

[1 Introduction](https://docs.google.com/document/d/1t3ya-7m7V8R49UEdH5Hfs28f1Dy3HJG7HVeEEM2sUOk/edit#heading=h.siqei25mwfk2)

[About our App](https://docs.google.com/document/d/1t3ya-7m7V8R49UEdH5Hfs28f1Dy3HJG7HVeEEM2sUOk/edit#heading=h.qdwgqbw3a7zg)

[Topics Covered](https://docs.google.com/document/d/1t3ya-7m7V8R49UEdH5Hfs28f1Dy3HJG7HVeEEM2sUOk/edit#heading=h.qaau17qpzxl0)

[1. Create React Native application using create-react-native-app CLI (CRNA)](https://docs.google.com/document/d/1t3ya-7m7V8R49UEdH5Hfs28f1Dy3HJG7HVeEEM2sUOk/edit#heading=h.q2dmg8qc70nq)

[Opening the CRNA app on a real device via Expo](https://docs.google.com/document/d/1t3ya-7m7V8R49UEdH5Hfs28f1Dy3HJG7HVeEEM2sUOk/edit#heading=h.kdk5dzb0cz6n)

[Opening the CRNA app in a simulator](https://docs.google.com/document/d/1t3ya-7m7V8R49UEdH5Hfs28f1Dy3HJG7HVeEEM2sUOk/edit#heading=h.irqxlgj7iold)

[2. Setup prettier, eslint, and airbnb style guide](https://docs.google.com/document/d/1t3ya-7m7V8R49UEdH5Hfs28f1Dy3HJG7HVeEEM2sUOk/edit#heading=h.4ma5tjfqowgl)

[Configure VS Code to run eslint on save](https://docs.google.com/document/d/1t3ya-7m7V8R49UEdH5Hfs28f1Dy3HJG7HVeEEM2sUOk/edit#heading=h.mxtgll63srxl)

[3. Drawer and Navigation using react-navigation](https://docs.google.com/document/d/1t3ya-7m7V8R49UEdH5Hfs28f1Dy3HJG7HVeEEM2sUOk/edit#heading=h.z15hf8wnutv1)

[Directory Structure](https://docs.google.com/document/d/1t3ya-7m7V8R49UEdH5Hfs28f1Dy3HJG7HVeEEM2sUOk/edit#heading=h.d5rnb3rpzj86)

[Tab Navigation](https://docs.google.com/document/d/1t3ya-7m7V8R49UEdH5Hfs28f1Dy3HJG7HVeEEM2sUOk/edit#heading=h.m8c9d1rhjkgz)

[Stack Navigation](https://docs.google.com/document/d/1t3ya-7m7V8R49UEdH5Hfs28f1Dy3HJG7HVeEEM2sUOk/edit#heading=h.njv0nd3bse0q)

[4. jest and Enzyme](https://docs.google.com/document/d/1t3ya-7m7V8R49UEdH5Hfs28f1Dy3HJG7HVeEEM2sUOk/edit#heading=h.l38kqwlo2pzr)

[5. Mobx Store](https://docs.google.com/document/d/1t3ya-7m7V8R49UEdH5Hfs28f1Dy3HJG7HVeEEM2sUOk/edit#heading=h.kgjy0736ojis)

[Author List View](https://docs.google.com/document/d/1t3ya-7m7V8R49UEdH5Hfs28f1Dy3HJG7HVeEEM2sUOk/edit#heading=h.hpkkroudk8pd)

[Book List View](https://docs.google.com/document/d/1t3ya-7m7V8R49UEdH5Hfs28f1Dy3HJG7HVeEEM2sUOk/edit#heading=h.to5hgqjhixs3)

[6a UI Framework - nativebase](https://docs.google.com/document/d/1t3ya-7m7V8R49UEdH5Hfs28f1Dy3HJG7HVeEEM2sUOk/edit#heading=h.fvcpb3k5mmlc)

[6b UI Framework - react-native-elements](https://docs.google.com/document/d/1t3ya-7m7V8R49UEdH5Hfs28f1Dy3HJG7HVeEEM2sUOk/edit#heading=h.b98xdt79kw7w)

**1 Introduction**

This series is a bit different in that it covers development practices including directory structure, state management (via Mobx) code styling and linting tools (prettier, eslint, and arirbnb style guide), and an important, but often ignored part - testing your application (via jest and enzyme).

