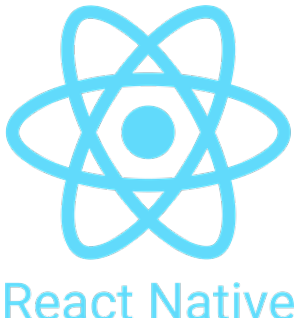
Introduction to React Native - Labs



Completed source code for all labs (for checking your work) can be found at:

`https://github.com/watzthisco/intro-to-react-native`

Version 1.0, October 2022
by Chris Minnick

www.watzthis.com

Disclaimers and Copyright Statement

Disclaimer

WatzThis? takes care to ensure the accuracy and quality of this courseware and related courseware files. We cannot guarantee the accuracy of these materials. The courseware and related files are provided without any warranty whatsoever, including but not limited to implied warranties of merchantability or fitness for a particular purpose. Use of screenshots, product names and icons in the courseware are for editorial purposes only. No such use should be construed to imply sponsorship or endorsement of the courseware, nor any affiliation of such entity with WatzThis?.

Third-Party Information

This courseware contains links to third-party web sites that are not under our control and we are not responsible for the content of any linked sites. If you access a third-party web site mentioned in this courseware, then you do so at your own risk. We provide these links only as a convenience, and the inclusion of the link does not imply that we endorse or accept responsibility for the content on those third-party web sites. Information in this courseware may change without notice and does not represent a commitment on the part of the authors and publishers.

Copyright

Help us improve our courseware

Please send your comments and suggestions via email to info@watzthis.com

Credits

About the Author

Chris Minnick (chris@minnick.com) is a prolific published author and the CEO of WatzThis, Inc.

He started his first company in 1996 and ran it successfully for over twenty years. During this time, he was responsible for the project management and development of hundreds of web and mobile projects for publishing and media clients.

Minnick has authored and co-authored over a dozen books including titles in the For Dummies series and several books for teaching kids to code. He has developed video courses for online training providers Pluralsight, O'Reilly Video, Ed2Go, and Skillshare. His current company, WatzThis, writes and maintains courseware that's used for training software developers at some of the largest companies in the world.

Minnick studied creative writing, literature, journalism, and film at the University of Michigan, and he worked in editorial, circulation, and online publishing capacities at newspapers and magazines in Ann Arbor, Detroit, and San Francisco. In addition to his technical writing and training, Chris is a novelist, painter, winemaker, swimmer, and musician.

Table of Contents

Setup Instructions

Course Requirements

To complete the labs in this course, you will need:

- A computer with MacOS, Windows, or Linux.
- Access to the Internet.
- A modern web browser.
- Ability to install software globally (or certain packages pre-installed as specified below).

Classroom Setup

These steps must be completed in advance if the students will not have administrative access to the computers in the classroom. Otherwise, these steps can be completed during the course as needed.

1. Install node.js on each student's computer. Go to nodejs.org and click the link to download the latest version from the LTS branch.
2. Install a code editor. We use Visual Studio Code in the course
3. Make sure Google Chrome is installed.
4. Install git on each student's computer. Git can be downloaded from *http://git-scm.com*. Select all the default options during installation.
5. Install Android Studio on each student's computer

Get the Repo

1. Open a command prompt.
    a. Use Terminal on MacOS (/Applications/Utilities/Terminal).
    b. Use gitbash on Windows (installed with git).
2. Enter `cd` to navigate to the user's home directory (or change to a directory where student files should be created).
3. Enter the following:

```
git clone https://github.com/chrisminnick/intro-to-react-native
```

The lab solution files for the course will download into a new directory called intro-to-react-native.

Lab 1 - Getting started with Expo

- ☐ 1. Open the integrated terminal in VS Code
- ☐ 2. Create a new Expo project

  ```
  npx create-expo-app my-app
  ```

  This will take some time. Now would be a good time to start installing Android Studio if you haven't already, so you'll have an emulator to test with.

  If you're on a Mac, install XCode too.

