## **EXERCISE 1:**

## **Objectives**

* 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

Answer:

**1.Define SPA and its benefits**

* A Single-Page Application (SPA) is a web app that loads a single HTML page and dynamically updates content without refreshing the page.
* Benefits: Faster navigation, improved user experience, reduced server load, and easier client-side routing.

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

* React is a JavaScript library for building user interfaces, mainly for SPAs.
* It works by creating reusable components and efficiently updating the UI using a virtual DOM.

**3. Identify the differences between SPA and MPA**

* SPA: Loads one page, updates content dynamically, uses client-side routing.
* MPA (Multi-Page Application): Loads a new page from the server for each navigation, uses server-side routing.

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

* Pros: Fast interactions, seamless user experience, less server traffic.
* Cons: Initial load may be slow, SEO challenges, browser history management can be complex.

**5. Explain about React**

* React enables developers to build complex UIs from small, isolated pieces of code called components. It uses a declarative approach and supports efficient updates via the virtual DOM.

**6. Define virtual DOM**

* The virtual DOM is a lightweight copy of the real DOM. React uses it to track changes and update only the necessary parts of the UI, improving performance.

**7. Explain Features of React**

* Component-based architecture
* Virtual DOM for efficient rendering
* Unidirectional data flow
* JSX syntax for UI definition
* Strong community and ecosystem

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.

1. To create a new React app, Install Nodejs and Npm from the following link:

<https://nodejs.org/en/download/>

1. Install Create-react-app by running the following command in the command prompt:



1. To create a React Application with the name of “myfirstreact”, type the following command:



1. Once the App is created, navigate into the folder of myfirstreact by typing the following command:



1. Open the folder of myfirstreact in Visual Studio Code
2. Open the App.js file in Src Folder of myfirstreact
3. Remove the current content of “App.js”
4. Replace it with the following:



1. Run the following command to execute the React application:



1. Open a new browser window and type “localhost:3000” in the address bar

CODE

App.js

import logo from './logo.svg';

import './App.css';

function App() {

  return (

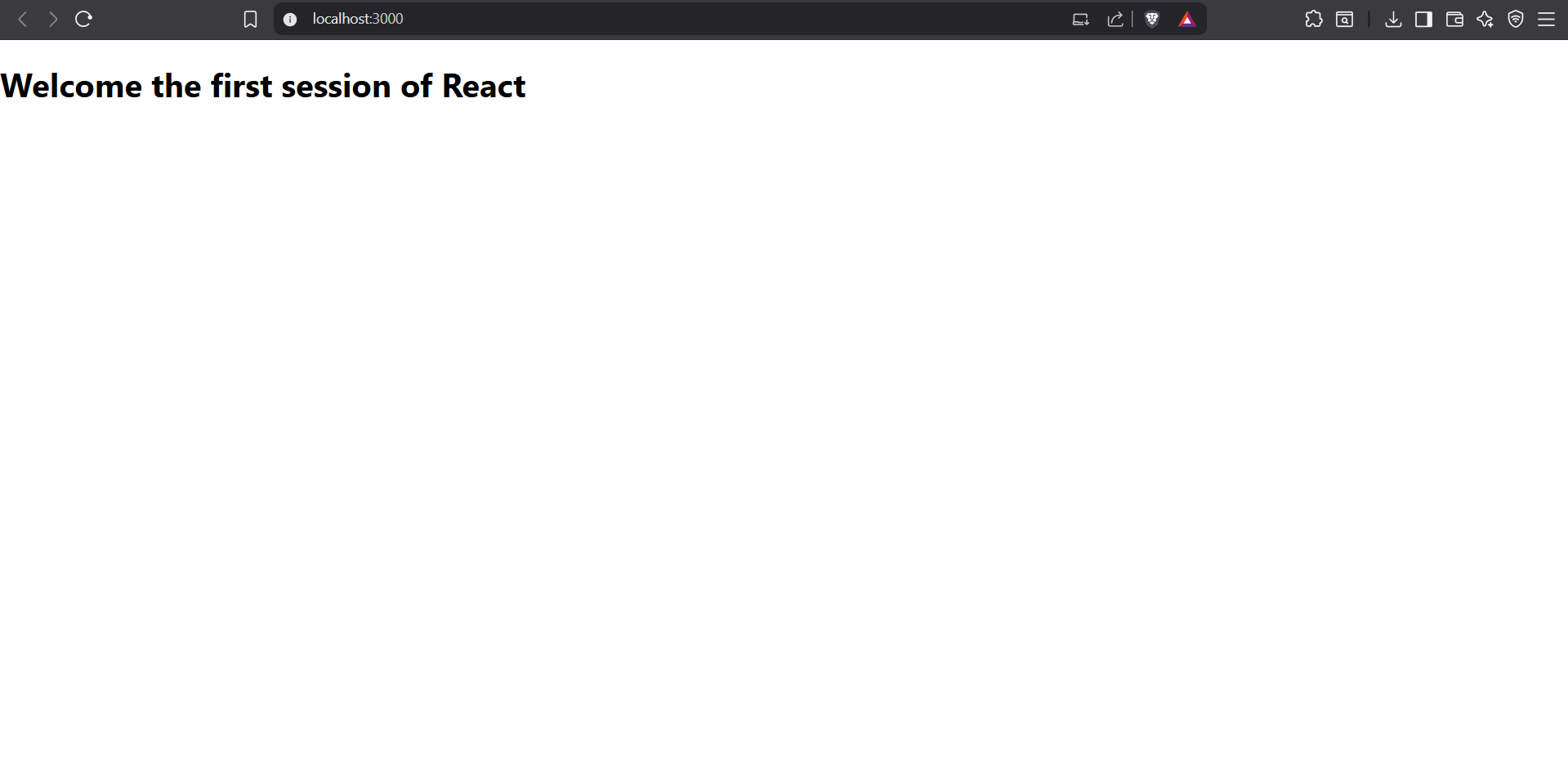
    <h1>Welcome the first session of React</h1>

