WELCOME TO scope

Welcome to our third React Native meeting! This is the last React Native curriculum, and we'll be talking more about functions, and a bit about APIs.

QR Code for Attendance

</>

Please fill out this attendance form! You and your Scope Cup team will be awarded points for attendance beginning today.

Announcements

Dues:

- Venmo $30 to @scopeusc

This week:

- Thursday: casual curriculum and then hang out, don't worry if you have the midterm/can't make it, Slack us beforehand
- Join the discord before the meeting!
- Office hours at 6:30pm Thurs to go over today's lesson

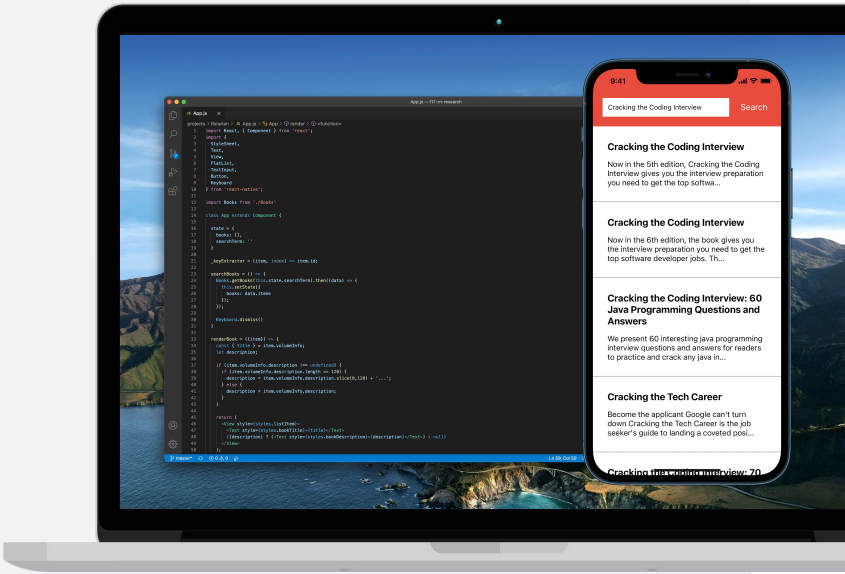
Today's Lesson

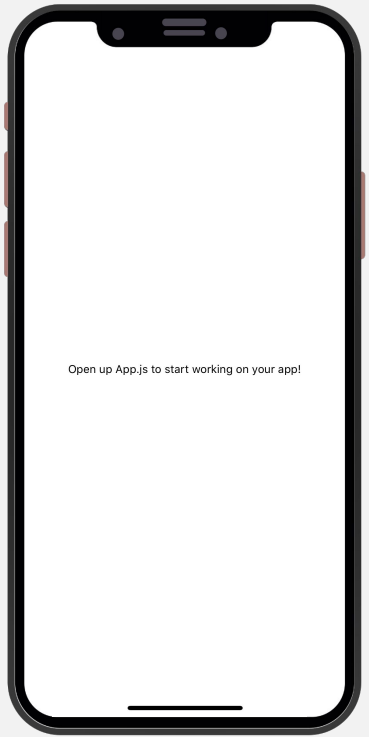
What you'll build:

A React Native app that integrates with the Google Books API

What you'll learn:

How to implement API calls, and how to process and display the resulting data in a React Native app





Create a new project

Open a terminal, navigate to your Scope folder and run the following commands:

```
1.  expo init Librarian
2.  cd Librarian
3.  npm start
```

Setting Up

First, we need to take several steps to set up our App.js before we start adding UI to the view. These are super similar to the previous week's.

1

Export a class, not a function

Replace the line

```
export default function App() {
```

With:

```
export default class App extends Component {
```

This allows us to place functions within the App class, rather than App being a singular function.

