1. ReactJS-HOL:-
2. Define SPA and its benefits

**SPA (Single Page Application)** is a web application that loads a single HTML page and dynamically updates content without refreshing the entire page.

**Benefits:**

* Faster user experience since only parts of the page update
* Smooth transitions without full reloads
* Reduced server load
* Can work offline with proper caching

1. Define React and identify its working

**React** is a JavaScript library for building user interfaces using reusable components.

**How it works:**

* Uses **components** to build UI
* Uses **JSX** to write HTML-like code in JavaScript
* Creates a **virtual DOM** to efficiently update the real DOM by only changing parts that need updating
* Supports **one-way data flow** from parent to child
* Uses **state** and **props** to manage and pass data

1. Identify the differences between SPA and MPA

| **Feature** | **SPA (Single Page Application)** | **MPA (Multi Page Application)** |
| --- | --- | --- |
| Page reload | No full reload; updates content dynamically | Full reload of new pages on navigation |
| User experience | Faster, smoother transitions | Slower, page refresh visible |
| Server requests | Fewer server requests after initial load | New request for every page navigation |
| Complexity | More complex on client-side | Simpler, traditional web apps |
| SEO | Needs extra setup for SEO | SEO is straightforward |

1. Explain Pros & Cons of Single-Page Application

**Pros:**

* Fast and responsive UI
* Reduced server load after initial page load
* Better user experience similar to desktop apps
* Works offline with proper caching

**Cons:**

* Initial loading time can be longer
* SEO can be challenging without server-side rendering
* More complex JavaScript code management
* Browser history and navigation handling need extra care

1. Explain about React

React is a popular JavaScript library developed by Facebook for building user interfaces with reusable UI components.

It allows developers to build complex, interactive UIs efficiently using JSX syntax and virtual DOM technology for optimized updates.

1. Define virtual DOM

The **virtual DOM** is a lightweight, in-memory copy of the real DOM. React uses it to compare changes in UI state and update only the necessary parts of the actual DOM, improving performance.

### Explain Features of React

· **Component-Based:** Build UIs using reusable components.

· **JSX Syntax:** Write HTML-like code in JavaScript.

· **Virtual DOM:** Efficient UI updates by diffing virtual and real DOM.

· **One-Way Data Binding:** Data flows from parent to child.

· **Declarative:** Describe UI in terms of states; React manages updates.

· **State Management:** Components can manage their own data.

· **Reusable Components:** Write code once, reuse it everywhere.

· **Strong Community and Ecosystem:** Many libraries and tools.

Create our First React App:-

* Open your **Command Prompt** - npm install -g create-react-app

This gives us the **toolbox** to create React apps.

#### 2. 🏗️ Create a New App

Type this:

npx create-react-app myfirstreact

#### 3. 📂 Go into Your App Folder

cd myfirstreact

#### 4. Open the App in VS Code

Type this:

code .

#### 5.Edit the App

Go to the src folder → Open the App.js file.

Delete everything inside App.js, then write **this new code**:

import React from 'react';

function App() {

return (

<div>

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

</div>

);

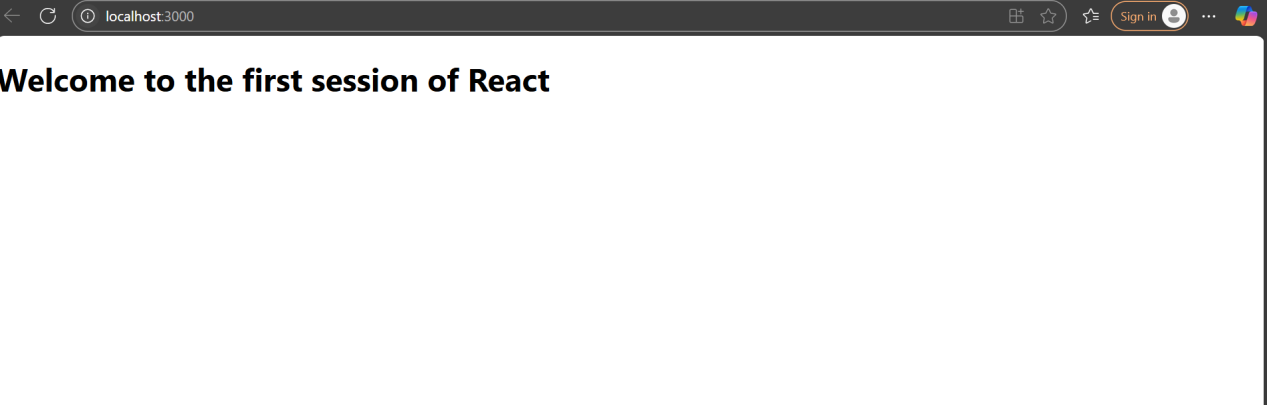
}

export default App;