  );

}

export default App;

Output:



**EXERCISE 2:**

## **Objectives**

* 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

Answer:

**React Components**

React components are reusable, self-contained pieces of UI. They accept inputs (props), manage their own state (if needed), and return React elements describing what should appear on the screen.

**Differences: Components vs JavaScript Functions**

* **React components** return JSX (React elements) and can manage state and lifecycle.
* **JavaScript functions** are general-purpose and do not have built-in support for state, props, or lifecycle.

**Types of Components**

1. **Class Components**: Use ES6 classes, can have state and lifecycle methods.
2. **Function Components**: Use plain functions, can use hooks for state and lifecycle.

**Class Component**

A class component is defined using a JavaScript class that extends [React.Component](vscode-file://vscode-app/c:/Users/saswa/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html). It must have a [render()](vscode-file://vscode-app/c:/Users/saswa/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) method that returns JSX.

**Function Component**

A function component is a JavaScript function that returns JSX. With React hooks, function components can also manage state and side effects.

**Component Constructor**

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

**render() Function**

The [render()](vscode-file://vscode-app/c:/Users/saswa/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) function is required in class components. It returns the JSX that defines the component’s UI.

In this hands-on lab, you will learn how to:

* Create a class component
* Create multiple components
* Render a component

## **Prerequisites**

The following is required to complete this hands-on lab:

* Node.js
* NPM
* Visual Studio Code

## **Notes**

Estimated time to complete this lab: **30 minutes.**

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.

1. Create a React project named “StudentApp” type the following command in terminal of Visual studio:



1. Create a new folder under Src folder with the name “Components”. Add a new file named “Home.js”
2. Type the following code in Home.js



1. Under Src folder add another file named “About.js”
2. Repeat the same steps for Creating “About” and “Contact” component by adding a new file as ”About.js”, “Contact.js” under “Src” folder and edit the code as mentioned for “Home” Component.
3. Edit the App.js to invoke the Home, About and Contact component as follows:



1. In command Prompt, navigate into StudentApp and execute the code by typing the following command:



1. Open browser and type “localhost:3000” in the address bar:



Code

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 Home from './Conponents/Home';

import About from './Conponents/About';

import Contact from './Conponents/Contact';

function App() {

  return (

    <div className="Container" style={{ justifyContent: 'center', textAlign: 'center', padding: '20px' }}>