- ☐ 3. Start Metro Bundler

  ```
  cd my-app
  expo start
  ```

- ☐ 4. Press w in the terminal to start the web view

  The first time you do this, you'll be asked if you want to install the required dependencies. Say Yes.

  If you get an error, quit Metro Bundler then run the following:

  ```
  npm install react-native-web react-dom --force
  ```

- ☐ 5. Try starting the web view again

  The app should open in your browser.

- ☐ 6. Open App.js in VS Code and try making some changes to the return value, then go back to the browser to see your changes.
- ☐ 7. Install the React Native Tools extension in VS Code

Lab 2: Using an emulator

☐ 1. Start Android Studio
☐ 2. In Android Studio, open the Android Device Manager
☐ 3. Create a new device.

   Make sure to choose a recent Android version and a new phone. If you need to install a new Android system image, now might be a good time to go to lunch or to start working on your app using the web view.

☐ 4. Once you have a device emulator, return to VS Code and try starting your app in it by pressing a in the terminal with Metro Bundler running.

   After a little while your app should open in the emulator.

☐ 5. If you're on macOS, you can install XCode and use the iPhone Simulator.
☐ 6. Press CTRL-M or CMD-M while the app is running to open the developer tools.
☐ 7. Enable remote debugging.

   A browser window will open. Open the Chrome Developer tools in that window and switch to the Sources tab.

Note: Remote debugging will slow down your simulated app and shouldn't be used all the time, but it's the best way to debug your code while it's running in a device.

Lab 3: Set up a server

In this course, we'll be connecting a React Native app with a data source using GraphQL. Follow these steps to set up the API server and the GraphQL server.

- ☐ 1. Copy the /api-server directory from the course repo and run npm install inside it.
- ☐ 2. Run npm start in the api-server directory to confirm that it works.

  The server should be accessible at localhost:3001

- ☐ 3. Copy the /graphql-server directory from the repo and run npm install inside it.
- ☐ 4. Run npm start in graphql-server to confirm that it works.

  The server should be accessible at localhost:4000

Lab 4: Connect the client to the GraphQL server

☐ 1. In the React Native app, install the apollo client and graphql

```
npm install @apollo/client graphql
```

☐ 2. Import ApolloClient, InMemoryCache, and ApolloProvider into App.js

```
import { ApolloClient, InMemoryCache, ApolloProvider }
from '@apollo/client';
```

☐ 3. Create a client instance

```
const client = new ApolloClient({
  uri: 'http://[your local IP address]:4000',
  cache: new InMemoryCache(),
  connectToDevTools: true,
});
```

Note: You have to use your local IP address (not localhost or 127.0.0.1) in the above or it won't work in the Android emulator. To find your local IP, enter `ipconfig` into the terminal (on Windows) or `ipconfig getifaddr en1` on macOS.

☐ 4. Wrap the contents of the return statement in App.js with ApolloProvider and pass the client instance to it.

```
export default function App() {
  return (
    <ApolloProvider client={client}>
      <View style={styles.container}>
        <Text>Something here.</Text>
        <StatusBar style="auto" />
      </View>
    </ApolloProvider>
  );
}
```

The app we're going to start building will be a fitness app / activity tracker for programmers called CoderFit. We'll create a new root component that we'll render inside App to make passing the Apollo client easier.

☐ 5. Create a new file in src called CodersList.js
☐ 6. Move the React Native imports from App.js to CodersList.js
☐ 7. Move the styles object from App.js to CodersList.js
☐ 8. Import useQuery and gql into CodersList.js

```
import { useQuery, gql } from '@apollo/client';
```

☐ 9. Move everything between <ApolloProvider> and </ApolloProvider>, except the status bar component from App.js and into CodersList.js

☐ 10. Import CodersList into App.js

```
import CodersList from './CodersList';
```

☐ 11. Render CodersList inside the ApolloProvider in App.js

```
export default function App() {
  return (
    <ApolloProvider client={client}>
      <CodersList />
      <StatusBar style="auto" />
    </ApolloProvider>
  );
}
```

