**WEEK 6:**

**Hands-on Assessment:**

**1. Define SPA and its benefits**

**Q:** What is SPA?  
**A:** **SPA (Single-Page Application)** is a web app that loads a single HTML page and dynamically updates the content without reloading the entire page.

**Benefits:**  
 Faster navigation (no full-page reloads)  
 Better user experience (like a desktop app)  
 Reduced server load after initial load  
 Easy to cache & handle offline features  
 Smooth transitions between pages

**2. Define React and identify its working**

**Q:** What is React?  
**A:** **React** is a **JavaScript library** developed by Facebook for building fast, reusable, and interactive **UI components** for web applications.

**How does it work?**

* React uses **components** to build UI.
* It uses a **Virtual DOM** to detect changes and update only the required parts of the UI instead of reloading the whole page.
* It follows a **unidirectional data flow**.

**Identify the differences between SPA and MPA**

| **Feature** | **SPA (Single Page Application)** | **MPA (Multi Page Application)** |
| --- | --- | --- |
| **Page Loading** | Loads once & updates dynamically | Reloads full page on every request |
| **Speed** | Faster after initial load | Slower due to repeated full-page loads |
| **Server Requests** | Few API calls | Multiple server requests per page |
| **User Experience** | Smooth, app-like | Traditional web experience |
| **SEO** | Harder (requires SSR) | Easier for SEO |
| **Example** | Gmail, Facebook | Amazon, Wikipedia |

**4. Explain Pros & Cons of Single-Page Application**

**Pros:**  
 Fast & smooth navigation  
 Reduced bandwidth usage after initial load  
 Better UX (like a native app)  
 Easier front-end & API separation

**Cons:**  
 Initial load can be slow  
 SEO is challenging (needs extra setup)  
 Browser back/forward buttons handling can be tricky  
 Heavier JavaScript required

**5. Explain about React**

**Q:** Why use React?  
**A:** React is a **component-based library** that allows developers to build **reusable UI elements**, manage **state efficiently**, and update the DOM **fast using Virtual DOM**. It’s widely used for SPAs.

Key points:

* Developed by Facebook
* Open-source & widely adopted
* Works well with other libraries/frameworks
* Helps build scalable UI

**6. Define Virtual DOM**

**Q:** What is Virtual DOM?  
**A:** **Virtual DOM** is an in-memory lightweight copy of the **real DOM**.

* When something changes, React updates the **Virtual DOM first**, then compares it with the previous version (**diffing**), and updates **only the changed parts** in the real DOM.

**Benefits:**  
 Faster updates  
 Improves performance  
 Avoids unnecessary DOM manipulation

**7. Explain Features of React**

**Component-Based Architecture** – Build reusable UI components  
 **Virtual DOM** – Improves performance  
 **Unidirectional Data Flow** – Predictable state management  
 **JSX (JavaScript XML)** – Write HTML inside JS easily  
 **Declarative UI** – Describe what UI should look like  
 **Fast Rendering** – Updates only necessary parts of the DOM  
 **Huge Community & Ecosystem** – Lots of libraries & tools

**Q. Create a new React Application with the name “myfirstreact”, Run the application to print “welcome to the first session of React” as heading of that page.**

* **Solution:**

# 1. Install Node.js first (manual)

# 2. Install create-react-app globally

npm install -g create-react-app

# 3. Create React app

npx create-react-app myfirstreact

# 4. Go to folder

cd myfirstreact

# 5. Open in VS Code

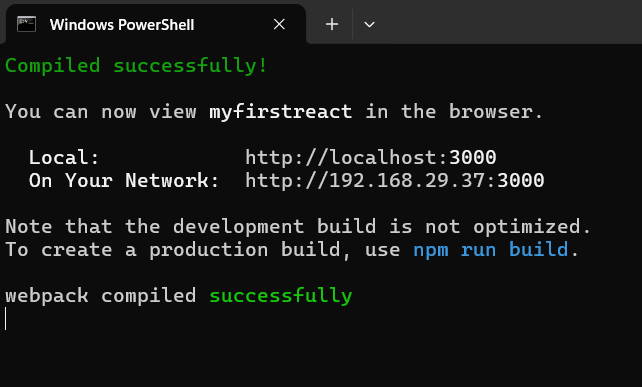
code .

