**1. ReactJS-HOL**

**Define SPA and its Benefits**

A Single Page Application, commonly known as SPA, is a type of web application that loads a single HTML page initially and then dynamically updates the content without refreshing the whole page. It interacts with the browser by rewriting sections of the current page, which makes the experience smoother and faster.

Some of the main benefits of using SPA include improved performance after the first load, a better user experience with seamless navigation, reduced server load due to fewer full-page requests, and overall faster response times for users.

**Define React and identify its working**

React is an open-source JavaScript library developed by Facebook. It is used for building user interfaces, particularly for single-page applications. React focuses on building components, which are like building blocks of the UI.

React works by using a concept called the virtual DOM. When a user interacts with the application and something changes, React compares the new virtual DOM with the previous one and updates only the part of the page that has changed. This makes the application faster and more efficient.

**Identify the differences between SPA and MPA**

Single Page Applications and Multi Page Applications differ mainly in how they load and display content. In an SPA, the entire application runs on a single HTML page, and only the content is updated dynamically using JavaScript. This means there is no full-page reload, resulting in a faster and smoother experience.

On the other hand, in a Multi Page Application (MPA), each interaction or link opens a new page from the server. This means the entire page reloads every time the user navigates, which can be slower and affect the user experience. However, MPAs are better suited for applications that require strong SEO support or have a large number of pages like blogs or e-commerce websites.

**Explain the Pros and Cons of Single Page Application**

Single Page Applications offer several advantages. They provide a fast and smooth user experience by eliminating full-page reloads. SPAs also reduce the load on the server since only data is sent and not the entire page. Developers often find them easier to work with when building highly interactive interfaces.

However, SPAs also have some drawbacks. SEO optimization is more difficult because search engines prefer traditional page structures. The initial loading time may be slightly higher since the whole app needs to load at once. Also, browser history management and deep linking require additional effort.

**Explain About React**

React is a powerful JavaScript library for creating user interfaces, especially dynamic and interactive front-ends. It allows developers to break the UI into small, reusable components that manage their own state. This makes the code more maintainable and modular.

React is particularly useful in Single Page Applications because it can update only the parts of the page that change, making it very efficient. It also supports features like JSX and hooks.

**Define Virtual DOM**

The Virtual DOM is a key feature of React. It is a lightweight copy of the actual DOM Document Object Model that exists in memory. When a change occurs in the application, React first makes the changes in the virtual DOM. Then it compares the updated virtual DOM with the previous version to find what exactly has changed.

Only those changes are then applied to the actual DOM. This approach avoids unnecessary updates and makes the application much faster and more responsive.

**Explain Features of React**

React comes with several useful features that make it a popular choice among developers. One of the core features is its component-based architecture, which allows breaking down the UI into reusable pieces. Another important feature is the use of the virtual DOM, which makes rendering more efficient.

React also supports unidirectional data flow, which means data flows in a single direction and makes debugging easier. JSX is another useful feature that lets you write HTML-like syntax directly inside JavaScript. With the introduction of hooks, React allows developers to manage state and side effects in functional components. Lastly, React has a strong community and ecosystem, along with great developer tools.

**Commands:**

npm install -g create-react-app

npx create-react-app myfirstreact

cd myfirstreact

**Project Name: myfirstreact**

**Code:**

**App.js**

import React from 'react';

import './App.css';

function App() {

return (

<div className="container">

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

</div>

);

}

export default App;

