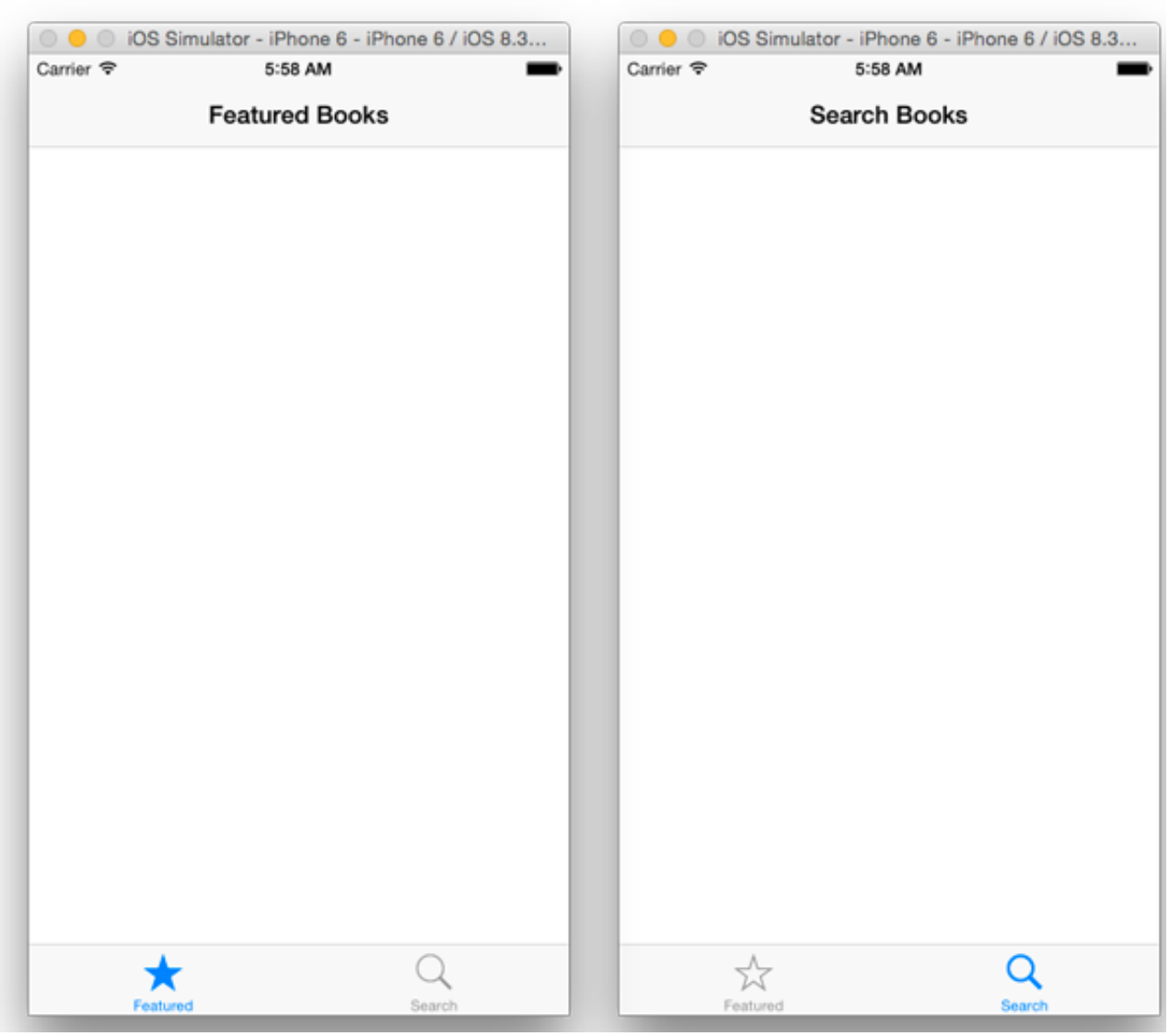
Next modify Search.js as shown below.

```
'use strict';

var React = require('react-native');
var SearchBooks = require('./SearchBooks');

var {
    StyleSheet,
    NavigatorIOS,
    Component
   } = React;

var styles = StyleSheet.create({
    container: {
        flex: 1
    }
});

class Search extends Component {
    render() {
        return (
```

```
21              <NavigatorIOS
22                  style={styles.container}
23                  initialRoute={{
24              title: 'Search Books',
25              component: SearchBooks
26          }}/>
27          );
28      }
29  }
30
31  module.exports = Search;
```

Just as in Featured.js, the above created a navigation controller, sets its initial route and se
it.

Reload the app and you should have the following.



Fetching and Displaying Data

Now we'll start adding data to our views. At first we'll construct the view with fake data anc

on use real data from an API.

In BookList.js add the following at the top of the file with the other variable declarations.

```
var FAKE_BOOK_DATA = [
    {volumeInfo: {title: 'The Catcher in the Rye', authors: "J. D. Salinger"
];
```

Modify the destructuring assignment as shown to include more components that we'll use.

```
var {
    Image,
    StyleSheet,
    Text,
    View,
    Component,
   } = React;
```

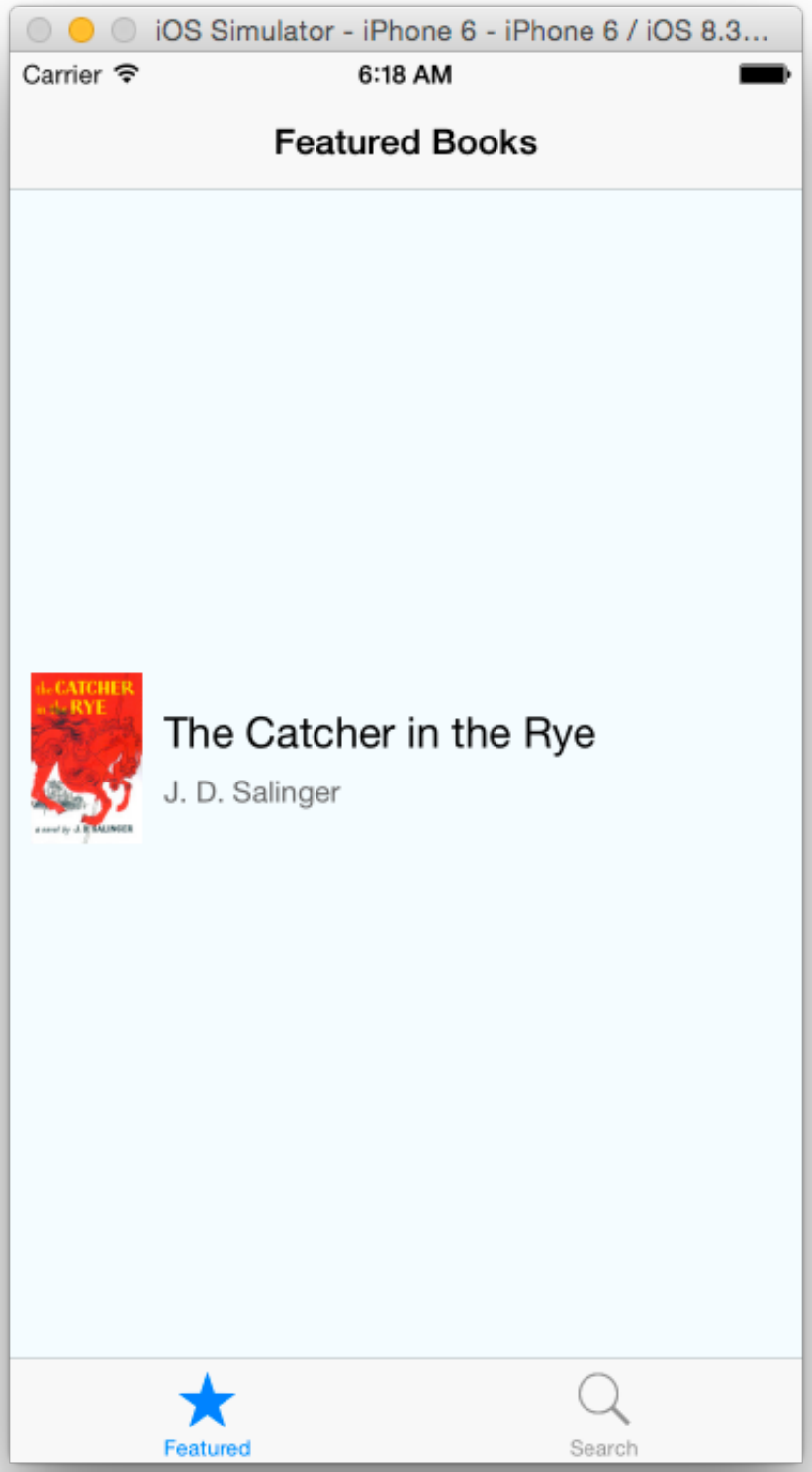
Add the following styles.

```
var styles = StyleSheet.create({
    container: {
        flex: 1,
        flexDirection: 'row',
        justifyContent: 'center',
        alignItems: 'center',
        backgroundColor: '#F5FCFF',
        padding: 10
    },
    thumbnail: {
        width: 53,
        height: 81,
        marginRight: 10
    },
    rightContainer: {
        flex: 1
    },
    title: {
        fontSize: 20,
        marginBottom: 8
    },
    author: {
        color: '#656565'
    }
});
```