      <Home />

      <About />

      <Contact />

    </div >

  );

}

export default App;

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>

);

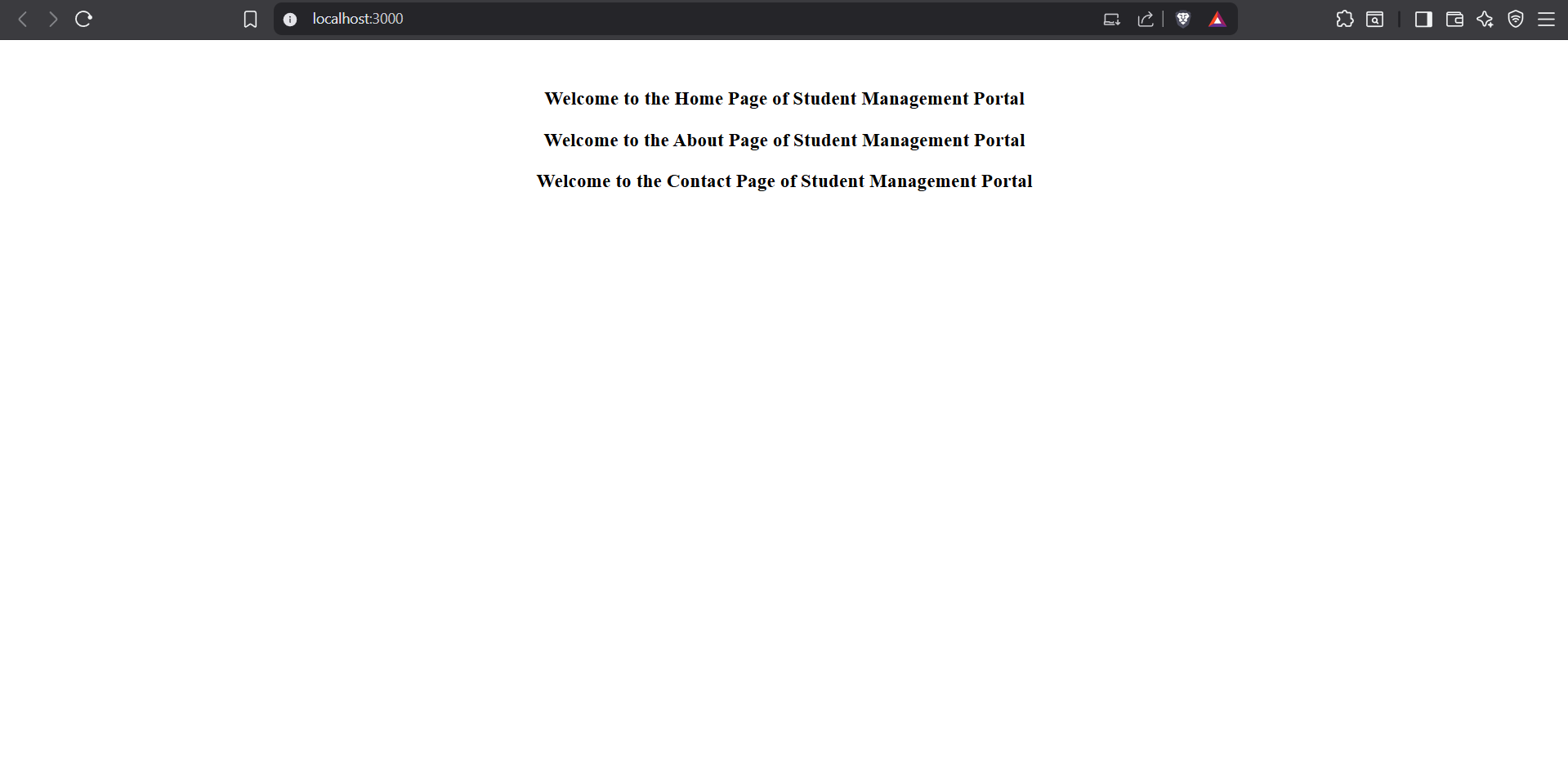
// If you want to start measuring performance in your app, pass a function

// to log results (for example: reportWebVitals(console.log))

// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals

// reportWebVitals();

Output:



**EXERCISE 3:**

## **Objectives**

* 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

Answer:

**React Components**

React components are reusable, self-contained pieces of UI. They accept inputs (props), manage their own state (if needed), and return React elements describing what should appear on the screen.