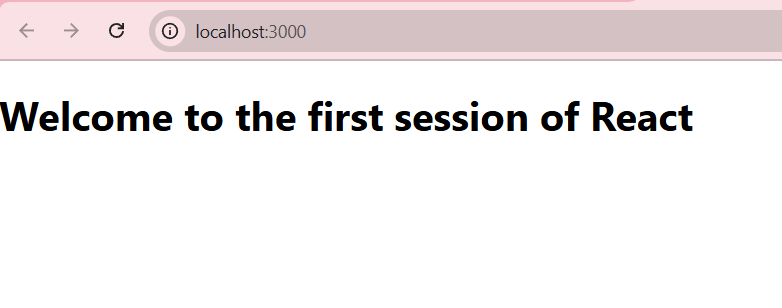
# 6. Edit App.js

# 7. Start the app

npm start

* **OUTPUT:**





**1. Explain React components**

**Q:** What are React components?  
**A:** React components are reusable, independent pieces of UI that define how a part of the application should look and behave. Each component can have its own logic, state, and styling, and can be combined to build complex UIs. Components help in maintaining modularity and reusability in React applications.

**2. Identify the differences between components and JavaScript functions**

**Q:** What is the difference between React components and JavaScript functions?  
**A:**

| **React Components** | **JavaScript Functions** |
| --- | --- |
| Used to define reusable UI elements | Used to perform a specific task or calculation |
| Can return JSX (UI elements) | Return primitive values or objects |
| Can manage state and lifecycle methods (in class components) | Do not have state or lifecycle by default |
| Can be rendered in the DOM | Cannot directly render UI without integration |
| Can be either class-based or function-based | Only function-based |

**3. Identify the types of components**

**Q:** What are the types of React components?  
**A:** React components are mainly of two types:

1. **Class Components** – Defined using ES6 classes, can use state and lifecycle methods.
2. **Function Components** – Defined using functions, initially stateless but can use state and lifecycle using React Hooks.

**4. Explain class component**

**Q:** What is a class component in React?  
**A:** A class component is a React component created using an ES6 class. It extends React.Component and must have a render() method that returns JSX. Class components can maintain their own internal state and use lifecycle methods like componentDidMount, componentDidUpdate, and componentWillUnmount.

Example:

javascript

CopyEdit

class Welcome extends React.Component {

render() {

return <h1>Hello, {this.props.name}</h1>;

}

}

**5. Explain function component**

**Q:** What is a function component in React?  
**A:** A function component is a simpler way to define a React component using a JavaScript function. It accepts props as arguments and returns JSX. Function components were initially stateless, but with React Hooks, they can now manage state and lifecycle.

Example:

javascript

CopyEdit

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

**6. Define component constructor**

**Q:** What is a component constructor in React?  
**A:** In a React class component, the constructor is a special method used to initialize the component’s state and bind event handlers. It is called before the component is mounted.

Example:

javascript

CopyEdit

constructor(props) {

super(props);

this.state = { count: 0 };

}

**7. Define render() function**

**Q:** What is the render() function in React?  
**A:** The render() function is a required method in class components that returns the JSX (UI) to be displayed on the screen. It should be a pure function, meaning it should not modify the component state or interact with the browser directly. It is called every time the component’s state or props change.

