1. **React HandsOn-1**

* Define SPA and its benefits
* Define React and identify its working
* Identify the differences between SPA and MPA
* Explain Pros & Cons of Single-Page Application
* Explain about React
* Define virtual DOM
* Explain Features of React

**Define SPA and its benefits**

**SPA (Single Page Application)** is a web application that loads a single HTML page and dynamically updates the content as the user interacts with the app, without reloading the entire page from the server.

#### ****Benefits of SPA:****

**Faster user experience**: Only necessary data is loaded, not full pages.

**Reduced server load**: Fewer HTTP requests to the server.

**Smooth navigation**: No full-page refreshes, transitions feel like a native app.

**Better caching**: Data can be cached efficiently, improving performance.

**Seamless user experience**: Faster interactions and real-time updates.

**Define React and indentify its working**

**React** is a JavaScript library developed by Facebook for building fast, interactive user interfaces for web and mobile applications using reusable UI components.

#### ****How React Works:****

React uses a **component-based architecture**.

It utilizes a **Virtual DOM** to optimize updates.

When data changes, React calculates the difference (**diffing**) and efficiently updates only the necessary parts of the DOM (**reconciliation**).

Components can have **state** and **props**, which control how they render and behave.

**Difference between SPA and MPA**

· **Page Reload:**

**SPA:** Loads a single HTML page; no full-page reload on navigation.

**MPA:** Reloads the entire page from the server on each user action.

· **Speed:**

**SPA:** Faster after initial load due to dynamic content loading.

**MPA:** Slower navigation as each action reloads a new page.

· **User Experience:**

**SPA:** Smooth and app-like experience.

**MPA:** Feels more traditional and less fluid.

· **SEO (Search Engine Optimization):**

**SPA:** SEO is harder; needs server-side rendering or pre-rendering.

**MPA:** Better SEO since each page has a unique URL and metadata.

· **Routing:**

**SPA:** Handled on the client-side using JavaScript (e.g., React Router).

**MPA:** Handled by the server (each page has a route).

· **Initial Load Time:**

**SPA:** Higher (downloads JS bundle and app logic upfront).

**MPA:** Faster (loads only what's required for the current page).

· **Development Complexity:**

**SPA:** More complex (requires client-side routing, state management).

**MPA:** Simpler and more straightforward for small to medium projects.

· **Server Load:**

**SPA:** Lower server load after initial load; fewer HTTP requests.

**MPA:** Higher server load due to frequent page reloads.

· **Technology Stack:**

**SPA:** Built using frameworks like React, Angular, Vue.

**MPA:** Built using server-side technologies like PHP, ASP.NET, JSP.

**Pros and Cons of SPA**

#### ****Pros:****

Fast, seamless user experience.

Reduced server load.

Easy to build progressive web apps (PWA).

Better front-end control with frameworks like React or Vue.

#### ****Cons:****

Poor SEO by default (requires SSR).

Slower initial loading.

Requires JavaScript to be enabled.

More complex client-side routing and state management.

**Explain about React**

React is a **front-end JavaScript library** mainly used to build **user interfaces** for web applications.

#### ****Key Aspects:****

Developed by **Facebook**.

Uses a **Virtual DOM** for efficient rendering.

Emphasizes **reusable components**.

Supports **hooks** for functional component state and side-effects.

Enables **one-way data binding**.

Can be used with libraries like Redux for state management.

**Define Virtual DOM**

**Virtual DOM (VDOM)** is a lightweight JavaScript representation of the actual DOM.

#### ****How it works:****

React creates a VDOM tree of components.

On state change, a new VDOM is created.

The new and old VDOM are compared (diffing).

Only changed parts are updated in the real DOM (reconciliation).

This results in **faster updates** and better performance than directly manipulating the DOM.

**Features of React**

· **JSX** – A syntax extension that lets you write HTML in JavaScript.

· **Component-Based Architecture** – Build encapsulated components that manage their own state.

