**WEEK 6**

**Exercise:1**

**1.Define SPA and Its Benefits**

SPA (Single-Page Application) is a web application or website that interacts with the user by dynamically rewriting the current page, rather than loading entire new pages from the server. This provides a smoother and faster user experience.

Benefits of SPA:

* Fast Loading: Only data is exchanged with the server, not full HTML pages.
* Improved User Experience: Feels more like a desktop app with smooth transitions.
* Reduced Server Load: Server provides data through APIs; rendering happens on the client.
* Easy Debugging: Tools like Chrome DevTools help in inspecting SPA behavior.

**2. Define React and Identify Its Working**

React is an open-source JavaScript library developed by Facebook for building user interfaces, especially for single-page applications.

How React Works:

* Uses a component-based architecture, breaking the UI into reusable pieces.
* Uses a virtual DOM to efficiently update the UI.
* React detects changes in the data and updates only the necessary parts of the DOM.

**3. Identify the Differences Between SPA and MPA**

| Feature | SPA (Single Page App) | MPA (Multi Page App) |
| --- | --- | --- |
| Page Reload | No full reload; only updates content | Full page reload on each navigation |
| Speed | Faster once loaded | Slower due to frequent server requests |
| Development Approach | Client-side rendering | Server-side rendering |
| Complexity | More JavaScript-heavy | Easier backend logic but more page transitions |
| Examples | Gmail, Facebook, Twitter | Amazon, LinkedIn (older version), Wikipedia |

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

Pros:

* Faster interactions after initial load
* Better user experience with dynamic content
* Easy to develop mobile-friendly apps
* Efficient use of APIs

Cons:

* Poor SEO (though can be improved with SSR)
* Initial loading might take longer
* Browser back/forward buttons need handling

**5. Explain About React**

React is a front-end JavaScript library for building interactive UIs. It allows developers to create large web applications that can change data without reloading the page.

Key Highlights:

* Developed by Facebook
* Component-based structure
* Supports hooks for functional components
* Uses JSX (JavaScript + XML)

**6. Define Virtual DOM**

Virtual DOM (Document Object Model) is a lightweight copy of the actual DOM.

* React maintains a virtual DOM in memory.
* When a component's state changes, a new virtual DOM is created and compared with the previous one.
* Only the differences (called “diff”) are updated in the real DOM.
* This makes updates faster and more efficient.

**7. Explain Features of React**

* Component-Based: UI is broken into small, reusable components.
* Virtual DOM: Optimizes performance by reducing direct DOM manipulation.
* Reusable Code: Components can be reused across the app.
* One-Way Data Binding: Data flows in one direction, making debugging easier.
* JSX Syntax: Allows HTML-like code in JavaScript, improving readability.
* React Hooks: Add state and lifecycle methods to functional components.
* Ecosystem Support: Rich ecosystem with tools like Redux, React Router, etc

.