Example:

javascript

CopyEdit

render() {

return <h1>Hello World</h1>;

}

**Q. Create a react app for Student Management Portal named StudentApp and create a component named Home which will display the Message “Welcome to the Home page of Student Management Portal”. Create another component named About and display the Message “Welcome to the About page of the Student Management Portal”. Create a third component named Contact and display the Message “Welcome to the Contact page of the Student Management Portal”. Call all the three components.**

**Step 1: Create the React Project**

Open terminal and run:

npx create-react-app studentapp

This will create a new React app named **studentapp**.  
 After creation, move into the folder:

cd studentapp

**Step 2: Create Components Folder**

Inside the **src/** folder, create a new folder called **Components**.  
In VS Code:

**Step 3: Create Home Component**

Inside src/Components/, create a new file named **Home.js**.

import React from 'react';

function Home() {

return (

<div>

<h1>Welcome to the Home page of Student Management Portal</h1>

</div>

);

}

export default Home;

**Step 4: Create About Component**

Inside src/Components/ **About.js**.

import React from 'react';

function About() {

return (

<div>

<h1>Welcome to the About page of the Student Management Portal</h1>

</div>

);

}

export default About;

**Step 5: Create Contact Component**

Inside src/Components/ **Contact.js**.

import React from 'react';

function Contact() {

return (

<div>

<h1>Welcome to the Contact page of the Student Management Portal</h1>

</div>

);

}

export default Contact;

**Step 6: Edit App.js to Call All Components**

In **src/App.js**

import React from 'react';

// Importing the components

import Home from './Components/Home';

import About from './Components/About';

import Contact from './Components/Contact';

function App() {

return (

<div>

<Home />

<About />

<Contact />

</div>

);

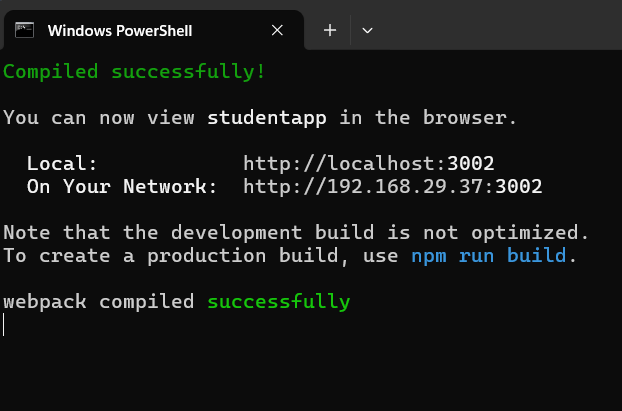
}

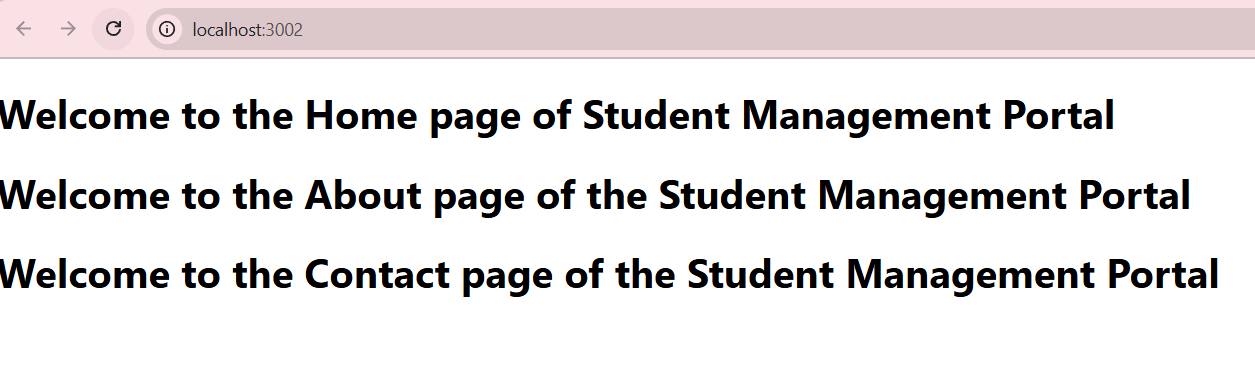
export default App;

**Step 7: Run the App**

npm start

**Step 8: Output**





**1. Explain React components**

**Q:** What are React components?  
**A:** React components are the building blocks of a React application. They are reusable, independent pieces of code that describe how a part of the user interface (UI) should appear and behave. Components can manage their own data (state), receive data through props, and return JSX that defines the UI. By combining multiple components, complex UIs can be created efficiently.