**App.css**

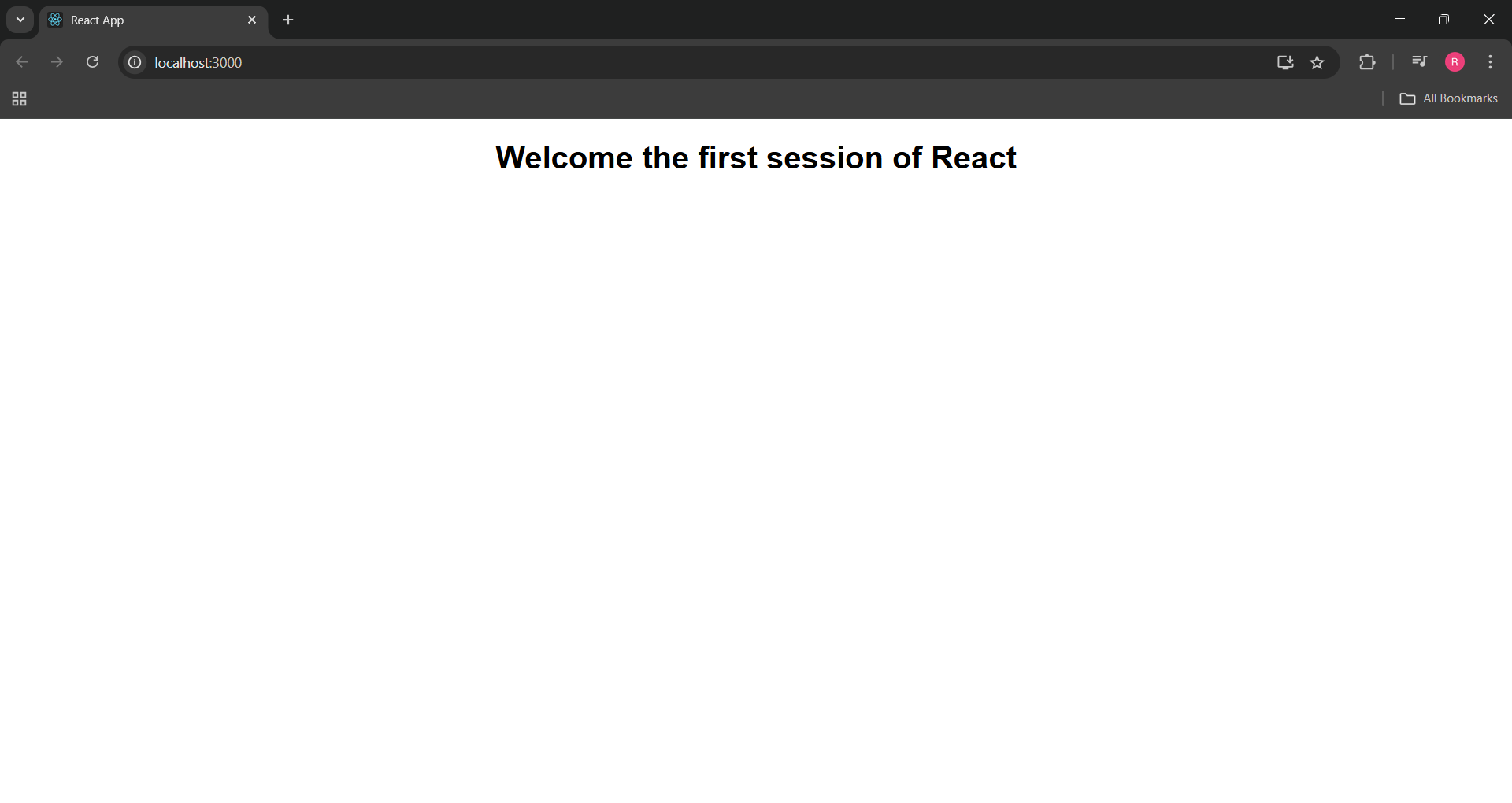
.container {

text-align: center;

font-family: Arial, sans-serif;

**}**

**Output:**

****

**2. ReactJS-HOL**

**Explain React components**

React components are reusable and independent building blocks in a React application. Each component controls its own structure, logic, and styling and can be reused multiple times throughout the app.

**Identify the differences Between Components and JavaScript Functions**

While both are JavaScript functions, React components are specifically written to return JSX (a syntax extension used with React) and manage UI logic. Unlike ordinary JavaScript functions, components can maintain state and lifecycle methods.