· **Virtual DOM** – Efficient UI rendering.

· **One-Way Data Binding** – Data flows from parent to child components only.

· **Hooks** – Add state and lifecycle methods to functional components.

· **High Performance** – Faster DOM updates using Virtual DOM and diffing.

· **Declarative UI** – Design views for each state, and React handles rendering.

· **Large Ecosystem** – Rich ecosystem with tools like Redux, React Router, Next.js.

**App.js**

import React from 'react'

import './App.css'

function App(){

  return (

    <h1>Welcome to the first session of React</h1>

  )

}

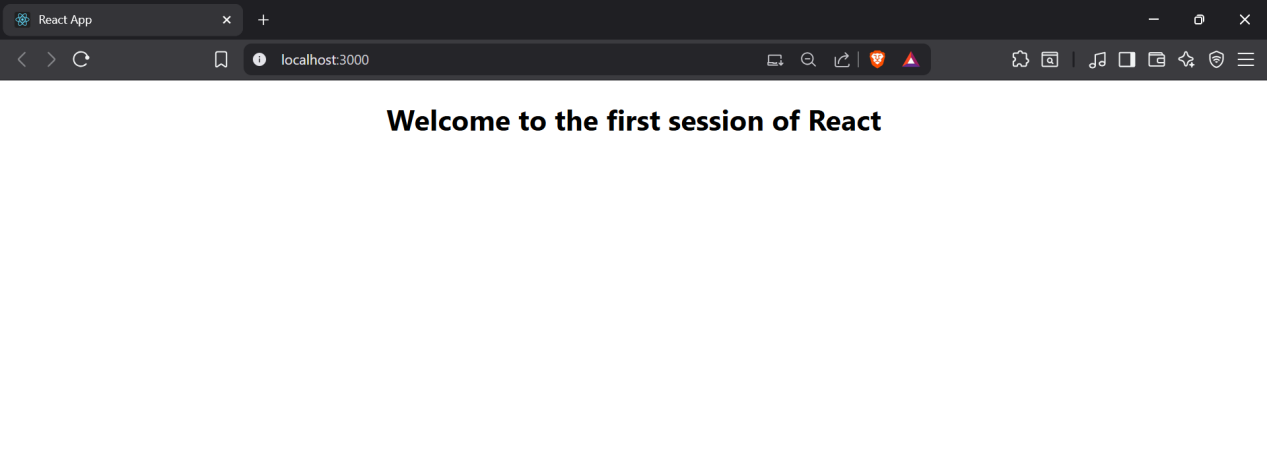
export default App

**App.css:**

h1{

  text-align: center;}

**Output:**



1. **React HandsOn-2**

* Explain React components
* Identify the differences between components and JavaScript functions
* Identify the types of components
* Explain class component
* Explain function component
* Define component constructor
* Define render() function

**Explain React components**

**React Components** are the building blocks of any React application. Each component represents a reusable piece of the UI (User Interface) and can manage its own state and logic.

#### Features:

Components let you split the UI into independent, reusable pieces.

They can be either **class-based** or **function-based**.

Each component can accept inputs called **props**.

Components can be **nested** inside one another.

**Difference between React Components and Javascript functions**

· **Purpose:**

**React Component:** Designed to render UI in React applications.

**JavaScript Function:** Used for general-purpose logic and operations.

· **Return Type:**

**React Component:** Returns **JSX** (which represents UI elements).

**JavaScript Function:** Returns any **data type** like numbers, strings, objects, etc.

· **Lifecycle Methods:**

**React Component:** **Class components** have lifecycle methods (e.g., componentDidMount, componentDidUpdate).

**JavaScript Function:** Does **not** have any lifecycle behavior.

· **React Integration:**

**React Component:** Automatically part of the **React rendering system**.

**JavaScript Function:** Not integrated with React by default.

· **Props:**

**React Component:** Accepts props to render dynamic data.

**JavaScript Function:** Accepts **parameters** (not React-aware).

· **State Management:**

**React Component:** Can manage state using **hooks (in function components)** or this.state (in class components).

**JavaScript Function:** Does **not manage state** internally.

**Identify the types of components:**

There are **two main types** of React components:

**Class Components**

**Function Components**

**Explain Class component:**

A **class component** is a React component defined using a JavaScript class that extends React.Component.