2

Import Component

In the

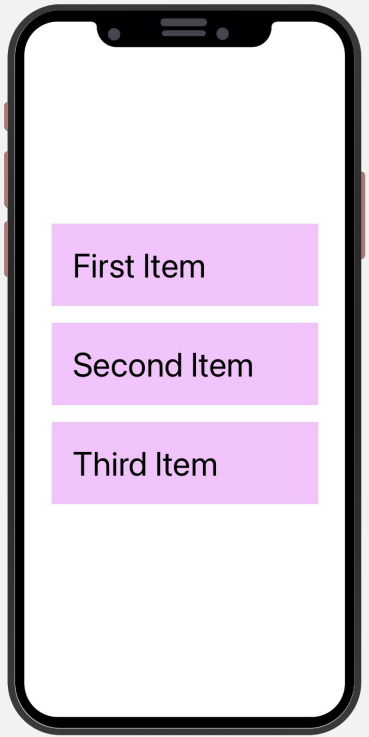
```
import React from 'react';
```

line, add `React, { Component } from 'react';` so that we are importing Component from React. A component is a type of class that provides additional features.

3

Create the render() method

Surround the `return ( );` method with the render method. Add `render(){` on line 6 before the 'return', and add a closing curly brace after return's closing parenthesis on line 11 (before the semicolon). The return method is required to implement the Component.



FlatList

Today's new UI element. This allows you to display a list of items stored in an array.

```
<FlatList
   data={/* some data in an array to display */}
   renderItem={/* a function to return UI components */}
   keyExtractor={/* a function to get a unique key for
      each element in the list */}
   style={/* how you want the list to be displayed */}
/>
```

Add elements

Now we can add the TextInput field, our first UI element today, as well as state.

1

Import the UI components

Add

```
FlatList, TextInput, Button, Keyboard
```

To the list of React element imports at the top of App.js, so the line reads::

```
import { StyleSheet, Text, View, FlatList, TextInput, Button, Keyboard } from 'react-native';
```

2

Add a TextInput

```
<TextInput placeholder="Search for a book..."
onChangeText={onChangeText={(text) => {
                          this.setState({
                            searchTerm: text
                          });
                        }
onSubmitEditing={this.searchBooks}
/>
```

3

Add a State

We need a state for the App class so we can track the search term and books returned from Google Books, since they are variables that will update the view. Add the following to the App

```
constructor(){
   super()
   this.state = {
     books: [],
     searchTerm: ''
   }
 };
```

Add styles

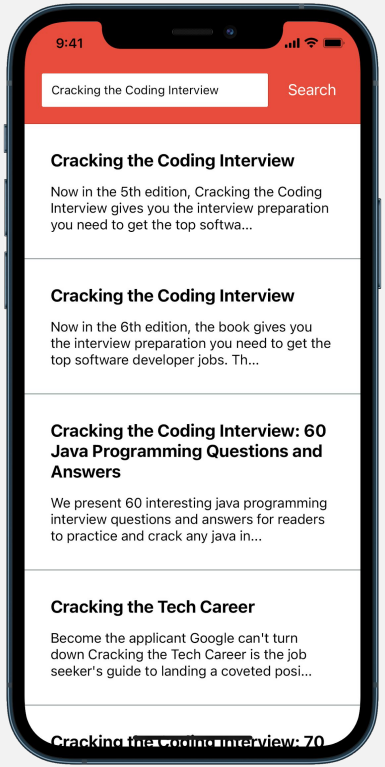
We learned new ways to do this last week, so you can pick your favorite or stick to the basics, and declare these as a const at the bottom of your App.js

1

2

3

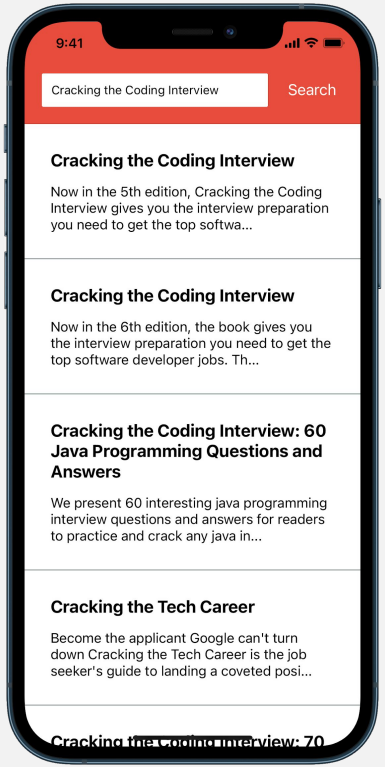
```
const styles = StyleSheet.create({
 container: {
   flex: 1,
   backgroundColor: "#fff",
   alignItems: "center",
   justifyContent: "center",
 },
 listItem: {
   // TODO
 },
 bookTitle: {
   // TODO
 },
 bookDescription: {
   // TODO
 }
});
```