**Identify the types of Components**

There are mainly two types of components in React:

* **Class Components**: These are ES6 classes that extend React.Component and use the render() method to return JSX.
* **Function Components**: These are simpler components written as JavaScript functions that return JSX directly.

**Explain Class Component**

A class component is a React component defined using a JavaScript class. It has access to features like state, lifecycle methods, and uses the render() method to return JSX.

**Explain Function Component**

A function component is a simpler and more modern way to define components in React. It is a plain JavaScript function that accepts props and returns JSX. With React hooks, function components can now manage state and side effects too.

**Define Component Constructor**

In class components, a constructor is a special method used to initialize the component’s state and bind methods. It is called once when the component is created.

constructor(props) {

super(props);

this.state = { /\* initial state \*/ };

}

**Define render() Function**

The render() function is a required method in class components. It tells React what to display on the screen by returning JSX**.**

**Commands:**

npx create-react-app studentapp

cd studentapp

**Project Name: studentapp**

**Code:**

**App.js**

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="container">

<Home />

<About />

<Contact />

</div>

);

}

export default App;

**Home.js**

import React from 'react';

function Home() {

return (

<div className="page">

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

</div>

);

}

export default Home;

**About.js**import React from 'react';

function About() {

return (

<div className="page">

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

</div>

);

}

export default About;

**Contact.js**import React from 'react';

function Contact() {

return (

<div className="page">

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

</div>

);

}export default Contact;

**App.css**

.container {

text-align: center;

padding: 40px;