Then modify the class as shown.

```
class BookList extends Component {
    render() {
    var book = FAKE_BOOK_DATA[0];
        return (
            <View style={styles.container}>
                <Image source={{uri: book.volumeInfo.imageLinks.thumbnail}}
                            style={styles.thumbnail} />
                <View style={styles.rightContainer}>
                    <Text style={styles.title}>{book.volumeInfo.title}</Text
                    <Text style={styles.author}>{book.volumeInfo.authors}</T
                </View>
            </View>
        );
    }
}
```

Reload the app and you should have the following.



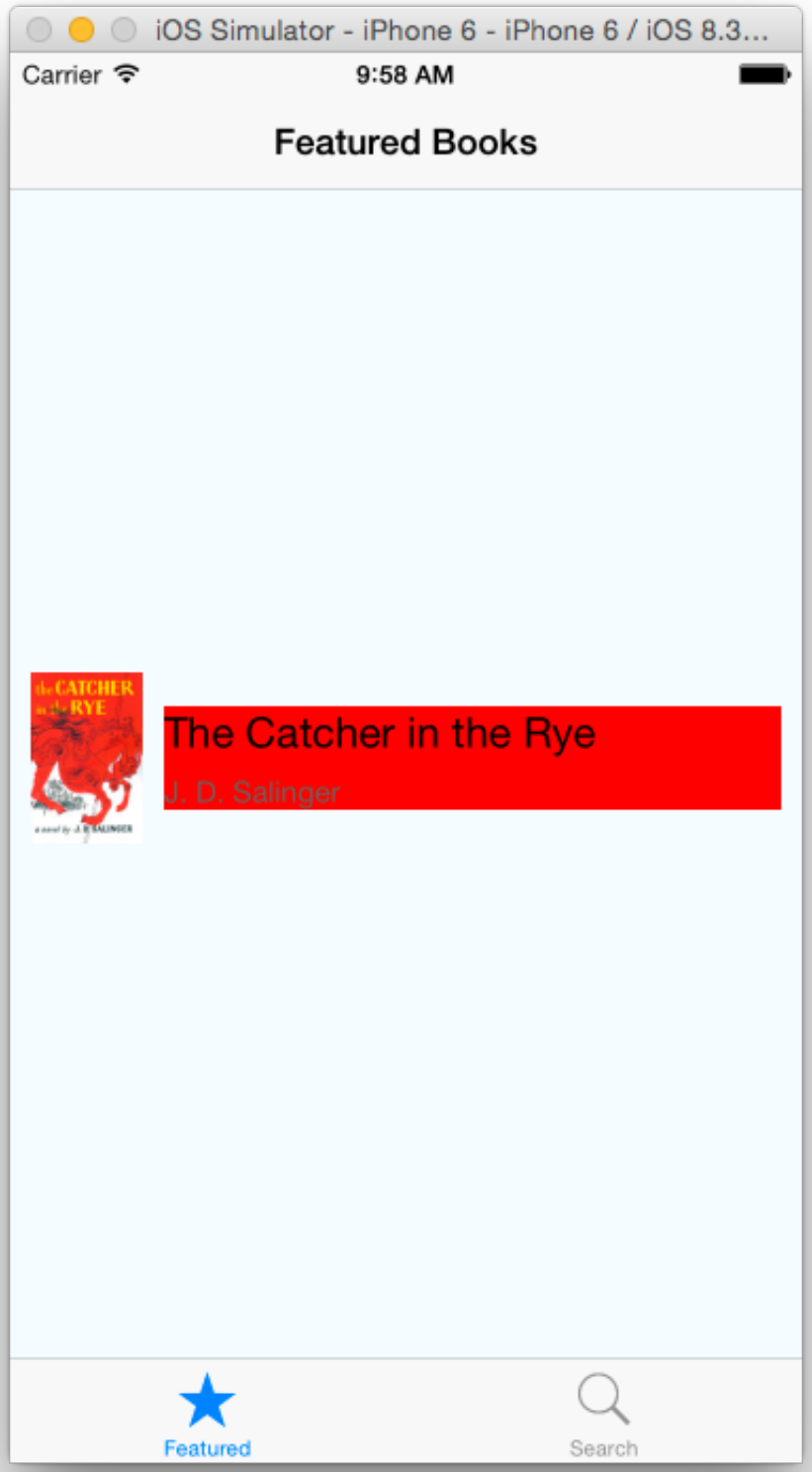
In the above code, we created a JSON object that is similar to the object we'll get from the A created properties and values for the object for a single book. In the class file, we use the fa only get the first element and use this to populate our view. We use the Image component image onto the view. Note that we set its width and height in the stylesheet. If you don't sp image size in the stylesheet, it will not appear in the view.

We specify a styling of *flexDirection: 'row'* for *container*. This will make the children of the with that style to be laid out horizontally rather than vertically which is the default. Note h wrap components inside other components. In the above there is the main container view children – an Image and a View. The View has has two children of its own – two Text comp

