React Advanced (Hooks, Firebase, Redux)

Hooks:

Question 1: What are React hooks? How do use State() and useEffect() hooks work in functional components?

Answer: React Hooks are special functions that allow functional components to use state, lifecycle features, and other React capabilities without writing class components.

• use State(): Lets you create and manage state inside functional components. It returns a state value and a setter function.

```jsx
import { useState } from "react";

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <button onClick={() => setCount(count + 1)}>Count: {count}</button>
  );
}
```

• use Effect(): Allows you to perform side effects such as fetching data, updating the DOM, timers, subscriptions, etc. It runs after render and can re-run when dependencies change.

```jsx
import { useEffect } from "react";

useEffect(() => {
  console.log("Component mounted or updated");
}, []);
```

Question 2: What problems did hooks solve in React development? Why are hooks important?

Answer: Hooks solved many issues:

• Avoided complex class components.

• Allowed sharing logic without Higher-Order Components or render props.

• Made code cleaner, easier to test, and more reusable.

Hooks are important because they bring stateful logic to functional components and simplify component structure.

Question 3: What is useReducer? How is it used in React?

Answer: use Reducer is a hook for managing complex state logic. It works like Redux but inside a component.

• It takes a reducer function and an initial state.

• It returns the current state and a dispatch function.

```jsx
import { useReducer } from "react";

const initialState = { count: 0 };

function reducer(state, action) {
  switch (action.type) {
    case "increment":
      return { count: state.count + 1 };
    case "decrement":
      return { count: state.count - 1 };
    default:
      return state;
  }
}

function Counter() {
  const [state, dispatch] = useReducer(reducer, initialState);

  return (
    <>
      <p>{state.count}</p>
      <button onClick={() => dispatch({ type: "increment" })}>+</button>
      <button onClick={() => dispatch({ type: "decrement" })}>-</button>
    </>
  );
}
```

Question 4: What is the purpose of useCallback & useMemo Hooks? (Easy Explanation)

Answer: useCallback

- It **remembers a function**.
- React normally creates a **new function every time** the component runs.
- useCallback stops this and **keeps the same function**, unless something important changes.

  Use it when:
  You pass a function to a child component and you want to **stop unnecessary re-renders**.

useMemo

- It **remembers a value**.

- If you have a big calculation (slow or heavy), useMemo **saves the result**.
- It only re-calculates when needed.

Use it when:
You want to **avoid heavy calculations** every time the component renders.

Question 5: Difference between useCallback & useMemo?

Answer:

Feature	`useMemo`	`useCallback`
Purpose	Memoizes a **value** or result of computation	Memoizes a **function**
Use Case	Optimizing **expensive calculations** or derived data	Preventing **child component re-renders**
Returns	Cached value	Cached function
Example Use	Filtering a large list	Passing memoized functions to child components

Question 6: What is useRef? How does it work?

Answer:

useRef creates a mutable reference object whose .current value persists across renders.

Uses:

• Accessing DOM elements

• Storing values that shouldn’t trigger re-renders

• Holding previous values

```jsx
import { useRef } from "react";

function InputFocus() {
  const inputRef = useRef();

  const focusInput = () => {
    inputRef.current.focus();
  };

  return (
    <>
      <input ref={inputRef} />
      <button onClick={focusInput}>Focus Input</button>
    </>
  );
}
```

LAB EXERCISE

Task 1: Create a functional component with a counter using the useState()hook. Include buttons to increment and decrement the counter.

```jsx
import React, { useState } from "react";

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div style={{ textAlign: "center" }}>
      <h2>Counter: {count}</h2>

      <button onClick={() => setCount(count + 1)}>Increment</button>
      <button onClick={() => setCount(count - 1)}>Decrement</button>
    </div>
  );
}

export default Counter;
```

Task 2: Use the useEffect()hook to fetch and display data from an API when the component mounts.

```jsx
import React, { useEffect, useState } from "react";

function FetchData() {
  const [data, setData] = useState([]);

  useEffect(() => {
    fetch("https://jsonplaceholder.typicode.com/posts")
      .then((res) => res.json())
      .then((result) => setData(result));
  }, []); // empty array → run only once

  return (
    <div>
      <h2>API Data</h2>
      {data.slice(0, 5).map((item) => (
        <p key={item.id}>{item.title}</p>
      ))}
    </div>
  );
}

export default FetchData;
```