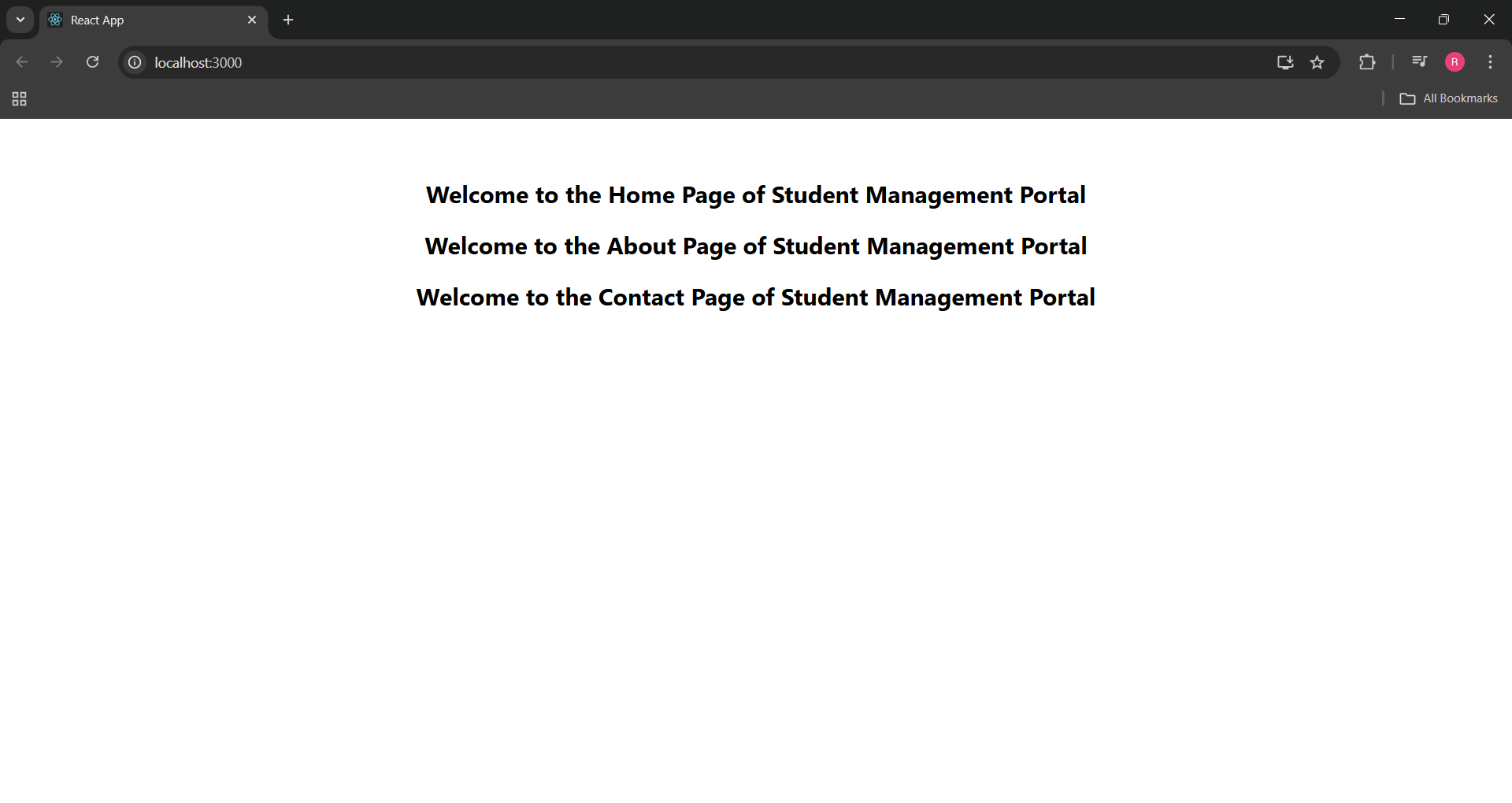
}

.page {

margin: 20px 0;

}

**Output:**

****

**3. ReactJS-HOL**

**Explain React components**

React components are reusable and independent building blocks in a React application. Each component controls its own structure, logic, and styling and can be reused multiple times throughout the app.

**Identify the differences Between Components and JavaScript Functions**

While both are JavaScript functions, React components are specifically written to return JSX (a syntax extension used with React) and manage UI logic. Unlike ordinary JavaScript functions, components can maintain state and lifecycle methods.

**Identify the types of Components**

There are mainly two types of components in React:

* **Class Components**: These are ES6 classes that extend React.Component and use the render() method to return JSX.
* **Function Components**: These are simpler components written as JavaScript functions that return JSX directly.

**Explain Class Component**

A class component is a React component defined using a JavaScript class. It has access to features like state, lifecycle methods, and uses the render() method to return JSX.

**Explain Function Component**

A function component is a simpler and more modern way to define components in React. It is a plain JavaScript function that accepts props and returns JSX. With React hooks, function components can now manage state and side effects too.

**Define Component Constructor**

In class components, a constructor is a special method used to initialize the component’s state and bind methods. It is called once when the component is created.

constructor(props) {

super(props);

this.state = { /\* initial state \*/ };

}

**Define render() Function**

The render() function is a required method in class components. It tells React what to display on the screen by returning JSX**.**

**Commands:**

npx create-react-app scorecalculatorapp

cd scorecalculatorapp

**Project Name: scorecalculatorapp**

**Code:**

**App.js**

import React from 'react';

import { CalculateScore } from './Components/CalculateScore';

function App() {

  return (

    <div>

      <CalculateScore

        Name="Steve"

        School="DNV Public School"

        total={284}

        goal={3}

      />

    </div>

  );

}

export default App;

**Components/CalculateScore.js**

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

const percentToDecimal = (decimal) => {

  return (decimal).toFixed(2) + '%';

}

const calcScore = (total, goal) => {

  return percentToDecimal((total / goal));

}

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

  <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>

);

**Stylesheets/mystyle.css**

.Name {

  font-weight: 300;

  color: blue;

}

.School {

  color: crimson;

}

.Total {

  color: darkmagenta;

}

.formatstyle {

  text-align: center;

  font-size: large;

