Setting up the Store

Using our knowledge from the previous chapter, we'll set up the store for our app, along with our render and store.subscribe functions.

Let’s quickly go over the process of setting up the store of the App, so that we can retrieve the data required to build the list of users within the sidebar.

One of the first steps when creating a Redux app is setting up the Redux store. Since this is where data will be read from, it becomes imperative to resolve this.

So, please install **redux** from the **cli** with:

```
yarn add redux
```

Once the installation is done, create a new folder called store and in the directory, create a new **index.js** file.

Don’t forget the analogy of having the major Redux actors in their own directory.

Like you already know, the store will be created via the **createStore** factory function from redux like this:

store/index.js:

```
import { createStore } from "redux";
const store = createStore(someReducer, initialState);
export default store;
```

store/index.js

The Redux createStore needs to be aware of the reducer - remember the store and reducer relationship I explained earlier.

Now, edit the second line to look like this:

```
const store = createStore(reducer, {contacts});
```

Now, import the reducer, and contacts from the static data:

```
import reducer from "../reducers";
import { contacts } from "../static-data";
```

Here's what we have a for our **store** up till now:

```
import { createStore } from "redux";
import reducer from "../reducers";
import { contacts } from "../static-data";

const store = createStore(someReducer, initialState);
export default store;
```

store/index.js

Since we actually haven't created any reducers directory, please do so now. Also create an index.js file within this reducers directory.

Now, create the reducer.

reducers/index.js:

```
export default (state, action) => {
  return state;
};
```

reducers/index.js

A reducer is just a function that takes in state and action, and returns a new state.

If I lost you in the creation of the store,

```
const store = createStore(reducer, {contacts});
```

you should remember that the second argument in createStore is the initial state of the app.

I have set this to the object, {contacts}.

This is an ES6 syntax, similar to this: {contacts: contacts} i.e a contacts key and a value of contacts from static-data.

There’s no way to ascertain that what we’ve done is right. Let’s attempt to fix that.

In the primary **index.js** , here’s what you should have now:

```
import React from "react";
import ReactDOM from "react-dom";
import "./index.css";
import App from "./App";
import registerServiceWorker from "./registerServiceWorker";
ReactDOM.render(<App />, document.getElementById("root"));
registerServiceWorker();
```

index.js

Like we did with the first example, refactor the **ReactDOM.render** call to sit inside a render function.

```
const render = () => {
  return ReactDOM.render(<App />, document.getElementById("root"));
};
```

Then involve the render function to have the App render correctly.

```
render()
```

Now, import the store you created earlier

```
import store from "./store";
```

And make sure any time the store is updated, the render function is invoked.

```
store.subscribe(render);
```

Good! Now we have our **store** set up. In the next lesson, let’s take advantage of this.