**2. Identify the differences between components and JavaScript functions**

**Q:** What is the difference between React components and JavaScript functions?

| **React Components** | **JavaScript Functions** |
| --- | --- |
| Used to create UI elements in a React app | Used for performing calculations or operations |
| Can return JSX (which gets rendered in the browser) | Returns values like strings, numbers, or objects |
| Can maintain state (class components) and use lifecycle methods | Stateless, they do not have lifecycle by default |
| Can receive props to display dynamic data | Can receive parameters but cannot directly render UI |
| Can be either class-based or function-based | Always function-based |

**3. Identify the types of components**

**Q:** What are the types of components in React?  
**A:** React components are mainly of two types:

1. **Class Components** – Defined using ES6 classes, support state and lifecycle methods.
2. **Function Components** – Defined using JavaScript functions. Initially stateless but with React Hooks, they can also manage state and lifecycle.

**4. Explain class component**

**Q:** What is a class component in React?  
**A:** A class component is a React component created using the ES6 class syntax. It extends React.Component and must have a render() method that returns JSX. Class components can have their own internal state and use lifecycle methods such as componentDidMount, componentDidUpdate, and componentWillUnmount.

Example:

class Welcome extends React.Component {

render() {

return <h1>Hello, {this.props.name}</h1>;

}

}

**5. Explain function component**

**Q:** What is a function component in React?  
**A:** A function component is a simpler React component written as a JavaScript function. It accepts props as an argument and returns JSX. Initially, function components were stateless, but with React Hooks (like useState and useEffect), they can now manage state and lifecycle features.

Example:

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

**6. Define component constructor**

**Q:** What is a component constructor in React?  
**A:** In a class component, the constructor is a special method used to initialize the component’s state and bind event handlers. It is called before the component is mounted. When using a constructor, super(props) must be called to properly inherit this.props from the parent class.

Example:

constructor(props) {

super(props);

this.state = { count: 0 };

}

**7. Define render() function**

**Q:** What is the render() function in React?  
**A:** The render() function is a required method in class components that returns JSX to define what should appear in the UI. It is called every time the component’s state or props change. The render() method should be pure, meaning it should not modify state or directly interact with the browser.

Example:

render() {

return <h1>Hello World</h1>;

}

**Q. Create a react app for Student Management Portal named scorecalculatorapp and create a function component named “CalculateScore” which will accept Name, School, Total and goal in order to calculate the average score of a student and display the same.**

**Step 1: Create React Project**

In (VS Code terminal or cmd):

npx create-react-app scorecalculatorapp

cd scorecalculatorapp

**Step 2: Create Components Folder**

Inside **src**, create a folder named **Components**.  
Then inside it, create a file named **CalculateScore.js**.

**Step 3: Code for CalculateScore.js**

inside src/Components/CalculateScore.js

import React from 'react';

import '../Stylesheets/mystyle.css'; // Import the CSS file

function CalculateScore(props) {

// Destructure props

const { name, school, total, goal } = props;

// Calculate average score

const averageScore = total / goal;

return (

<div className="score-container">

<h2>Student Score Details</h2>

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

<p><strong>School:</strong> {school}</p>

<p><strong>Total Marks:</strong> {total}</p>

<p><strong>Goal:</strong> {goal}</p>

<p className="average"><strong>Average Score:</strong> {averageScore.toFixed(2)}</p>

</div>

);

}

export default CalculateScore;

**Step 4: Add Styles**

Inside **src**, create a folder named **Stylesheets**.  
Inside it, create a file named **mystyle.css**.