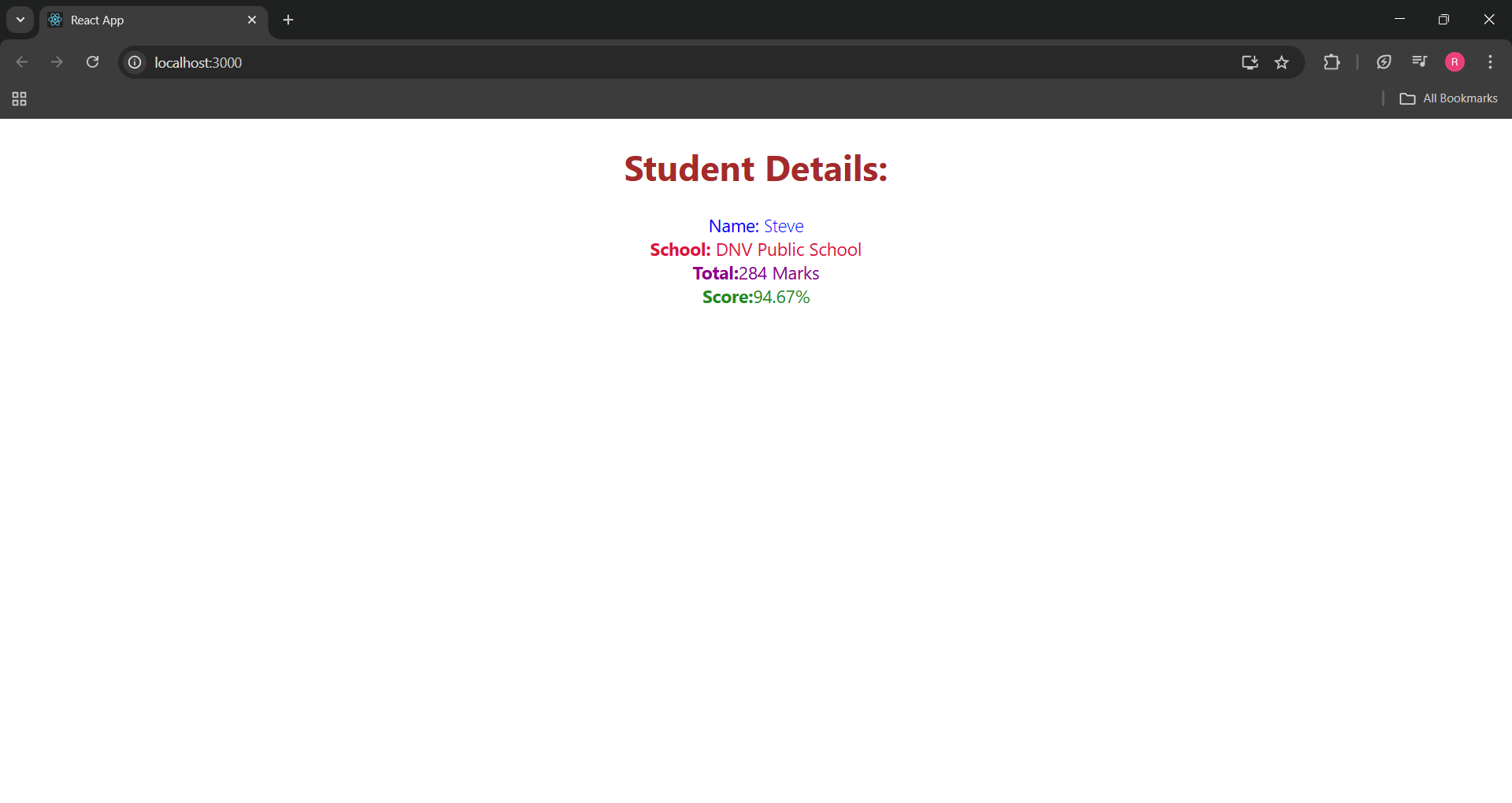
}

.Score {

  color: forestgreen;

}

**Output:**



**4. ReactJS-HOL**

**Explain the need and Benefits of Component Life cycle**

In React, components go through different stages — like being created, updated, and removed. Lifecycle methods help us hook into these stages and control what happens when. They’re useful for things like fetching data after a component loads, cleaning up timers or listeners, or handling errors gracefully.  
Using lifecycle methods keeps our code clean, predictable, and easier to debug.