API calls

How do you make HTTP requests and parse the data in React?

```
return fetch('https://www.googleapis.com/books/v1/volumes?q='
+ query).then((res) => res.json());
```



API calls: example

Let's say we wanted to search for Daniel Keyes' Flowers for Algernon, with keyword "flower" and author "Keyes."

This is how we would construct the request.

```
return
fetch('https://www.googleapis.com/books/v1/volumes?q=flowers+
inauthor:keyes').then((res) => res.json());
```

JSON Format

Format data will be returned in.

Similar to a dictionary in Python/C++.

We can use each of these items as a struct, and access elements directly.


```
{
 "kind": "books#volumes",
 "items": [
  {
   "kind": "books#volume",
   "id": "_ojXNuzgHRcC",
   "etag": "OTD2tB19qn4",
   "selfLink": "https://www.googleapis.com/books/v1/volumes/_ojXNuzgHRcC",
   "volumeInfo": {
    "title": "Flowers",
    "authors": [
     "Vijaya Khisty Bodach"
    ],
   ...
  },
  {
   "kind": "books#volume",
   "id": "RJxWIQOvoZUC",
   "etag": "NsxMT6kCCVs",
   "selfLink": "https://www.googleapis.com/books/v1/volumes/RJxWIQOvoZUC",
   "volumeInfo": {
    "title": "Flowers",
    "authors": [
     "Gail Saunders-Smith"
    ],
    ...
  },
  {
   "kind": "books#volume",
   "id": "zaRoX10_UsMC",
   "etag": "pm1sLMgKfMA",
   "selfLink": "https://www.googleapis.com/books/v1/volumes/zaRoX10_UsMC",
   "volumeInfo": {
    "title": "Flowers",
    "authors": [
     "Paul McEvoy"
    ],
    ...
  },
  "totalItems": 3
}
```

Functions!

getBooks()

- Query API by making an HTTP request with the appropriate parameters

searchBooks()

- Allow user to search. Call getBooks(), and store the list of books returned.

renderBook()

- Bonus! Check out the Google Books API, and see if you can return appropriately rendered tags

1

```
_keyExtractor = (item, index) => item.id;

getBooks = (query) => {
  // TODO
}

searchBooks = () => {
  // TODO

  Keyboard.dismiss()
}

 renderBook = ({item}) => {
  const { title } = item.volumeInfo;
  let description;

  if (item.volumeInfo.description !== undefined) {
    if (item.volumeInfo.description.length >= 120) {
      description = item.volumeInfo.description.slice(0,120) + '...';
    } else {
      description = item.volumeInfo.description;
    }
  }

  return (
    // BONUS TODO
  );
}
```

Functions!

Here's how we did it. Questions?

1

```
_keyExtractor = (item, index) => item.id;

getBooks = (query) => {
  return fetch('https://www.googleapis.com/books/v1/volumes?q=' + query)
          .then((res) => res.json());
}

searchBooks = () => {
  this.getBooks(this.state.searchTerm).then((data) => {
    this.setState({
      books: data.items
    });
  });

  Keyboard.dismiss()
}

 renderBook = ({item}) => {
  const { title } = item.volumeInfo;
  let description;

  if (item.volumeInfo.description !== undefined) {
    if (item.volumeInfo.description.length >= 120) {
      description = item.volumeInfo.description.slice(0,120) + '...';
    } else {
      description = item.volumeInfo.description;
    }
  }

  return (
    <View style={styles.listItem}>
      <Text style={styles.bookTitle}>{title}</Text>
      {(description) ? (<Text style= {styles.bookDescription}>
         {description}</Text>) : null}
    </View>
  );
}
```