**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.**

**App.js**

function App() {

return (

<div>

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

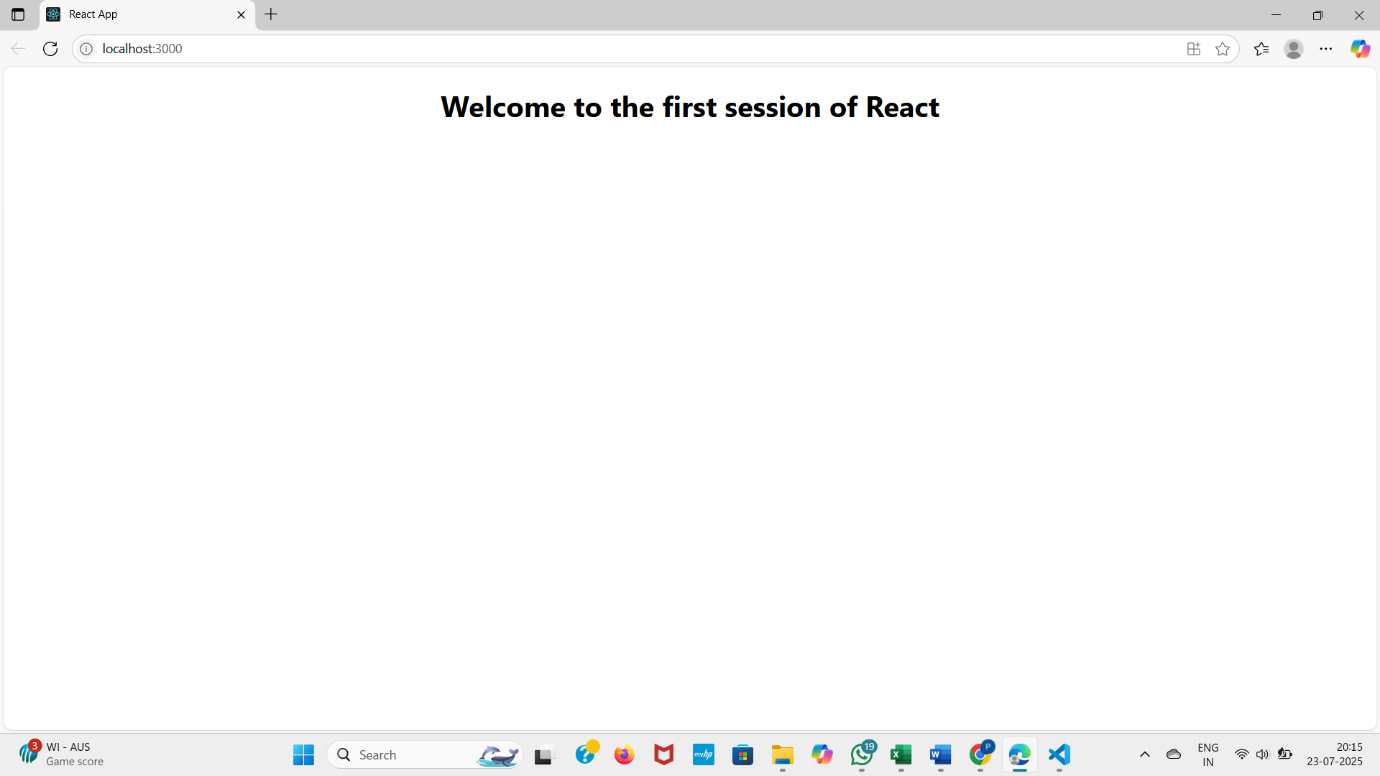
</div>

);

}

export default App;

**OUTPUT**



**Exercise:2**

**1. Explain React Components**

**React Components** are the building blocks of a React application. They allow you to divide the UI into smaller, reusable pieces that manage their own state and logic.

* A component can be thought of as a **JavaScript function or class** that returns a portion of the UI (written in JSX).
* React renders components to build the full user interface.

Example:

function Welcome() {

return <h1>Hello, Welcome!</h1>;

}

**2. Differences Between Components and JavaScript Functions**

| **Aspect** | **React Component** | **JavaScript Function** |
| --- | --- | --- |
| Purpose | Returns UI (JSX) | Executes logic or returns data |
| Return Type | Returns JSX (React elements) | Returns any data type |
| React Integration | Works within React rendering system | Standalone, not tied to React |
| Lifecycle Methods | Supports lifecycle methods (in class components) | No lifecycle awareness |
| State Handling | Can have state (using hooks or class state) | Doesn’t handle UI state |

**3. Types of Components**

React has two main types of components:

**Class Components**

* Uses ES6 class syntax
* Can hold state and use lifecycle methods

**Function Components**

* Simpler, written as JavaScript functions
* Can use hooks (useState, useEffect) for state and effects

**4. Explain Class Component**

A **class component** is a React component defined using ES6 class syntax.

**Example:**

import React, { Component } from 'react';

class Welcome extends Component {

render() {

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

}

}

**Key Features:**

* Can use lifecycle methods like componentDidMount
* Uses this.state and this.setState for managing state
* Requires a render() method to return JSX

**5. Explain Function Component**

A **function component** is a simpler way to create a React component using a function.

**Example:**

function Welcome() {

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

}

**With Hooks:**

jsx

CopyEdit

import { useState } from 'react';

function Counter() {

const [count, setCount] = useState(0);

return <button onClick={() => setCount(count + 1)}>Count: {count}</button>;

}

**Advantages:**

* Shorter syntax
* Easier to understand
* Preferred in modern React using hooks

**6. Define Component Constructor**

The **constructor** is a special method used in class components to:

* Initialize the component's state
* Bind event handlers

**Example:**

class Welcome extends React.Component {

constructor(props) {

super(props);

this.state = { name: 'User' };

}

}

**7. Define render() Function**

The render() function is a **mandatory method** in class components that:

* Describes what to display on the screen
* Returns JSX

**Example:**

render() {

return <h1>Hello World</h1>;

}

**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.**

**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 React from 'react';

import Home from './Components/Home';

import About from './Components/About';

import Contact from './Components/Contact';

function App() {

return (

<div>

<Home />

<About />

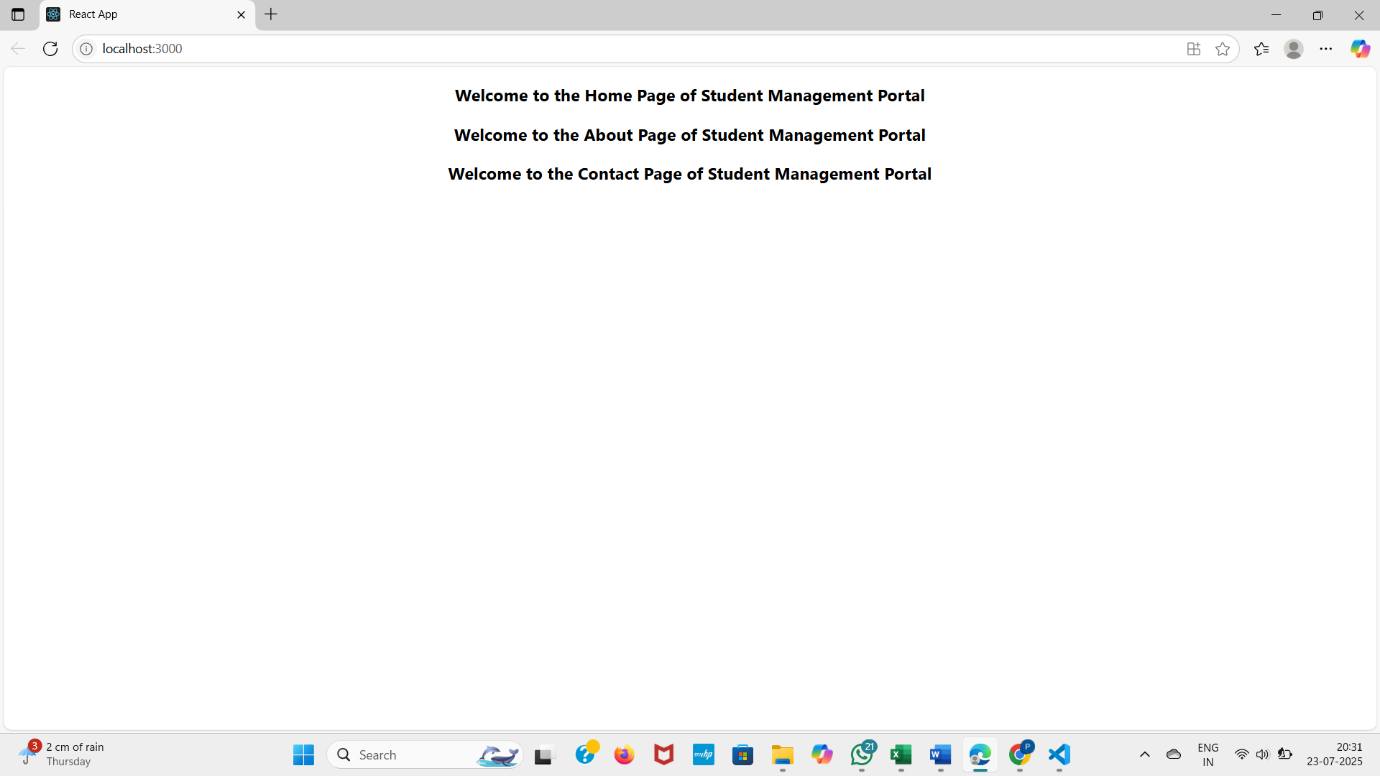
<Contact />

</div>

);

}

export default App;

**OUTPUT**

**Exercise:3**

**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.**

**Create CalculateScore.js**

import React from 'react';

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

const calcScore = (total, goal) => {

return (total / goal).toFixed(2);

};

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>

);

**4. Create Stylesheet**

.formatstyle {

font-family: Arial;

margin: 20px;

padding: 20px;

border: 2px solid #333;

border-radius: 10px;

background-color: #f7f7f7;

}

.Name, .School, .Total, .Score {

margin: 10px 0;

font-size: 18px;

}

**5. Modify App.js**

import React from 'react';