#### Features:

Has access to **lifecycle methods** (e.g., componentDidMount)

Can manage **state** directly via this.state

Uses this.props to access properties

**Explain Function Component:**

A **function component** is a simpler way to write components using JavaScript functions.

#### Features:

Stateless by default, but can use **React Hooks** (e.g., useState, useEffect) to manage state and lifecycle.

Easier to write and understand.

Preferred in modern React development.

**Define Component Constructor**

The **constructor** is a special method used in **class components** to initialize state and bind methods.

#### Usage:

Called before the component is mounted.

Initializes this.state and binds this context for methods.

**Define render() function:**

The render() method is used **in class components** to describe what the UI should look like.

#### Features:

It returns **JSX**.

Called every time the component's **state or props** change.

Must return a **single root element**.

**Home.js**

import React ,{Component} from 'react';

class Home extends Component {

  render() {

    return(

    <div>

    <h3>Welcome to the Home Page of Student Management Portal</h3>

    </div>

    );

  }

}

export default Home;

**About.js**

import React ,{Component} from 'react';

class About extends Component {

  render() {

    return(

    <div>

    <h3>Welcome to the About Page of Student Management Portal</h3>

    </div>

    );

  }

}

export default About;

**Contact.js**

import React ,{Component} from 'react';

class Contact extends Component {

  render() {

    return(

    <div>

    <h3>Welcome to the Contact Page of Student Management Portal</h3>

    </div>

    );

  }

}

export default Contact;

**App.js**

import React from 'react'

import Home from './Components/Home'

import './App.css'

import About from './Components/About';

import Contact from './Components/Contact';

function App (){

  return(

    <div className='container'>

      <Home/>

      <About/>

      <Contact/>

    </div>

  );

}

export default App

**App.css**

.container {

  display: flex;

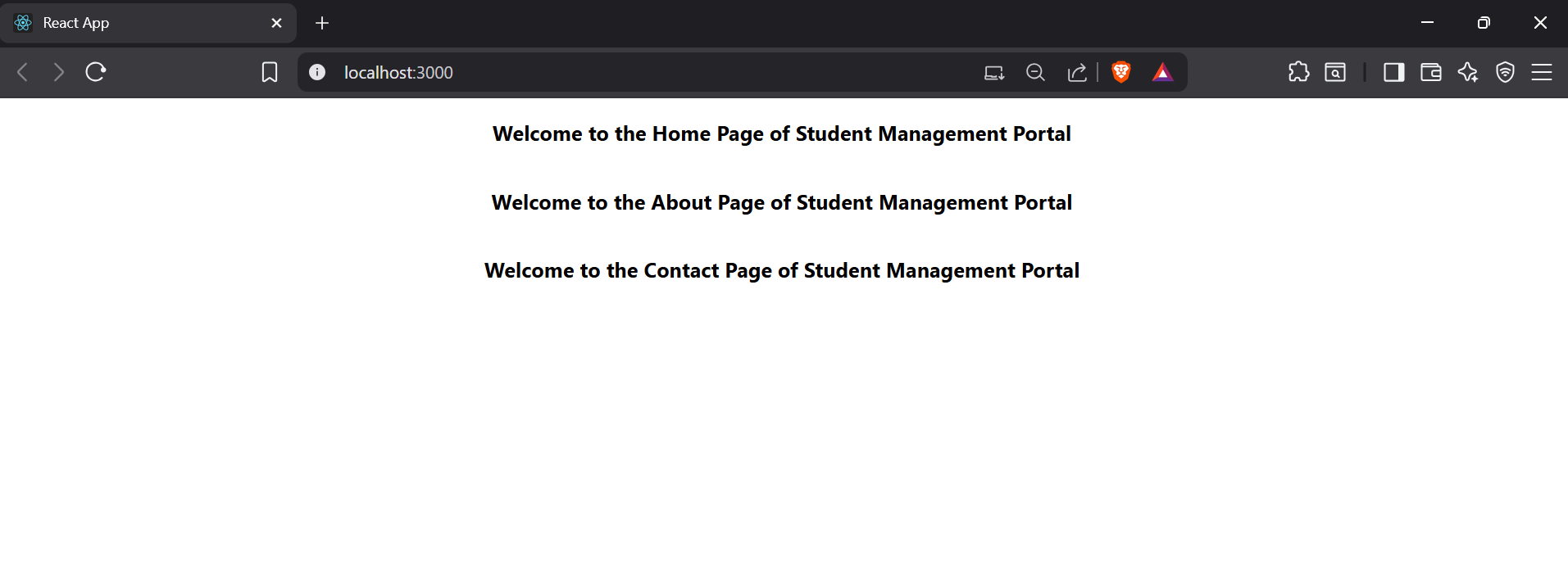
  flex-direction: column;