Task 3: Create react app to avoid re-renders in react application by useRef ?

```jsx
import React, { useRef, useState } from "react";

function AvoidRerender() {
  const renderCount = useRef(0);
  const [value, setValue] = useState("");

  renderCount.current += 1; // changes but does NOT re-render

  return (
    <div style={{ textAlign: "center" }}>
      <h2>Avoid Re-Renders</h2>

      <input
        type="text"
        onChange={(e) => setValue(e.target.value)}
        placeholder="Type something"
      />

      <p>Component Rendered: {renderCount.current} times</p>
    </div>
  );
}

export default AvoidRerender;
```

Redux:

THEORY EXERCISE

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

Answer: Redux is a state management library used in React application to manage global state in a predictable way.

- It helps when an app has a lot of components that need to share data.

- Data is usually passed from a parent to child components using props.

- But as the app grows, managing state across multiple components becomes complicated.

- Redux provides a centralized store where all components can access the necessary data without passing props manually.

The components of redux architecture

Action:

- It is like a message that tells Redux what to do.

- It is just a simple JavaScript object with a “type” property and sometimes extra data(payload)

Reducer:

- A reducer is a function that decides how the state should change when an action happens.

- It takes the current state and an action, then returns the new state.

Store:

- Store is like a big storage room that holds the entire state of the app.

- All components that need data can get it from the store instead of relying on props

LAB EXERCISE

Task 1: Create a simple counter application using Redux for state management. Implementactions to increment and decrement the counter.

counterSlice.jsx:

```jsx
import { createSlice } from "@reduxjs/toolkit";

const counterSlice = createSlice({
  name: "counter",
  initialState: { value: 0 },
  reducers: {
    increment: (state) => { state.value++ },
    decrement: (state) => { state.value-- }
  }
});

export const { increment, decrement } = counterSlice.actions;
export default counterSlice.reducer;
```

store.jsx:

```jsx
import { configureStore } from "@reduxjs/toolkit";
import counterReducer from "./counterSlice";

export const store = configureStore({
  reducer: {
    counter: counterReducer,
  },
});
```

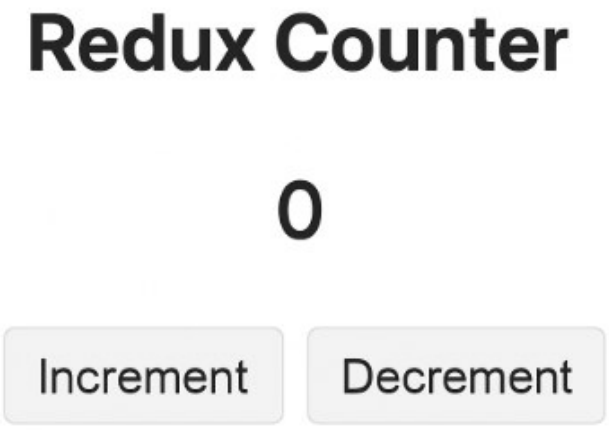
index.jsx:

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

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

App.jsx:

```jsx
import React from "react";
import { useSelector, useDispatch } from "react-redux";
import { increment, decrement } from "./counterSlice";

function App() {
  const count = useSelector((state) => state.counter.value);
  const dispatch = useDispatch();

  return (
    <div style={{ textAlign: "center", marginTop: "50px" }}>
      <h1>Simple Redux Counter</h1>
      <h2>{count}</h2>

      <button onClick={() => dispatch(increment())}>+</button>
      <button onClick={() => dispatch(decrement())}>-</button>
    </div>
  );
}

export default App;
```

Output:



Firebase:

THEORY EXERCISE

1.What is Firebase? What features does Firebase offer?

Answer:

Firebase is a **backend platform by Google** that helps you build web and mobile apps without creating your own server. It provides ready-made backend services.

Features of Firebase

- **Authentication** – Login using email, phone, Google, etc.
- **Firestore / Realtime Database** – Store and sync data in real time.
- **Cloud Storage** – Store images, videos, and files.
- **Hosting** – Host websites and apps fast and securely.
- **Cloud Messaging (FCM)** – Send push notifications.
- **Cloud Functions** – Run backend code without servers.
- **Analytics** – Track user behavior and app performance.