**About our App**

The application we're building is for a BookStore, and will consist of two simple views, Books View and Authors View. The app will contain a navigation drawer with two menu options, allowing the user to switch between the two views. The first option would be for navigating to the Books View, and the other option would be for the Authors View. The Books View will contain the list of books, while the Authors View will containing the list of authors. The Book View will further contain tab navigation allowing the user to switch between Fiction and Non-Fiction books.

**Topics Covered**

We'll cover different parts for a full blown React Native application.

1. Create React Native application using create-react-native-app CLI (CRNA)

2. Setup prettier, eslint, and arirbnb style guide

3. Add Drawer and Tabs Navigation using react-navigation

4. Add mobx

5. Add

We'll be installing everything on a Mac OS. Most of the commands will be the same when you have node installed, but if you face any issues, add a comment or just google it.

So let's get started by bootstrapping our React Native BookStore application.

**1. Create React Native application using create-react-native-app CLI (CRNA)**

We'll use **yarn** as our package manager. The same commands can be run via npm. The reason I prefer yarn is that I've found yarn to be faster than npm, and the **facebook**, the creators of yarn **say so themselves**, although **some suggest** that npm is now just as fast. To install

If there are any issues, follow the **yarn installation guide**. To look at the **commands documentation** to learn more about it.

npm install -g create-react-native-app

Then, to create our app,

Now, we can use the CRNA command to create our new React Native project called "my-rn-app":

create-react-native-app my-rn-app

Once CRNA is done bootstrapping our React Native application, it will show the helpful commands. Let's change directory to the newly created CRNA app, and start it.

cd my-rn-app

npm start

And that's it! Easy peasy!

If you face any issues, please refer to either the React Native's getting started guide <https://facebook.github.io/react-native/docs/getting-started.html>

or Create React Native app (CRNA) guide <https://github.com/react-community/create-react-native-app/blob/master/react-native-scripts/template/README.md>

Opening the CRNA app on a real device via Expo

When the app is started via "npm start", a QR code will be displayed in your terminal. The easiest way to look at our bootstrapped app is using the Expo app. To do that:

1. Install the [Expo](https://expo.io/) client app on your iOS or Android phone

2. Make sure that you are connect to the same wireless network as your computer

3. Using the Expo app, scan the QR code from your terminal to open your project

Opening the **CRNA** app in a simulator

To run the app on iOS Simulator, you'll need to install Xcode. To run the app on an Android Virtual Device, you need to setup the Android development environment. Look at the react-native getting started guide for for both the setups. <https://facebook.github.io/react-native/docs/getting-started.html#installing-dependencies>

**2. Setup prettier, eslint, and airbnb style guide**

why is linting tool important

why is Code styling important

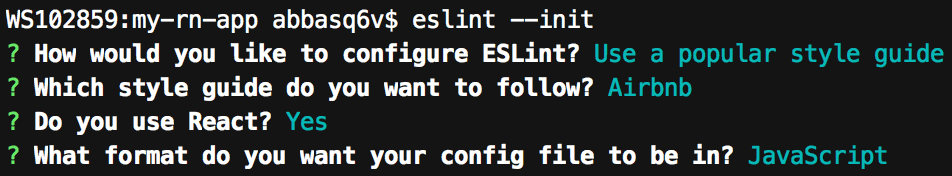
Being a dynamic language, JavaScript is prone to developer error. Since it's not a compile language, error are discovered when the JavaScript code is executed. Linting tools like ESLint allow developers to discover problems with their JavaScript code without executing it. This is why a good linting tool is extremely important to ensure that quality is baked in from the beginning. In order to do that, we'll start our tutorial series by implementing linting tools. You can learn more about ESLint here. <https://eslint.org/docs/about/>

ESLint is configurable and you can set your rules according to your preferences. However different linting rules configuration have have been provided by the community. One of the popular ones is the [airbnb styles guides](https://github.com/airbnb/javascript). The [eslint-config-airbnb package](https://github.com/airbnb/javascript/tree/master/packages/eslint-config-airbnb) provides the contains all of the airbnb ESLint rules, including ECMAScript 6+ and React. We'll use that. Go to the repo to follow the instructions, or if you are on a Mac or Linux, simply run this command in the terminal:

(  
  export PKG=eslint-config-airbnb;  
  npm info "$PKG@latest" peerDependencies --json | command sed 's/[\{\},]//g ; s/: /@/g' | xargs npm install --save-dev "$PKG@latest"  
)

To create the eslint configuration file,

eslint --init



A file .eslintrc.js will be created with the following configurations:

module.exports = {

   "extends": [

     "prettier"

   ],

   "plugins": [

       "react",

       "jsx-a11y",

       "import"

   ]

}

You might be using experimental features not supported in ESLint itself yet. To avoid linting errors on those experimental features, let's also add [babel-eslint](https://www.npmjs.com/package/babel-eslint)

npm install --save-dev babel-eslint

Add "babel-eslint" to .eslintrc.js

module.exports = {

   "parser": "babel-eslint",

   "extends": [

...

