React Hooks:

Theory exercise

Question.1

What are react hooks? How do usestate() and useeffect() work?

Ans: **React Hooks** are special functions that allow **functional components** to use **state and lifecycle features** that were earlier only possible in class components.

useState()

- Used to manage **state** in functional components.
- Returns **state value + function to update it**.

```
const [count, setCount] = useState(0);
```

useEffect()

- Used to handle **side effects** like:
  - API calls
  - DOM updates
  - subscriptions
- Runs after render.

```
useEffect(() => {
  console.log("Component mounted");
```

}, []);

Q2. What problems did Hooks solve? Why are Hooks important?

Hooks solved:

1. **Complex class components**
2. **Code duplication**
3. **Confusing lifecycle methods**
4. **Difficult state sharing**

Why important?

- Cleaner code
- Reusable logic
- No this keyword
- Better performance & readability

Q3. What is useReducer? How is it used?

useReducer is used for **complex state logic** (similar to Redux).

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

Example:

function reducer(state, action) {

```
  switch (action.type) {
    case "INC": return state + 1;
    case "DEC": return state - 1;
    default: return state;
  }
}
```

Q4. Purpose of useCallback & useMemo

Hook	Purpose
useCallback	Memoizes **functions**
useMemo	Memoizes **computed values**

Used to **avoid unnecessary re-renders**.

Q5. Difference between useCallback & useMemo

useCallback	useMemo
Returns function	Returns value
Prevents function recreation	Prevents recalculation

Q6. What is useRef?

- Stores **mutable values**
- Does **not cause re-render**
- Access DOM elements

```
const inputRef = useRef();
```

LAB EXERCISE (HOOKS)

Task 1: Counter using useState

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

  return (
    <>
      <h2>{count}</h2>
      <button onClick={() => setCount(count + 1)}>+</button>
      <button onClick={() => setCount(count - 1)}>-</button>
    </>
  );
}
```

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

{count}

setCount(count + 1)}>+ setCount(count - 1)}>-); }

```
useEffect(() => {
  fetch("https://jsonplaceholder.typicode.com/users")
    .then(res => res.json())
    .then(data => setUsers(data));
}, []);
```

const count = useSelector(state => state.counter); const dispatch = useDispatch(); dispatch({ type: "INC" })}>+

```
const count = useSelector(state => state.counter);
const dispatch = useDispatch();

<button onClick={() => dispatch({ type: "INC"
})}>+</button>
```

const count = useSelector(state => state.counter); const dispatch = useDispatch(); dispatch({ type: "INC" })}>+

REACT ROUTER

THEORY

Q1. What is React Router?

React Router enables **navigation** in **Single Page Applications** without reloading the page.

Q2. Difference

Component	**Purpose**
BrowserRouter	Wraps app
Route	Maps path to component
Link	Navigation
Switch	Renders first match

LAB

Task 1: Home & About Routing

```
<BrowserRouter>
  <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/about" element={<About />} />
```

```
  </Routes>
</BrowserRouter>
```

```
} /> } />
```

```
<Link to="/">Home</Link>
<Link to="/about">About</Link>
<Link to="/contact">Contact</Link>
```

```
Home About Contact
```

JSON-SERVER & FIREBASE

THEORY

Q1. RESTful Web Services

REST is an architecture that uses **HTTP methods**:

- GET
- POST
- PUT
- DELETE

Q2. JSON-Server

JSON-Server creates a **fake REST API** using a JSON file.

npx json-server --watch db.json --port 3001

Q3. Fetch data using Axios

axios.get("http://localhost:3001/users")

Q4. Firebase

Firebase is a **Backend-as-a-Service** offering:

- Authentication
- Realtime Database

- Hosting
- Cloud functions

Q5. Error & Loading State Importance

- Improves UX
- Prevents crashes
- Shows proper feedback

LAB

Task 1: Fetch & Display Users

```
if (loading) return <p>Loading...</p>;
if (error) return <p>Error!</p>;
```

if (loading) return

Loading...

; if (error) return

Error!

;

Task 2: Firebase CRUD + Authentication

- Email/Password login
- Firestore CRUD
- Google Auth

signInWithPopup(auth, provider);

signInWithPopup(auth, provider);

Task 3: Loading Spinner

{loading && <Spinner />}

Task 3: Loading Spinner {loading && }

CONTEXT API

THEORY

Q1. What is Context API?

Context API manages **global state** without prop drilling.

Q2. createContext & useContext

const ThemeContext = createContext();

const theme = useContext(ThemeContext);

LAB

Task 1: Theme Toggle

const [theme, setTheme] = useState("light");

Task 2: Auth Context

{user ? <h1>Welcome</h1> : <Login />}

STATE MANAGEMENT

THEORY

Q1. Redux

Redux manages **global state** using:

- Actions
- Reducers
- Store

Q2. Recoil vs Redux

Recoil	**Redux**
Simple	Boilerplate
Hooks-based	Complex
Fine-grained state	Central store

LAB

Task 1: Redux Counter

dispatch({ type: "INC" });

Task 2: Recoil Todo App

```
const todoListState = atom({ key: "todos", default: [] });
```

Task 3: Redux Toolkit CRUD

```
createSlice({
  name: "users",
  reducers: {
    addUser: (state, action) => {
      state.push(action.payload);
    }
  }
});
```