☐ 12. Create a query in CodersList.js

```
export const CODERS_QUERY = gql`
  query coders {
    coders {
      id
      name
      description
    }
  }
`;
```

☐ 13. Inside the CodersList function, pass the query to useQuery and deconstruct loading, data, and error.

```
  const { loading, error, data } =
useQuery(CODERS_QUERY);
```

☐ 14. Handle loading and error by returning early if either are true.

```
  if (loading) return <Text>Loading...</Text>;
  if (error) return <Text>Error!
{error.message}</Text>;
```

☐ 15. Replace the dummy text in CodersList's return statement with a map function that will render a list of coders from the graphql datasource.

```
      {data.coders.map((coder) => (
        <View className="coder" key={coder.id}>
          <Text>{coder.name}</Text>
          <Text>{coder.description}</Text>
        </View>
      ))}
```

☐ 16. Make sure that the api-server and the graphql-server are both running, then start the metro bundler

```
expo start
```

☐ 17. Press w to view your app in a browser.
☐ 18. If everything works in the browser, try it in your Android or iOS emulator.

Lab 05: Set up Navigation

- ☐ 1. Create a new folder in your project called navigation.
- ☐ 2. Install @react-navigation

```
npm install @react-navigation/native
```

- ☐ 3. Install navigation screens and context

```
npx expo install react-native-screens react-native-safe-area-context
```

- ☐ 4. Install native-stack

```
npm install @react-navigation/native-stack
```

- ☐ 5. Import dependencies into App.js

```
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
```

- ☐ 6. Create a stack navigator in App.js (outside the function).

```
const Stack = createNativeStackNavigator();
```

- ☐ 7. Wrap the CodersList element in App.js's return with NavigationContainer and Stack.Navigator and render CodersList in a Stack.Screen

```
<ApolloProvider client={client}>
  <NavigationContainer>
    <Stack.Navigator>
      <Stack.Screen name="All Coders" component={CodersList} />
    </Stack.Navigator>
  </NavigationContainer>
  <StatusBar style="auto" />
</ApolloProvider>
```

- ☐ 8. Test it out and make sure your app still looks the same as before, but now with the screen name in the header.

Lab 06: Set up another route

☐ 1. Add a new Screen to the router in App.js for an AddCoder component and pass initialRouteName to the Stack Navigator to set the initial route to the All Coders route.

```
<ApolloProvider client={client}>
  <NavigationContainer>
    <Stack.Navigator initialRouteName="All Coders">
      <Stack.Screen name="All Coders" component={CodersList} />
      <Stack.Screen name="Add a Coder" component={AddCoder} />
    </Stack.Navigator>
  </NavigationContainer>
</ApolloProvider>
```

☐ 2. Create an AddCoder component and import it into App.js

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

export default function AddCoder() {
  return (
    <View style={styles.container}>
      <Text>Coder Name</Text>
      <View>
        <TextInput placeholder="Coder Name" />
        <TextInput placeholder="Coder Description" />
      </View>
      <Button title="Add Coder" />
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
});
```

☐ 3. Move AddCoder and CodersList into a new directory named screens

☐ 4. Run your app, then try changing the initial route to view the Add Coder screen.

Lab 7: Changing Screens

- ☐ 1. Import Button from react-native into CodersList.
- ☐ 2. Deconstruct the navigation object CoderList's props object

```
function CoderList({navigation}) {
```

- ☐ 3. Render a button and set its onPress to call navigation.navigate and pass in the name of your other route.

```
<Button title="Add a Coder" onPress={
() =>
  navigation.navigate('Add a Coder')
} />
```

- ☐ 4. Put a button on Add a Coder to return to the Coders List
- ☐ 5. Test it out and verify that you can switch between screens.

Lab 8: Hooking up the Add Coder form to a GraphQL mutation

Lab 10: Using FlexBox and React Native Styling