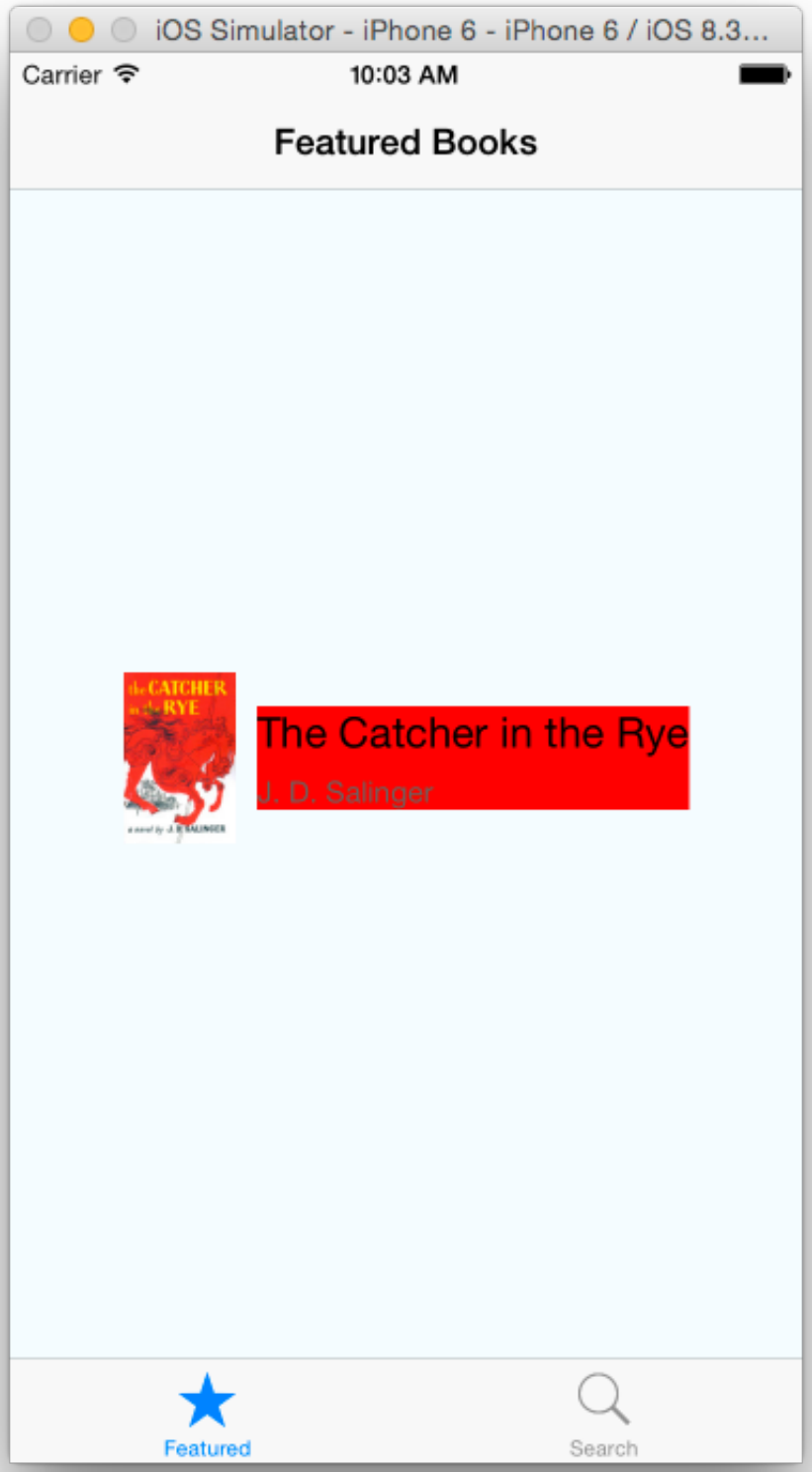
The Image component is laid out first and then the View (rightContainer) is placed horizon to it. We specify a style of *flex: 1* for the *rightContainer*. This makes that view occupy the re space not occupied by the image. If you want to see the effects of the flex styling then add following to *rightContainer*.

```
backgroundColor: 'red'
```

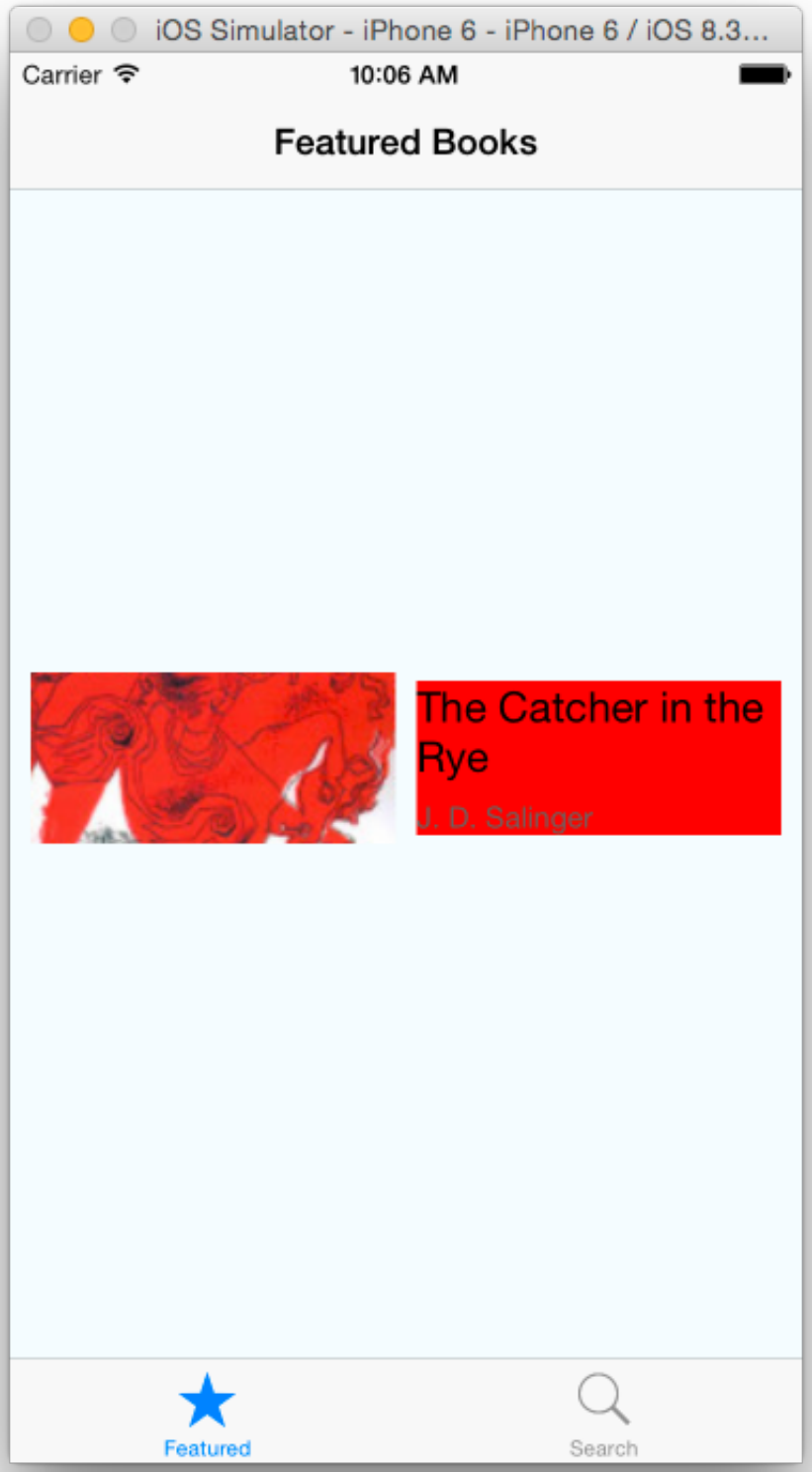
Reload the app and you will see the space occupied by the component with *rightContainer* takes up the entire space not occupied by its sibling. It doesn't stretch to the end of the scr because the outer container has some padding set on it and the image has a margin set to



Remove *flex: 1* from *rightContainer* and reload the app. Now the component only occupies space to fit its content.



If you put a style of *flex: 2* for both *thumbnail* and rightContainer they will occupy the same space as their widths will have a ratio of 2:2 (or 1:1). You can specify whatever value you want ratio that will be taken into account.



You can try out different ratios to get the styling you like. For the tutorial, we'll continue w had before the step to add a red background to *rightContainer*.

Adding a ListView

React Native has a component called ListView which displays scrollable rows of data – basi

table view in iOS terms.

To get started, modify the destructuring statement as shown to include more components use.

```
var {
    Image,
    StyleSheet,
    Text,
    View,
    Component,
    ListView,
    TouchableHighlight
  } = React;
```

Add the following style to the style sheet.

```
separator: {
      height: 1,
      backgroundColor: '#dddddd'
  }
```

Add the following constructor to the BookList class.

```
constructor(props) {
      super(props);
      this.state = {
          dataSource: new ListView.DataSource({
              rowHasChanged: (row1, row2) => row1 !== row2
          })
      };
  }
```

Then add the following function.

```
componentDidMount() {
    var books = FAKE_BOOK_DATA;
    this.setState({
        dataSource: this.state.dataSource.cloneWithRows(books)
    });
  }
```

In the constructor, we create a *ListView.DataSource* object and assign it to the *dataSource* The DataSource is an interface that ListView uses to determine which rows have changed course of updates to the UI. We provide a function that compares the identity of a pair of r

this is used to determine the changes in a list of data.

componentDidMount() is called when the component is loaded/mounted onto the UI view function is called, we set the *datasource* property with data from our data object.