A closer look

renderBook()

- When we call this, we will be calling it with a JSON element, item, which is how we can get each of these elements
- The ternary operator (?) is a shorthand way to write, if description is not blank, display in this style

```
renderBook = ({item}) => {
 const { title } = item.volumeInfo;
 let description;

 if (item.volumeInfo.description !== undefined) {
   if (item.volumeInfo.description.length >= 120) {
     description = item.volumeInfo.description.slice(0,120) + '...';
   } else {
     description = item.volumeInfo.description;
   }
 }

 return (
   <View style={styles.listItem}>
     <Text style={styles.bookTitle}>{title}</Text>
     {(description) ? (<Text style= {styles.bookDescription}>
       {description}</Text>) : null}
   </View>
 );
}
```

render()

Similar to what we've done before... try it!

```
render() {
   return (
     <View style={styles.container}>
       <TextInput
         onChangeText={ // TODO }
         value={ // TODO }
         style={styles.searchTextInput}
         onSubmitEditing={ // TODO }
         placeholder="Search for a book..."
       />
     </View>
   );
}
```

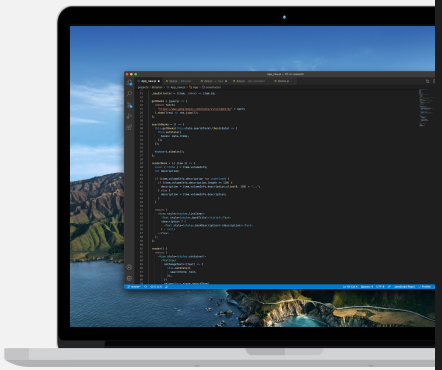
render()

Here's how we did it. Questions?

```
render() {
  return (
    <View style={styles.container}>
      <TextInput
        onChangeText={(text) => {
          this.setState({
            searchTerm: text,
          });
        }}
        value={this.state.searchTerm}
        style={styles.searchTextInput}
        onSubmitEditing={this.searchBooks}
        placeholder="Search for a book..."
      />
    </View>
  );
 }
}
```

Code

Your current code should read as shown:



```
import React, { Component } from "react";
import {
 StyleSheet,
 Text,
 View,
 FlatList,
 TextInput,
 Button,
 Keyboard,
} from "react-native";

export default class App extends Component {
 constructor() {
   super();
   this.state = {
     books: [],
     searchTerm: "",
   };
 }

 _keyExtractor = (item, index) => item.id;

 getBooks = (query) => {
   return fetch(

"https://www.googleapis.com/books/v1/volumes?q="+
query
   ).then((res) => res.json());
 };

 searchBooks = () => {

this.getBooks(this.state.searchTerm).then((data)
=> {
     this.setState({
       books: data.items,
     });
   });

   Keyboard.dismiss();
 };
```

```
 renderBook = ({ item }) => {
   const { title } = item.volumeInfo;
   let description;
   if (item.volumeInfo.description !== undefined) {
     if (item.volumeInfo.description.length >= 120) {
       description = item.volumeInfo.description.slice(0,
120) + "...";
     } else {
       description = item.volumeInfo.description;
     }
   }
   return (
     <View style={styles.listItem}>
       <Text style={styles.bookTitle}>{title}</Text>
       {description ? (
         <Text
style={styles.bookDescription}>{description}</Text>
       ) : null}
     </View>
   );
 };
 render() {
   return (
     <View style={styles.container}>
       <TextInput
         onChangeText={(text) => {
           this.setState({
             searchTerm: text,
           });
         }}
         value={this.state.searchTerm}
         style={styles.searchTextInput}
         onSubmitEditing={this.searchBooks}
         placeholder="Search for a book..."
       />
     </View>
   );
 }
}
const styles = StyleSheet.create({
 // check previous slides
});
```

Search Button

Add a button to allow the user to search. Do this right below the text input. Hopefully you remember this from last week! Don't forget to link the button to the function you just wrote.

