Redux in a Nutshell Lab

It's unfortunate that we don't have time to really get into Redux because it is a fantastic library that complements React so well. But at least we got a high-level introduction and we are hoping that after the course you'll make time to dig into understanding Redux enough to actually write your own Redux code. To help with that, we are providing you with a prewritten implementation of Redux that we will use for subsequent labs.

Install Redux

1. Open a command window and cd to your "client" project. Install Redux.

```
npm install redux
```

2. Take a look in package.json. Make sure redux is listed as a dependency.
3. Look in node_modules. Make sure you can see the redux library.

Of course there is no way for you to learn Redux in just one lecture, so we won't ask you to write your own redux store, reducers, action creators, and middleware. Instead we've built them for you. Let's see what we can learn by reading through them.

Copy the prebuilt Redux files

It is a best practice to keep your Redux things together in a folder so let's group them.

4. Look in the starters folder. Notice that there's a folder in there called 'store'. This is a pre-built Redux store.
5. Copy all of starters/store to your project under src. Something like this might work:

```
cp myProject/starters/store myProject/client/src
```

   (Hint: don't actually execute that command. It is just trying to tell you to copy the entire store folder to your src folder under your project.)
6. Go ahead and look around in that folder. You'll find certain files. Let's examine them.

store.js

7. Crack open store.js in your IDE. Discuss these questions with your pair programming partner.
8. What is being imported from redux? _____
9. What are each of them for? What will they do? _____

Notice that we're importing a bunch of reducers and middleware. Let's take a look at them.

Looking at the reducers

10. Edit reducers.js. Take a look at each reducer. Notice that they all have the same shape. What is being received and returned?
    It is _____
    But before we return a new state, we are altering the state.
11. See if you can follow through the reducer composition, going backwards through the file, tracing each sub-reducer's part back to its origin.

Notice that we're importing from ./actions.js. Let's look at what's in there.

actions.js

This file holds an enumeration of sorts of all of the types of actions that can be taken. All it does is prevent hard-to-debug typos and enables your IDE to prompt you with intellisense.

12. Scan through this file with your partner. Discuss why it helps. What if you didn't have it? What might happen?

middleware.js

13. Open this file and read through. Notice first that it imports actions and actionTypes from actions. We now know what those are.
14. Take a look at one or two of the middleware functions that are being exported. Discuss with your partner how they work.
15. Notice the shape of every middleware function. They have the same required shape. What is that shape?
    _____

Redux middleware is a very advanced subject, so don't fret if you don't understand it. It really takes days to get a deep enough understanding of Redux to be able to apply it. Maybe you can come back for a multiple-day intro at a later time.

16. There are several middleware functions that begin with fetch. Notice how they are fetching some data from an API and in the promise callback, it is dispatching a couple of actions. Where are those actions coming from? _____

Wiring them up with React hooks

Finally, let's make them work. To do so we're going to use some fairly advanced techniques that will become much more clear toward the end of the React class. But how cool that you get to a preview of React hooks, right?

17. Open App.js and import your store. Redux itself isn't needed, just your store. Interesting, right? And while you're there, import useState and useEffect from React.

```
import { useState, useEffect } from 'react';
import { store } from './store/store';
import { actions } from './store/actions';
```

18. Inside the App function at the top, you may already have a useState and useEffect lines. If you do, replace them with these lines. If not, add them.

```
const [state, setState] = useState(store.getState());
useEffect(() => {
  const unsubscribe = store.subscribe(() => setState({ ...store.getState() }));
  store.dispatch(actions.fetchInitialData());
  return unsubscribe;
}, []);
```

19. Run and test. You won't see anything different on the page but if you look in the console, you should be seeing lots of wonderful data about theaters and films and showings.

I know that's a lot to take in, but some of it will become more clear later in the course and other parts are beyond the scope of the course (so don't sweat it). In a nutshell, this line says "Whenever any action is dispatched, set App.js's state variable to whatever is in Redux after the

reducer runs." In other words, refresh this page with the latest state whenever that state might change. Then it says to use the fetchInitialData middleware to hit our API to grab all the initial data our users will need.