Module 02

1. Define a basic React Router setup with two routes, one for the homepage ("/") and one for a contact page ("/contact").

→

- To use React Router, install react-router-dom
code:- > npm install react-router-dom

- Basic Setup:-

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

const Home = () => <h1> Welcome to us </h1>;
const Contact = () => <h1> Contact us </h1>;

function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/contact"
               element={<Contact />} />
      </Routes>
    </Router>
  );
}
export default App;
```

2. Write a simple React component that takes a prop called "name" and renders it inside a element.

→
- import React from "react";
const Greeting = ({name}) => {
return Hello, {name}! ;
};
export default Greeting;

- Used in a Parent Components:
<Greeting name="Utsav" />

3. What is the component tree in React?

→
- The component tree in React represents the hierarchical structure of components in an application.
- Example:-

APP
|- Navbar
| |- Logo
| |- Menu
|- Main Content
| |- Sidebar
| |- Article
|- Footer

4. Explain the concept of state in React components.

→ State is an object that holds dynamic data in a React component and determines how the component behaves and renders

```
import React, { useState } from "React";
const Counter = () => {
    const [count, setCount] = useState(0);
    return (
        <div>
            <p> Count : {count} </p>
            <button onClick = {() => setCount
                (count + 1)}> Increment
            </button>
        </div>
    );
};
export default Counter;
```

5. What are props in React and how are they used?

→ - Props (short for properties) are read-only data pased from a parent component to a child component.

- Example:

```
const UserProfile = ({name, age}) => {
    return (
        <div>
            <h2> Name : {name} </h2>
```

```
    <p> Age: {age} </p>
  </div>
  );
};

//usage
<UseProfile name="Utsav" age={22} />
```

6. Define React JS and explain its significance in web development.

→ Answer After this.

- React Hooks allow functional components to manage state and side effects without using class components.
- Common React Hooks & Use Cases:-

1. useState() :- Manages states in a functional components.
   - const [count, setCount] = useState(0);
   - Handling user input, toggling UI elements
2. useEffect() :- Performs side effects like data fetching or DOM updates.
   - useEffect(() => { console.log("Component Mounted"); }, []);
   - Fetching API Data, updating document title.
3. useRef() :- References DOM elements or stores value without re-rendering.
   - const InputRef = useRef(null);
   - Managing focus, storing previous state.

7. Discuss the different type of React Hook and their use cases.
→ Answer above.

Now ⑥.

→
- React JS is a JavaScript library developed by Facebook for building user interface (UI) especially for single-page applications (SPAs).
- It enables developers to create reusable UI components and manage application state efficiently.
- Significance in web Development:-

1. Component-Based Architecture
2. Virtual DOM
3. Unidirectional Data Flow
4. Hooks
5. Strong Community & Ecosystem.

8. Discuss the difference between props & state in React components.

→

Feature	Props	State
- Definition	Read-only data passed from parent to child.	Component-managed mutable data.
- Mutability	Immutable	Mutable.

- Who controls it?	Controlled by Parent Components	Managed within the components.
- Rerendering	Changing props triggers rerendering	Updating state triggers rerendering.
- Usage	Passing data to components.	Handling dynamic behavior.
- Example.	`const Greeting = ({name}) => <h1> hello, {name} </h1>; <Greeting name= "utsav" />;`	`const [count, setCount] = useState(2);`

9. Discuss the benefits of using functional components with React Hooks. over class components.

→

- 1. Less Code & Simplicity :- No need to define constructor, this or lifecycle methods.

- 2. Better Readability & maintainability :- functional components are easier to read.

- 3. Performance Optimizations :- Hooks allow memoization (useMemo(), useCallback()).

- 4. No this Keyword Issues:- Functional components do not require this.

- 5. Reusability of Logic:- Custom hooks make code modular and reusable.

- 6. Faster Execution:- Functional components execute faster as they don't extend React.Component.

- Example:-

- class Component with state:-

