react-redux

We have a barebones React app created and have added a Redux store to it. We're even reading the preloaded state and logging it to the console. We can begin fleshing out the reducer, subscribing to changes, and dispatching actions. But all of that stuff is much more streamlined if we use the glue code in react-redux. In this lab, let's install react-redux and wire it up.

Set up react-redux

1. First install the library itself:

```
npm install react-redux
```

Add the root Provider

The Provider is a component that ... well ... "*provides"* the store to all the components underneath it no matter how many levels deep. Let's make our store and a Provider known at the highest level and then include them in the JSX.

2. Edit index.js. Put this at the top:

```
import { Provider } from 'react-redux';
import { store } from './store/store';
```

3. Still in index.js, find the <App /> component and wrap it with a <Provider>. Do something like this:

```
<Provider store={store}><App /></Provider>
```

4. If you run and test at this point you won't see anything different. Go ahead and try though. Just prove to yourself that there are no transpile errors.

Adding a useSelector hook

useSelector is a hook included in the react-redux library. It essentially reaches up into its component ancestry to find the Provider and accesses any part of the state you want. It is called a "selector" because it *selects* only the slice of state that you instruct it to.

5. Edit App.js. At the top, import { useSelector } from 'react-redux'.
6. Find where you're getting state from the store and change it to this:

```
const state = useSelector(state => state);
```

7. Run and test again. Verify that you're still able to read the state. It should look no different than getting state directly from the store.

But there is a huge difference! This selector not only gets you the state, but it also subscribes so that all changes to state will result in this component being redrawn.

Dispatching with the useDispatch hook

8. We will soon have a need to dispatch some actions to the store. With react-redux, we can use the useDispatch hook to create a dispatch method. Let's experiment with that right now.
9. Edit App.js. Inside the useEffect, go ahead and do this:

```
useEffect(() => {
  console.log("State is", state);
  store.dispatch({type:'DUMMY_ACTION'});
}, []);
```

Of course this does nothing useful; we haven't written the reducer to handle an action type of DUMMY_ACTION. But notice that it runs without error. If you debugged it, you'd see that the reducer is actually called!

You may have noticed that using this methodology, the store is still needed. But if we use react-redux's useDispatch hook we can completely eliminate the store from this component. Let's do that now.

10. Change the useEffect to read like this:

```
useEffect(() => {
  console.log("State is", state);
  dispatch({ type: 'DUMMY_ACTION' });  // <-- Remove 'store.'
}, []);
```

All you've done is remove "store." Notice two things. First, there are no longer any references to the store itself. You could remove the import if you wanted to. Second, There's a transpile error; it doesn't know what dispatch() is. Let's fix that next.

11. import useDispatch from react-redux.
12. Inside the App function at the very top, add this:

```
const dispatch = useDispatch();
```

    Notice that your error goes away and you're back to running again.

Bonus!! Seeing a change

13. Add this inside the App function but outside of the useEffect()

```
console.log('drawing', Math.random())
```

14. Run and test. Look at the console. You see a random number there -- one (1) random number.
15. Now edit your store.js and change the reducer to this:

```
const reducer = (state, action) => ({ ...state });  // <-- Spread the old state.
```

16. Now run and test again. How many random numbers did you get this time? It should be two.

See if you can explain why it got redrawn now. (Hint: useSelector redraws the component when what is being selected changes.) If you can't logic it out, Ask your instructor before the next lecture. There's some really good insights into redux and into React that you can get from this exercise!

In all the subsequent labs we'll be using react-redux's glue code since it is so much easier than coding it all brute force but keep in mind that you'll still learn all the tools you need to write without it if that is what you prefer.