**Differences: Components vs JavaScript Functions**

* **React components** return JSX (React elements) and can manage state and lifecycle.
* **JavaScript functions** are general-purpose and do not have built-in support for state, props, or lifecycle.

**Types of Components**

1. **Class Components**: Use ES6 classes, can have state and lifecycle methods.
2. **Function Components**: Use plain functions, can use hooks for state and lifecycle.

**Class Component**

A class component is defined using a JavaScript class that extends [React.Component](vscode-file://vscode-app/c:/Users/saswa/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html). It must have a [render()](vscode-file://vscode-app/c:/Users/saswa/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) method that returns JSX.

**Function Component**

A function component is a JavaScript function that returns JSX. With React hooks, function components can also manage state and side effects.

**Component Constructor**

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

**render() Function**

The [render()](vscode-file://vscode-app/c:/Users/saswa/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) function is required in class components. It returns the JSX that defines the component’s UI.

In this hands-on lab, you will learn how to:

* Create a function component
* Apply style to components
* Render a component

## **Prerequisites**

The following is required to complete this hands-on lab:

* Node.js
* NPM
* Visual Studio Code

## **Notes**

Estimated time to complete this lab: **30 minutes.**

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.

1. Create a React project named “scorecalculatorapp” type the following command in terminal of Visual studio:



1. Create a new folder under Src folder with the name “Components”. Add a new file named “CalculateScore.js”
2. Type the following code in CalculateScore.js





1. Create a Folder named Stylesheets and add a file named “mystyle.css” in order to add some styles to the components:



1. Edit the App.js to invoke the CalculateScore functional component as follows:



1. In command Prompt, navigate into scorecalculatorapp and execute the code by typing the following command:



1. Open browser and type “localhost:3000” in the address bar:



CODE:

CalculateScore.js

import React from 'react';

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 }) => {

    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>

    );