```
⇒ class Counter extends React.Component {
    state = { count: 0 };
    Increment = () => this.setState({ count:
        this.state.count + 1 });
    render() {
        return <button onClick={this.
            Increment}>{this.state.count}
        </button>;
    }
}
```

- Functional Components with Hooks:-

```
⇒ const Counter = () => {
    const [count, setCount] = useState(0);
    return <button onClick={() =>
        setCount(count + 1)}>{count}
        </button>;
};
```

10. Explain the concept of virtual DOM in React and its advantages.

→

- Definition :- The virtual DOM (vDOM) is an in memory representation of the real DOM. React updates the virtual DOM instead of directly modifying the real DOM, leading to efficient rendering.

- 1. State/props change :- Triggers rerender in virtual DOM.
2. Diffing Algorithm :- Compares the previous and new virtual DOM.
3. Efficient Updates :- Updates only the changed parts in the real DOM.

- Advantages :-
- Performance Improvement
- Efficient Updates.
- Faster Rendering
- Smoth UI changes.

- Example :-

```
const [count, setCount] = useState(21);
<button
    onClick = {
        () => setCount(count + 1)
    }
> Increment </button>
```

11. Compare and contrast class components and functional components in React.

Feature	class Components	Functional components.
- Syntax	Uses ES6 class syntax (extends React.Component)	Uses function syntax.
- State Management	Uses this.state & setState()	uses useState() Hooks.
- Lifecycle Methods	Uses lifecycle methods (componentDidMount, componentDidUpdate).	Uses useEffect() for side effects.
- Performance	Slightly slower due to complex structure.	Faster due to simpler execution.
- Hooks Support	Not compatible with hooks.	uses Hooks (useState, useEffect).

- Example:-

→ class Greeting extends React.Component {
render() {
return <h1> Hello, {this.props.name}
</h1>;
}
}

→ const Greeting = ({name}) => <h1> Hello,
{name} </h1>;

12. Create a React component that uses state to display a counter that increments by 1 every time a button is called.

→

```
import {useState} from "react";
const Counter = () => {
    const [count, setCount] = useState(0);
    return (
        <div>
            <h2> Counter: {count} </h2>
            <button onClick={() => setCount(
                count + 1)}> Increment
            </button>
        </div>
    );
};
export default Counter;
```

13. Create a React form component that captures user input for a username and password, and updates state accordingly.

→

```
Import {useState} From "react";
const LoginForm = () => {
    const [formData, setFormData]
        = useState({ username: "",
                     password: "" });
    const handleChange = (e) => {
        SetFormData({... formData, [e.
            target.name]: e.target.value});
    };
```

```
return(
  <form>
    <label> Username: </label>
    <input type="text" name="username"
      value={formData.username}
      onChange={handleChange}
    />

    <label> password: </label>
    <input type="password" name="password"
      value={formData.password}
      onChange={handleChange}
    />

    <button type="submit"> login </button>
  </form>
);
};
export default LoginForm;
```

14. Write a React component that fetches data from an API using the useEffect hook and display it in a list.

→

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

const DataFetcher = () => {
  const [data, setData] = useState([]);

  useEffect(() => {
    fetch("https://jsonplaceholder.typicode
      com/Posts")
```

```
.then((response) => response.json())
.then((json) => setData(json));
}, []);

return(
  <div>
    <h2>Fetched Data:</h2>
    <ul>
      {data.slice(0,5).map((item)=>(
        <li key={item.id}>{item.
          title}</li>
      ))}
    </ul>
  </div>
);
};
export default DataFetcher;
```

15. Develop a React component that uses React Context to manage user authentication state across multiple components.

→ Create context & Provider :-

```
import {createContext, useState} from "react";

export const AuthContext = createContext();
export const AuthProvider = ({children}) => {
  const [isAuthenticated, setIsAuthenticated]
    = useState(false);

  return(
```

```
    <AuthContext.Provider value={{
        isAuthenticated, setIsAuthenticated
    }}>
        <children />
    </AuthContext.Provider>
  );
};
```

