Redux Options

Component Types

- Presentational components
  - get all their data through props
  - do not make REST calls
  - do not dispatch actions
  - typically do not maintain state
    unless it is for temporary UI data
- Container components
  - make REST calls
  - dispatch actions
  - get data from the Redux store
  - only render presentational components
    with the exception of wrapping divs
- For more, google
  Dan Abramov smart dumb components

React-Redux

- There are many ways to use Redux for managing application state
- One way is to use react-redux
  - billed as the "Official React bindings for Redux"
  - https://github.com/reactjs/react-redux
  - http://redux.js.org/docs/basics/UsageWithReact.html
- Includes a **`Provider`** component and a **`connect`** function
- To install
  - **`npm install --save react-redux`**

Provider

- Wrap topmost component in a **Provider** component

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


const store = createStore(...);

// Render this.
  <Provider store={store}>
    ... topmost component ...
  </Provider>
```

Connect

- Create "container components" that are "connected"
- The `connect` function is a "higher order component"
  - it takes a component and returns a new component

```
import {connect} from 'react-redux';
import React from 'react';

... definition of MyContainer ...

const mapStateToProps = state => {
  // Extract data from state and use it to
  // create an object where each key is a
  // prop to be passed to a presentational component.
  return props;
};
const mapDispatchToProps = dispatch => {
  // Return an object where each key is
  // the name of a function to be passed
  // as a prop to a presentational component.
  // These functions call dispatch to
  // dispatch an action to Redux.
  return props;
};

export default connect(mapStateToProps, mapDispatchToProps)(MyContainer);
```

Container Component Options

- Option #1 to load initial state
  - get data from Redux store using `mapStateToProps`
  - access this data in the component via props
- Option #2 to load initial state
  - make REST calls in `componentWillMount` lifecycle method, perhaps using Fetch API
  - wait for promises returned REST calls to resolve with the data
  - store data in the state of the component by calling `this.setState`
- To dispatch actions that that update the Redux store
  - return props from `mapDispatchToProps`
    whose values are functions that call `dispatch`
  - call these functions from component event handling methods

What Did We Gain?

- No need to pass data through the entire component hierarchy to provide it where it is needed
- Components that need data from the Redux store get it by being "connected" and using their `mapStateToProps` function
- No need to find a way to make the `dispatch` method of the Redux store available to all components that need to dispatch actions
- Components that need to dispatch actions can do so in methods of the object returned by their `mapDispatchToProps` function

What Did We Lose?

- With connected components it is not the case that all components are functions of their props
  - only presentational components fit this description
- Instead, connected components are functions of their props AND the current state of the Redux store
- This is impure from a functional programming perspective
- It treats the Redux store as one big global