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

function App() {

return (

<div>

<CalculateScore

Name="Jessy"

School="ABC High School"

total={450}

goal={500}

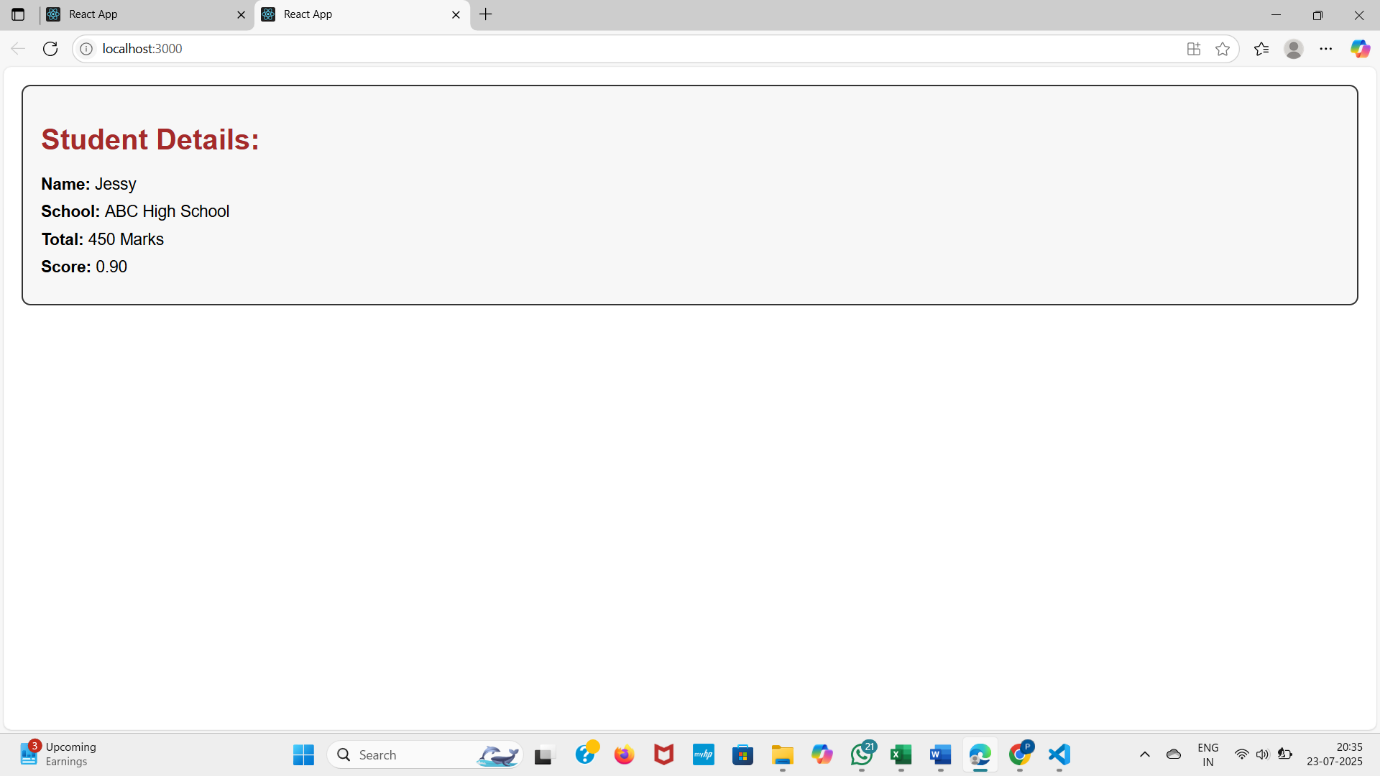
/>

</div>

);

}

export default App;

**OUTPUT**

**Exercise:4**

**Objectives**

**1.Explain the Need and Benefits of Component Life Cycle**

React components go through a series of stages during their existence — from creation to updating and removal. These stages are collectively known as the Component Life Cycle.

Why it's needed:

* To control and monitor component behavior during its life span.
* To fetch data, manipulate DOM, or clean up resources at the right time.
* To improve performance and avoid memory leaks.
* To integrate with external systems or APIs safely.

Benefits:

* Precise control over the app's behavior.
* Organized handling of tasks like fetching data or timers.
* Better debugging and testing.
* Efficient app updates and clean unmounting.

