Wiring to React components

This lesson introduces the react-redux module and how it is used to connect Redux with React components

WE'LL COVER THE FOLLOWING

Wiring to React components

If we talk about Redux in the context of React we almost always mean react-redux module. It provides two things that help connecting Redux to our components.

- `<Provider>` component - it's a component that accepts our store and makes it available for the children down the React tree via the React's context API. For example:

```
<Provider store={ myStore }>
  <MyApp />
</Provider>
```

We usually have a single place in our app where we use it.

- `connect` function - it is a function that does the subscribing for updates in the store and re-renders our component. It implements a higher-order component. Here is its signature:

```
connect(
  [mapStateToProps],
  [mapDispatchToProps],
  [mergeProps],
  [options]
)
```

`mapStateToProps` parameter is a function that accepts the current state and

must return a set of key-value pairs (an object) that are being sent as props to

our React component. For example:

```
const mapStateToProps = state => ({
  visible: state.visible
});
```

`mapDispatchToProps` is a similar one but instead of the `state`, it receives a `dispatch` function. Here is the place where we can define a prop for dispatching actions.

```
const mapDispatchToProps = dispatch => ({
  changeVisibility: value => dispatch(changeVisibility(value))
});
```

`mergeProps` combines both `mapStateToProps` and `mapDispatchToProps`. The props sent to the component gives us the opportunity to accumulate better props. Like for example if we need to fire two actions we may combine them to a single prop and send that to React. `options` accepts couple of settings that control how the connection works. In the following section, we will look into creating an app using Redux.

Let's use all we have learned about Redux to make a simple counter app in the next lesson