Modify the *render()* function as shown.

```
render() {
    return (
        <ListView
            dataSource={this.state.dataSource}
            renderRow={this.renderBook.bind(this)}
            style={styles.listView}
            />
    );
}
```

Then add the following function to the BookList class.

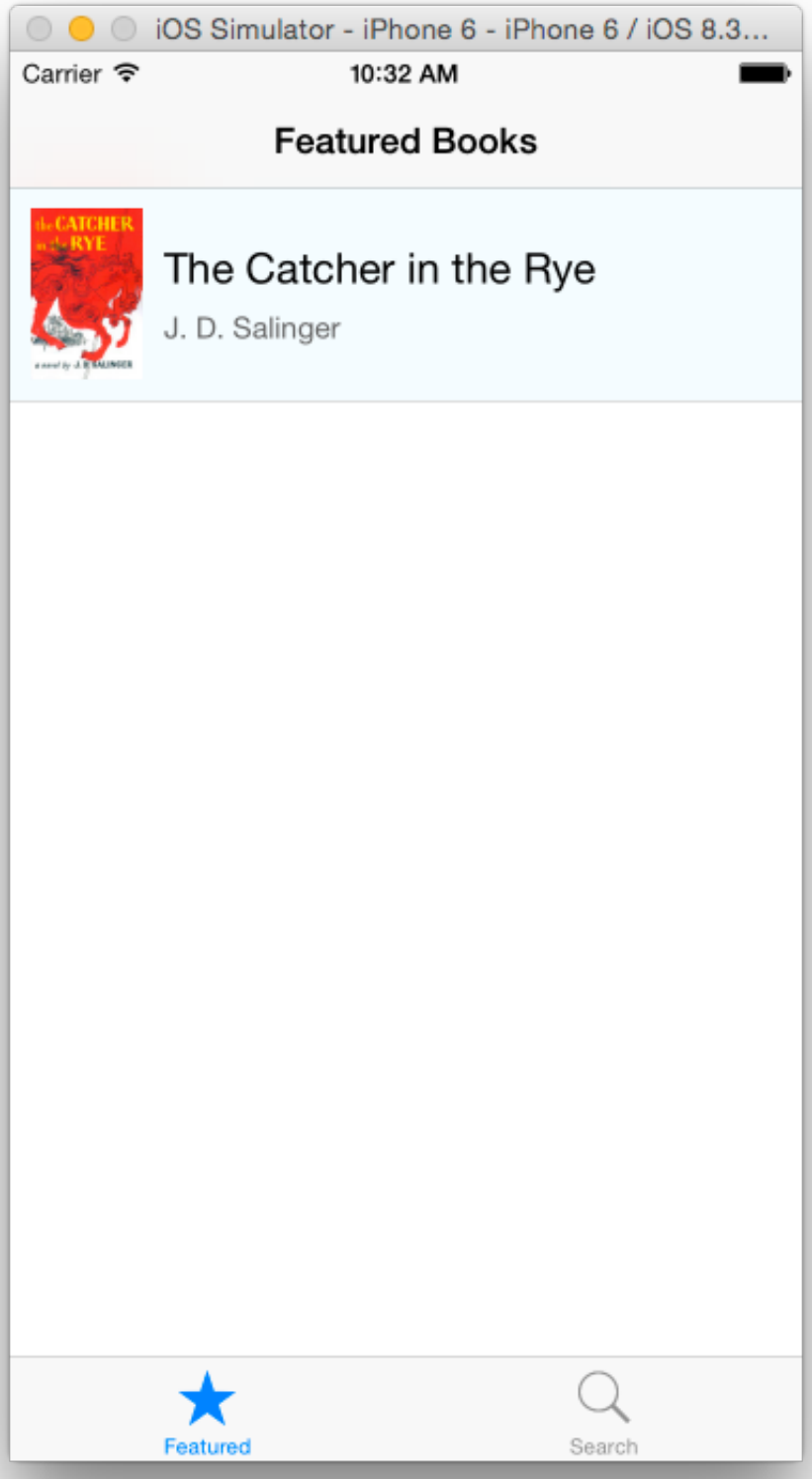
```
renderBook(book) {
       return (
            <TouchableHighlight>
                <View>
                    <View style={styles.container}>
                        <Image
                            source={{uri: book.volumeInfo.imageLinks.thumbna
                            style={styles.thumbnail} />
                        <View style={styles.rightContainer}>
                            <Text style={styles.title}>{book.volumeInfo.titl
                            <Text style={styles.author}>{book.volumeInfo.aut
                        </View>
                    </View>
                    <View style={styles.separator} />
                </View>
            </TouchableHighlight>
       );
   }
```

The above creates a ListView component in *render()*. Here it's *datasource* attribute is set to of the *dataSource* property we defined earlier and the function *renderBook()* is called to re rows of the ListView.

In *renderBook()* we use the TouchableHighlight component. This is a wrapper for making v respond properly to touches. On press down, the opacity of the wrapped view is decreased allows the underlay color to show through, darkening or tinting the view. With this, if you

on a ListView row, you will see the highlight color, just as what we are used to when we sel view cell. We add an empty View component at the bottom of the row with a styling of sep With styling, this view will simply be a grey horizontal line that will act like a partition betw row.

Reload the app and you should see the table view with only one cell.



Now to load real data into the app.

Remove the FAKE_BOOK_DATA variable from the file and instead add the following. This i we will load data from.

```
var REQUEST_URL = 'https://www.googleapis.com/books/v1/volumes?q=subject:fic
```

Modify the destructuring statement.

```
var {
    Image,
    StyleSheet,
    Text,
    View,
    Component,
    ListView,
    TouchableHighlight,
    ActivityIndicatorIOS
  } = React;
```

Add the following styles.

```
listView: {
        backgroundColor: '#F5FCFF'
  },
  loading: {
        flex: 1,
        alignItems: 'center',
        justifyContent: 'center'
  }
```

Modify the constructor as shown. We add another property to the component's state objec this to determine whether the view is loading or not.

```
constructor(props) {
        super(props);
        this.state = {
            isLoading: true,
            dataSource: new ListView.DataSource({
                rowHasChanged: (row1, row2) => row1 !== row2
            })
        };
  }
```

Modify the *componentDidMount()* function as shown and add the *fetchData()* function belc

