React Redux

Download Demo Code

Goals

- Combine a Redux store with a React app
- Pass state from the store to a component with ***useSelector***
- Dispatch actions to the store with ***useDispatch***

React-Redux Setup

React-Redux

- We'll use a library called ***react-redux***
- Allows us to connect a store to a React app
- We need to wrap our app in a top-level component called ***<Provider>***
- In order to make this work, we need a reducer

Our Counting Reducer

Here's our reducer from before

demo/counter/src/rootReducer.js

```
const INITIAL_STATE = { count: 0 };

function rootReducer(state = INITIAL_STATE, action) {
  switch (action.type) {
    case "INCREMENT":
      return { ...state, count: state.count + 1 };

    case "DECREMENT":
      return { ...state, count: state.count - 1 };

    default:
      return state;
  }
}

export default rootReducer;
```

<Provider>

- ***react-redux*** gives us ***Provider*** component
  - ***Provider*** accepts a prop of a Redux store
- We should wrap our top-level ***App*** component in a ***Provider***

demo/counter/src/index.js

```
import React from "react";
import ReactDOM from "react-dom";

import App from "./App";

import rootReducer from "./rootReducer";
import { createStore } from "redux";
import { Provider } from "react-redux";

const store = createStore(rootReducer);

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

Connecting Components

Connecting to the Store

- We can access values from the store with the ***useSelector*** hook
- ***useSelector*** accepts a callback
- The callback has access to the store as its first argument
- The callback should return whatever data we want from the store

useSelector Example

demo/counter/src/FirstCounter.js

```
import React from "react";
import { useSelector } from "react-redux";

function FirstCounter() {
  // let's pull in the value of count from the store
  const count = useSelector(store => store.count);

  return (
    <div>
      <h2>The count is: {count}.</h2>
    </div>
  );
}

export default FirstCounter;
```

Dispatching to the Store

- ***useSelector*** reads from the store, but what about making changes?
- For this we can use another hook: ***useDispatch***
- ***useDispatch*** lets us dispatch actions to the store

useDispatch Example

demo/counter/src/SecondCounter.js

```
import React from "react";
import { useSelector, useDispatch } from "react-redux";

function SecondCounter() {
  const count = useSelector(st => st.count);
  const dispatch = useDispatch();
  const up = () => dispatch({ type: "INCREMENT" });
  const down = () => dispatch({ type: "DECREMENT" });
  console.log("COUNT", count);
  return (
    <div>
      <h2>The count is: {count}.</h2>
      <button onClick={up}> + </button>
      <button onClick={down}> - </button>
    </div>
  );
}

export default SecondCounter;
```

Data Flow

Data Flow with React-Redux

- Store is created, which dispatches an initial action
- Reducers returns the initial state
- ***useSelector*** runs for all components connected to store
  - Provides the data for these components
  - Triggers ***render***
- On dispatch, any connected components that receive new data from ***useSelector*** will re-render

How to Connect

- Not every component needs ***useSelector*** and ***useDispatch***!
  - Some will only need to read data
  - Some will only need to dispatch actions
- If you're mapping over an array to render components, continue to pass props directly from React
- Sometimes passing props down will be easier than having ***useSelector*** everywhere

Redux Dev Tools

The Redux Dev Tools are a great way to debug your react-redux applications

Redux Dev Tools

Including the Redux Dev Tools

src/index.js

```
const store = createStore(rootReducer,
  window.__REDUX_DEVTOOLS_EXTENSION__
  && window.__REDUX_DEVTOOLS_EXTENSION__()
);
```

- Once this is done, go to Chrome to see your Redux state!