#### 7.Run the App

Back in the terminal, type:

npm start



1. ReactJS-HOL:-
2. Explain React components

**React components** are the building blocks of a React application. They are reusable pieces of UI that define what should be rendered on the screen. Components can be thought of as JavaScript functions or classes that return React elements (JSX) describing what the UI should look like.

1. Identify the differences between components and JavaScript functions

· **React Components**:

* Return React elements (JSX) to be rendered on the UI.
* Can manage their own state and lifecycle (especially class components).
* Must start with a capital letter.
* Can be class-based or function-based.

· **JavaScript Functions**:

* Are generic blocks of code that perform tasks or return values.
* Don’t necessarily return UI elements.
* No special naming convention needed.
* Do not have lifecycle methods or state unless used as React function components with hooks.

### 3. Identify the types of components

There are mainly two types of React components:

**Class Components**:  
Created using ES6 classes. They can have state and lifecycle methods.  
Example:

class MyComponent extends React.Component {

render() {

return <div>Hello</div>;

}

}

**Function Components**:  
Created using JavaScript functions. They are simpler and can use hooks for state and lifecycle features.  
Example:

function MyComponent() {

return <div>Hello</div>;

}

1. Explain class component

A **class component** is a React component defined as an ES6 class that extends from React.Component. It must have a render() method which returns the JSX to be rendered. Class components can hold internal state and use lifecycle methods like componentDidMount().

Example:

class MyComponent extends React.Component {

render() {

return <h1>Hello from Class Component</h1>;

}

}

### 5. Explain function component

A **function component** is a React component defined as a JavaScript function that returns JSX. It is simpler and, with React hooks, can also handle state and side effects.

Example:

function MyComponent() {

return <h1>Hello from Function Component</h1>;

}

### 6. Define component constructor

The **constructor** in a class component is a special method that is called when the component is created. It is mainly used to initialize state or bind methods.

Example:

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = { count: 0 };

}

// ...

}

### 7. Define render() function

The **render()** function is a required method in class components. It defines what the UI looks like by returning JSX elements. React calls this method to know what to display on the screen.

Example:

render() {

return <h1>Hello World</h1>;

}

Create our React App:-

* In the terminal, type the command

npx create-react-app studentapp

* Open the Project

cd studentapp

* Open this project in VS Code:

code .

* Create Components Folder

StudentApp > src > Components

* Create Home.js inside Component

import React from 'react';

class Home extends React.Component {

render() {

return (

<div>

<h2>Welcome to the Home page of Student Management Portal</h2>

</div>

);

}

}

export default Home;

* Create About.js inside Component

import React from 'react';

class About extends React.Component {

render() {

return (

<div>

<h2>Welcome to the About page of the Student Management Portal</h2>

</div>

);

}

}

export default About;

* Create Contact.js inside Component

import React from 'react';

class Contact extends React.Component {

render() {

return (

<div>

<h2>Welcome to the Contact page of the Student Management Portal</h2>

</div>

);

}

}

export default Contact;

* Show Components in App.js

· Go to src > App.js

· Replace all the code in App.js with this:

import React from 'react';

import './App.css';

import Home from './Components/Home';

import About from './Components/About';

import Contact from './Components/Contact';

function App() {

return (

<div className="App">

<h1>Student Management Portal</h1>

<Home />

<About />

<Contact />

</div>

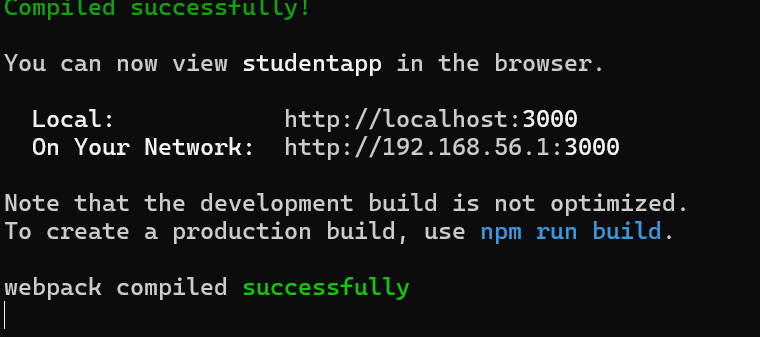
);