**2. Identify Various Life Cycle Hook Methods**

In class components, the most commonly used lifecycle methods are:

| Phase | Lifecycle Method | Purpose |
| --- | --- | --- |
| Mounting | constructor() | Initialize state, bind methods |
| Mounting | componentDidMount() | Run code after the component is rendered (e.g., fetch data) |
| Updating | shouldComponentUpdate() | Decide whether to re-render component |
| Updating | componentDidUpdate() | Act after the component has updated |
| Unmounting | componentWillUnmount() | Cleanup like clear timers or unsubscribe |
| Error Handling | componentDidCatch() | Catch and handle errors gracefully |

**3. List the Sequence of Steps in Rendering a Component**

The typical rendering sequence for a class component is:

1. constructor()
2. render()
3. componentDidMount()
4. (State/props change) → shouldComponentUpdate()
5. render() (again)
6. componentDidUpdate()
7. When component is removed → componentWillUnmount()

**1. Create Post.js**

import React from 'react';

export class Post extends React.Component {

render() {

return (

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

<h2>{this.props.title}</h2>

<p>{this.props.body}</p>

</div>

);

}

}

**2. Create Posts.js**

import React from 'react';

import { Post } from './Post';

export class Posts extends React.Component {

constructor(props) {

super(props);

this.state = {

posts: [],

hasError: false

};

}

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);

this.setState({ hasError: true });

});

};

componentDidMount() {

this.loadPosts();

}

componentDidCatch(error, info) {

alert("Something went wrong: " + error.message);

this.setState({ hasError: true });

}

render() {

if (this.state.hasError) {

return <h3>Error loading posts.</h3>;

}

return (

<div>

<h1>Blog Posts</h1>

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

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

))}

</div>

);

}

}

**3. App.js**

import React from 'react';

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

import { Posts } from './Posts';

function App() {

return (

<div>

{/\* <CalculateScore

Name="Jessy"

School="ABC High School"

total={450}

goal={500}

/> \*/}

<Posts />

</div>

);

}

export default App;

**CalculateScore.js**

import React from 'react';

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

const calcScore = (total, goal) => {

return (total / goal).toFixed(2);

};

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>

);

export default CalculateScore;

.formatstyle {

font-family: Arial;

padding: 20px;

margin: 20px;

background-color: #f9f9f9;

border: 1px solid gray;

border-radius: 10px;