  align-items: center;

  height: 100vh;

}

**Output:**



1. **React HandsOn-3**

**CalculateScore.js**

import React from 'react'

import "../mystyle.css";

const percentToDecimal=(decimal)=>{

    return(decimal.toFixed(2)+'%');

}

const calcScore=(total,goal)=>{

    return percentToDecimal(total/goal)

}

export const CalculateScore = ({Name,School,total,goal}) => {

  return (

    <div className='formatstyle'>

        <h1>

            <font color="Brown">

                Student Details:

            </font>

        </h1>

        <div className='Name'> <b><span>Name:</span></b><span>{Name}</span></div>

        <div className='School'><b><span>School:</span></b><span>{School}</span></div>

        <div className='Total'><b><span>Total:</span></b><span>{total}</span><span>Marks</span></div>

        <div className='Score'><b>Score:</b>

        <span>

            {calcScore(total,goal)}

        </span>

        </div>

    </div>

  )

}

export default CalculateScore

**mystyle.css**

.Name{

    font-weight: 300;

    color:blue;

}

.School{

    color:crimson;

}

.Total{

    color:darkmagenta;

}

.formatstyle{

    text-align: center;

    font-size:large;

}

.Score{

    color:forestgreen;

}

**App.js**

import CalculateScore from "./components/CalculateScore";

function App(){

  return(

    <div>

      <CalculateScore Name={'Steeve'} School={'DNV Public School'} total={284}goal={3}/>