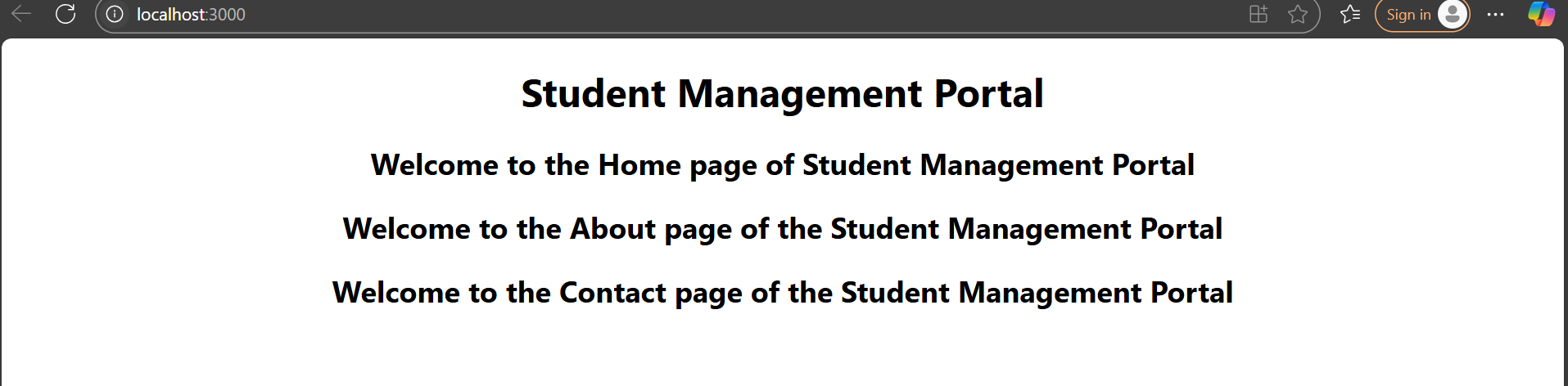
}

export default App;

* Run the App

npm start





3.ReactJS-HOL:- 1. **Create a React app:**

**npx create-react-app scorecalculatorapp**

##### 2. ****Create a function component:**** CalculateScore

Place it in: src/Components/CalculateScore.js

import React from 'react';

import '../Stylesheets/mystyle.css';

function CalculateScore(props) {

const average = props.Total / props.Goal;

return (

<div className="score-card">

<h2>Student Score Details</h2>

<p><strong>Name:</strong> {props.Name}</p>

<p><strong>School:</strong> {props.School}</p>

<p><strong>Total Score:</strong> {props.Total}</p>

<p><strong>Goal:</strong> {props.Goal}</p>

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

</div>

);

}

export default CalculateScore;

1. **Add Styling**

· Create a folder named Stylesheets.

· Add a file named mystyle.css to style your component.

.score-card {

background-color: #f0f8ff;

border: 2px solid #0074d9;

border-radius: 10px;

padding: 20px;

margin: 30px auto;

width: 300px;

box-shadow: 2px 2px 12px rgba(0,0,0,0.1);

font-family: Arial, sans-serif;

text-align: left;

}

.score-card h2 {

color: #0074d9;

margin-bottom: 15px;

}

Use the Component

Modify App.js to invoke the CalculateScore functional component.

import React from 'react';

import './App.css';

import CalculateScore from './Components/CalculateScore';

function App() {

return (

<div className="App">

<h1>Welcome to Score Calculator App</h1>

<CalculateScore Name="Alice Johnson" School="Greenwood High" Total={450} Goal={5} />

</div>

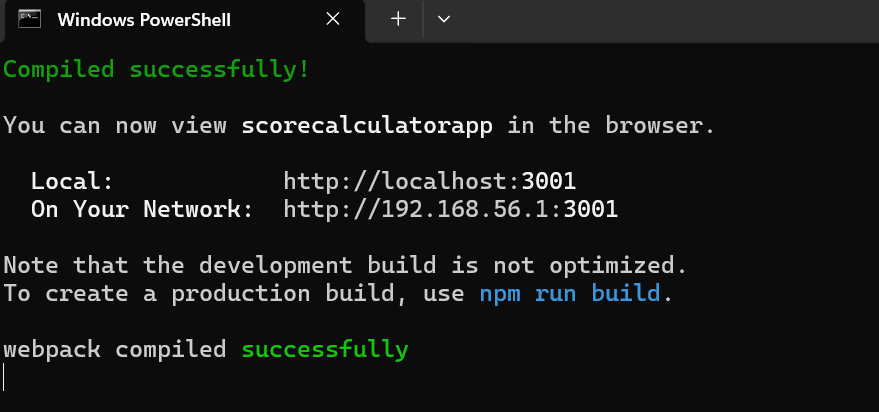
);