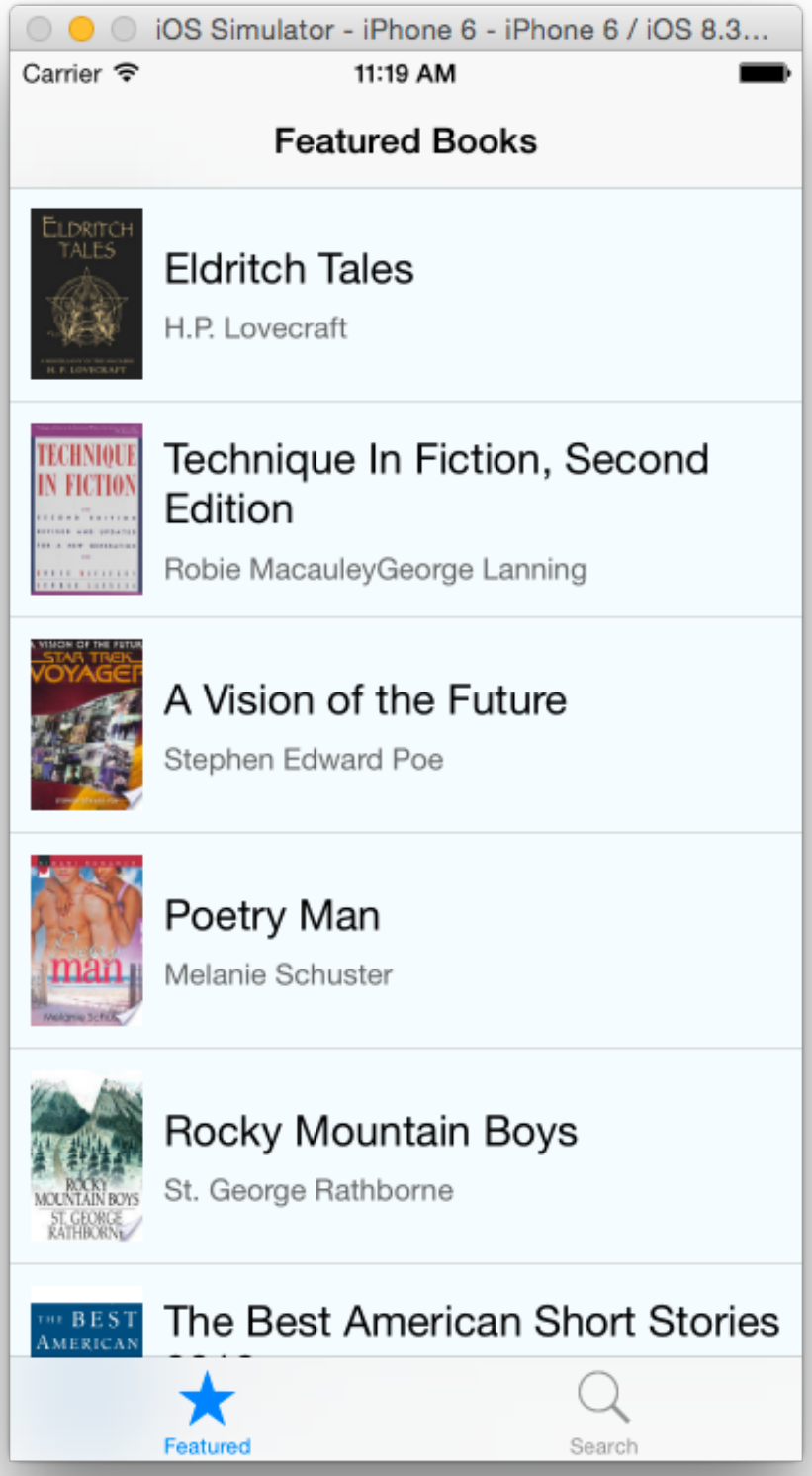
fetchData() makes the call to the Google books API and sets the dataSource property with t gets from the response. It also sets isLoading to true.

```
componentDidMount() {
        this.fetchData();
    }

    fetchData() {
        fetch(REQUEST_URL)
        .then((response) => response.json())
        .then((responseData) => {
            this.setState({
                dataSource: this.state.dataSource.cloneWithRows(responseData.
                isLoading: false
            });
        })
        .done();
    }
```

Modify *render()* as shown and add the *renderLoadingView()* function shown below. We add *isLoading* and if it's set to true, we return the view that is returned by the *renderLoadingVi* will be a view that shows an activity indicator (a spinner) with the text 'Loading books…'. W loading is done, you should see a list of books in a table.

```
render() {
        if (this.state.isLoading) {
            return this.renderLoadingView();
        }

        return (
             <ListView
                 dataSource={this.state.dataSource}
                 renderRow={this.renderBook.bind(this)}
                 style={styles.listView}
                 />
         );
}

renderLoadingView() {
    return (
        <View style={styles.loading}>
            <ActivityIndicatorIOS
                size='large'/>
            <Text>
                Loading books...
            </Text>
        </View>
    );
}
```

Reload the app and you should have something similar to what's shown below.



Adding the Detail View

If you tap on a cell in the table, the cell will be highlighted but nothing will happen. We'll ad view that will show us details of the book we select.

Add a file to the project and name it BookDetail.js. Paste the following in the file.

```
'use strict';

var React = require('react-native');

var {
    StyleSheet,
    Text,
    View,
    Component,
    Image
   } = React;

var styles = StyleSheet.create({
    container: {
        marginTop: 75,
        alignItems: 'center'
    },
    image: {
        width: 107,
        height: 165,
        padding: 10
    },
    description: {
        padding: 10,
        fontSize: 15,
        color: '#656565'
    }
});

class BookDetail extends Component {
    render() {
        var book = this.props.book;
        var imageURI = (typeof book.volumeInfo.imageLinks !== 'undefined') ?
        var description = (typeof book.volumeInfo.description !== 'undefined
        return (
            <View style={styles.container}>
                <Image style={styles.image} source={{uri: imageURI}} />
                <Text style={styles.description}>{description}</Text>
            </View>
        );
    }
}

module.exports = BookDetail;
```

Most of what is in the above code we've been through it so I won't go over it all. What we h is the use of the props property to extract data from. We will pass data into this class by se props property. In the above, we get this data and populate the view with the data.

Notice we set a top margin on container. If you don't then the view will start from the top c screen and this might result in some elements being hidden by the navigation bar.

In BookList.js add the following to the file.

```
var BookDetail = require('./BookDetail');
```

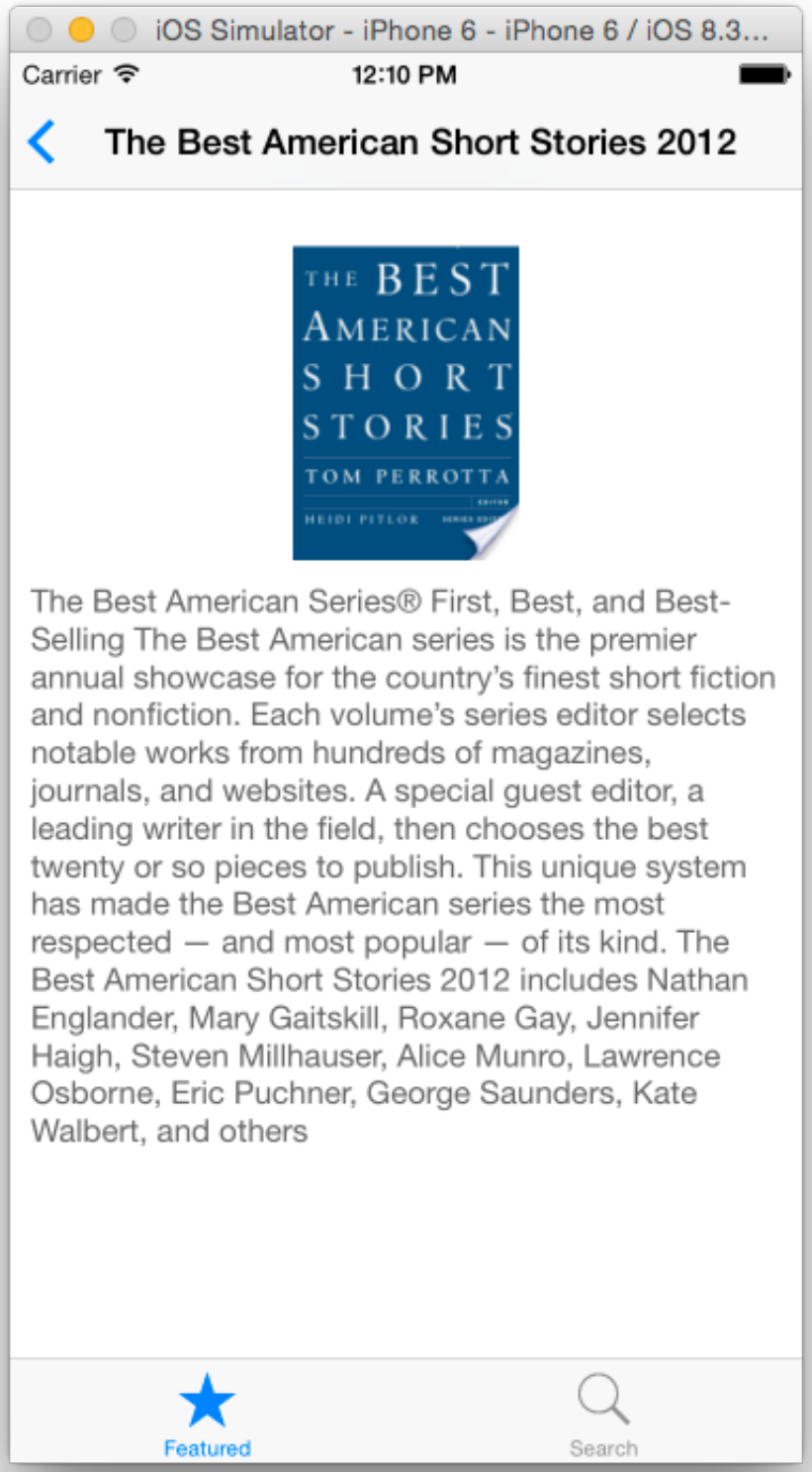
Modify the TouchableHighlight in the *render()* function of the BookList class as shown.

```
<TouchableHighlight onPress={() => this.showBookDetail(book)}  underlayColor
```