**Identify various life cycle hook methods**

Here are the main lifecycle methods I’ve worked with:

* **Mounting:** constructor(), componentDidMount()
* **Updating:** componentDidUpdate()
* **Unmounting:** componentWillUnmount()
* **Error Handling:** componentDidCatch()

Each of these lets us perform certain actions at the right time, like loading data, updating the UI, or catching runtime errors.

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

When a class component renders for the first time, it goes through these steps:

constructor() — initializes state and props

render() — returns what to show on the screen

componentDidMount() — runs after the UI is ready; perfect for fetching data

This flow ensures that data is loaded and displayed efficiently without blocking the UI.

**Commands:**

npx create-react-app blogapp

cd blogapp

**Project Name: blogapp**

**Code:**

**App.js**

import React from 'react';

import './App.css';

import Posts from './Posts';

function App() {

  return (

    <div className="App">

      <h1>My Blog Application</h1>

      <Posts />

    </div>

  );

}

export default App;

**Post.js**

import React from 'react';

class Post extends React.Component {

  render() {

    const { title, body } = this.props;

    return (

      <div style={{ marginBottom: '20px' }}>

        <h3>{title}</h3>

        <p>{body}</p>

      </div>

    );

  }

}

export default Post;

**Posts.js**

import React, { Component } from 'react';

import Post from './Post';

class Posts extends Component {

  constructor(props) {

    super(props);

    this.state = {

      postsList: [],

      hasError: false

    };

  }

  loadPosts = async () => {

    try {

      const response = await fetch('https://jsonplaceholder.typicode.com/posts');

      const data = await response.json();

      this.setState({ postsList: data });

    } catch (err) {

      console.error('Failed to fetch posts:', err);

    }

  };

  componentDidMount() {

    this.loadPosts();

  }

  componentDidCatch(error, info) {

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

    alert('Something went wrong while rendering the posts.');

    this.setState({ hasError: true });

  }

  render() {

    const { postsList, hasError } = this.state;

    if (hasError) {

      return <p>Unable to display posts at the moment.</p>;

