CS-Web Programming

Redux

Assignment: 03

Team :

Laiba Asif (21i2560)

Submission date :4 may,2024

Introduction:

Redux is a JavaScript application's predictable state container. It acts as a central repository for an application's state, **simplifying** the management and manipulation of **state data** throughout the programme. Redux offers a unidirectional data flow and predictable state changes by adhering to the **Flux** architectural principles.

Importance of Redux:

Redux is a well-liked option for managing application state because of its many advantages:

> **1.** <u>Stable State Administration:</u> Redux makes the application state more predictable and logical by enforcing a single source of truth. Redux makes state changes clear and standardised, which improves the predictability of the application's behaviour.
>
> **2.** <u>A state that is centralised:</u> Redux centralises the application's state into a store, removing the need for complicated data transfer between components and enabling all components to access and edit state data. This lowers the possibility of inconsistent state throughout the application and streamlines state management.
>
> **3.** <u>Time Travel Debugging:</u> The design of Redux makes it possible to perform strong debugging operations, including time travel debugging. Debugging and problem solving are made easier for developers by the ability to replay operations and examine the application's state at various points in time.
>
> **4**. The capacity to scale Redux is ideal for big and complicated applications because of its tremendous scalability. Even as the application increases in size and complexity, developers can handle state logic effectively because to its modular structure and separation of responsibilities.

How Redux Works:

<u>Redux functions on the basis of three primary tenets:</u>

> <u>1. Single Verdict Source:</u> Redux stores the state of an application in a single JavaScript object known as the store. By doing this, the state is guaranteed to have a single source of truth, which facilitates management and manipulation.
>
> <u>2. Read-only status:</u>Redux's state is unchangeable and cannot be changed directly. Rather, sending actions—plain JavaScript objects with information about the state change—triggers state modifications.
>
> <u>3. Use Pure Functions to Make Changes:</u> Redux reducers are pure functions that define the actions that cause the state of the application to change. Reducers don't

change the original state; instead, they return a new state object based on the action and the existing state as input.

Example: Redux E-Commerce Shopping Cart

I'll utilise React and Redux to develop an e-commerce shopping cart. The application will allow users to add and remove goods from their carts as well as modify the amount of each product.

Step 1: Setting up the Redux Store

I'll implement actions and reducers in this step.

```
export const ADD_TO_CART = 'ADD_TO_CART';
export const REMOVE_FROM_CART = 'REMOVE_FROM_CART';
export const ADJUST_ITEM_QUANTITY = 'ADJUST_ITEM_QUANTITY';

export const addToCart = (product) => ({
  type: ADD_TO_CART,
  payload: {
    product,
  },
});

export const removeFromCart = (productId) => ({
  type: REMOVE_FROM_CART,
  payload: {
    productId,
  },
});

export const adjustItemQuantity = (productId, quantity) => ({
  type: ADJUST_ITEM_QUANTITY,
  payload: {
    productId,
    quantity,
  },
```

```
import { ADD_TO_CART, REMOVE_FROM_CART, ADJUST_ITEM_QUANTITY } from
'./actions';
```

```
const initialState = {
  cart: [],
};

const cartReducer = (state = initialState, action) => {
  switch (action.type) {
    case ADD_TO_CART:
      return {
        ...state,
        cart: [...state.cart, { ...action.payload.product, quantity: 1 }],
      };
    case REMOVE_FROM_CART:
      return {
        ...state,
        cart: state.cart.filter(item => item.id !== action.payload.productId),
      };
    case ADJUST_ITEM_QUANTITY:
      return {
        ...state,
        cart: state.cart.map(item =>
          item.id === action.payload.productId ? { ...item, quantity: action.payload.quantity } :
item
        ),
      };
    default:
      return state;
  }
};

export default cartReducer;
```

Step 2: Integrating Redux with React

Here i'll integrate Redux with React components to build the shopping cart interface.

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

const App = () => {
  return (
    <Provider store={store}>
      <div>
        <h1>E-commerce Shopping Cart</h1>
        <ShoppingCart />
      </div>
    </Provider>
  );
};

export default App;
```

```
// ShoppingCart.js
import React from 'react';
import { connect } from 'react-redux';
import { addToCart, removeFromCart, adjustItemQuantity } from './actions';

const ShoppingCart = ({ cart, addToCart, removeFromCart, adjustItemQuantity }) => {
  return (
    <div>
      <h2>Shopping Cart</h2>
      <ul>
        {cart.map(item => (
          <li key={item.id}>
            <span>{item.name} - ${item.price}</span>
            <button onClick={() => removeFromCart(item.id)}>Remove</button>
            <input
              type="number"
              value={item.quantity}
              onChange={(e) => adjustItemQuantity(item.id, parseInt(e.target.value))}
            />
          </li>
        ))}
      </ul>
    </div>
  );
};

const mapStateToProps = state => ({
  cart: state.cart,
});

const mapDispatchToProps = {
  addToCart,
  removeFromCart,
  adjustItemQuantity,
};

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

References:

- Official Redux documentation: Redux - A JS library for predictable and maintainable global state management | Redux
- React Redux library documentation: React Redux | React Redux (react-redux.js.org)