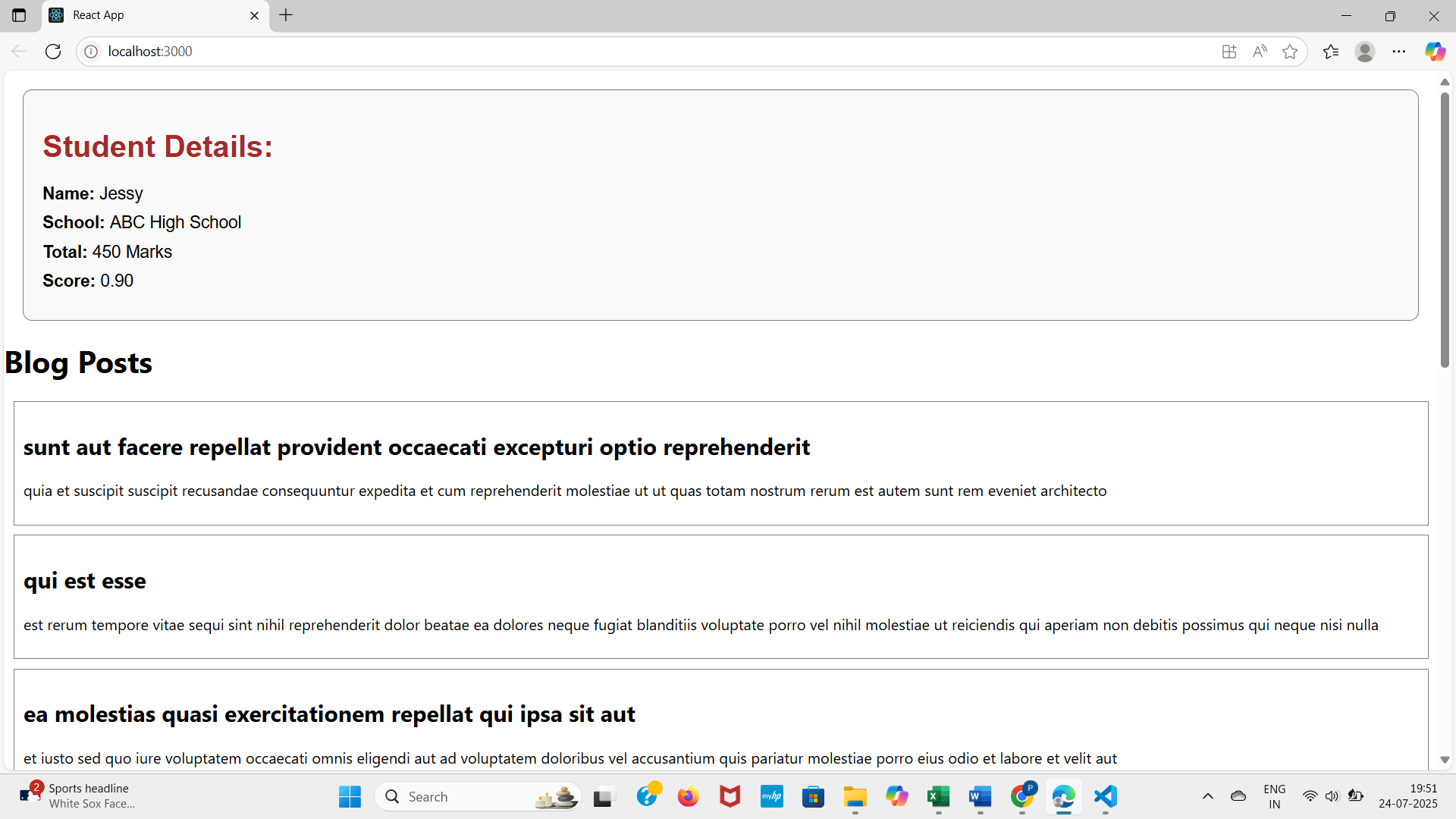
}

.Name, .School, .Total, .Score {

margin-bottom: 10px;

font-size: 18px;

**OUTPUT**



**Exercise :5**

**1. Understanding the Need for Styling React Components**

Styling is essential in React to:

* Improve user experience and visual appeal
* Clearly differentiate component states (e.g., active, completed)
* Maintain a consistent look and feel
* Make components more readable and interactive
* Help in branding and responsive layouts

React supports various styling methods like:

* Plain CSS files
* CSS Modules
* Inline styling
* CSS-in-JS libraries (e.g., styled-components**)**

**2. Working with CSS Module and Inline Styles**

CSS Modules:

* CSS Modules allow scoped and reusable styles specific to a component.
* You import styles from a .module.css file and use them as JavaScript objects.
* This prevents class name conflicts and promotes modularity.

**Example:**

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

<div className={styles.box}></div>

**Inline Styles:**

* Inline styles are written directly in JSX using JavaScript objects.
* Useful for dynamic styling based on props or state.

**Example:**

const statusStyle = {

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

};

<h3 style={statusStyle}>Cohort Name</h3>

**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.**

**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, trainer, startDate, endDate }) => {

const statusStyle = {

color: status.toLowerCase() === 'ongoing' ? 'green' : 'blue'

};

return (

<div className={styles.box}>

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

<dl>

<dt>Status:</dt>

<dd>{status}</dd>

<dt>Trainer:</dt>

<dd>{trainer}</dd>

<dt>Start Date:</dt>

<dd>{startDate}</dd>

<dt>End Date:</dt>

<dd>{endDate}</dd>

</dl>

</div>

);

};

export default CohortDetails;

**App.js**

import React from 'react';

import CohortDetails from './Components/CohortDetails';

function App() {

return (

<div>

<h1>Academy Cohort Dashboard</h1>

<CohortDetails

name="React Training - July 2025"

status="ongoing"

trainer="Mr. Rahul"

startDate="2025-07-01"

endDate="2025-08-15"

/>

<CohortDetails

name="Java Bootcamp - May 2025"

status="completed"

trainer="Ms. Priya"

startDate="2025-05-01"

endDate="2025-06-20"

/>

</div>

);

}

export default App;

**App.js**

import React from 'react';

import CohortDetails from './Components/CohortDetails';

import './App.css'; // Add this if you want to style the container

function App() {

return (

<div>

<h1>Academy Cohort Dashboard</h1>

{/\* New container div for side-by-side layout \*/}

<div className="cohort-container">

<CohortDetails

name="React Training - July 2025"

status="ongoing"

**OUTPUT**