    }

    return (

      <div>

        <h2>All Blog Posts</h2>

        {postsList.slice(0, 10).map(post => (

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

        ))}

      </div>

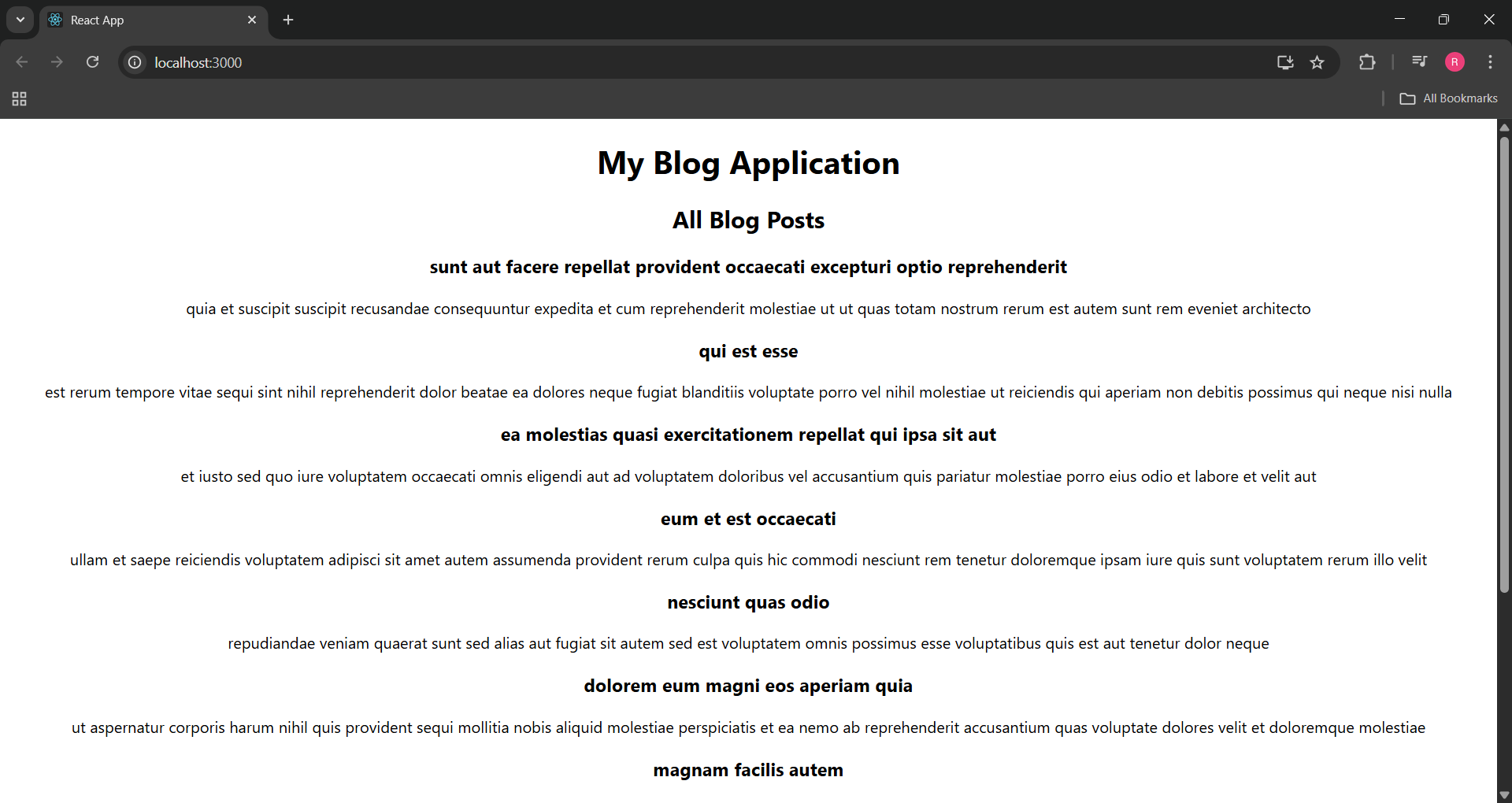
    );

  }

}

export default Posts;

**Output:**



**5. ReactJS-HOL**

**Why Styling is Needed in React Components**

Styling makes React components look clean, organized, and user-friendly. Without styles, even a good layout can look plain or hard to use. Applying styles helps improve readability and gives a professional appearance to the UI.

**Working with CSS Modules and Inline Styles**

React allows styling in multiple ways — two common ones are:

* **CSS Modules:** These keep styles local to a component. So, styles don't accidentally leak into other components.
* **Inline Styles:** You can add styles directly in JSX using the style={{}} prop, which is helpful when styling depends on logic or conditions.

**Commands:**

npx create-react-app cohort-dashboard

cd cohort-dashboard

**Project Name: cohort-dashboard**

**Code:**

**App.js**

import React from 'react';

import CohortDetails from './CohortDetails';

function App() {

  return (

    <div className="App">

      <h2>Cohort Dashboard</h2>

      <CohortDetails

        name="React Bootcamp"

        status="ongoing"

        startDate="2025-07-01"

        endDate="2025-08-15"

      />

      <CohortDetails

        name="Java Basics"

        status="completed"

        startDate="2025-06-01"

        endDate="2025-07-10"

      />

    </div>

  );

}

export default App;