The above specifies a callback function that will be called when the row it pressed. Paste th function into the class. This will push the BookDetail view onto the navigation stack and se seen on the navigation bar. It passes the book object that corresponds to that particular ro BookDetail class.

```
showBookDetail(book) {
       this.props.navigator.push({
           title: book.volumeInfo.title,
           component: BookDetail,
           passProps: {book}
       });
   }
```

Reload the app and you should now be able to see details of a selected book.



Searching

Now that we have completed the Master-Detail view of the Featured tab, we'll work on the to enable the user to query the API for books of their choice.

Open SearchBooks.js and modify it as shown.

```
'use strict';

var React = require('react-native');
var SearchResults = require('./SearchResults');
var {
    StyleSheet,
    View,
    Text,
    Component,
    TextInput,
    TouchableHighlight,
    ActivityIndicatorIOS
    } = React;

var styles = StyleSheet.create({
    container: {
        marginTop: 65,
        padding: 10
    },
    searchInput: {
        height: 36,
        marginTop: 10,
        marginBottom: 10,
        fontSize: 18,
        borderWidth: 1,
        flex: 1,
        borderRadius: 4,
        padding: 5
    },
    button: {
        height: 36,
        backgroundColor: '#f39c12',
        borderRadius: 8,
        justifyContent: 'center',
        marginTop: 15
    },
    buttonText: {
        fontSize: 18,
        color: 'white',
        alignSelf: 'center'
    },
    instructions: {
        fontSize: 18,
        alignSelf: 'center',
        marginBottom: 15
    },
    fieldLabel: {
        fontSize: 15,
        marginTop: 15
    },
    errorMessage: {
        fontSize: 15,
        alignSelf: 'center',
        marginTop: 15,
```

```
        color: 'red'
    }
});

class SearchBooks extends Component {

    constructor(props) {
        super(props);
        this.state = {
            bookAuthor: '',
            bookTitle: '',
            isLoading: false,
            errorMessage: ''
        };
    }


    render() {
        var spinner = this.state.isLoading ?
            ( <ActivityIndicatorIOS
                hidden='true'
                size='large'/> ) :
            ( <View/>);
        return (
            <View style={styles.container}>
                <Text style={styles.instructions}>Search by book title and/
                <View>
                    <Text style={styles.fieldLabel}>Book Title:</Text>
                    <TextInput style={styles.searchInput} onChange={this.bo
                </View>
                <View>
                    <Text style={styles.fieldLabel}>Author:</Text>
                    <TextInput style={styles.searchInput} onChange={this.bo
                </View>
                <TouchableHighlight style={styles.button}
                                    underlayColor='#f1c40f'
                                    onPress={this.searchBooks.bind(this)}>
                    <Text style={styles.buttonText}>Search</Text>
                </TouchableHighlight>
                {spinner}
                <Text style={styles.errorMessage}>{this.state.errorMessage}
            </View>
        );
    }

    bookTitleInput(event) {
        this.setState({ bookTitle: event.nativeEvent.text });
    }

    bookAuthorInput(event) {
        this.setState({ bookAuthor: event.nativeEvent.text });
    }

    searchBooks() {
```

```
109         this.fetchData();
110     }
111
112     fetchData() {
113
114         this.setState({ isLoading: true });
115
116         var baseURL = 'https://www.googleapis.com/books/v1/volumes?q=';
117         if (this.state.bookAuthor !== '') {
118             baseURL += encodeURIComponent('inauthor:' + this.state.bookAuth
119         }
120         if (this.state.bookTitle !== '') {
121             baseURL += (this.state.bookAuthor === '') ? encodeURIComponent(
122         }
123
124         console.log('URL: >>> ' + baseURL);
125         fetch(baseURL)
126             .then((response) => response.json())
127             .then((responseData) => {
128                 this.setState({ isLoading: false});
129                 if (responseData.items) {
130
131                     this.props.navigator.push({
132                         title: 'Search Results',
133                         component: SearchResults,
134                         passProps: {books: responseData.items}
135                     });
136                 } else {
137                     this.setState({ errorMessage: 'No results found'});
138                 }
139             })
140             .catch(error =>
141                 this.setState({
142                     isLoading: false,
143                     errorMessage: error
144                 }))
145             .done();
146     }
147
148 }
149
150 module.exports = SearchBooks;
```

In the above, we set some properties in the constructor: *bookAuthor*, *bookTitle*, *isLoading* *errorMessage*. We'll see how these are used shortly.

In the *render()* method, we check if *isLoading* is true and create an activity indicator if it is, we create an empty view. This will be used later. We then create the search form that will l insert queries. TextInput is used for input. We specify a callback function for each TextInp component that will be called when the value of the component changes i.e. when the user

some text. When called, the callback functions *bookTitleInput()* and *bookAuthorInput()* will *bookAuthor* and *bookTitle* state properties with the data entered by the user. *searchBooks* when the user presses the Search button. Note that React Native does not have a button c Instead we use TouchableHighlight and wrap it around Text which we then style to look lik When the Search button is pressed, a URL is constructed depending on the data entered. A search by title, author or both title and author. If results are returned, SearchResults will b on to the navigation stack otherwise an error message will be displayed. We also pass the r data to the SearchResults class.

Create a file named SearchResults.js and paste the following in.

```
'use strict';

var React = require('react-native');
var BookDetail = require('./BookDetail');
var {
    StyleSheet,
    View,
    Text,
    Component,
    TouchableHighlight,
    Image,
    ListView
    } = React;

var styles = StyleSheet.create({
    container: {
        flex: 1,
        justifyContent: 'center',
        alignItems: 'center'
    },
    title: {
        fontSize: 20,
        marginBottom: 8
    },
    author: {
        color: '#656565'
    },
    separator: {
        height: 1,
        backgroundColor: '#dddddd'
    },
    listView: {
        backgroundColor: '#F5FCFF'
    },
    cellContainer: {
        flex: 1,
        flexDirection: 'row',
        justifyContent: 'center',
```

```
        alignItems: 'center',
        backgroundColor: '#F5FCFF',
        padding: 10
    },
    thumbnail: {
        width: 53,
        height: 81,
        marginRight: 10
    },
    rightContainer: {
        flex: 1
    }
});

class SearchResults extends Component {

    constructor(props) {
        super(props);

        var dataSource = new ListView.DataSource(
            {rowHasChanged: (row1, row2) => row1 !== row2});
        this.state = {
            dataSource: dataSource.cloneWithRows(this.props.books)
        };
    }

    render() {

        return (
            <ListView
                dataSource={this.state.dataSource}
                renderRow={this.renderBook.bind(this)}
                style={styles.listView}
                />
        );
    }

    renderBook(book) {
        var imageURI = (typeof book.volumeInfo.imageLinks !== 'undefined')

        return (
            <TouchableHighlight onPress={() => this.showBookDetail(book)}
                                underlayColor='#dddddd'>
                <View>
                    <View style={styles.cellContainer}>
                        <Image
                            source={{uri: imageURI}}
                            style={styles.thumbnail} />
                        <View style={styles.rightContainer}>
                            <Text style={styles.title}>{book.volumeInfo.tit
                            <Text style={styles.author}>{book.volumeInfo.au
                        </View>
                    </View>
                    <View style={styles.separator} />
```

```
                </View>
            </TouchableHighlight>
        );
    }

    showBookDetail(book) {

        this.props.navigator.push({
            title: book.volumeInfo.title,
            component: BookDetail,
            passProps: {book}
        });
    }

}

module.exports = SearchResults;
```

