codecademy

Connect to React with React Redux

react-redux Package

The `react-redux` package, the official Redux-UI binding package for React, lets your React components interact with a Redux store without writing the interaction logic yourself. This allows an application to rely on Redux to manage the global state and React to render the UI based on the state.
Interactions may include reading data from a Redux store and dispatching actions to the store.

Install react-redux

The `react-redux` package is added to a React/Redux project by first installing it with `npm`.
A few of the resources imported from `react-redux` are:

- `Provider`
- `useSelector`
- `useDispatch`

```
npm install react-redux
```

The Provider Component

The `<Provider />` component makes the Redux store available to a nested child component. The store should be passed in as the `store` prop.
All child components of the `<Provider />` component can now use the resources provided by `react-redux` to access the Redux store including retrieving data and dispatching actions.

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Provider } from 'react-redux';

import { App } from './App';
import { createStore } from 'redux';

const store = createStore();

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

The useSelector() Hook

The `useSelector()` hook extracts state data from the Redux store using a selector function each time the state is updated. Any component nested within `<Provider />` may access state using `useSelector()`. Selectors used with `useSelector()` can be pre-defined functions or inline selectors.
When called within a React component `useSelector(selector)` accomplishes two things:

- Returns the data retrieved by the selector
- Subscribes the React component to changes in the store and forces a re-render if the selector's result changes

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

// Pre-defined selector function
const selectPosts = state => state.posts;

const App = (props) => {
  const posts = useSelector(selectPosts);

  // Alternatively, selectors can be used
inline
  // const posts = useSelector(state =>
state.posts);

  // code to render posts is omitted...
};
```

The useDispatch() Hook

The `useDispatch()` hook returns a reference to the `store.dispatch()` method. It must be used within a React component that is nested within the `<Provider />` component.
By convention, a React component defines `dispatch` and assigns the reference returned by `useDispatch()`. `dispatch()` can then be used within the component to dispatch action objects.

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

const MyComponent = () => {
  const dispatch = useDispatch();

  return (
    <button
        onClick={() => dispatch(
            { type: 'buttonClicked' }
    )} >
      Click Me
    </button>
  );
};
```

Selectors

In Redux, selectors are user-defined functions that extract specific pieces of information from a store state value. Selectors are pure functions that take the `state` as an argument and React components can use selectors to get specific data from the store.
By convention selector function names start with `select` and describe the data they retrieve from the store.

```
/*
state = {
    todos: [
    {id: 1, content: 'Work'},
    {id: 2, content: 'Shop'},
    {id: 3, content: 'Sleep'}
  ]
}
*/

// This selector retrieves the todos
array.
const selectTodos = state => state.todos;


// This selector retrieves an array of
todo ids.
const selectTodoIDs = state =>
state.todos
  .map(todo => todo.id);
```