<https://medium.freecodecamp.org/adding-some-air-to-the-airbnb-style-guide-3df40e31c57a>

We'll a

<https://github.com/airbnb/javascript/tree/master/packages/eslint-config-airbnb#eslint-config-airbnb-1>

Code styling

No one like ugly code. And when you're working in a team, you want to make sure that the code formatting and indentation is consistent throughout the team. Prettier is just the tool for that. It ensures that all the code conforms to a consistent style.

We'll also add the [ESLint plugin for prettier](https://github.com/prettier/eslint-plugin-prettier), which will add Prettier as an ESLint rule and report differences as individual ESLint issues.

Now, there may be conflicts between the ESLint rules and the code formatting done by prettier. Fortunately, there is a plugin available called [eslint-config-prettier](https://github.com/prettier/eslint-config-prettier) that turns off all rules that are unnecessary or might conflict with prettier. So let's install all the necessary tools, prettier and eslint-plugin-prettier

npm install --save-dev prettier eslint-plugin-prettier eslint-config-prettier

To enable eslint-plugin-prettier plugin, update your .eslintrc.js file to add the "prettier" plugin. And to show linting error on Prettier formatting rules, add the "rule" to show error on "prettier/prettier". Here's our updated .eslintrc.js:

module.exports = {

"parser": "babel-eslint",

"extends": [

  "airbnb",

  "prettier"

],

"plugins": [

  "react",

  "prettier",

  "jsx-a11y",

  "import"

 ],  
  "rules": {  
    "prettier/prettier": "error"  
  }  
}

[eslint-config-prettier](https://github.com/prettier/eslint-config-prettier) also ships with a CLI tool to help you check if your configuration contains any rules that are unnecessary or conflict with Prettier. Let's be proactive and do just that.

First, add a script for it to package.json:

{  
  "scripts": {  
    "eslint-check": "eslint --print-config .eslintrc.js | eslint-config-prettier-check"  
  }  
}

Now, run the "eslint-check" command:

npm run eslint-check

This will list the conflicting rules in the terminal. Let's turn off the conflicting rules by updating the .eslintrc.js file. I also prefer singleQuote and trailingComma, so I'll configure those rules as well. This is what our .eslintrc.js file looks like now:

module.exports = {

"parser": "babel-eslint",

"extends": [

  "airbnb",

  "prettier"

],

"plugins": [

  "react",

  "jsx-a11y",

  "import"

],

"rules": {

  "prettier/prettier": "error",

  "react/jsx-closing-bracket-location": "off",

  "react/jsx-curly-spacing": "off",

  "react/jsx-equals-spacing": "off",

  "react/jsx-first-prop-new-line": "off",

  "react/jsx-indent": "off",

  "react/jsx-indent-props": "off",

  "react/jsx-max-props-per-line": "off",

  "react/jsx-tag-spacing": "off",

  "react/jsx-wrap-multilines": "off"

}

}

Configure VS Code to run eslint on save

We can configure any IDE to automatically run eslint on Save or as we type, and since we have also configured prettier along with eslint, our code will automatically be pretiffied. VS Code is an IDE popular in the JavaScript community, so I'll show how to setup eslint auto-fix on save using VS Code, but the steps would be similar in any IDE.

To configure VS Code to automatically run eslint on Save, we first need to install the eslint extension. Go to Extensions, search for "ESLint" extension, and install it. Once the ESLint extension is installed, go to Preferences > User Settings, and set "eslint.autoFixOnSave" to true. Also make sure that "files.autoSave" is either "off", "onFocusChange" or "onWindowChange".