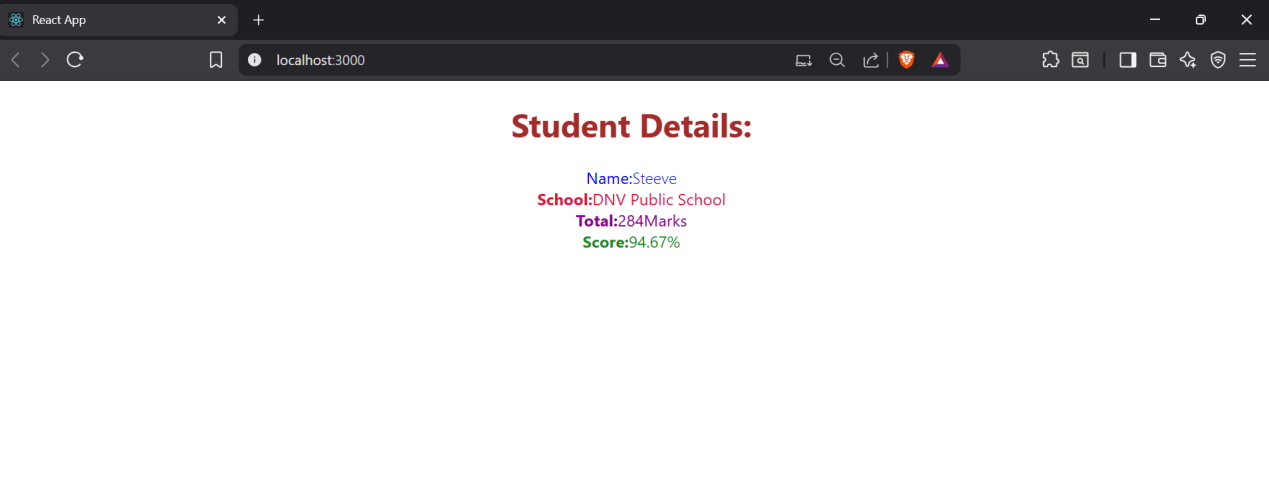
    </div>

  )

}

export default App

**Output:**



1. **React HandsOn-4**

**Objective Questions**

1. **Explain the need and Benefits of component life cycle**

The component lifecycle in React is essential because it gives developers structured control over how a component behaves during different stages of its existence. These stages include mounting (when a component is added to the DOM), updating (when state or props change), and unmounting (when it is removed from the DOM). Lifecycle methods allow you to run specific code at each of these phases—for example, fetching data after a component mounts or cleaning up resources before it unmounts. This structure helps in writing predictable, efficient, and maintainable code. The key benefits include optimizing performance, managing side effects like API calls or event listeners, and improving error handling, all of which contribute to building robust and scalable applications.

1. **Identify various life cycle hook methods**

React provides several lifecycle hook methods in class components that correspond to different phases of a component’s existence. In the mounting phase, the primary methods are constructor(), getDerivedStateFromProps(), render(), and componentDidMount(). These are used for setting up initial state, reading props, rendering UI, and performing side effects like data fetching. During the updating phase, which occurs when props or state change, the key methods are getDerivedStateFromProps(), shouldComponentUpdate(), render(), getSnapshotBeforeUpdate(), and componentDidUpdate(). These allow developers to optimize re-renders and respond to changes. In the unmounting phase, the componentWillUnmount() method is used to clean up timers, subscriptions, or other resources. Additionally, the error-handling phase includes componentDidCatch(), which catches runtime errors during rendering or in lifecycle methods and allows for graceful fallback UI.

1. **List the sequence of steps in rendering a component**

The sequence of lifecycle methods in React follows a well-defined order depending on the component’s phase. When a component is first mounted, the sequence is: constructor(), getDerivedStateFromProps(), render(), and then componentDidMount(). This order allows the component to initialize, read inputs, render content, and perform any post-render logic like fetching data. When the component is updated, either due to a change in props or state, the order is: getDerivedStateFromProps(), shouldComponentUpdate(), render(), getSnapshotBeforeUpdate(), and componentDidUpdate(). This sequence ensures that updates are controlled and optimized. Finally, when the component is unmounted, the componentWillUnmount() method is called to clean up. If an error occurs at any point, componentDidCatch() is invoked, allowing the app to recover gracefully. Understanding this sequence helps developers write precise and efficient React components.

**Post.js**

import React from 'react';

export default class Post extends React.Component {

  render() {

    const { title, body } = this.props;

    return (

      <div style={{ margin: '20px 0' }}>

        <h2>{title}</h2>

        <p>{body}</p>

      </div>

    );

  }

}

**Posts.js**

import React from 'react';

import Post from './Post';

export default class Posts extends React.Component {

  constructor(props) {

    super(props);

    this.state = {

      posts: [],

      hasError: false

    };

  }

  loadPosts = () => {

    fetch('https://jsonplaceholder.typicode.com/posts')

      .then(response => response.json())