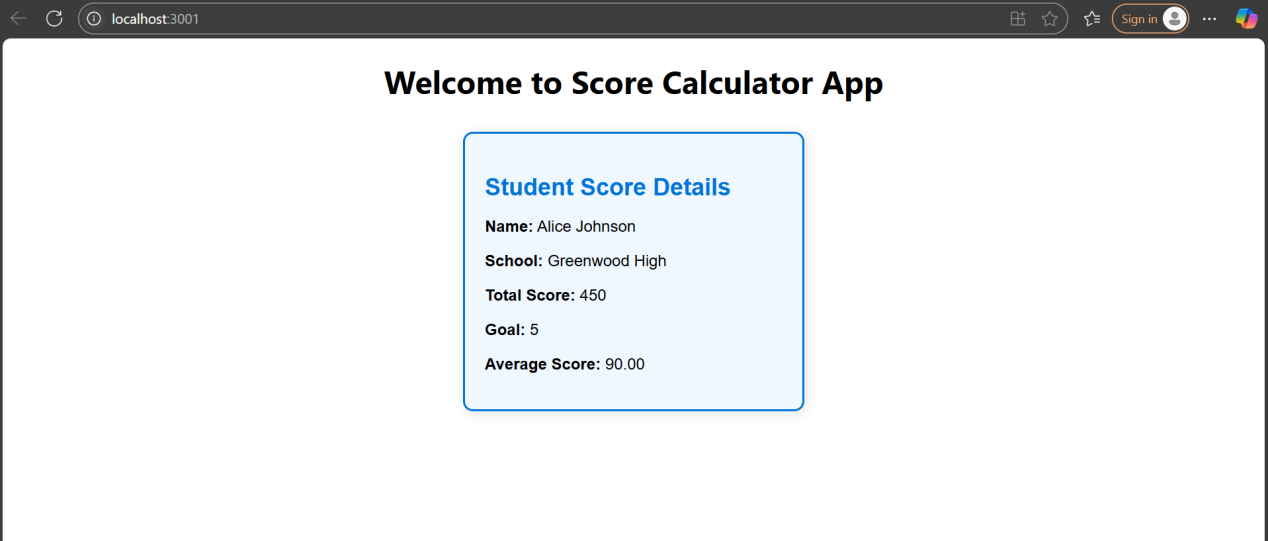
}

export default App;

Run the App

npm start





1. ReactJS-HOL:-
2. **Explain the Need and Benefits of Component Lifecycle**

#### What is a Component Lifecycle?

In React, every class component goes through a **series of steps** from creation to destruction. These steps make up the **component lifecycle**.

#### Why It's Needed:

* To run **specific code at specific times** (e.g., when component loads, updates, or errors).
* Helps control side effects (e.g., API calls, timers, event listeners).

#### Benefits:

* Allows **data fetching** after the component is mounted.
* Helps with **clean-up** tasks before component is removed.
* Makes debugging easier by tracking lifecycle stages.
* Enables powerful features like **error handling** and **conditional rendering**.

1. Identify Various Lifecycle Hook Methods

In **class components**, lifecycle methods are special functions that get called automatically during each phase.

Major Lifecycle Hooks:

| **Phase** | **Lifecycle Methods** | **Purpose** |
| --- | --- | --- |
| **Mounting** | constructor()componentDidMount() | Setup state, fetch data after component loads |
| **Updating** | shouldComponentUpdate()componentDidUpdate() | Respond to changes in props or state |
| **Unmounting** | componentWillUnmount() | Cleanup like removing event listeners or timers |
| **Error** | componentDidCatch()getDerivedStateFromError() | Catch and handle errors gracefully |

1. List the Sequence of Steps in Rendering a Component

When a class component is rendered, it follows this sequence:

#### 🔁 Rendering Lifecycle (Mounting Phase):

* constructor()  
  → Sets initial state and binds methods.
* render()  
  → Returns JSX to display.
* componentDidMount()  
  → Runs **after** the component appears on the screen.