Now, open the file App.js. If the eslint is configured correctly, you should see some linting error, like the "react/prefer-stateless-function", "react/jsx-filename-extension", and "no-use-before-define" errors. Let's turn those "off" in the .eslintrc.js file. I also prefer singleQuote and trailingComma, so I'll configure those rules as well. Here is the updated .eslintrc.js file.

module.exports = {

"parser": "babel-eslint",

"extends": [

  "airbnb",

  "prettier"

],

"plugins": [

  "react",

  "prettier",

  "jsx-a11y",

  "import"

],

"rules": {

  "prettier/prettier": [

    "error",

    {

      "singleQuote": true,

      "trailingComma": "all",

    }

  ],

  "react/prefer-stateless-function": "off",

  "react/jsx-filename-extension": "off",

  "no-use-before-define": "off",

  "react/jsx-closing-bracket-location": "off",

  "react/jsx-curly-spacing": "off",

  "react/jsx-equals-spacing": "off",

  "react/jsx-first-prop-new-line": "off",

  "react/jsx-indent": "off",

  "react/jsx-indent-props": "off",

  "react/jsx-max-props-per-line": "off",

  "react/jsx-tag-spacing": "off",

  "react/jsx-wrap-multilines": "off"

}

}

I know this was a lot of work, considering that we haven't even started working on our app yet, but trust me this setup would be very beneficial in the long run, even if you're a one person team, and when you're working with other developers, linting and coding standards will go a long way in reducing code defects and ensuring consistency in code style.

<https://github.com/qaiser110/react-native-mobx-starter/tree/1-eslint-airbnb-prettier>

**3. Drawer and Tabs Navigation using react-navigation**

Our app will contain a navigation drawer with two menu options, the first option would be for the BookView, containing the list of books. The other option would be for the AuthorView, containing the list of authors. For navigation between the different views, we'll use [React Navigation](https://reactnavigation.org). So let's install it:

npm install --save react-navigation

Open App.js, and make the following changes:

1. Rename the default export

2. Add two react stateless components for our screens, for the BookView and AuthorView respectively

3. Add stateless components for our DrawerComponent. This component will render the Drawer content, along with the menu options (Books and Authors) inside <DrawerItems> imported from react-navigation

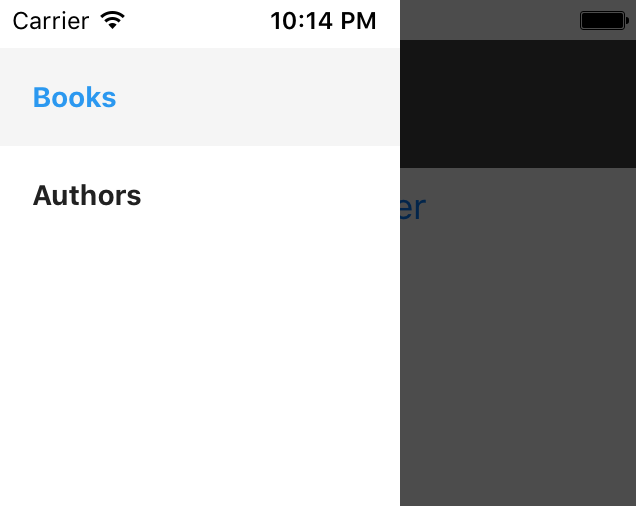
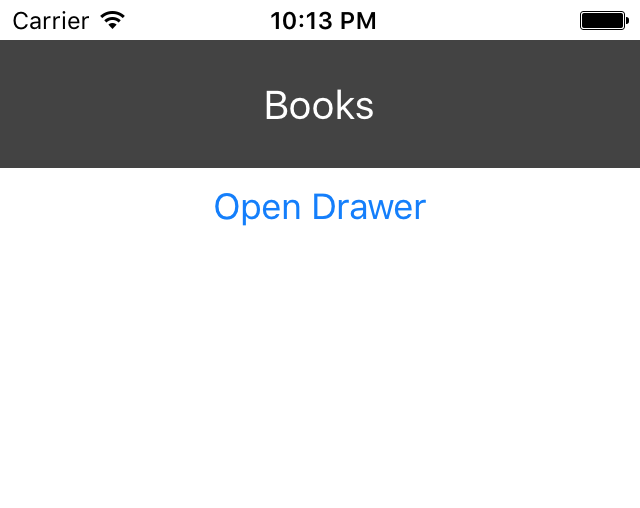
4. Use the [DrawerNavigator](https://reactnavigation.org/docs/navigators/drawer) from react-navigation to implement the Drawer Navigation

5. Add relevant styles

Here's a diff of our latest changes:

[github.com/qaiser110/react-native-mobx-starter/commit/7ebb2b5a5d8448e7fd75d6c72a66c1c7f09232b1](https://github.com/qaiser110/react-native-mobx-starter/commit/7ebb2b5a5d8448e7fd75d6c72a66c1c7f09232b1)

And after making the above changes, here's what our UI looks like when before and after we click on the "Open Drawer" button



**Directory Structure**

It's important to think about your application and how you'll structure of your files and resources in the beginning of the project. While there could be several ways of structuring your application code, I prefer co-locating files and tests using a feature-based architecture. Co-locating files related to a particular feature or module has **several benefits**.