1

Add the Button

Here's how we did it:

1

```
{/* TextInput is right here */}
    <Button
      title="Search"
      onPress={this.searchBooks}
    />
```

Results?

So, the app will accept input, request a list of books matching the title inputted from the Google Books API, but won't display them. What UI component do you think should display the books? *Hint: we learned a new UI component earlier today*

Add a FlatList

Yes, a FlatList! Here's a reminder how to use one. Go ahead and add one below your search button, and see if you can set some of the attributes.

```
<FlatList
   data={/* elements stored somewhere in the class*/}
   keyExtractor={/* function you wrote earlier */}
   renderItem={/* function you wrote earlier */}
/>
```

Add a FlatList

Here's how we did it:

```
<FlatList
   data={this.state.books}
   keyExtractor={this._keyExtractor}
   renderItem={this.renderBook}
/>
```

Code

Your current code should read as shown:

```
import React, { Component } from "react";
import {StyleSheet, Text, View, FlatList, TextInput, Button, Keyboard}
from "react-native";

export default class App extends Component {
 constructor() {
   super();
   this.state = {
     books: [],
     searchTerm: "",
   };
 }

  keyExtractor = (item, index) => item.id;
 getBooks = (query) => {
   return fetch(
    "https://www.googleapis.com/books/v1/volumes?q="+ query
   ).then((res) => res.json());
 };
 searchBooks = () => {
   this.getBooks(this.state.searchTerm).then((data) => {
     this.setState({
       books: data.items,
     });
   });

   Keyboard.dismiss();
 };
 renderBook = ({ item }) => {
   const { title } = item.volumeInfo;
   let description;
   if (item.volumeInfo.description !== undefined) {
     if (item.volumeInfo.description.length >= 120) {
       description = item.volumeInfo.description.slice(0, 120) + "...";
     } else {
       description = item.volumeInfo.description;
     }
   }

   return (
     <View style={styles.listItem}>
       <Text style={styles.bookTitle}>{title}</Text>
       {description ? (
         <Text style={styles.bookDescription}>{description}</Text>
       ) : null}
     </View>
   );
 };

render() {
   return (
     <View style={styles.container}>
       {/* <View> these views are a hint for
         later */}
       {/* <View> */}

       {/* </View> */}

       <TextInput
         onChangeText={(text) => {
           this.setState({
             searchTerm: text,
           });
         }}
         value={this.state.searchTerm}
         style={styles.searchTextInput}
         onSubmitEditing={this.searchBooks}
         placeholder="Search for a book..."
       />

       <Button title="Search"
onPress={this.searchBooks} />

       {/* </View> */}

       <FlatList
         data={this.state.books}
         keyExtractor={this. keyExtractor}
         renderItem={this.renderBook}
       />
     </View>
   );
 }
}

const styles = StyleSheet.create({
 // check previous slides
});
```

Run your app!

In your terminal/command prompt:

```
npm start
```

Once Expo DevTools opens in the browser, either:

1. Scan the QR code with your phone to open Expo Go
2. Click “Run on iOS Simulator” or “Run on Android device/emulator”
3. App is now fully functional, but you can’t type into the search field on some devices, and it’s just horribly designed

Advanced Styling

So, what's the remedy for this? We can set up some enclosing views for components we want to group together, and then style both the views and the components.

1

Put some components into their own views:

Our recommendations: Inside the container view, wrap the search input field in its own view, and wrap this new view and your search button in another new view.

Hint 1: when styling, use a flexbox, and the flexDirection and flexBasis properties to orient the input field and button.

Hint 2: Add some margin to the search bar view so we can see the input field.

