**JAVASCRIPT Assignment**

**MODULE: 18 State Management (Redux, Redux-Toolkit or Recoil)**

1. What is Redux, and why is it used in React applications? Explain the core concepts of actions, reducers, and the store.

Ans. Redux is a state management library used in React applications to manage global application state in a predictable way. It is particularly useful for large applications where managing state across multiple components becomes complex.

***Use of Redux in React applications:***

* Centralized State Management: Redux provides a single source of truth (store) that makes state management easier across components.
* Predictable State Changes: State updates follow a strict pattern using pure functions (reducers), making debugging easier.
* Easier Debugging & Testing: Redux DevTools allow time-travel debugging, and pure reducers make state updates predictable.
* Improved Performance: Reduces unnecessary re-renders by controlling how and when components update.

***Core Concepts of Redux***

1. **Actions**

* Actions are plain JavaScript objects that describe what should happen in the application.
* They have a type property (a string) and optionally a payload to send data.
* Example:

const addProduct = (product) => {

return {

type: "ADD\_PRODUCT",

payload: product

};

};

1. **Reducers**

* Reducers are pure functions that specify how the application's state should change in response to an action.
* They take the current state and an action, then return a new state.
* Example:

const productReducer = (state = [], action) => {

switch (action.type) {

case "ADD\_PRODUCT":

return [...state, action.payload]; // Adding new product to the state

default:

return state;

}

};

1. **Store**

* The store is a JavaScript object that holds the entire application state.
* It provides methods like:

getState() → Get the current state

dispatch(action) → Send an action to update the state

subscribe(listener) → Listen for state changes

* Example:

import { createStore } from "redux";

const store = createStore(productReducer);

store.dispatch(addProduct({ id: 1, name: "Laptop" }));

console.log(store.getState()); // Output: [{ id: 1, name: "Laptop" }]

1. How does Recoil simplify state management in React compared to Redux?

Ans. Recoil is a state management library for React that provides a more straightforward and flexible approach compared to Redux. It is designed to work seamlessly with React’s built-in state and context, making it easier to manage shared state across components without the complexity of Redux.

**How does Recoil works?**

Recoil introduces atoms and selectors to manage state more efficiently:

1. **Atoms (State Units)**

* Atoms are pieces of state that can be shared across components.
* Similar to useState, but globally accessible.
* Example:

import { atom } from "recoil";

const productState = atom({

key: "productState", // Unique ID

default: [], // Initial value

});

1. **Selectors (Derived State)**

* Selectors allow computed state based on atoms.
* Similar to Redux selectors but built-in and reactive.
* Example:

import { selector } from "recoil";

import { productState } from "./atoms";

const productCount = selector({

key: "productCount",

get: ({ get }) => get(productState).length,

});

1. **Usage in Components**

* Components subscribe to atoms/selectors using hooks (useRecoilState, useRecoilValue).
* Example:

import { useRecoilState } from "recoil";

import { productState } from "./atoms";

function ProductList() {

const [products, setProducts] = useRecoilState(productState);

return (

<div>

<h2>Products: {products.length}</h2>

<button onClick={() => setProducts([...products, { id: 1, name: "Laptop" }])}>

Add Product

</button>

</div>

);

}