Let's create a src directory, where we'll keep all our source files, and inside it, create two directories, one for the book view, named "book", and the other for the author view, named "author".

Create index.js files within each of the two directories we just added. These files will export the components for each of our views. Move the code from App.js for the BookView and AuthorView components into these files, and import them instead.

Our app should still work as it was before the refactor. Here's a file diff of our recent changes:

[github.com/qaiser110/react-native-mobx-starter/commit/99fbd537de1cefc3723b79fe25394063abe62a3c](https://github.com/qaiser110/react-native-mobx-starter/commit/99fbd537de1cefc3723b79fe25394063abe62a3c)

Each of the screen will have a title, which means that we'd be duplicating the same code along with the styles. To keep our code DRY, let's move the title to a separate file "src/components/Title.js", and reuse it where needed. We'll also move the main views into a new parent directory "src/views" to keep them separate from other components.

Here's a file diff of our recent changes:

[github.com/qaiser110/react-native-mobx-starter/commit/578f9bbcd2e946ca52e579c160aa17b0917947d8](https://github.com/qaiser110/react-native-mobx-starter/commit/578f9bbcd2e946ca52e579c160aa17b0917947d8)

**Tab Navigation**

We need two tabs in the books view to show the fiction and non-fiction books. Let's use the [TabNavigator](https://reactnavigation.org/docs/navigators/tab) from react-navigation to implement the Drawer Navigation.

With those changes we have our apps navigations all done. We can open the main drawer from all screens and tabs. We can also switch tabs by clicking on tabs or by clicking on the buttons we added in each tab view.

[github.com/qaiser110/react-native-mobx-starter/commit/7e5ef8c6dd12c64b9721a4e17050ed9c60c835c6](https://github.com/qaiser110/react-native-mobx-starter/commit/7e5ef8c6dd12c64b9721a4e17050ed9c60c835c6)

**Stack Navigation**

We'll need a top bar on Author View and Book View, and a menu icon to open the drawer. So for both the views, instead of returning a ReactNative View, we'll return a [StackNavigator](https://reactnavigation.org/docs/navigators/stack) instead.

All Books View and books data

We'll need data

a view for displaying a list of all books

I've also removed all the content from the book tabs, since we'll be adding the actual content there shortly.

**4. jest and Enzyme**

<https://medium.com/react-native-training/learning-to-test-react-native-with-jest-part-1-f782c4e30101>

npm install --save-dev jest babel-jest regenerator-runtime

"scripts": {

 "test": "node node\_modules/jest/bin/jest.js --watch"

},

"jest": {

 "preset": "jest-expo"

}

Expo includes all the jest related libraries and configurations

We'll install enzyme, and also add react-dom which is a dependency. We'll also add jest-serializer-enzyme which makes ...

npm install --save-dev enzyme@2.9.1

npm install --save-dev react-dom@16.0.0-rc.3 react-test-renderer@16.0.0-alpha.12

jest-serializer-enzyme

// pacakge.json

"jest": {  
    "preset": "jest-expo",  
    "snapshotSerializers": [  
      "./node\_modules/jest-serializer-enzyme"  
    ]  
  },

Also, at the time of this tutorial, there is an open ticket which would cause

<https://github.com/facebook/react-native/issues/13034>

Enzyme allows for direct manipulation of the props and state of our components so we can create snapshots for multiple renders of the same component.

<https://blog.joinroot.com/mounting-react-native-components-with-enzyme-and-jsdom/>

<http://airbnb.io/enzyme/docs/api/ShallowWrapper/dive.html>

**5. Mobx Store**

There are several state management libraries out there, most popular of them are Redux, MobX, and Saga.

