**Explain the need and benefits of React Router**

In Single Page Applications (SPAs), navigating between different views or pages doesn’t reload the entire webpage. React Router is a standard routing library for React that enables this client-side navigation. It allows developers to define routes, map them to components, and manage navigation without full page reloads. The main benefits include faster navigation, better user experience, and the ability to create dynamic, nested, and protected routes. React Router also supports route-based code splitting, parameterized routing, and deep linking, making it ideal for building scalable web apps.

**Identify the Components in React Router**

React Router provides several key components to manage routing in a React application:

* **<BrowserRouter>**: The top-level component that enables history-based navigation using the HTML5 history API. It wraps the entire application.
* **<Routes>**: A container for all <Route> elements. It ensures only the first matching route is rendered.
* **<Route>**: Defines the path and the component to render when the URL matches the route.
* **<Link>**: Replaces anchor tags to create navigation links without reloading the page.
* **<Navigate>**: Programmatically redirects the user to another route.
* **useParams**, **useNavigate**, **useLocation**: Useful hooks for accessing route parameters, navigating programmatically, and reading the current location.

**List the types of Router Components**

React Router provides different types of routers based on the environment and use case:

1. **BrowserRouter**: Uses the HTML5 history API for clean URLs (e.g., /about). Best for modern web applications.
2. **HashRouter**: Uses URL hash (#) for routing (e.g., /#/about). Useful in static file hosting environments where server support is limited.
3. **MemoryRouter**: Stores history in memory. Ideal for testing and non-browser environments like React Native.
4. **StaticRouter**: Used for server-side rendering (SSR). It doesn’t change the URL and is used to render static content based on the route.

**Parameter passing via url**

React Router allows dynamic values to be passed through the URL using route parameters. These are defined in the route path using a colon (:) before the parameter name.

**Example of defining a parameter:**

<Route path="/user/:id" element={<User />} />

**Accessing the parameter in the component:**

import { useParams } from 'react-router-dom';

function User() {

const { id } = useParams();

return <h1>User ID: {id}</h1>;

}

In this example, navigating to /user/42 will render User ID: 42. This feature is useful for detail pages, dynamic routes, or passing values like user IDs, product IDs, etc.

**Index.js :-**

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

  <React.StrictMode>

    <App />

  </React.StrictMode>

);

reportWebVitals();

**App.js :-**

// src/App.js

import React from 'react';

import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';

import Home from './Home';

import TrainersList from './TrainersList';

import TrainerDetails from './TrainerDetails';

import trainers from './TrainersMock';

function App() {

  return (

    <BrowserRouter>

      <div>

        <nav style={{ marginBottom: '1rem' }}>

          <Link to="/" style={{ marginRight: '10px' }}>Home</Link>

          <Link to="/trainers">Trainers</Link>

        </nav>

        <Routes>

          <Route path="/" element={<Home />} />

          <Route path="/trainers" element={<TrainersList trainers={trainers} />} />

          <Route path="/trainers/:id" element={<TrainerDetails />} />

        </Routes>

      </div>

    </BrowserRouter>

  );

}

export default App;

**Home.js :-**

// src/Home.js

import React from 'react';

const Home = () => (

  <div>

    <h2>Welcome to Trainers App</h2>

    <p>Use the navigation bar to browse trainers.</p>

  </div>

);

export default Home;

**Trainer.js :-**

// src/Trainer.js

class Trainer {

  constructor(trainerId, name, email, phone, technology, skills) {

    this.trainerId = trainerId;

    this.name = name;

    this.email = email;

    this.phone = phone;

    this.technology = technology;

    this.skills = skills;

  }

}

export default Trainer;

**TrainerDetails :-**

// src/TrainerDetails.js

import React from 'react';

import { useParams } from 'react-router-dom';

import trainers from './TrainersMock';

const TrainerDetails = () => {

  const { id } = useParams();

  const trainer = trainers.find(t => t.trainerId === id);

  if (!trainer) return <p>Trainer not found</p>;

  return (

    <div>

      <h2>Trainer Details</h2>

      <p><strong>ID:</strong> {trainer.trainerId}</p>

      <p><strong>Name:</strong> {trainer.name}</p>

      <p><strong>Email:</strong> {trainer.email}</p>

      <p><strong>Phone:</strong> {trainer.phone}</p>

      <p><strong>Technology:</strong> {trainer.technology}</p>

      <p><strong>Skills:</strong> {trainer.skills.join(', ')}</p>

    </div>

  );

};

export default TrainerDetails;

**TrainersList.js :-**

// src/TrainersList.js

import React from 'react';

import { Link } from 'react-router-dom';

const TrainersList = ({ trainers }) => (

  <div>

    <h2>Trainers List</h2>

    <ul>

      {trainers.map(trainer => (

        <li key={trainer.trainerId}>

          <Link to={`/trainers/${trainer.trainerId}`}>{trainer.name}</Link>

        </li>

      ))}

    </ul>

  </div>

);

export default TrainersList;

TrainersMock.js :-

// src/TrainersMock.js

import Trainer from './Trainer';

const trainers = [

  new Trainer("T101", "John Doe", "john@example.com", "9876543210", "React", ["JavaScript", "Hooks"]),

  new Trainer("T102", "Jane Smith", "jane@example.com", "9876543211", "Angular", ["TypeScript", "RxJS"]),

  new Trainer("T103", "Mark Wilson", "mark@example.com", "9876543212", "Vue", ["Vuex", "Composition API"])

];

export default trainers;

Output :\_