.score-container {

background-color: #f4f4f4;

border: 1px solid #ccc;

padding: 20px;

margin: 20px;

border-radius: 10px;

text-align: left;

width: 300px;

}

.score-container h2 {

color: #333;

text-align: center;

}

.score-container p {

font-size: 16px;

margin: 5px 0;

}

.average {

color: green;

font-weight: bold;

}

**Step 5: Edit App.js**

Now **src/App.js**,

import React from 'react';

import CalculateScore from './Components/CalculateScore';

function App() {

return (

<div>

<h1>Welcome to Student Management Portal</h1>

{/\* Pass props for student details \*/}

<CalculateScore

name="John Doe"

school="ABC High School"

total={450}

goal={5}

/>

<CalculateScore

name="Jane Smith"

school="XYZ Public School"

total={380}

goal={4}

/>

</div>

);

}

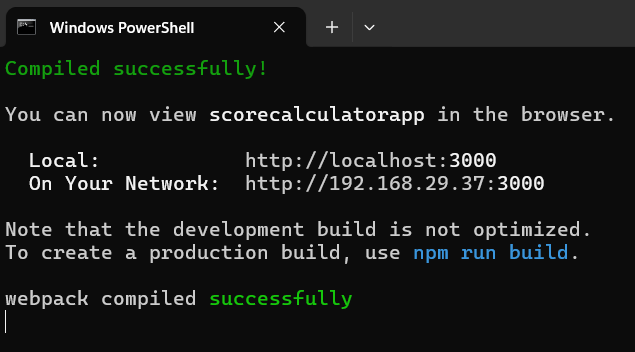
export default App;

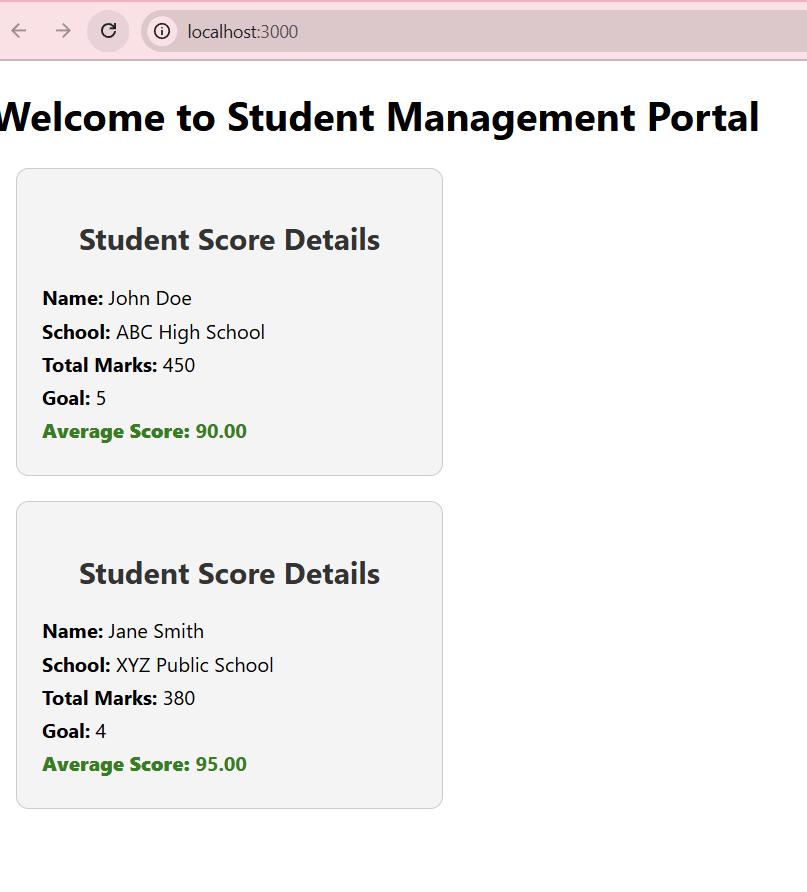
**Step 6: Run the Project**

cd scorecalculatorapp

npm start

* **OUTPUT:**





**Q: Explain the need and benefits of component life cycle**  
**A:** The component lifecycle in React defines the stages a component goes through from creation to removal from the DOM. It is needed to perform specific actions at the right time during these stages. For example, you might need to fetch data when a component mounts or clean up resources when it unmounts. The benefits include better control over the component behavior, optimized performance by managing updates efficiently, the ability to handle side effects like API calls, and ensuring proper cleanup of resources to prevent memory leaks.

**Q: Identify various life cycle hook methods**  
**A:** React provides several lifecycle hook methods categorized into three phases:

1. **Mounting Phase** (when the component is created and added to the DOM)
   * constructor()
   * static getDerivedStateFromProps()
   * render()
   * componentDidMount()
2. **Updating Phase** (when the component is re-rendered due to state or props change)
   * static getDerivedStateFromProps()
   * shouldComponentUpdate()
   * render()
   * getSnapshotBeforeUpdate()
   * componentDidUpdate()
3. **Unmounting Phase** (when the component is removed from the DOM)
   * componentWillUnmount()
4. **Error Handling Phase**
   * componentDidCatch()
   * static getDerivedStateFromError()

**Q: List the sequence of steps in rendering a component**  
**A:** When rendering a component, React follows these steps in order:

1. **Initialization**: The component is created, and the constructor is called to initialize state and props.
2. **Mounting**:
   * static getDerivedStateFromProps() is called before render.
   * render() method is called to return JSX.
   * The component is added to the DOM.
   * componentDidMount() is called after the component is mounted.
3. **Updating** (when state or props change):
   * static getDerivedStateFromProps() is called.
   * shouldComponentUpdate() decides whether to re-render.
   * render() updates the UI.
   * getSnapshotBeforeUpdate() captures information before DOM updates.
   * componentDidUpdate() is called after DOM updates.
4. **Unmounting**:
   * componentWillUnmount() is called before removing the component from the DOM.

**Q.Create a new react application using *create-react-app* tool with the name as “blogapp**”

**Step 1: Create React App**

Run command in your terminal:

npx create-react-app blogapp

Move into the folder:

cd blogapp

**Step 2: Open in VS Code**

code .

**Step 3: Create Post.js**

Inside the **src** folder, create a file named **Post.js**.

class Post {

constructor(id, title, body) {

this.id = id;

this.title = title;

this.body = body;

}

}

export default Post;

**Step 4: Create Posts.js Component**

Inside **src**, create another file **Posts.js**.

**Step 5: Full Posts.js Code**

import React, { Component } from 'react';

import Post from './Post';

class Posts extends Component {

constructor(props) {

super(props);

// Initialize state with empty posts list

this.state = {

posts: [],

error: null

};

// Bind loadPosts method

this.loadPosts = this.loadPosts.bind(this);

}

// Step 6: Method to fetch posts using Fetch API

loadPosts() {

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

.then(response => {

if (!response.ok) {

throw new Error('Network response was not ok');

}

return response.json();

})

.then(data => {

// Convert fetched data into Post objects

const postsData = data.map(

p => new Post(p.id, p.title, p.body)

);

this.setState({ posts: postsData });

})

.catch(error => {

this.setState({ error: error.message });

alert('Error fetching posts: ' + error.message);

});

}

// Step 7: componentDidMount hook calls loadPosts

componentDidMount() {

this.loadPosts();

}

// Step 9: componentDidCatch to catch rendering errors

componentDidCatch(error, info) {

alert('Something went wrong: ' + error);

console.error('Error details:', info);

}

// Step 8: Render method to display posts

render() {

const { posts } = this.state;

return (

<div>

<h1>Blog Posts</h1>

{posts.length === 0 ? (

<p>Loading posts...</p>

) : (

posts.map(post => (

<div key={post.id} style={{ marginBottom: '20px' }}>

<h2>{post.title}</h2>

<p>{post.body}</p>

</div>

))

)}

</div>

);

}

}

export default Posts;

**Step 10: Add Posts Component to App.js**

In **src/App.js**

import React from 'react';

import Posts from './Posts';

function App() {

return (

<div>

<Posts />

</div>

);