→ Create component to consume context:-

```
import {useContext} from "react";
import {AuthContext} from "./AuthProvider";

const AuthState = () => {
  const {isAuthenticated,
         setIsAuthenticated} = useContext(AuthContext);

  return(
    <div>
      <h2> User is {isAuthenticated ?
        "Logged In" : "Logged Out"} </h2>
      <button onClick={() => setIsAuth-
        enticated(!isAuthenticated)}>
        {isAuthenticated ? "Logout" :
          "Login"}
      </button>
    </div>
  );
};
export default AuthState;
```

→ App.js :-

```
import {AuthProvider} from "./Authprovider";
import {Authstate} from "./Authstate";

function App(){
    return(
        <AuthProvider>
            <Authstate />
        </AuthProvider>
    );
};
export default App;
```

16. Demonstrate how React context can be used as an alternative to prop drilling for state management in deeply nested components

→

- when passing state down multiple levels in a component hierarchy, it can be cumbersome
-

```
const Parent = () => {
    const user = "Ravi";
    return <Child user={user} />;
};

const Child = ({user}) => {
    return <Grandchild user={user} />;
};

const Grandchild = ({user}) => {
    return <h2>Hello, {user}!</h2>
};
```

→ Using React Context (Efficient Approach):

Step-1 Create Context :-

```
import { createContext, useContext } from
                    "react";
const UserContext = createContext();
```

Step-2 Create a Provider :-

```
const UserProvider = ({children}) => {
    const user = "Utsav";
    return <UserContext.Provider value=
        {user}>{children}</UserContext.Provide
};
```

Step-3 Consume context in Grandchild

```
const Grandchild = () => {
    const user = useContext(UserContext);
    return <h2>Hello, {user}</h2>
};
```

Step-4 Use Provider in App

```
const App = () => {
    <UserProvider>
        <Grandchild />
    </UserProvider>
};
```

- Benefits of React context over Prop Drilling
  - No need to pass props through multiple components.
  - cleaner, scalable code.
  - centralized state management.

17. Discuss the error handling strategies in React applications, including React Error Boundaries and error logging.

→

1. Using Error Boundaries :- React provides Error Boundaries to catch JavaScript error in components.

- Example:-

```
Important React from "react";
class ErrorBoundary extends React.
    Component {
    constructor(props) {
        super(props);
        this.state = { hasError : false };
    }

    static getDerivedStateFromError(error)
        return { hasError : true };
    }

    componentDidCatch (error, info) {
        console.error ("Error caught:",
            error, info);
    }

    render() {
        if (this.state.hasError) {
            return <h2> Something went
            wrong. </h2>;
    }
```

```
        return this.props.children;
    }
}
export default ErrorBoundary;
```

- Using the Error Boundary in App:

```
<ErrorBoundary>
    <ComponentThatMayCrash />
</ErrorBoundary>
```

2. Try & Catch Blocks:-

```
const fetchData = async () => {
    try {
        const response = await fetch("https://
            jsonplaceholder.typicode.com/posts");
        const data = await response.json();
        console.log(data);
    } catch (error) {
        console.error("Failed to fetch data:",
            error);
    }
};
```

3. Logging Errors:- Errors should be logged to a service like Sentry, LogRocket or Firebase.

```
- const handleClick = () => {
    try {
        throw new Error("something went
            wrong");
```

```
} catch (error) {
    console.error("Error:", error);
}
};
```

18. Describe the useState and useEffect hooks in React and their usage patterns with examples.

→ useState Hook (manages state in Functional components):-

- The useState Hooks allows components to have local state.