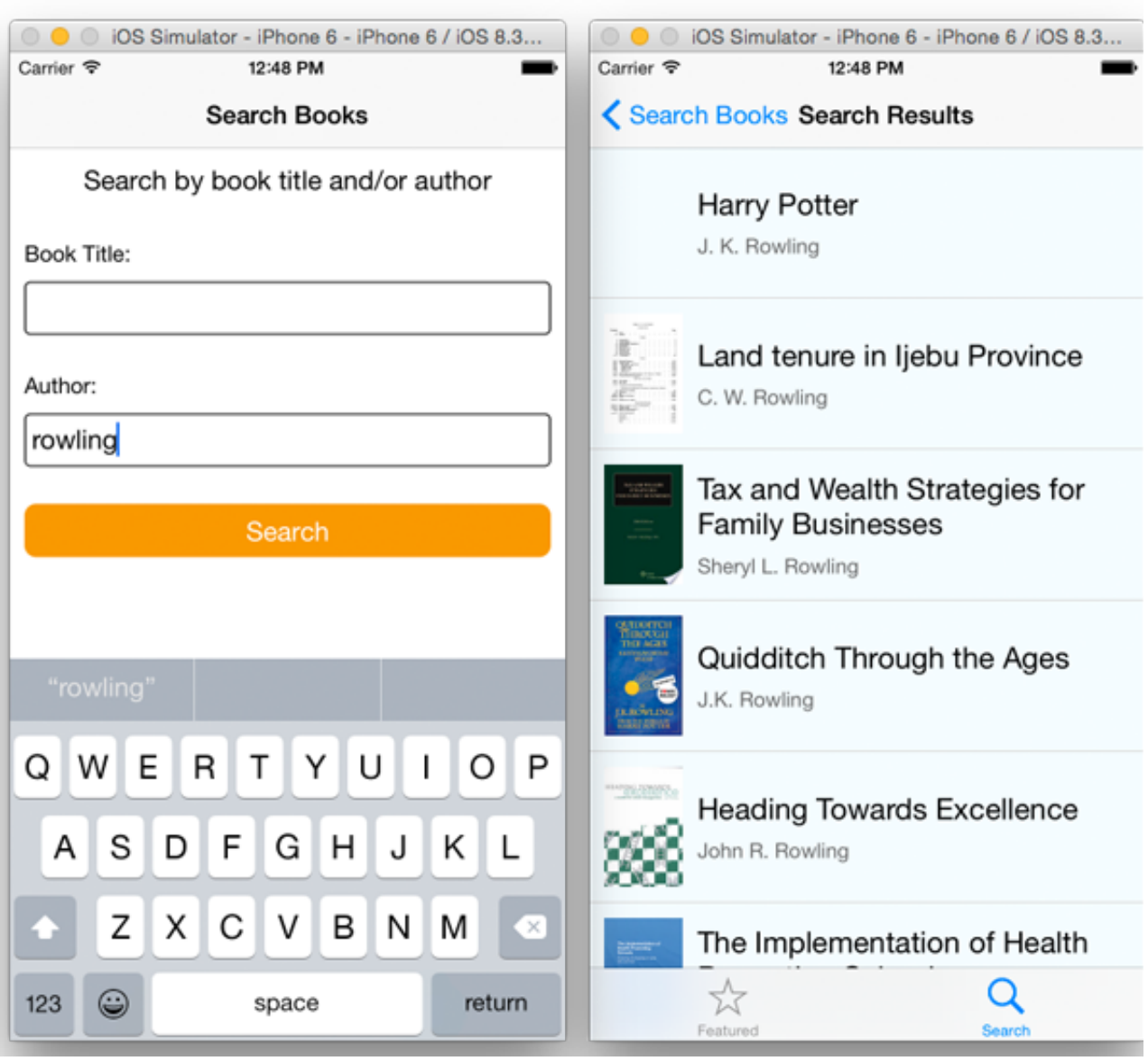
We've gone through a lot of what we use in the code above so I won't get into every detail. gets the data passed into the class via the props property and creates a ListView populated data.

One thing I noticed in the API is that when you search by author, some results will not be b but data on the author themselves. This means that for some rows *book.volumeInfo.imageLinks.thumbnail* and *book.volumeInfo.description* will have a value undefined. So we make a check for this and show an empty image view if there is no image didn't and the app tried to load an image that wasn't there, it would crash.

We use the same BookDetail component we created earlier to show the details of each boo should put the above check for missing data incase we try to load the BookDetail view with data. Open BookDetail.js and modify the *render()* function as shown. This checks if the data has an image and description before populating the view with this data. If we try to view a without a description or image, the respective area will be blank. You might want to put in message to the user, but we'll leave it as it is here.

```
render() {
    var book = this.props.book;
    var imageURI = (typeof book.volumeInfo.imageLinks !== 'undefined') ? boo
    var description = (typeof book.volumeInfo.description !== 'undefined') ?
    return (
        <View style={styles.container}>
            <Image style={styles.image} source={{uri: imageURI}} />
            <Text style={styles.description}>{description}</Text>
        </View>
    );
}
```

Reload the app and you should be able to search for a book.



Conclusion

Although it's still a work-in-progress, React Native looks promising as another option for b mobile applications. It opens the door to web developers looking to dip their toes in the mo development waters; even for mobile developers, it could offer a way to streamline their de workflow.

As the project grows, it will be interesting to see where it goes and the apps (iOS and Andro perhaps other platforms as well) that will be built with React Native. In the meantime, if yo convincing on whether web technologies can be used to achieve truly native experiences, y check out these apps that were built by react native: Facebook Ads Manager (completely w React Native) and Facebook Groups (a hybrid app created with React Native and Objective-

"Learn once, write anywhere". This alone might make it worth learning how to use the fram

You can download the completed project here.

To learn more about React Native, you could watch the following videos and also read the documentation.

- Introducing React Native.
- A Deep Dive into React Native.
- React Native and Relay: Bringing Modern Web Techniques to Mobile.

For reference, you can download the Xcode project here.

What do you think about this tutorial and React Native? Leave me comment and share you

Note: This tutorial is also available in Chinese. We're going to support other languages soo want to join our translation team, please contact us.

You May Like These:


iOS Programming Tutorial: Create a Simple Table View App

Creating Hello World App Using Xcode 5 and Interface Builder

Creating Hello World App in Swift Using Xcode 6

Building a Custom Pull To Refresh Control for Your iOS Apps

filed under: course, intermediate, tutorials + tagged with: facebook, ios programming, javascript, react r

About Joyce Echessa

I am a web developer who also does mobile development from time to time. I have also been writing o some time now, a process I find enjoyable and challenging, for, at times, it is when you have to explain to someone, that you find there is still a lot to learn. You can find me on Twitter.

see more

Reply • Share ›

Neeraj Khanna • a month ago

Joyce, great writeup, really helpful. Quick question - can we control the iOS native app throu JS without app update, i.e. submitting to app store? it will be beneficial that we are avoiding from the picture but controlling the app features and UI without going through app store an customers to update their native apps. Please let's know, if it is possible through React Native

Reply • Share ›

Ari Corner • 2 months ago

Thank you so much! This was super helpful! Quick question about BookDetail: if the book's d long, it will get cut off at the bottom, so is there a way to scroll/view the entire description in (For example, Focus by Arthur Miller cuts off mid sentence at the bottom of the view.) Thank

Reply • Share ›

Greg Thompson Jr. • 3 months ago

So this is for people who are already on iOS, right? Not people who are using Linux (particul in my case)? You start out the tutorial using Brew, and you mention something Xcode-specif after.