}

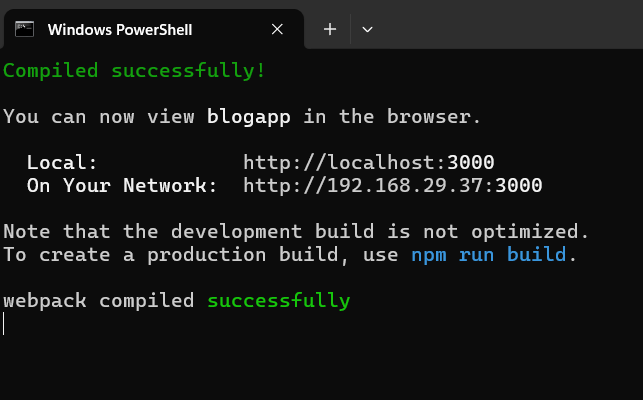
export default App;

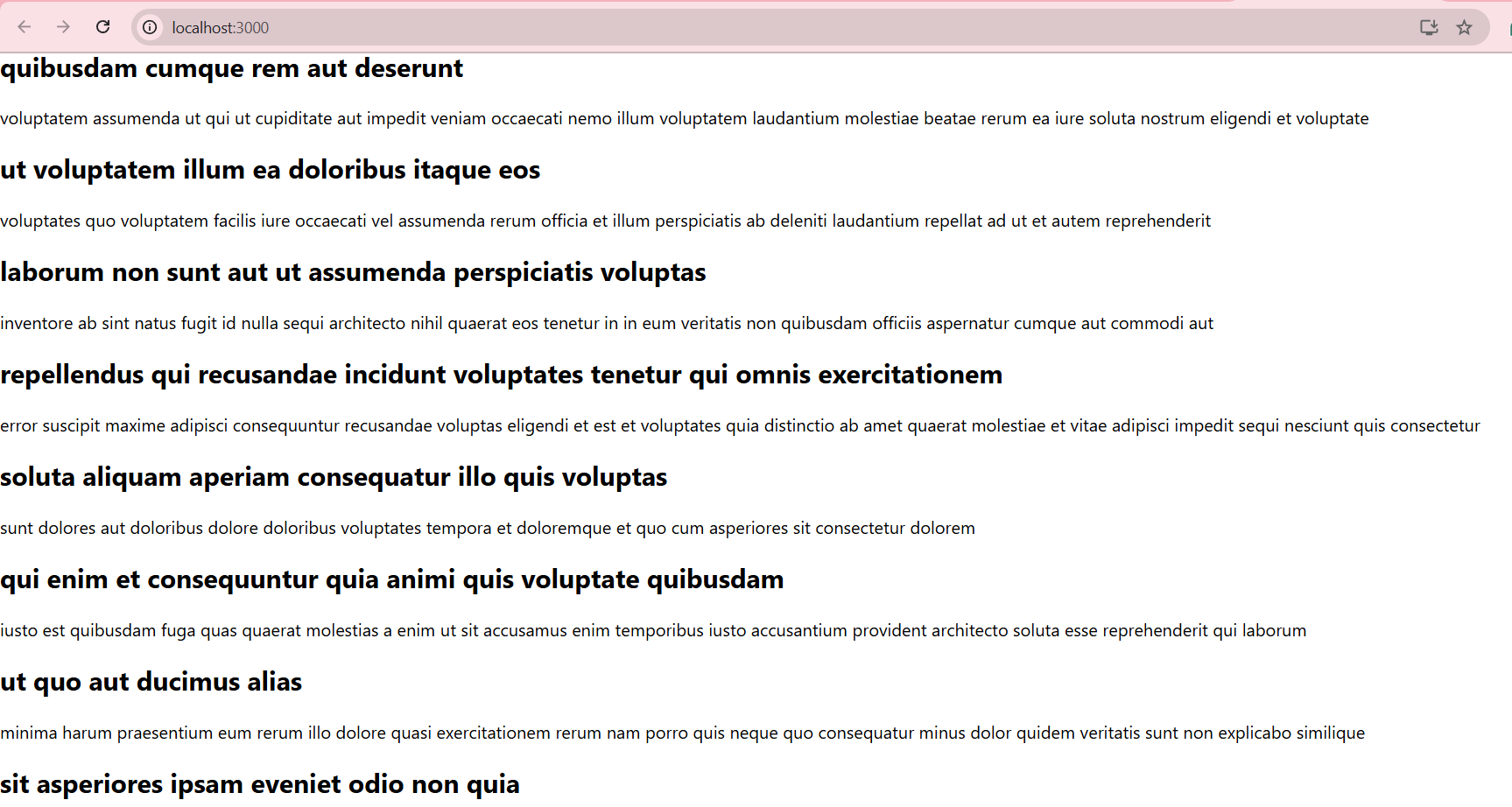
**Step 11: Run the App**

In the terminal, run:

npm start

* **Output:**





**Q. Cognizant Academy teams want to maintain a list of trainers along with their expertise in a SPA using React as the technology. You are assigned the task of creating this React app.**

**1. Create Trainer.js**

Inside **src/Trainer.js** define the Trainer model class:

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;

**2. Create TrainersMock.js**

Inside **src/TrainersMock.js**,

import Trainer from "./Trainer";

const trainers = [

new Trainer(

1,

"Syed Khaleelullah",

"khaleelullah@cognizant.com",

"97676516962",

".NET",

["C#", "SQL Server", "React", ".NET Core"]

),

new Trainer(

2,

"John Doe",

"johndoe@cognizant.com",

"9876543210",

"Java",

["Java", "Spring Boot", "Hibernate", "React"]

),

new Trainer(

3,

"Jane Smith",

"janesmith@cognizant.com",

"9123456789",

"Frontend",

["HTML", "CSS", "JavaScript", "React"]

)

];

export default trainers;

**3. Create TrainersList.js**

import React from "react";

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

import trainers from "./TrainersMock";

function TrainersList() {

return (

<div>

<h2>Trainers List</h2>