}

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}

      />

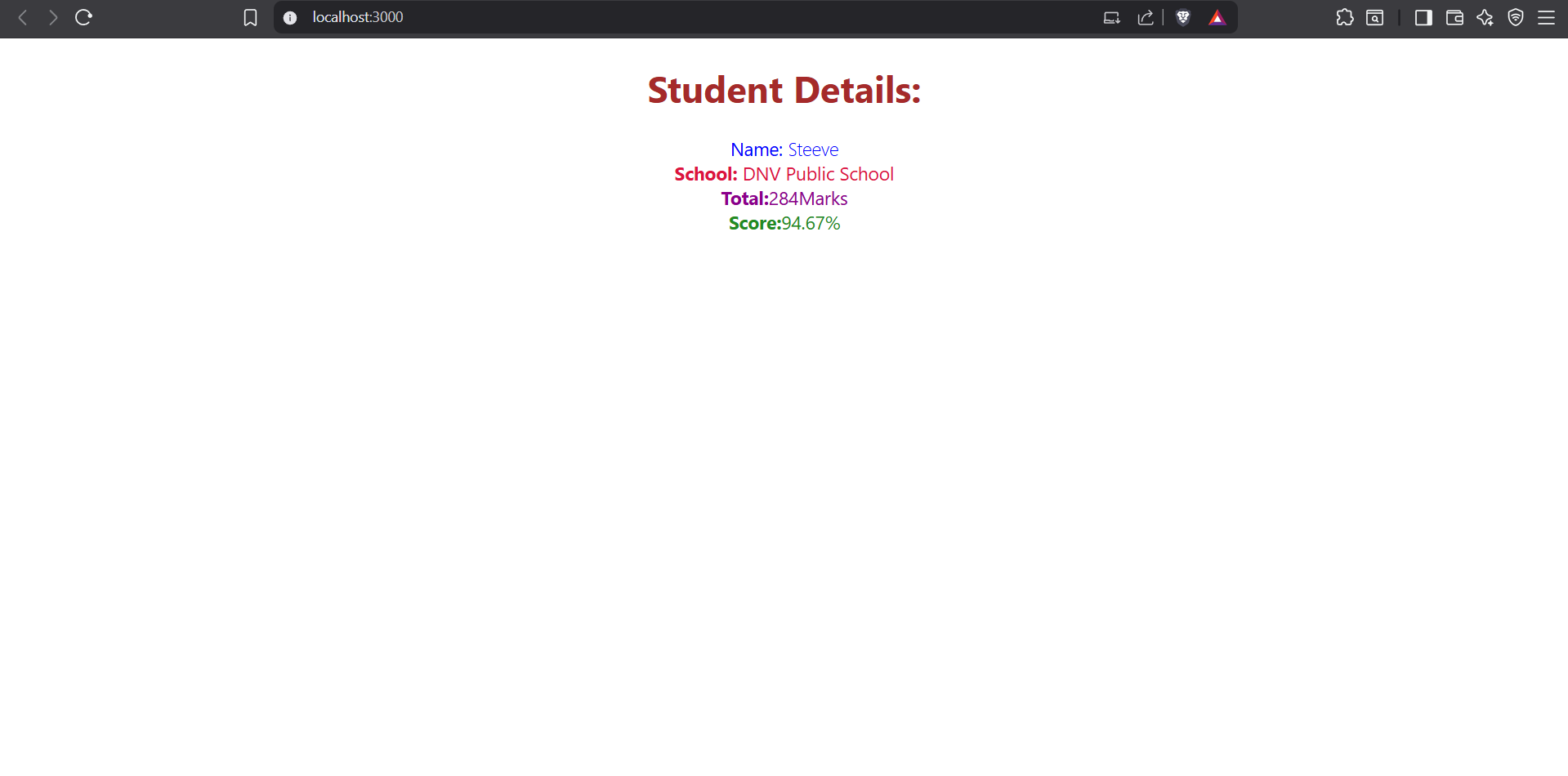
    </div>

  );

}

export default App;

Output:



**EXERCISE 4:**

**Objectives**

* Explain the need and Benefits of component life cycle
* Identify various life cycle hook methods
* List the sequence of steps in rendering a component

**Need and Benefits of Component Lifecycle**

* The component lifecycle allows you to run code at specific points in a component’s existence (creation, update, destruction).
* It helps manage side effects (e.g., data fetching, subscriptions), resource cleanup, and DOM updates.
* Lifecycle methods make components predictable and maintainable by organizing code for different phases.

**Various Lifecycle Hook Methods**

For class components, the main lifecycle methods are:

* **constructor**: Initialization (state, bindings).
* **componentDidMount**: Runs after the component is added to the DOM (good for data fetching).
* **shouldComponentUpdate**: Decides if a re-render is needed.
* **componentDidUpdate**: Runs after updates (good for reacting to prop/state changes).
* **componentWillUnmount**: Cleanup before the component is removed.
* **componentDidCatch**: Error handling in rendering or lifecycle methods.

**Sequence of Steps in Rendering a Component**

1. **constructor** (initialization)
2. **render** (returns JSX)
3. **componentDidMount** (after first render)
4. On state/prop change:
   * **shouldComponentUpdate**
   * **render**
   * **componentDidUpdate**
5. On unmount:
   * **componentWillUnmount**

In this hands-on lab, you will learn how to:

* Implement componentDidMount() hook
* Implementing componentDidCatch() life cycle hook.

