BOSSCODER
ACADEMY

Quick Start to React

REDUX

7-DAY GUIDE

Disclaimer

Everyone learns uniquely.

Learn Redux in a structured manner and master it by practically applying your skills.

This Doc will help you with the same.

Day 1

Introduction to React Redux

What is React Redux?

React Redux is the official binding library for React and Redux. It enables React components to read data from a Redux store and dispatch actions to the store to update the state.

Benefits of using Redux with React

- **Predictable State:** Centralized state management allows for predictable state transitions.
- **Easier Debugging:** Redux DevTools can be used to inspect every action and state change.
- **Consistent Data Flow:** A unidirectional data flow makes the data more predictable and easier to manage.

Day 2

Core Concepts of Redux

Actions

Actions are payloads of information that send data from your application to your Redux store.

```
// action types
const ADD_TODO = 'ADD_TODO';

// action creators
const addTodo = (text) => ({
  type: ADD_TODO,
  payload: text,
});
```

Reducers

Reducers specify how the application's state changes in response to actions sent to the store.

```
// reducer function
const todos = (state = [], action) => {
  switch (action.type) {
    case ADD_TODO:
      return [...state, action.payload];
    default:
      return state;
  }
};
```

Store

The store holds the whole state tree of your application. The only way to change the state inside it is to dispatch an action on it.

```
import { createStore } from 'redux';
import todos from './reducers';

// create store
const store = createStore(todos);
```

Middleware

Middleware provides a third-party extension point between dispatching an action and the moment it reaches the reducer.

```
import { applyMiddleware, createStore } from 'redux';
import thunk from 'redux-thunk';
import rootReducer from './reducers';

const store = createStore(
  rootReducer,
  applyMiddleware(thunk)
);
```

Day 3

Setting Up Redux with React

Installing Redux and React-Redux

First, install the required packages.

```
npm install redux react-redux
```

Setting up the Redux store

Create a file ‘store.js’ to configure the store.

```
import { createStore } from 'redux';
import rootReducer from './reducers';

const store = createStore(rootReducer);

export default store;
```

Connecting Redux to a React application

Wrap your root component with the ‘Provider’ component to give it access to the Redux store.

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Provider } from 'react-redux';
import store from './store';
import App from './App';

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

Actions and Action Creators

Defining actions

Actions are defined as objects with a ‘type’ property.

```
const ADD_ITEM = 'ADD_ITEM';
 const addItem = (item) => ({
  type: ADD_ITEM,
  payload: item,
});
```

Creating action creators

Action creators are functions that create and return an action object.

```
const REMOVE_ITEM = 'REMOVE_ITEM';
 const removeItem = (id) => ({
  type: REMOVE_ITEM,
  payload: id,
});
```

Using action creators in components

Dispatch actions using 'dispatch' from 'useDispatch' hook.

```
import React from 'react';
import { useDispatch } from 'react-redux';
import { addItem } from './actions';

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

  const handleAddItem = (item) => {
    dispatch(addItem(item));
  };

  return (
    <button onClick={() => handleAddItem('New
Item')}>Add Item</button>
  );
};
```

Day 5

Reducers

Defining reducers

Reducers are functions that take the current state and an action, and return a new state.

```
const itemsReducer = (state = [], action) =>
{
  switch (action.type) {
    case ADD_ITEM:
      return [...state, action.payload];
    case REMOVE_ITEM:
      return state.filter(item => item.id !==
action.payload);
    default:
      return state;
  }
};
```

Combining reducers

Use ‘combineReducers’ to combine multiple reducers into one.

```
import { combineReducers } from 'redux';
import itemsReducer from './itemsReducer';
import userReducer from './userReducer';
 const rootReducer = combineReducers({
  items: itemsReducer,
  user: userReducer,
});
 export default rootReducer;
```

Handling actions in reducers

Reducers should handle each action type appropriately.

```
const userReducer = (state = {}, action) => {
  switch (action.type) {
    case 'SET_USER':
      return { ...state, user: action.payload
};
    default:
      return state;
  }
```

The Redux Store

Creating the store

Create a store with ‘createStore’.

```
import { createStore } from 'redux';
import rootReducer from './reducers';

const store = createStore(rootReducer);

export default store;
```

Providing the store to a React application

Use ‘Provider’ to make the Redux store available to the rest of your app.

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

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

Accessing the store in components

Use ‘useSelector’ and ‘useDispatch’ to interact with the store.

```
import { useSelector, useDispatch } from
'react-redux';
 const MyComponent = () => {
  const items = useSelector((state) =>
state.items);
  const dispatch = useDispatch();
  const handleAddItem = (item) => {
    dispatch(addItem(item));
  };
  return (
    <div>
      <button onClick={() =>
handleAddItem('New Item')}>Add Item</button>
      <ul>
        {items.map((item, index) => (
          <li key={index}>{item}</li>
        ))}
      </ul>
    </div>
  ); };
```

React-Redux Connect and Hooks

The connect function

'connect' is a higher-order function that connects a React component to the Redux store.

```
import { connect } from 'react-redux';
import { addItem } from './actions';
const mapStateToProps = (state) => ({
  items: state.items,
});
const mapDispatchToProps = (dispatch) => ({
  addItem: (item) => dispatch(addItem(item)),
});
const MyComponent = ({ items, addItem }) => (
  <div>
    <button onClick={() => addItem('New 
Item')}>Add Item</button>
    <ul>
{items.map((item, index) => (
        <li key={index}>{item}</li>
      ))}
    </ul>
  </div>
);

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

mapStateToProps and mapDispatchToProps

These functions help connect the Redux store to React components.

```
const mapStateToProps = (state) => ({
  items: state.items,
});
const mapDispatchToProps = (dispatch) => ({
  addItem: (item) => dispatch(addItem(item)),
});
export default connect(mapStateToProps,
mapDispatchToProps)(MyComponent);
```

Using useSelector and useDispatch hooks

These hooks provide an alternative to ‘connect’ for accessing the store.

```
import { useSelector, useDispatch } from
'react-redux';
const MyComponent = () => {
  const items = useSelector((state) =>
state.items);
  const dispatch = useDispatch();
  const handleAddItem = (item) => {
    dispatch(addItem(item));
  };
   return (
    <div>
      <button onClick={() =>
handleAddItem('New Item')}>Add Item</button>
      <ul>
        {items.map((item, index) => (
          <li key={index}>{item}</li>
        ))}
      </ul>
    </div>
  );};
```

Did you find it Useful?

Leave a comment!

Alamin CodePapa

@CodePapa360

FOLLOW FOR MORE

Like

Comment

Repost