#### If the component ****updates**** (state/props change):

shouldComponentUpdate() (optional)

render()

componentDidUpdate()

#### When the component is removed:

componentWillUnmount()

#### If an error occurs:

componentDidCatch(error, info)  
→ Handles error and can show fallback UI.

* Create a New React App

npx create-react-app blogapp

* Open the Project in VS Code

cd blogapp

code .

* Create Post.js in src Folder

import React from 'react';

function Post({ title, body }) {

return (

<div className="post">

<h2>{title}</h2>

<p>{body}</p>

</div>

);

}

export default Post;

* Create Posts.js

import React, { Component } from 'react';

import Post from './Post';

class Posts extends Component {

constructor(props) {

super(props);

this.state = {

posts: [],

hasError: false

};

}

* Add the loadPosts() method-inside Posts.js, under the constructor, add:

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);

alert("Failed to fetch posts");

});

}

### Use componentDidMount() to Load Posts

Under your loadPosts() method, add:

componentDidMount() {

this.loadPosts();

}

### Handle Errors with componentDidCatch()

Below componentDidMount(), add:

componentDidCatch(error, info) {

console.error("Error caught in Posts component:", error);

alert("Something went wrong!");

this.setState({ hasError: true });

}

* Render the Posts

render() {

if (this.state.hasError) {

return <h1>Oops! Something went wrong.</h1>;

}

return (

<div>

<h1>All Blog Posts</h1>

{this.state.posts.map(post => (

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

))}

</div>

);

}

}

export default Posts;

* Use Posts in App.js

import React from 'react';

import './App.css';

import Posts from './Posts'; // Importing the Posts component

function App() {

return (

<div className="App">

<h1>Student Management Portal</h1>

<Posts />

</div>

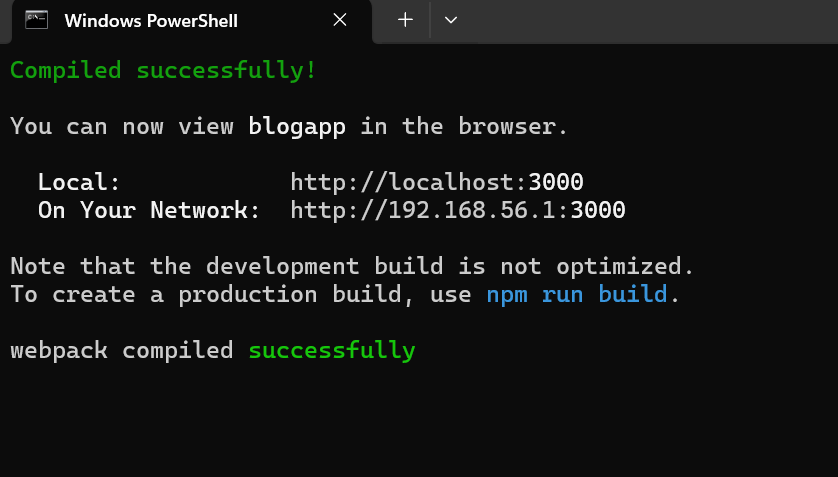
);