## **Prerequisites**

The following is required to complete this hands-on lab:

* Node.js
* NPM
* Visual Studio Code

## **Notes**

Estimated time to complete this lab: **60 minutes.**

1. Create a new react application using *create-react-app* tool with the name as “blogapp”
2. Open the application using VS Code
3. Create a new file named as **Post.js** in **src folder** with following properties



Figure 2: Post class

1. Create a new class based component named as **Posts** inside **Posts.js** file



Figure 3: Posts Component

1. Initialize the component with a list of Post in state of the component using the constructor
2. Create a new method in component with the name as **loadPosts()** which will be responsible for using Fetch API and assign it to the component state created earlier. To get the posts use the url (<https://jsonplaceholder.typicode.com/posts>)



Figure 4: loadPosts() method

1. Implement the **componentDidMount()** hook to make calls to **loadPosts()** which will fetch the posts



Figure 5: componentDidMount() hook

1. Implement the **render()** which will display the title and post of posts in html page using heading and paragraphs respectively.



Figure 6: render() method

1. Define a **componentDidCatch()** method which will be responsible for displaying any error happing in the component as alert messages.



Figure 7: componentDidCatch() hook

1. Add the Posts component to App component.
2. Build and Run the application using *npm start* command.

Code:

Post.js

class Post {

    constructor(id, title, body){

        this.id=id;

        this.title=title;

        this.body=body;

    }

}

export default Post;

Posts.js

import React, { Component } from 'react';

import Post from './Post';

class Posts extends Component {

    constructor(props) {

        super(props);

        this.state = {

            posts: []

        };

    }

    loadPosts = async () => {

        try {

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

            if (!response.ok) {

                throw new Error('Failed to fetch posts');

            }

            const data = await response.json();

            const posts = data.map(

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

            );

            this.setState({ posts });

        } catch (error) {

            this.setState({ error });

        }

    }

    componentDidMount() {

        this.loadPosts();

    }

    componentDidCatch(error, info) {

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

    }

    render() {

        if (this.state.error) {

            return <h1>Error: {this.state.error.message}</h1>;

        }

        return (

            <div>

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

                    <div key={post.id}>

                        <h1>{post.title}</h1>

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

                        <hr />

                    </div>

                ))}

            </div>

        );

    }

}

export default Posts;

App.js

import Posts from './Posts';