<ul>

{trainers.map((trainer) => (

<li key={trainer.trainerId}>

{/\* Link to TrainerDetails page with ID \*/}

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

</li>

))}

</ul>

</div>

);

}

export default TrainersList;

**4. Create Home.js**

import React from "react";

function Home() {

return (

<div>

<h2>Welcome to My Academy Trainers App</h2>

<p>Navigate to the Trainers List to view trainer details.</p>

</div>

);

}

export default Home;

**5. Update App.js**

import React from "react";

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

import Home from "./Home";

import TrainersList from "./TrainersList";

import TrainerDetails from "./TrainerDetails";

function App() {

return (

<Router>

<h1>My Academy Trainers App</h1>

{/\* Navigation Links \*/}

<nav>

<Link to="/">Home</Link> |{" "}

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

</nav>

{/\* Routes \*/}

<Routes>

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

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

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

</Routes>

</Router>

);

}

export default App;

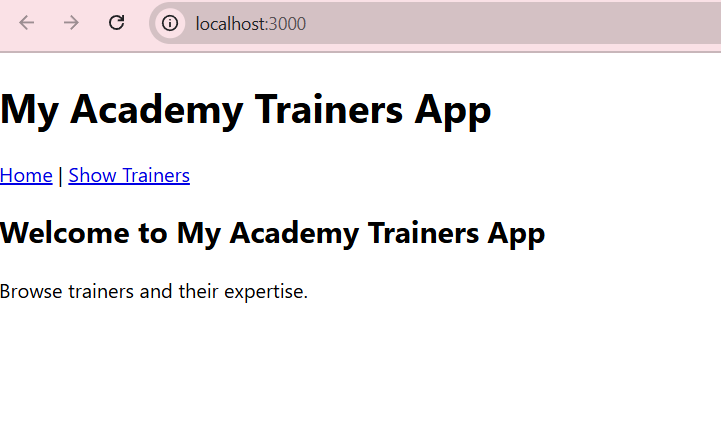
**6. Install React Router DOM**

npm install react-router-dom

**7. Run the App**

npm start

* **OUTPUT:**

****