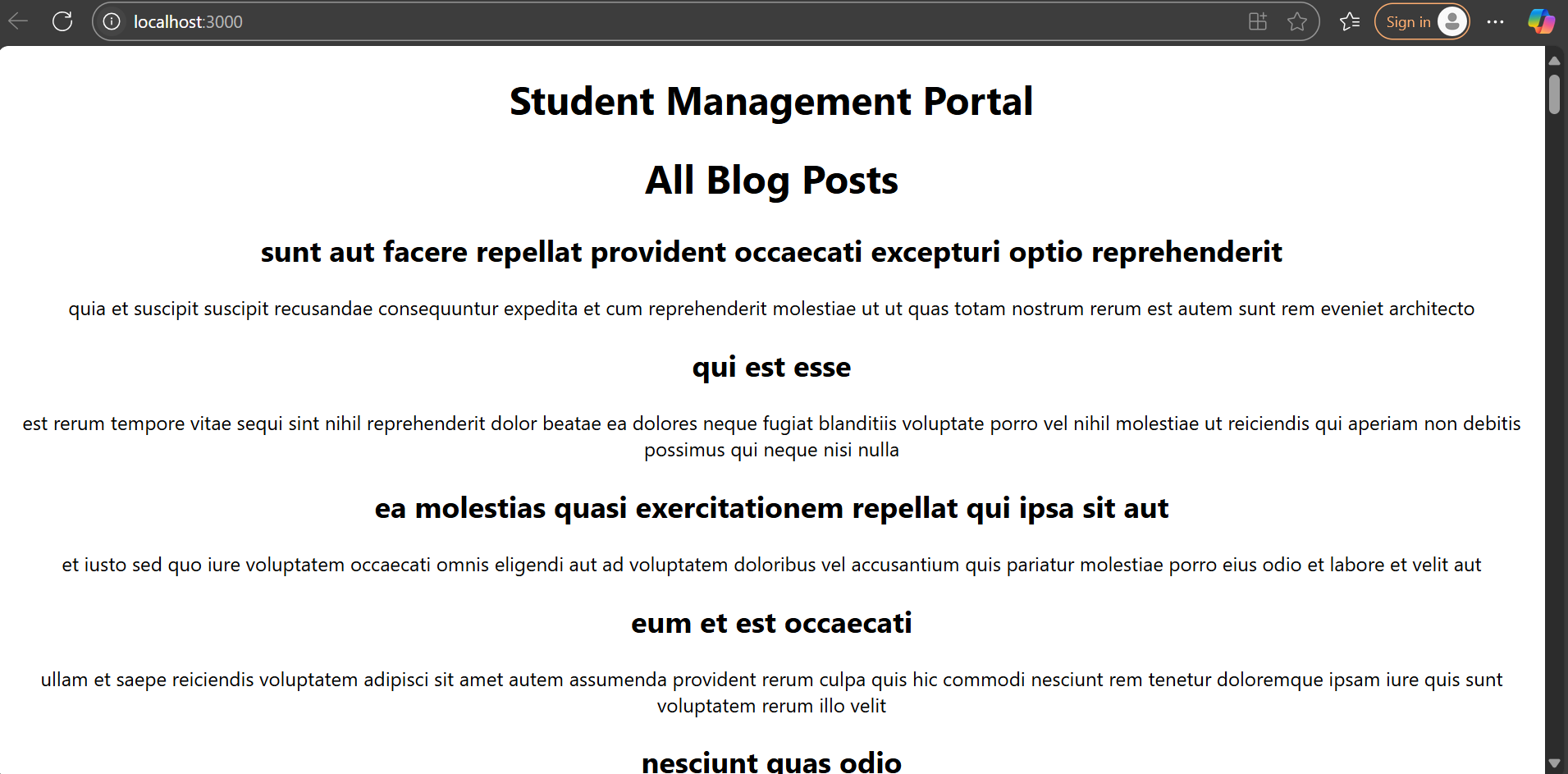
}

export default App;

* Run the Project

Npm start





1. ReactJS-HOL:-

Step 1: Unzip the Project

**Step 2: Open Command Prompt in That Folder**

**cd C:\Users\KIIT\Downloads\cohorttracker**

Step 3: Install Required Packages

npm install

Step 4: Open the App in VS Code

code .

### ****Step 5: Create the CSS Module****

Inside the src/ folder, create a file:

.box {

width: 300px;

display: inline-block;

margin: 10px;

padding: 10px 20px;

border: 1px solid black;

border-radius: 10px;

}

dt {

font-weight: 500;

}

Step 6: Update the CohortDetails Component

src/CohortDetails.js

import React from 'react';

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

const CohortDetails = ({ cohorts }) => {

return (

<div>

{cohorts.map((cohort) => (

<div key={cohort.id} className={styles.box}>

<h3 style={{ color: cohort.status === 'ongoing' ? 'green' : 'blue' }}>

{cohort.name}

</h3>

<dl>

<dt>Start Date</dt>

<dd>{cohort.startDate}</dd>

<dt>Status</dt>

<dd>{cohort.status}</dd>

</dl>

</div>

))}

</div>

);

};

export default CohortDetails;

src/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;

}

src/App.js

import React from 'react';

import './App.css';

import CohortDetails from './CohortDetails';

const sampleCohorts = [

{ id: 1, name: "React Basics", startDate: "2024-10-01", status: "ongoing" },

{ id: 2, name: "Node Bootcamp", startDate: "2024-08-15", status: "completed" },

{ id: 3, name: "Angular Advance", startDate: "2024-09-20", status: "ongoing" }

];

function App() {

return (

<div className="App">

<h1>My Academy Dashboard</h1>

<CohortDetails cohorts={sampleCohorts} />

</div>

);

}

export default App;

* Run the App

npm start