function App() {

  return (

    <div className="App">

      <Posts />

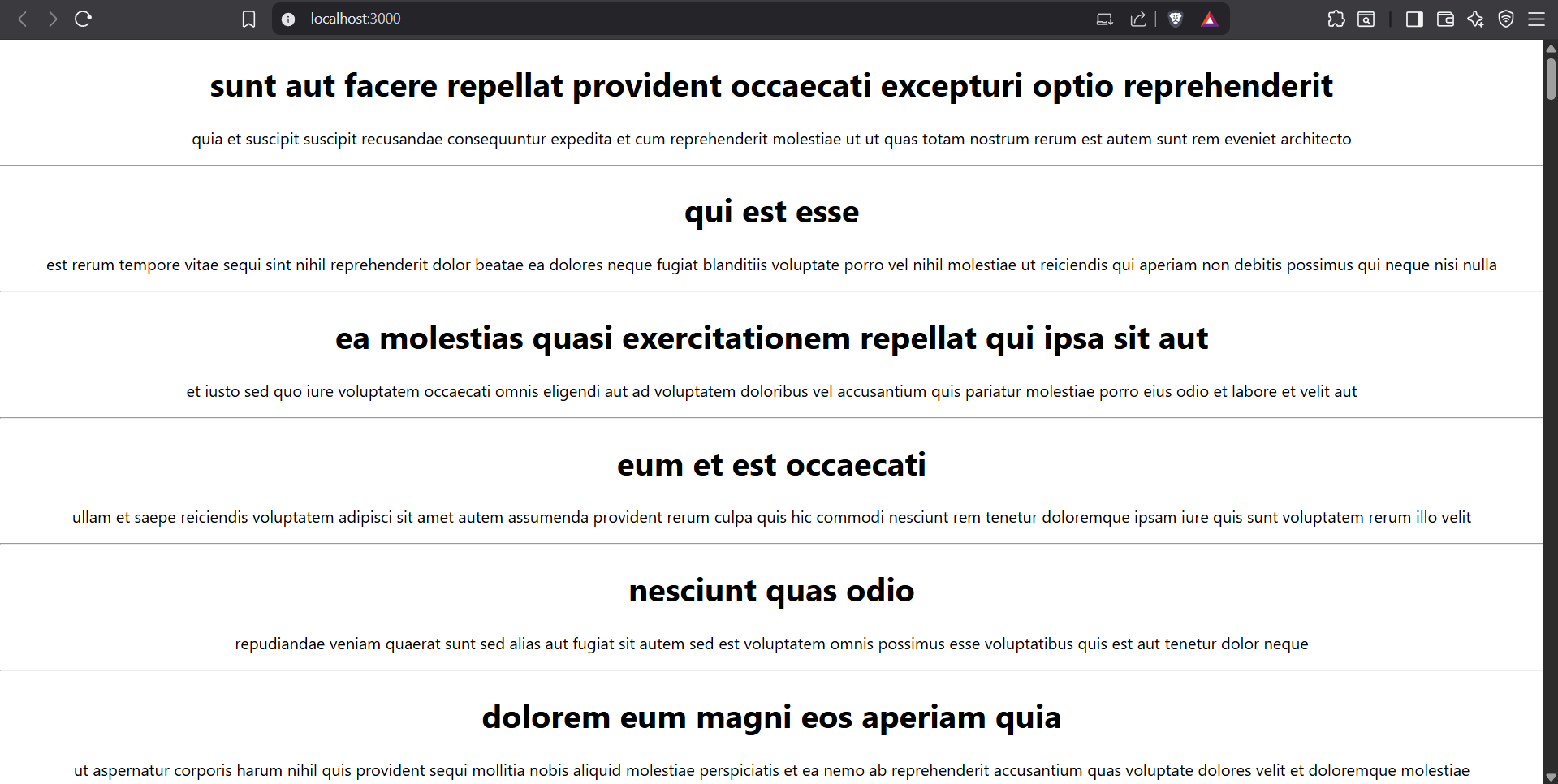
    </div>

  );

}

export default App;

Output:



**EXERCISE 5:**

**Objectives**

* Understanding the need for styling react component
* Working with CSS Module and inline styles

In this hands-on lab, you will learn how to:

* Style a react component
* Define styles using the CSS Module
* Apply styles to components using className and style properties

## **Prerequisites**

The following is required to complete this hands-on lab:

* Node.js
* NPM
* Visual Studio Code

## **Notes**

Estimated time to complete this lab: **30 minutes.**

My Academy team at Cognizant want to create a dashboard containing the details of ongoing and completed cohorts. A react application is created which displays the detail of the cohorts using react component. You are assigned the task of styling these react components.

Download and build the attached react application.



1. Unzip the react application in a folder
2. Open command prompt and switch to the react application folder
3. Restore the node packages using the following commands



Figure 1: Restore packages

1. Open the application using VS Code
2. Create a new CSS Module in a file called “CohortDetails.module.css”
3. Define a css class with the name as “box” with following properties

*Width = 300px;*

*Display = inline block;*

*Overall 10px margin*

*Top and bottom padding as 10px*

*Left and right padding as 20px*

*1 px border in black color*

*A border radius of 10px*

1. Define a css style for html <dt> element using tag selector. Set the font weight to 500.
2. Open the cohort details component and import the CSS Module
3. Apply the box class to the container div
4. Define the style for <h3> element to use “green” color font when cohort status is “ongoing” and “blue” color in all other scenarios.
5. Final result should look similar to the below image



Figure 2: Final Result

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;

}

.ongoing {

  color: green;

}

.otherStatus {

  color: blue;

}

CohortDetails.js

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

function CohortDetails(props) {

const statusClass = props.cohort.currentStatus === 'Ongoing' ? styles.ongoing : styles.otherStatus;

return (

<div className={styles.box}>

<h3 className={statusClass}>

{props.cohort.cohortCode} -

<span>{props.cohort.technology}</span>

</h3>

<dl>

<dt>Started On</dt>

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

<dt>Current Status</dt>

<dd>{props.cohort.currentStatus}</dd>

<dt>Coach</dt>

<dd>{props.cohort.coachName}</dd>

<dt>Trainer</dt>

<dd>{props.cohort.trainerName}</dd>

</dl>

</div>

);

}

export default CohortDetails;

Output:

