**Superset ID:6373322**

**COGNITZANT DIGITAL NURTURE 4.0 JAVA FSE**

**WEEK-6: ReactJs - HOL**

## **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

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

**Concept:**

* **SPA (Single Page Application)**: A web app that loads a single HTML page and dynamically updates it as the user interacts with it.
* **React**: A JavaScript library for building user interfaces, especially SPAs.
* **SPA vs MPA**:
  + SPA: Faster, dynamic, loads once.
  + MPA: Multiple HTML pages, more traditional, full reloads.
* **Pros of SPA**: Fast, seamless user experience.
* **Cons of SPA**: SEO issues, initial load time.
* **Virtual DOM**: A lightweight JavaScript object representing the actual DOM. React uses it to optimize updates.
* **Features of React**: Component-based, reusable code, virtual DOM, unidirectional data flow.

**CODE:**

**myfirstreact:**

**src/App.js**

import React from 'react';

function App() {

return (

<div>

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

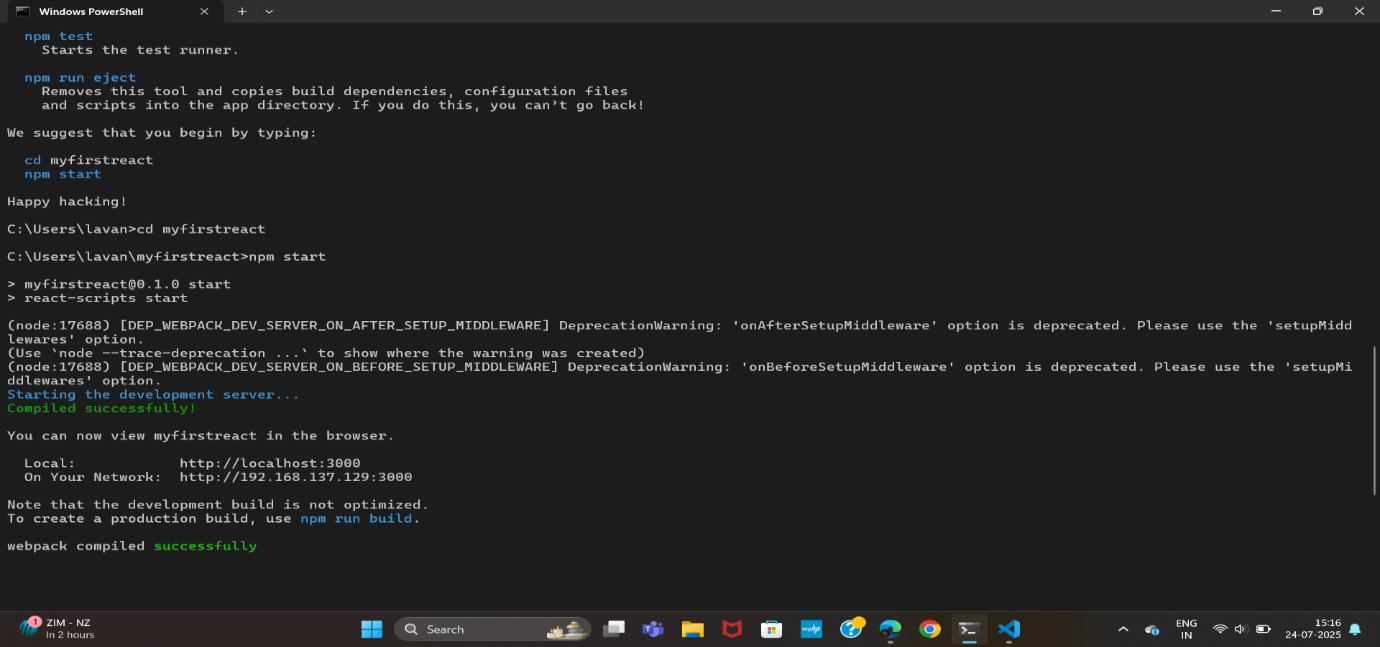
</div>

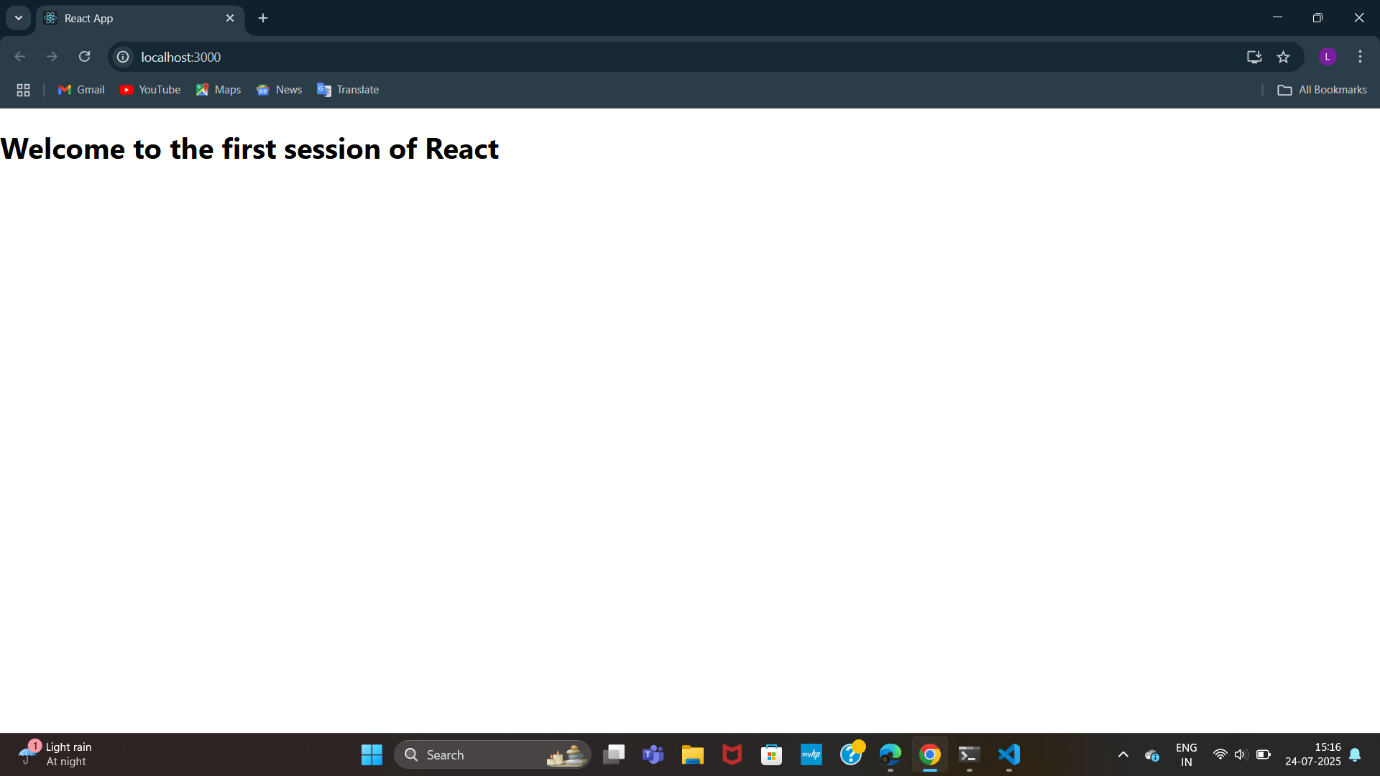
);

}

export default App;

**OUTPUT:**

****

****

## **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

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
3. Under Src folder add another file named “About.js”
4. 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.
5. Edit the App.js to invoke the Home, About and Contact component as follows:
6. 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:

**Concept:**

**React Components**

* A building block of a React UI.
* A JavaScript function or class that **returns JSX** (HTML-like code).
* Makes code **modular, reusable, and manageable**.

**Components vs JavaScript Functions**

* **React Component**: Returns JSX, can manage state & lifecycle.
* **JS Function**: Returns a value, doesn't manage UI or state.
* **Component is a special JS function** used to build UI.

**Types of Components**

* **Function Components**: Simpler, use Hooks, modern standard.
* **Class Components**: Older, use state & lifecycle with class syntax.

**Class Component**

* Defined using class keyword, extends React.Component.
* Uses constructor, this.state, and lifecycle methods.
* Has a required render() method to return JSX.

class Welcome extends React.Component {

render() {

return <h1>Hello</h1>;

}

}

**Function Component**

* Simple JS function that returns JSX.
* Can use useState(), useEffect() hooks for state/lifecycle.
* No need for this, constructor, or render().

function Welcome() {

return <h1>Hello</h1>;

}

**Component Constructor**

* Only in class components.
* Used to **initialize state** and **bind methods**.
* Always call super(props) first.

constructor(props) {

super(props);

this.state = { name: "React" };

}

**render() Function**

* Part of class components.
* **Returns JSX** to define what shows on the screen.
* React automatically calls render() on state/prop change.

render() {

return <div>Hello</div>;

}

**CODE:**

**StudentApp**

**Src/Components/Home.js**

import React, { Component } from 'react';

class Home extends Component {

render() {

return (

<div>

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

</div>

);

}

}

export default Home;

**About.js**

import React, { Component } from 'react';

class About extends Component {

render() {

return (

<div>

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

</div>

);

}

}

export default About;

**Contact.js**

import React, { Component } from 'react';

class Contact extends Component {

render() {

return (

<div>

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

</div>

);

}

}

export default Contact;

**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="App">

<Home />

<About />

<Contact />

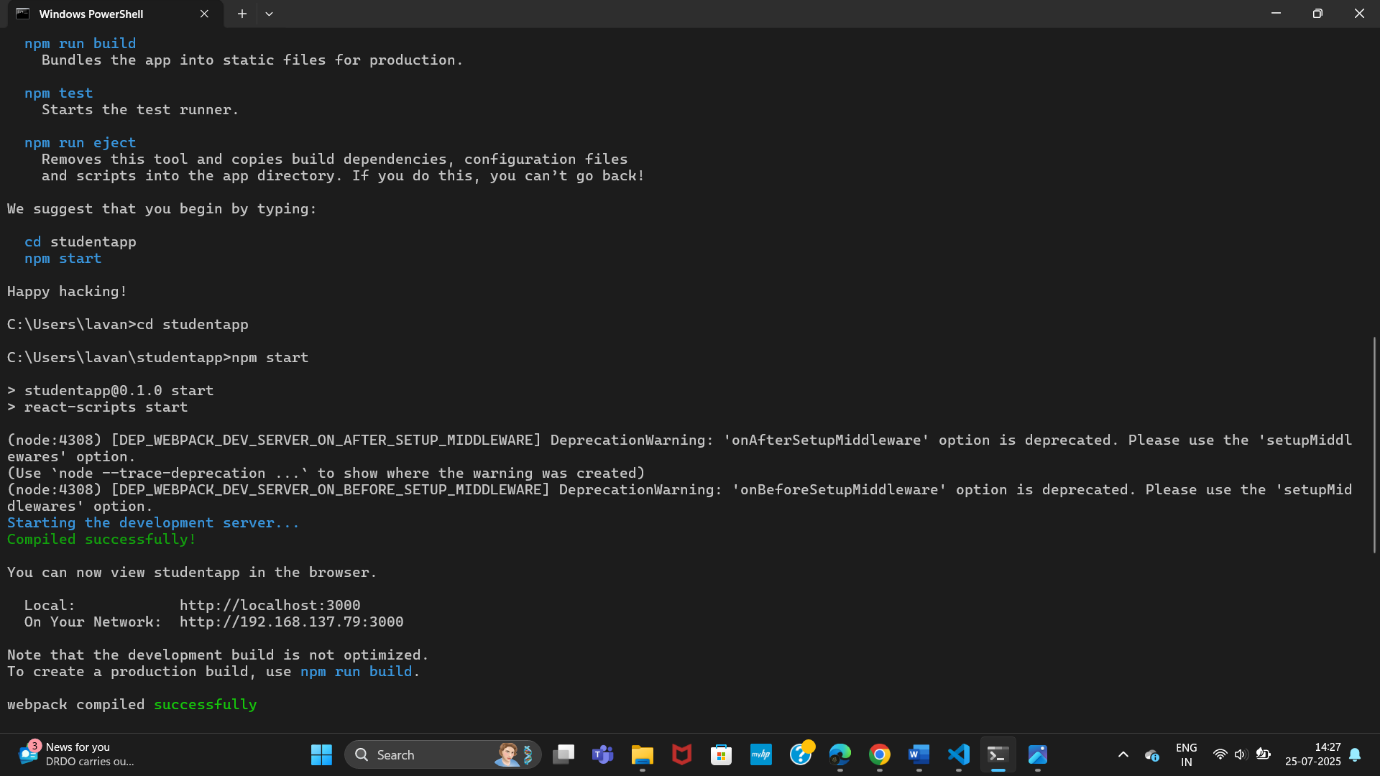
</div>

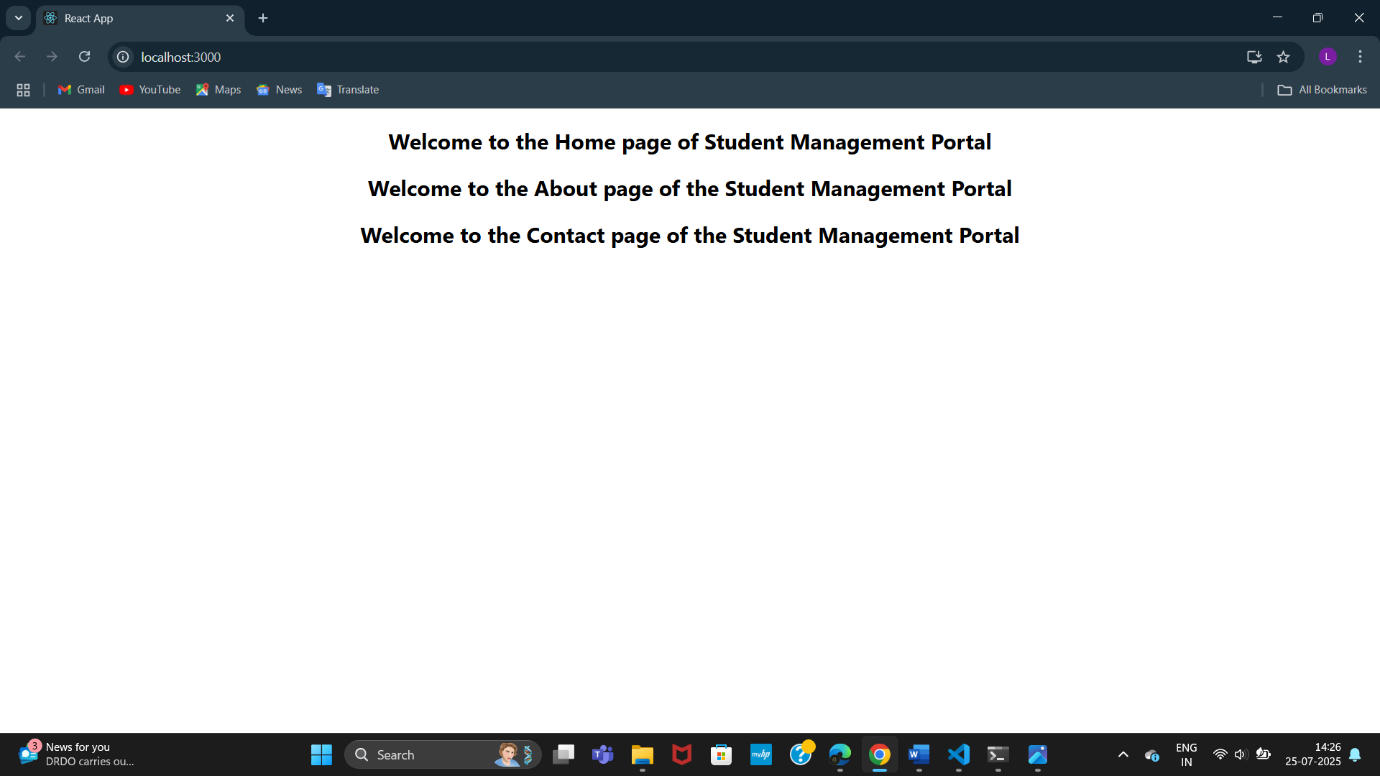
);

}

export default App;

**OUTPUT:**





## 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

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
3. Create a Folder named Stylesheets and add a file named “mystyle.css” in order to add some styles to the components:
4. Edit the App.js to invoke the CalculateScore functional component as follows:
5. 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:**

**Scorecalculatorapp**

**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-container">

<h2>Student Score Calculator</h2>

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

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

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

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

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

</div>

);

}

export default CalculateScore;

**src/Stylesheets/mystyle.css**

.score-container {

background-color: #f4f4f4;

border: 2px solid #ccc;

padding: 20px;

border-radius: 10px;

width: 400px;

margin: 30px auto;

font-family: Arial, sans-serif;

box-shadow: 0 0 10px rgba(0,0,0,0.1);

}

.score-container h2 {

color: #2e6da4;

text-align: center;

}

.score-container p {

font-size: 16px;

margin: 10px 0;

}

**App.js**

import React from 'react';

import './App.css';

import CalculateScore from './Components/CalculateScore';

function App() {

return (

<div className="App">

<CalculateScore

name="Divya"

school="Greenwood High School"

total={450}

goal={5}

/>

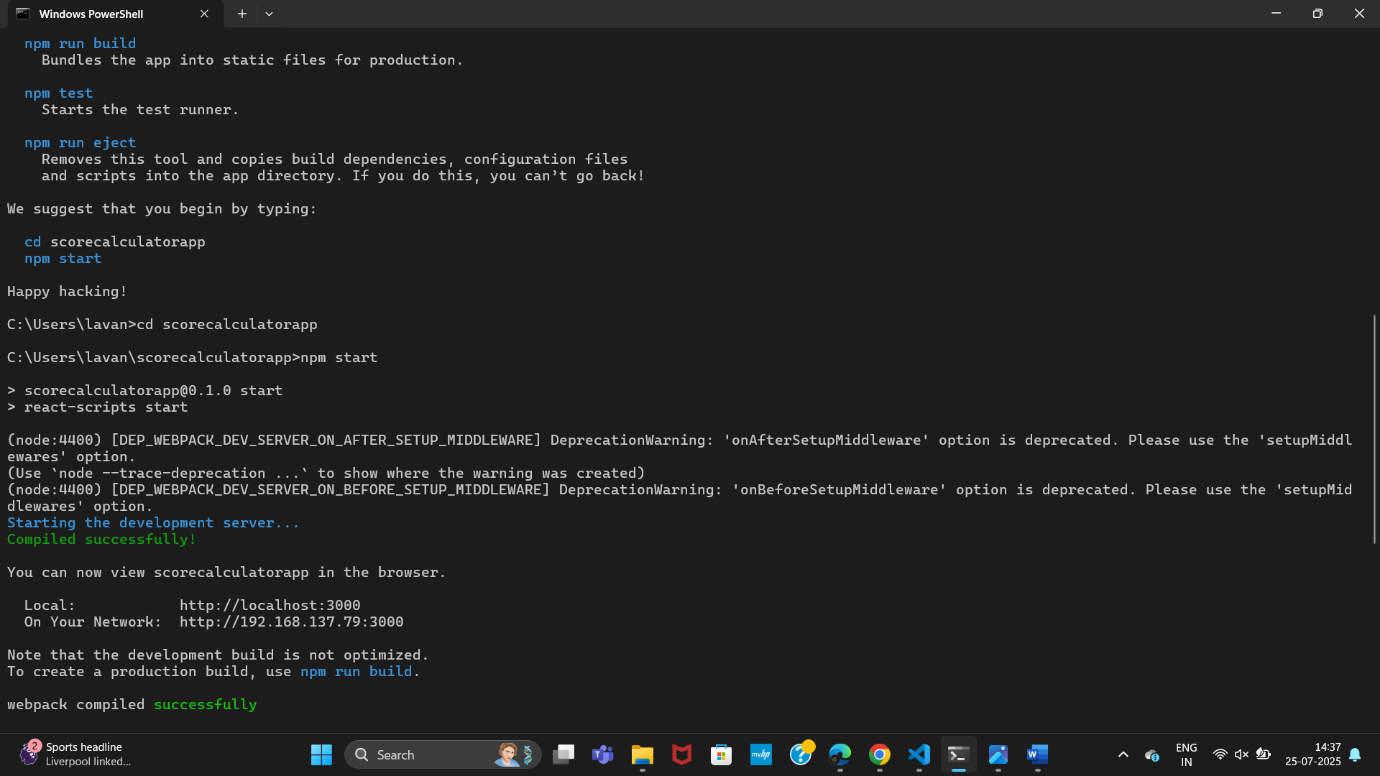
</div>

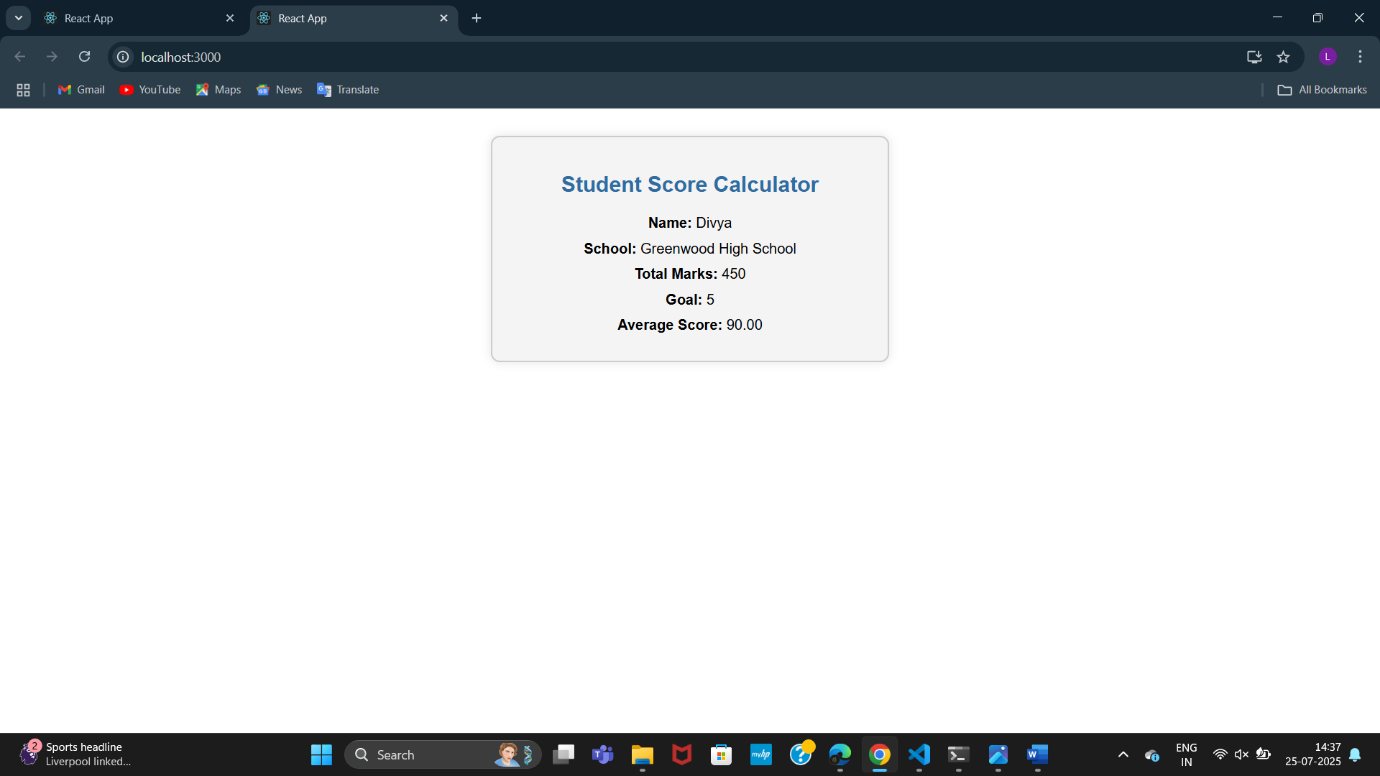
);

}

export default App;

**OUTPUT:**





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
* Create a new react application using *create-react-app* tool with the name as “blogapp”
* Open the application using VS Code
* Create a new file named as **Post.js** in **src folder** with following properties
* Create a new class based component named as **Posts** inside **Posts.js** file
* Initialize the component with a list of Post in state of the component using the constructor
* 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>)
* Implement the **componentDidMount()** hook to make calls to **loadPosts()** which will fetch the posts
* Implement the **render()** which will display the title and post of posts in html page using heading and paragraphs respectively.
* Define a **componentDidCatch()** method which will be responsible for displaying any error happing in the component as alert messages.
* Add the Posts component to App component.
* Build and Run the application using *npm start* command.

**Concept:**

**Need and Benefits of Component Lifecycle**

* Allows **controlled execution** of code at different phases (creation, update, deletion).
* Helps manage:
  + **Data fetching**
  + **DOM manipulation**
  + **Timers and subscriptions**
* Improves performance and **prevents memory leaks**.
* Enables **debugging and error handling** at specific points.

**Lifecycle Hook Methods (Class Components)**

🔹 **Mounting (Component is being created and inserted into DOM)**

* constructor()
* static getDerivedStateFromProps()
* render()
* componentDidMount()

🔹 **Updating (Component is re-rendered due to changes in props/state)**

* static getDerivedStateFromProps()
* shouldComponentUpdate()
* render()
* getSnapshotBeforeUpdate()
* componentDidUpdate()

🔹 **Unmounting (Component is removed from DOM)**

* componentWillUnmount()

🔹 **Error Handling**

* componentDidCatch()
* getDerivedStateFromError()

**Sequence of Steps in Rendering a Component**

1. constructor()  
   → Initialize state, bind functions.
2. getDerivedStateFromProps()  
   → Sync state with props (rarely used).
3. render()  
   → Return JSX for UI.
4. componentDidMount()  
   → Runs after initial render — used for API calls, DOM ops.
5. On updates:  
   → getDerivedStateFromProps()  
   → shouldComponentUpdate()  
   → render()  
   → getSnapshotBeforeUpdate()  
   → componentDidUpdate()
6. On unmounting:  
   → componentWillUnmount()

**CODE:**

**Blogapp**

**src/Post.js**

import React from 'react';

class Post extends React.Component {

render() {

const { title, body } = this.props;

return (

<div style={{ border: '1px solid #ccc', margin: '10px', padding: '10px' }}>

<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 = {

posts: [],

error: null

};

}

// Method to fetch data from API

loadPosts = () => {

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

.then(response => {

if (!response.ok) {

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

}

return response.json();

})

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

.catch(error => {

this.setState({ error });

});

};

// componentDidMount to load data after rendering

componentDidMount() {

this.loadPosts();

}

// Catch errors in children components

componentDidCatch(error, info) {

alert(`Something went wrong: ${error.toString()}`);

}

render() {

const { posts, error } = this.state;

if (error) {

return <h2>Error loading posts.</h2>;

}

return (

<div>

<h1>Blog Posts</h1>

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

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

))}

</div>

);

}

}

export default Posts;

**App.js**

import React from 'react';

import './App.css';

import Posts from './Posts';

function App() {

return (

<div className="App">

<Posts />

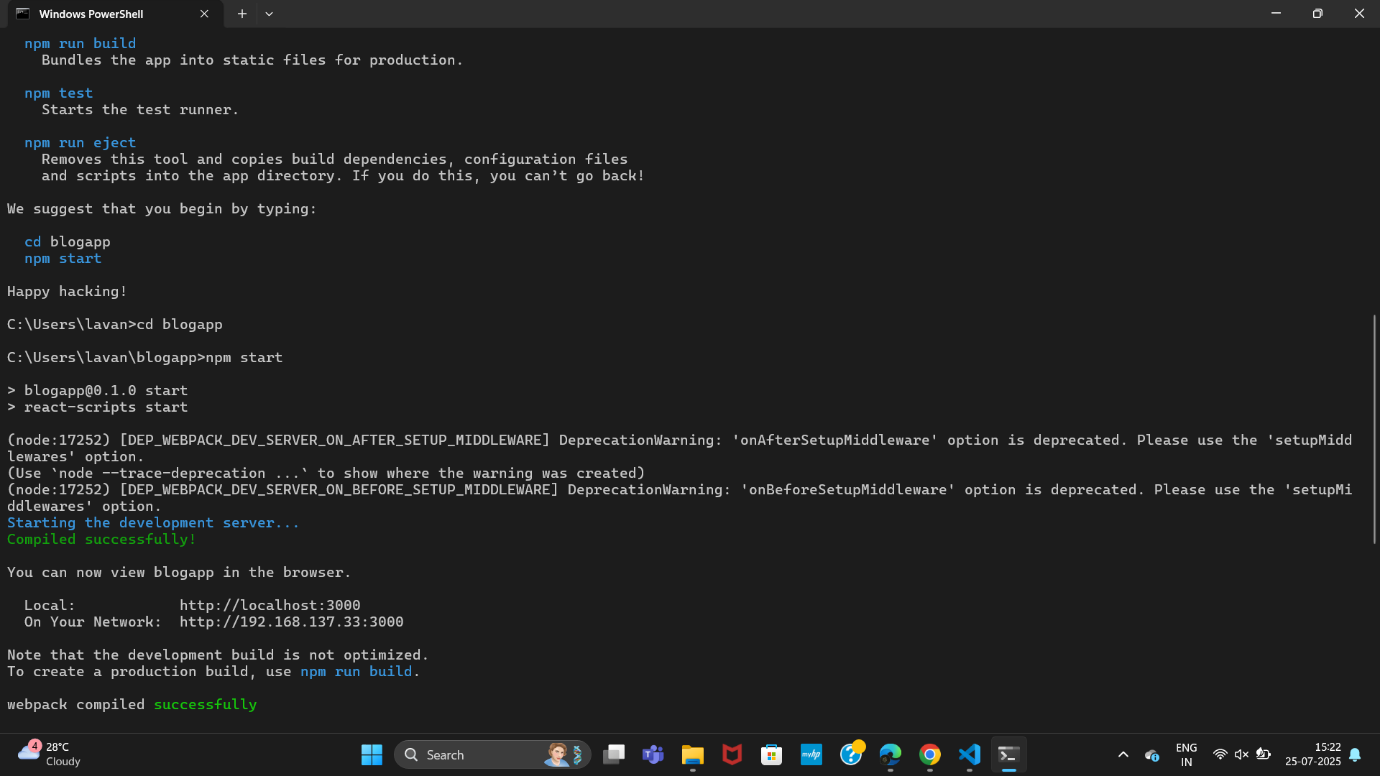
</div>

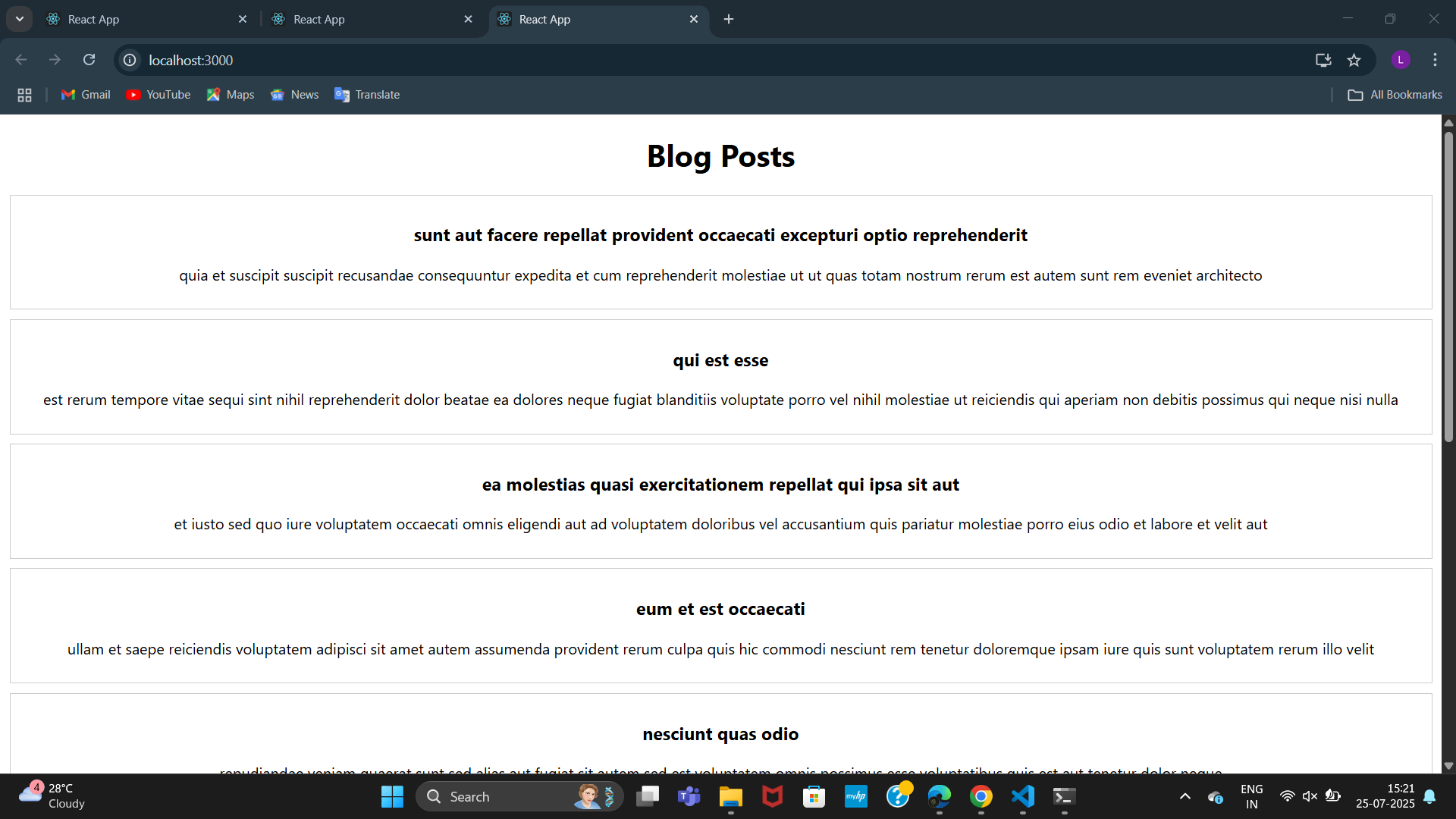
);

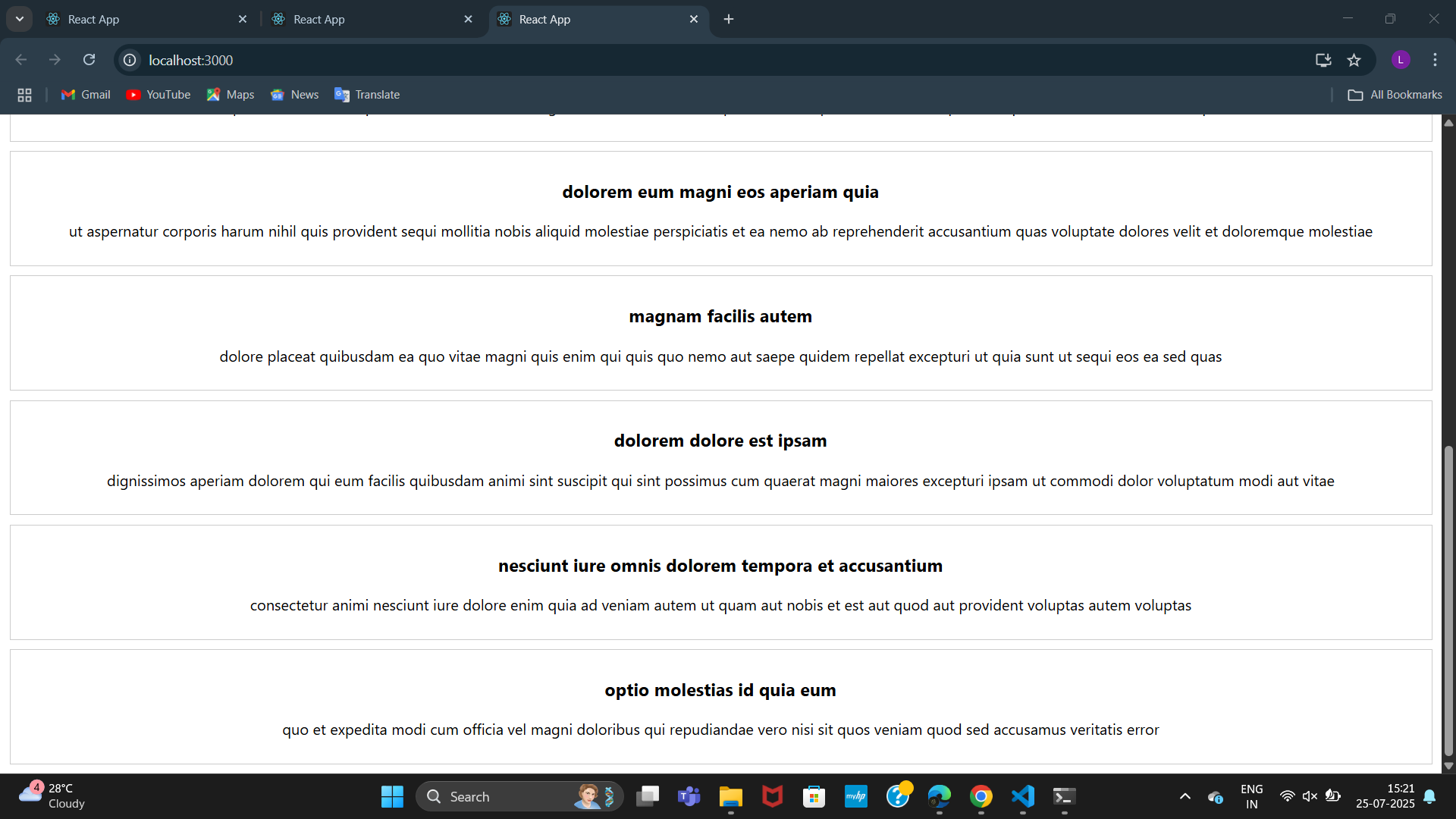
}

export default App;

**OUTPUT:**







5. **Objectives**

* Understanding the need for styling react component
* Working with CSS Module and inline styles
* Unzip the react application in a folder
* Open command prompt and switch to the react application folder
* Restore the node packages using the following commands



Figure 1: Restore packages

* Open the application using VS Code
* Create a new CSS Module in a file called “CohortDetails.module.css”
* 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*

* Define a css style for html <dt> element using tag selector. Set the font weight to 500.
* Open the cohort details component and import the CSS Module
* Apply the box class to the container div
* Define the style for <h3> element to use “green” color font when cohort status is “ongoing” and “blue” color in all other scenarios.

**Concept:**

**Need for Styling React Components**

* Makes the UI **visually appealing and user-friendly**.
* Ensures **brand consistency** across components.
* Helps in **responsive design** (works well across screen sizes).
* Enables **separation of concerns** — structure (JSX) and styling (CSS).
* Supports **dynamic styling** (based on state or props).

**Working with CSS Modules and Inline Styles in React**

**🔹 CSS Modules**

* A **modular CSS approach** in React.
* Styles are **scoped locally** to the component.
* Prevents **class name conflicts**.
* File name should end with .module.css

**CODE:**

**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, status, instructor }) => {

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>Instructor:</dt>

<dd>{instructor}</dd>

</dl>

</div>

);

};

export default CohortDetails;

**App.js**

import React from "react";

import CohortDetails from "./CohortDetails";

function App() {

return (

<div>

<CohortDetails name="React Bootcamp" status="ongoing" instructor="Divya Sharma" />

<CohortDetails name="Angular Training" status="completed" instructor="John Doe" />

</div>

);

}

export default App;

**OUTPUT:**

