MODULE: 7

React – Applying Redux

1. What is Redux?

Redux is a state management library for JavaScript applications, commonly used with React. It helps manage application state in a predictable way using a single store. Redux follows three core principles:

* **Single source of truth** (one central store for state)
* **State is read-only** (state can only be changed by dispatching actions)
* **Changes are made with pure functions** (reducers define how state updates based on actions)

2. What is Redux Thunk used for?

Redux Thunk is a middleware that allows us to write asynchronous logic, such as API calls, inside Redux action creators. Without Thunk, Redux actions must be synchronous. Thunk enables us to dispatch actions after asynchronous operations are completed.

Example usage of Redux Thunk:

const fetchData = () => {

return async (dispatch) => {

dispatch({ type: "FETCH\_REQUEST" });

try {

const response = await fetch("https://api.example.com/data");

const data = await response.json();

dispatch({ type: "FETCH\_SUCCESS", payload: data });

} catch (error) {

dispatch({ type: "FETCH\_ERROR", payload: error });

}

};

};

3. What is a Pure Component? When to use Pure Component over Component?

A **PureComponent** in React is similar to a regular **Component**, but it implements **shouldComponentUpdate** with a shallow comparison of props and state. This helps prevent unnecessary re-renders when props or state do not change.

Use **PureComponent** when:

* The component's state and props are simple and don’t involve deep objects.
* You want to optimize performance by avoiding unnecessary re-renders.

Example:

import React, { PureComponent } from "react";

class MyComponent extends PureComponent {

render() {

console.log("Rendered!");

return <h1>Hello, {this.props.name}</h1>;

} }

4. What is the second argument that can optionally be passed to setState and what is its purpose?

The second argument to setState is a **callback function** that gets executed after the state has been updated and the component has re-rendered. It ensures that we perform actions that depend on the updated state.

Example:

this.setState({ count: this.state.count + 1 }, () => {

console.log("State updated:", this.state.count);

});

5. Table with Search Functionality in React

import React, { useState } from "react";

const TableWithSearch = () => {

const [search, setSearch] = useState("");

const data = [

{ id: 1, name: "John Doe", age: 25 },

{ id: 2, name: "Jane Smith", age: 30 },

{ id: 3, name: "Mike Johnson", age: 35 },

];

const filteredData = data.filter((item) =>

item.name.toLowerCase().includes(search.toLowerCase())

);

return (

<div>

<input

type="text"

placeholder="Search by name..."

value={search}

onChange={(e) => setSearch(e.target.value)}

/>

<table border="1">

<thead>

<tr>

<th>ID</th>

<th>Name</th>

<th>Age</th>

</tr>

</thead>

<tbody>

{filteredData.map((item) => (

<tr key={item.id}>

<td>{item.id}</td>

<td>{item.name}</td>

<td>{item.age}</td>

</tr>

))}

</tbody>

</table>

</div>

);

};

export default TableWithSearch;

6.Create Login registration with CRUD Application using API (Redux)

import React from "react";

import { BrowserRouter as Router, Route, Routes } from "react-router-dom";

import { Provider } from "react-redux";

import store from "./redux/store";

import Login from "./components/Login";

import Register from "./components/Register";

import Dashboard from "./components/Dashboard";

import EditUser from "./components/EditUser";

const App = () => {

return (

<Provider store={store}>

<Router>

<Routes>

<Route path="/" element={<Login />} />

<Route path="/register" element={<Register />} />

<Route path="/dashboard" element={<Dashboard />} />

<Route path="/edit/:id" element={<EditUser />} />

</Routes>

</Router>

</Provider>

);

};

export default App;