2

As a reminder:

Define your style, and then set it in the element:

```
textInput: {
   fontSize: 20,
   color: 'red',
   flexBasis: '20%',
   backgroundColor: 'blue',
   width: '100%'
 }
```

```
<TextInput style={styles.textInput}> … />
```

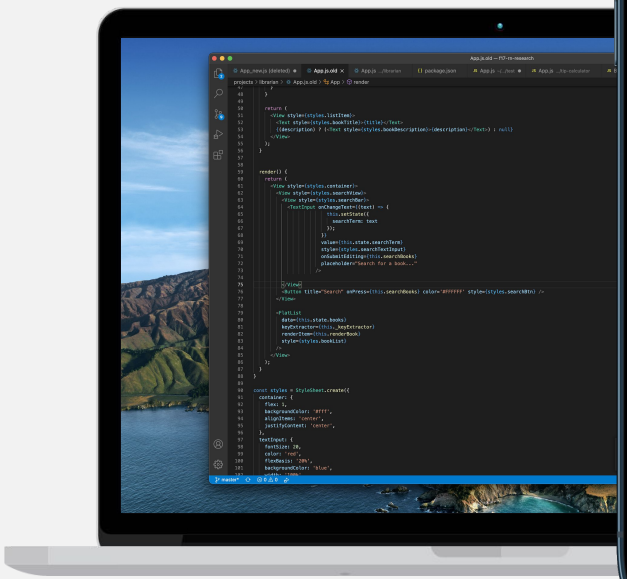
3

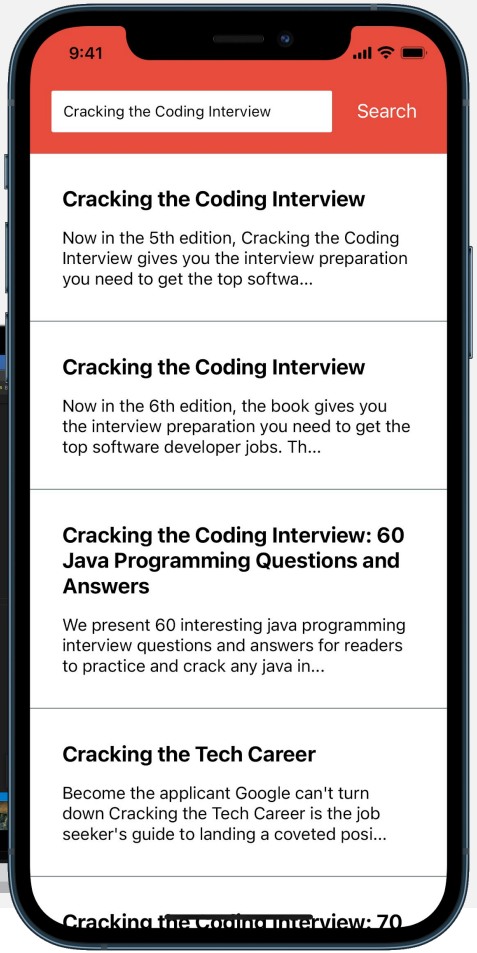
Get to customizing!

Here are some challenges, depending on how comfortable you feel. Change up some of the colors and fonts; try and add an external stylesheet (CSS); check out external styling APIs like Bootstrap, and see if you can figure out how to bring that in.

Final Views

Here's how we structured our views:

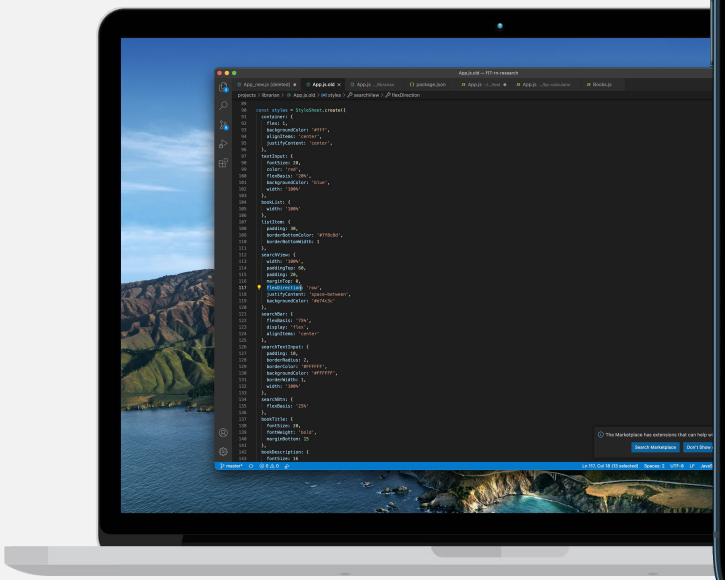


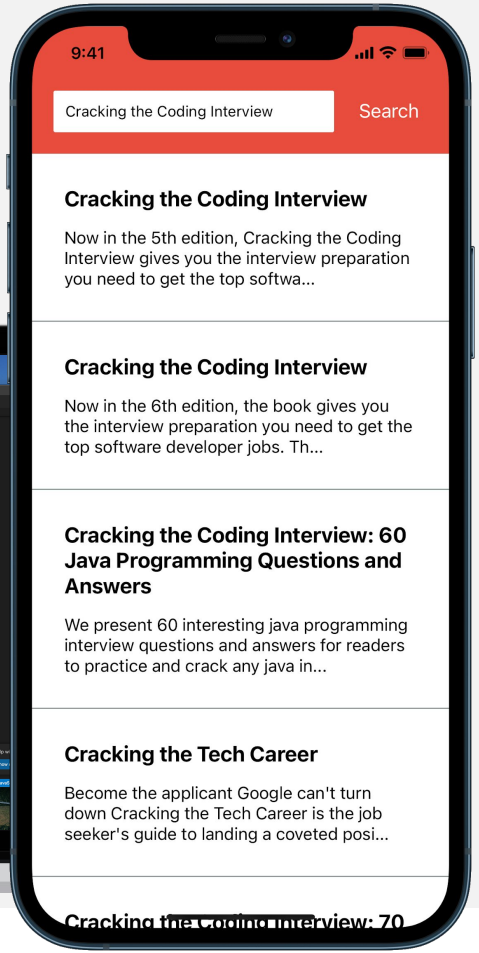



```
render() {
   return (
     <View style={styles.container}>
       <View style={styles.searchView}>
         <View style={styles.searchBar}>
           <TextInput onChangeText={(text) => {
                        this.setState({
                          searchTerm: text
                        });
                      }
               value={this.state.searchTerm}
               style={styles.searchTextInput}
               onSubmitEditing={this.searchBooks}
               placeholder="Search for a book..."
             />
         </View>
         <Button title="Search"
            onPress={this.searchBooks}
            color='#FFFFFF'
            style={styles.searchBtn} />
       </View>
       <FlatList
         data={this.state.books}
         keyExtractor={this. keyExtractor}
         renderItem={this.renderBook}
         style={styles.bookList}
       />
     </View>
   );
 }
```

Final Styles

And here's the styles we came up with:






```
const styles = StyleSheet.create({
 container: {
   flex: 1,
   backgroundColor: '#fff',
   alignItems: 'center',
   justifyContent: 'center',
 },
 textInput: {
   fontSize: 20,
   color: 'red',
   flexBasis: '20%',
   backgroundColor: 'blue',
   width: '100%'
 },
 bookList: { width: '100%' },
 listItem: {
   padding: 30,
   borderBottomColor: '#7f8c8d',
   borderBottomWidth: 1
 },
 searchView: {
   width: '100%',
   paddingTop: 60,
   padding: 20,
   marginTop: 0,
   flexDirection: 'row',
   justifyContent: 'space-between',
   backgroundColor: '#e74c3c'
 },
 searchBar: {
   flexBasis: '75%',
   display: 'flex',
   alignItems: 'center'
 },
 searchTextInput: {
   padding: 10,
   borderRadius: 2,
   borderColor: '#FFFFFF',
   backgroundColor: '#FFFFFF',
   borderWidth: 1,
   width: '100%'
 },
 searchBtn: { flexBasis: '25%' },
 bookTitle: {
   fontSize: 20,
   fontWeight: 'bold',
   marginBottom: 15
 },
 bookDescription: {
   fontSize: 16
 },
 clearInput: {
   position: 'absolute',
   top: 10,
   right: 10
 }
});
```

Wrap Up

With our final lesson, we hope that you've learned how to make HTTP requests from within React Native apps, and how to use encapsulating views along with the CSS flexbox to make more complex styles for your application.