- Example:- counter using useState

```
import {useState} from "react";
const Counter = () => {
  const [count, setCount] = useState(0);
  return (
    <div>
      <h2>Count: {count}</h2>
      <button onClick={() => setCount
        (count + 1)}>Increment
      </button>
    </div>
  );
};
export default Counter;
```

→ useEffect Hook (Handles side Effects in React)

- The useEffect Hook performs side effects like fetching data, setting timeouts, or interacting with the DOM.
- Example:- Fetch API Data

```
import {useState, useEffect} from "react";
const DataFetcher = () => {
  const [data, setData] = useState([]);

  useEffect (() => {
    fetch("https://jsonplaceholder.typicode.com/posts").
    .then((response) => response.json())
    .then((json) => setData(json));
  }, []);

  return (
    <div>
      <h2> Data Lists : </h2>
      <ul>
        {data.slice(0, 5).map((item) => {
          <li key={item.id}>
            {item.title}
          </li>
        })}
      </ul>
    </div>
  );
};
export default DataFetcher;
```

19. Explain the role of React Router in single-page applications (SPAs) and its key features.

→

- React Router is a library that enables client-side routing in React applications. Unlike traditional multi-page applications (MPAs), where each navigation request loads a new page, React Router allows SPAs to update the UI without reloading the page, improving performance and user experience.

- Example of Navigation without Reload:-

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

const Home = () => <h2>Home Page</h2>;
const About = () => <h2>About Page</h2>;
const App = () => {
  <Router>
    <nav>
      <Link to="/">Home</Link> |
      <Link to="/about">About</Link>
    </nav>
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/about" element={<About />} />
    </Routes>
  </Router>
};
export default App;
```

→ Key Features of React Router :-

1. Declarative Routing :- Uses <Routes> and <Route> components to define navigation paths.

2. Dynamic Routing :- Supports route parameters (/user/:id) for personalized pages.

3. Nested Routes :- Allows hierarchical route structures within components.

4. Redirects & Protected Routes :- Handle authentication-based access control.

5. History Management :- Uses browser history APIs for seamless navigation.

6. Lazy Loading Support :- Improves performance by loading components only when needed.

20. Build a React application that utilizes React Router for navigation, React Context for state management, and useEffect for fetching and displaying data from an external API.

Step-1 Setup React Router:-

```
Import {BrowserRouter as Router, Routes,
    Route, Link} from "react-router-dom";
Import {UserProvider} from "./UserContext";
Import Home from "./Home";
Import Profile from "./Profile";

const App = () => {
    <UserProvider>
        <Router>
            <nav>
                <Link to="/">Home</Link> |
                <Link to="/profile">Profile</Link>
            </nav>
            <Routes>
                <Route path="/" element=
                    {<Home/>} />
                <Route path="/profile" element=
                    {<Profile/>} />
            </Routes>
        </Router>
    </UserProvider>
);
export default App;
```

Step-2 Implement Context for User Management:-

```
Import {createContext, useContext, useState}
        from "react";
const UserContext = createContext();
export const UserProvider = ({children}) => {
```

```
const [user, setUser] = useState(
        "Utsav Vachhani");
return <UserContext.Provider
    value={{user, setUser}}>
    {children}</UserContext.Provider>
};
export const useUser = () => useContext
                (UserContext);
```

Step-3 Home.js :-

```
import {useEffect, useState} from "react
const Home = () => {
  const [users, setUsers] = useState([]);
  useEffect(() => {
    fetch("https://jsonplaceholder.
        typicode.com/users")
    .then((response) => response.json())
    .then((data) => setUsers(data));
  }, []);

  return (
    <div>
      <h2> GitHub Users </h2>
      <ul>
        {users.slice(0,5).map((user) => (
          <li key={user.id}>
            {user.name}
          </li>
        ))}
      </ul>
    </div>
```

```
    );
);
export default Home;
```

Step-4 Profile page :-

```
import {useUser} from "./UserContext";
const Profile = () => {
    const {user} = useUser();
    return <h2>Welcome, {user}!</h2>;
};
export default Profile;
```

×

@tair