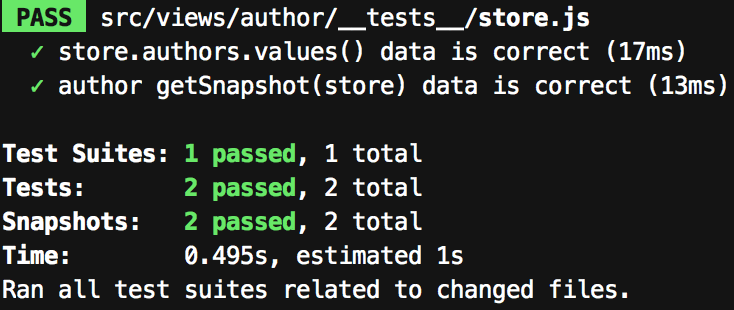
<https://bitbucket.org/abc110/reactnative-poc/commits/all>

npm install --save mobx mobx-react mobx-state-tree

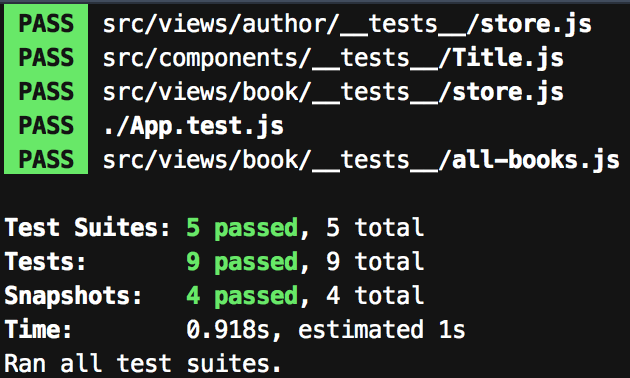
Since we'll be using decorators, we also need to add the "transform-decorators-legacy" plugin in .babelrc

"plugins": ["transform-decorators-legacy"],

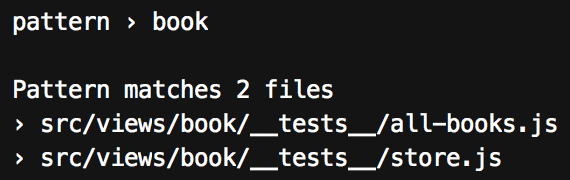
Now make a change in "src/views/author/store.js".



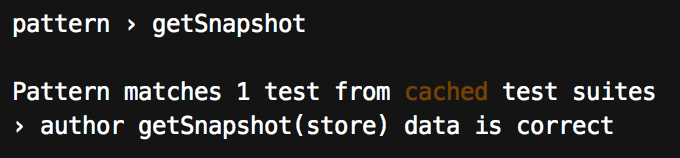
Press "a",



Now if you want to run tests for just the books module, press "p" followed by "book". Jest will filter the tests based on regex pattern on the file paths and list the files matching the pattern. Hit "Enter" to run all the tests in this file.



Similarly, you can run tests based on test name pattern matching. For instance, press "t" followed by "getSnapshot".



Now, we'll need the api to call so we'll move the mock api we created for our tests to a new file, and import it in the test. Rather than referencing that api in all our views and components, we'll make it the default api in our store. That way, once we have the real api ready, we just have to fix the api file reference in the store.

It is important to note that since we're building a small demo app, mocking our api this way is okay, but in a bigger project, we would use proper mocking. Jest provides good mocking capabilities, or you can use an external mocking and spying framework such as sinon. For making http calls to a Restful API, I like to use axios, along with its mocking companion, Moxios.

**Author List View**

The top level component should be a stateful component, so let's do that

**Book List View**

The only thing different in the three tabs, namely "All Books", "Fiction Books", "NonFiction Books", is the data that they'll display. The layout and format of the items would be the same. And since Mobx allows us to keep our data completely separate from the view. So we can get rid of the three separate views, and have the three tabs displayed using a single stateless component, with different data sets sent via props.

Let's also add a test for the setFilter

 render() {

   const { filter } = this.props

   this.store.setFilter(filter)

   return <BookList books={this.store.filteredBooks} />

 }

 render() {

   const { filter } = this.props

   return <BookList books={this.store.booksByGenre(filter)} />

 }

but I find setFilter() more expressive and explicit.

I've refactored our codebase a bit. I've `moved all-books.js`, `fiction-books.js`, `non-fiction-books.js` into the tabs folder. I've also moved BookList and Book into separate files, to better prepare ourselves for the changes to come. Our BookList and Book components will grow in size as we'll be adding a UI Framework to make our views look pretty. This brings me to two important practices, small component size and code refactor.

It's important to note that refactoring should be a major part of the development workflow. We should continuously refactor our code to prepare ourselves for future changes and challenges. This greatly impacts the development productivity in the long run.

Our store could also be moved into a separate directory, but since our app is quite small, we don't need to do that.

**6a UI Framework - nativebase**

npm install native-base --save

npm install @expo/vector-icons --save

**6b UI Framework - react-native-elements**

yarn add react-native-elements