Module 18 - ReactJS For Full Stack

THEORY ASSIGNMENT

14. State Management (Redux, Redux-Toolkit or Recoil):

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

Redux is a **state management library** often used in **React applications** to manage complex application state in a predictable and centralized way. It helps developers manage the state of their app across different components without prop-drilling (passing props down multiple component levels).

- **Why Redux is Used in React Applications:**
  - **Centralized State**: Redux keeps the application's state in a single store, making it easier to manage and debug.
  - **Predictable State Updates**: Changes to the state happen in a controlled manner using pure functions called reducers.
  - **Easier Debugging**: Redux offers time-travel debugging and logging with tools like Redux DevTools.
  - **Better Scalability**: As the application grows, Redux helps organize and scale state logic more effectively.
- **Core Concepts of Redux:**
  1. **Actions**
     - **Definition**: Plain JavaScript objects that describe *what happened*.
     - **Structure**:
     - {
     - type: 'ADD_TODO',

- payload: { id: 1, text: 'Learn Redux' }
- }
- **Purpose**: Actions are dispatched to signal that something has happened and the state might need to change.

2. **Reducers**
   - **Definition**: Pure functions that take the current state and an action, and return a new state.
   - **Example**:
   - const todoReducer = (state = [], action) => {
   - switch (action.type) {
   - case 'ADD_TODO':
   - return [...state, action.payload];
   - default:
   - return state;
   - }
   - };
   - **Purpose**: Reducers specify how the state changes in response to an action.
3. **Store**
   - **Definition**: An object that holds the entire state of the application.
   - **Created Using**:
   - import { createStore } from 'redux';
   - const store = createStore(todoReducer);
   - **Key Methods**:
     - getState() – Returns the current state.
     - dispatch(action) – Sends an action to the reducer.

- subscribe(listener) – Registers a callback to be invoked on every state change.

Question 2: How does Recoil simplify state management in React compared to Redux?

Recoil is a state management library developed by Facebook that simplifies and streamlines state handling in **React** applications, especially when compared to **Redux**.

🔹 **Key Ways Recoil Simplifies State Management Compared to Redux:**

1. No Boilerplate Code

- **Redux** often requires setting up actions, action creators, reducers, and a store.
- **Recoil** eliminates most of this boilerplate. You just define **atoms** (state units) and **selectors** (derived/computed state) directly where you need them.

2. Built for React

- **Recoil** is designed specifically for **React** and integrates directly with React's rendering and component tree.
- State values (atoms) are like React state but can be shared across components.

3. Granular State Updates

- In **Redux**, the entire reducer runs for any state change, and components often re-render more than necessary.
- **Recoil** allows components to **subscribe only to the specific atoms or selectors** they need, minimizing re-renders and improving performance.

4. Simpler Learning Curve

- **Redux** has a steeper learning curve, especially for beginners (middleware, immutability, reducers, etc.).
- **Recoil** uses more familiar concepts that feel like an extension of React's built-in state (e.g., useRecoilState feels like useState).

Redux example:

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

// Reducer
function itemsReducer(state = [], action) {
  switch (action.type) {
    case 'ADD_ITEM': return [...state, action.payload];
    default: return state;
  }
}

// Store setup, Provider, dispatch(), etc.
```

Recoil example:

```
// Atom
const itemListState = atom({
  key: 'itemListState',
  default: [],
});

// Inside component
```

```
const [items, setItems] = useRecoilState(itemListState);
setItems([...items, newItem]);
```