      .then(data => this.setState({ posts: data }))

      .catch(error => {

        console.error("Error fetching posts:", error);

        this.setState({ hasError: true });

      });

  }

  componentDidMount() {

    this.loadPosts();

  }

  componentDidCatch(error, info) {

    alert("Something went wrong: " + error.message);

  }

  render() {

    const { posts, hasError } = this.state;

    if (hasError) {

      return <h2>Something went wrong while loading posts.</h2>;

    }

    return (

      <div>

        <h1>Blog Posts</h1>

        {posts.map(post => (

          <Post key={post.id} title={post.title} body={post.body} />

        ))}

      </div>

    );

  }

}

**App.js**

import React from 'react';

import Posts from './Posts';

function App() {

  return (

    <div className="App">

      <Posts />

    </div>

  );

}

export default App;

Output

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

1. **React HandsOn-5**

**CohortDetails.module.css**

.box {

  width: 300px;

  display: inline-block;

  margin: 10px;

  padding: 10px 20px;

  border: 1px solid black;

  border-radius: 10px;

}

dt {

  font-weight: 500;

}

**CohortDetails.js**

import React from 'react';

import styles from './CohortDetails.module.css';

const CohortDetails = ({ name, startDate, endDate, status }) => {

const statusStyle = {

color: status === 'ongoing' ? 'green' : 'blue'

};

return (

<div className={styles.box}>

<h3 style={statusStyle}>{name}</h3>

<dl>

<dt>Start Date:</dt>

<dd>{startDate}</dd>

<dt>End Date:</dt>

<dd>{endDate}</dd>

<dt>Status:</dt>

<dd>{status}</dd>

</dl>

</div>

);

};

export default CohortDetails;

**App.js**

import React from 'react';

import CohortDetails from './CohortDetails';

function App() {

  return (

    <div>

      <CohortDetails name="React Bootcamp" startDate="2025-07-01" endDate="2025-07-31" status="ongoing" />

      <CohortDetails name="Node.js Training" startDate="2025-06-01" endDate="2025-06-30" status="completed" />

    </div>

  );

}

export default App;

Output

A screenshot of a computer

AI-generated content may be incorrect.

1. **React HandsOn-6**

**Trainer.js**

export default 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;

  }

}

**TrainersMock.js**

import Trainer from './Trainer';

const trainers = [

  new Trainer(1, "Alice Johnson", "alice@example.com", "1234567890", "React", ["React", "Redux", "Hooks"]),

  new Trainer(2, "Bob Smith", "bob@example.com", "0987654321", "Node.js", ["Node.js", "Express", "MongoDB"]),

  new Trainer(3, "Charlie Brown", "charlie@example.com", "1112223333", "Java", ["Java", "Spring Boot", "Microservices"])

];

export default trainers;

**Home.js**

import React from 'react';

const Home = () => {

  return (

    <div>

      <h2>Welcome to Cognizant Academy Trainers Portal</h2>

      <p>This portal helps manage trainers and their expertise.</p>

    </div>

  );

};

export default Home;

**TrainerList.js**

import React from 'react';

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

const TrainerList = ({ trainers }) => {

  return (

    <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 TrainerList;

**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.toString() === id);

  if (!trainer) {

    return <h2>Trainer not found</h2>;

  }

  return (

    <div>

      <h2>{trainer.name}</h2>

      <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;

**App.js**

import React from 'react';

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

import Home from './Home';

import TrainerList from './TrainerList';

import TrainerDetails from './TrainerDetails';

import trainers from './TrainersMock';

function App() {

  return (

    <Router>

      <div>

        <nav>

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

          <Link to="/trainers" style={{ margin: '10px' }}>Trainers</Link>

        </nav>

        <hr />

        <Routes>

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

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

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

        </Routes>

      </div>

    </Router>

  );

}

export default App;

Output

**A screenshot of a computer

AI-generated content may be incorrect.**