Reply • Share ›

Joyce Echessa ➚ Greg Thompson Jr. • 3 months ago

Yes, you need a Mac to run this. When you create a project with the React Native CLI generates an Xcode project, so you'll need Xcode.

Reply • Share ›

Evgeniy D • 3 months ago

Very useful tutorial. Everything work. Unfortunately there is no proper IDE for such a syntax

Reply • Share ›

Joyce Echessa ➚ Evgeniy D • 3 months ago

Thanks. You're right about the IDE. But getting one with JSX support helps a lot. The this http://nuclide.io/ from Facebook which is an IDE for React and React Native, bu

this http://nuclide.io/ from Facebook which is an IDE for React and React Native, bt under development.

Reply • Share ›

Leo Chen • 3 months ago

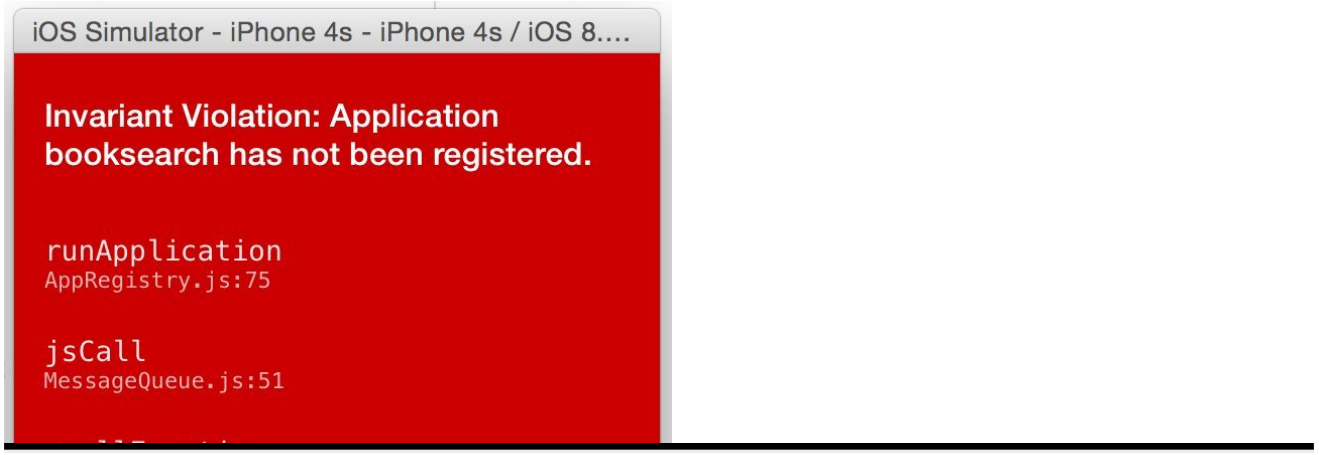
Nice tutorials,

but I am having something like this

http://imgur.com/rldMKeS

after the line : "In index.ios.js delete everything and paste in the following."

do you know why?



see more

Reply • Share ›

Leo Chen ➚ Leo Chen • 3 months ago

I solve it after change the app name from

AppRegistry.registerComponent('BookSearch', () => BookSearch);

to

AppRegistry.registerComponent('booksearch', () => BookSearch);

once again this tutorial is great, just minor typo needs to be fixed.
thank you.

1 Reply • Share ›

iCreative Kid • 3 months ago

Getting following error while trying to install the React Native CLI tool

module.js:340

throw err;

^

Error: Cannot find module 'child-process-close'

Error: Cannot find module 'child_process_close'

at Function.Module._resolveFilename (module.js:338:15)

at Function.Module._load (module.js:280:25)

at Module.require (module.js:364:17)

at require (module.js:380:17)

at /usr/local/lib/node_modules/npm/lib/npm.js:15:1

at Object.<anonymous> (/usr/local/lib/node_modules/npm/lib/npm.js:472:3)

at Module._compile (module.js:456:26)

at Object.Module._extensions..js (module.js:474:10)

at Module.load (module.js:356:32)

at Function.Module._load (module.js:312:12)

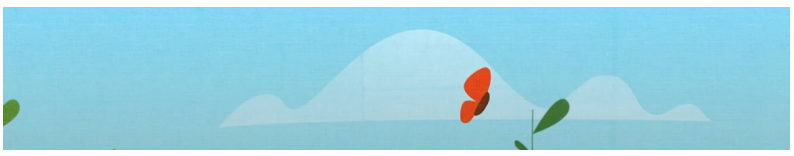
• Reply • Share ›

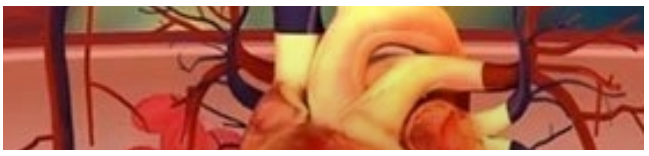
Joyce Echessa ➚ iCreative Kid • 3 months ago

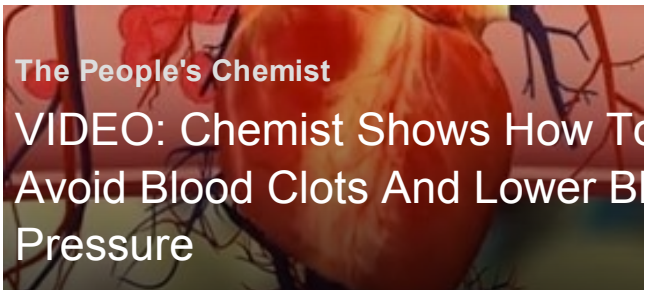
Could you try uninstalling node then installing it again. Look at this post http://stackoverflow.com/quest...
Also look at this http://stackoverflow.com/quest...

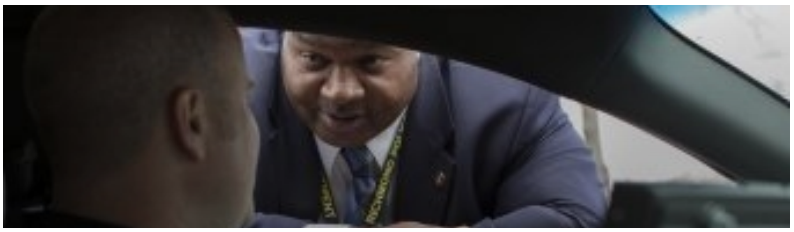
I would recommend you use this to completely remove node from your machine http://stackoverflow.com/quest...
When I was experimenting with react native back when it first came out, I had a proj fine; I then updated Yosemite to the recent version 10.10.3 and the same project woul tried reinstalling node with Homebrew, but still that didn't help (it seems like 'brew ur still leaves some stuff on your computer). It was only until I manually removed everyt with the instructions from the above link and installed it again that the project started again.
So it could be worth a try to try reinstalling node. Hope that helps.

• Reply • Share ›

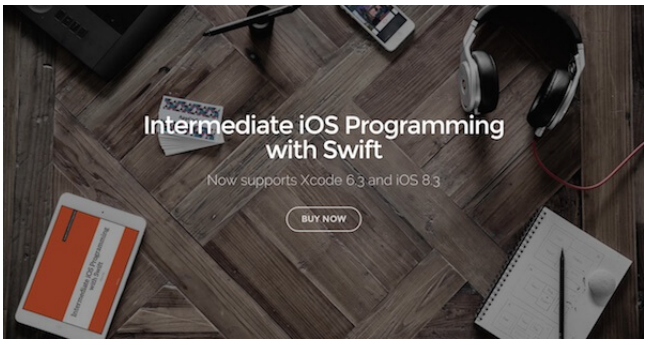








Our New Book



Connect With Us

About AppCoda

About

Tutorials

Course

Book
Beginning iOS 8 Programming with Swift

Intermediate iOS 8 Programming with Swift

App Template

Marketplace

Contact

Get Our New Book

Connect With Us