**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';

function CohortDetails({ name, status, startDate, endDate }) {

const headingStyle = {

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

};

return (

<div className={styles.box}>

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

<dl>

<dt>Status:</dt>

<dd>{status}</dd>

<dt>Start Date:</dt>

<dd>{startDate}</dd>

<dt>End Date:</dt>

<dd>{endDate}</dd>

</dl>

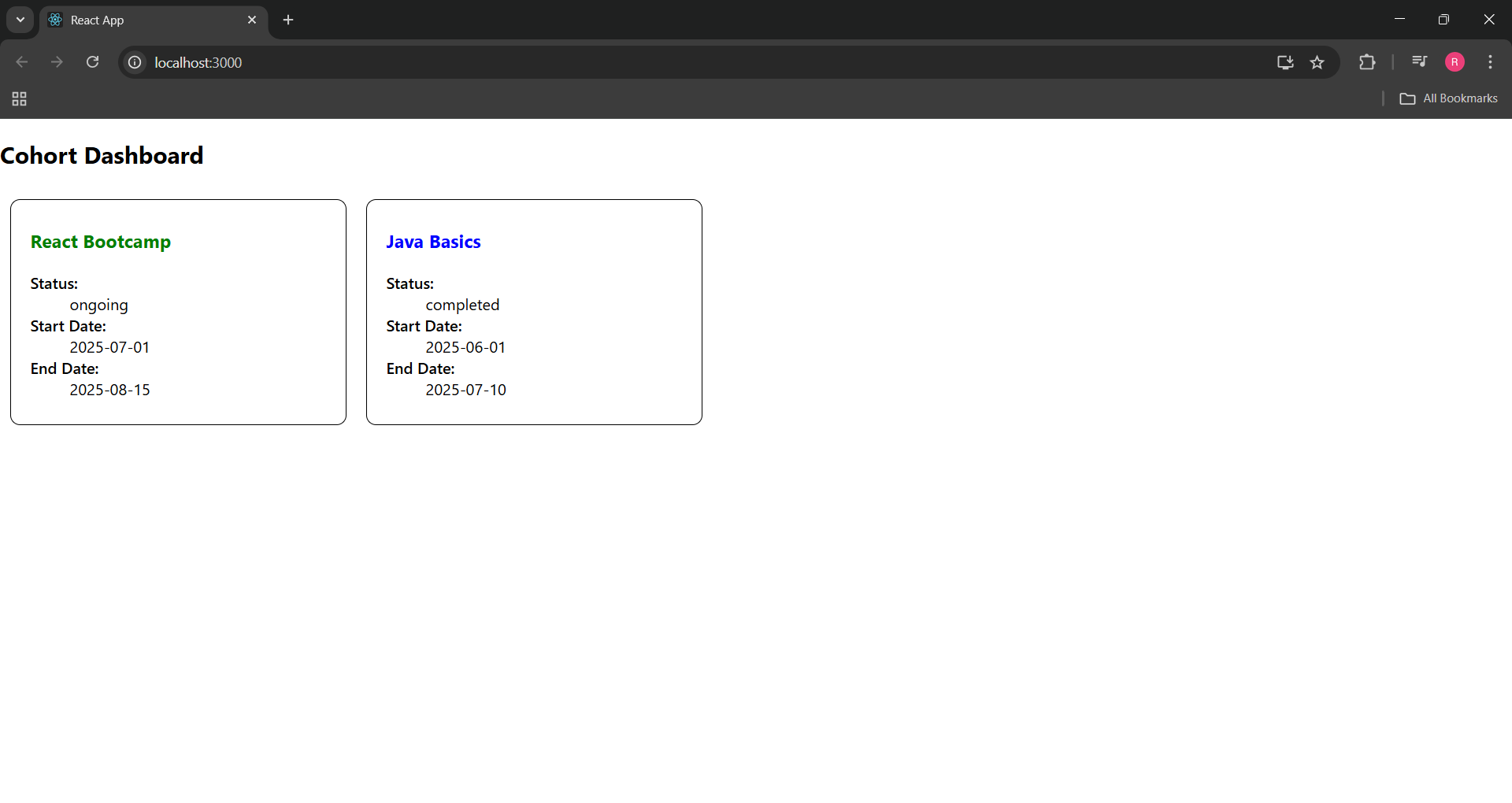
</div>

);

}

export default CohortDetails;

**Output:**

****