**WEEK 060**

**React**

**Exercise-01**

**1. Define SPA and its Benefits**

SPA (Single Page Application) is a web application that loads a single HTML page and dynamically updates content without requiring full page reloads.

**Benefits:**

* Faster user experience after the initial load
* Reduces server load by minimizing full page requests
* Smooth navigation without reloading
* Uses AJAX or APIs for dynamic data updates

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

React is a JavaScript library developed by Facebook for building user interfaces, especially for Single Page Applications.

**Working of React:**

* Builds UIs using reusable components
* Maintains a virtual DOM for efficient updates
* Only updates parts of the UI that change
* Handles routing and state management in the client-side

**3. Differences Between SPA and MPA**

| **Feature** | **SPA (Single Page App)** | **MPA (Multi Page App)** |
| --- | --- | --- |
| Navigation | No full page reload | Reloads entire page |
| Performance | Fast after initial load | Slower, especially on navigation |
| Data Handling | Uses AJAX or APIs for updates | Loads new HTML on every request |
| Development | Best for dynamic apps | Ideal for content-heavy or SEO-focused sites |
| Examples | Gmail, Facebook | Amazon, LinkedIn (classic), Wikipedia |

**4. Pros and Cons of Single Page Application**

**Pros:**

* Smooth and fast user interaction
* Efficient updates using client-side logic
* Minimal server communication
* Better performance after initial load

**Cons:**

* SEO limitations
* Larger initial bundle size
* Client-side routing complexity

**5. Explain About React**

* React is a declarative, component-based JavaScript library used to build interactive UIs efficiently.
* It helps in building fast-loading SPAs with reusable UI blocks called components.
* Emphasizes performance using virtual DOM
* Encourages code reuse with components
* Supports modern features like Hooks and JSX

**6. Define Virtual DOM**

Virtual DOM is an in-memory representation of the real DOM elements. It allows React to:

* Detect changes efficiently using diffing algorithms
* Update only modified elements in the real DOM
* Improve rendering performance by avoiding full reloads

**7. Features of React**

* **Component-Based Architecture:** Build reusable and manageable UI blocks
* **Virtual DOM:** Boosts performance with minimal DOM manipulation
* **One-Way Data Binding:** Predictable flow of data
* **JSX Syntax:** Combines HTML with JavaScript for ease of development
* **Hooks Support:** Functional component state and side-effect handling
* **Fast Rendering :** Only changes the necessary part of the page
* **Rich Ecosystem:** Compatible with libraries like Redux, React Router

**CODE:**

**App.js:**

import React from 'react';

function App() {

  return (

    <div style={{ display: 'flex', height: '100vh', justifyContent: 'center', alignItems: 'center' }}>

      <h1 style={{ fontWeight: 'bold' }}>Welcome to the first session of React</h1>

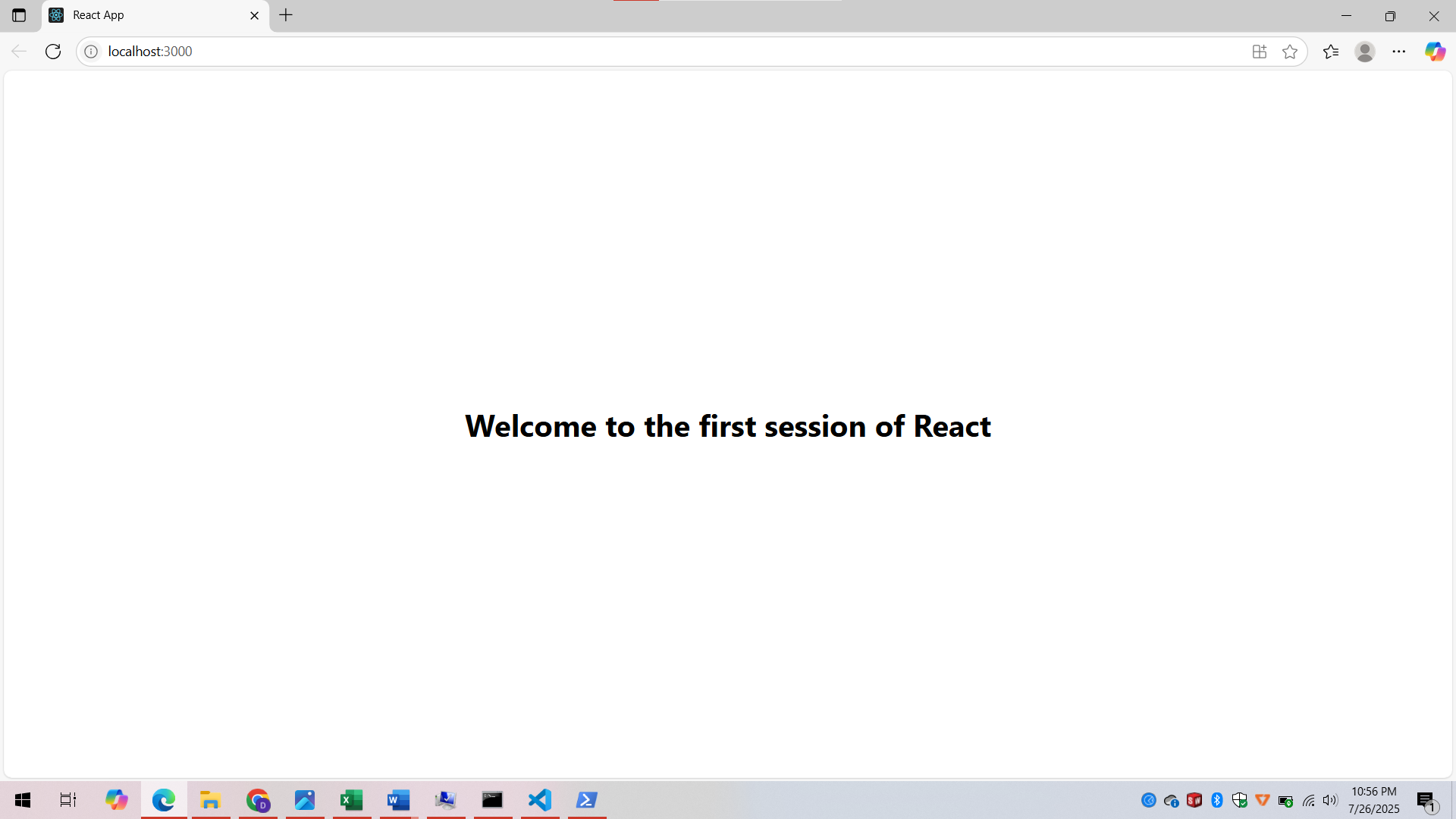
    </div>

  );

}

export default App;

**Output:**

****

**Explanation:**

* Installed Node.js and npm to set up the React development environment.
* Created a new React project using the command: npx create-react-app myfirstreact.
* Opened the project in Visual Studio Code and modified the App.js file.
* Replaced the default content with a custom heading: “Welcome to the first session of React”, styled to appear bold and centered using inline CSS.
* Ran the application using npm start, and the output was successfully displayed in the browser at http://localhost:3000.

**Exercise 02:**

**1.Explain React Components**

React components are the building blocks of any React application. A component is a reusable, self-contained piece of UI that can render HTML elements, manage its own state, and respond to user interactions.

React components help in:

* Breaking the UI into smaller parts
* Reusing code
* Managing complex interfaces efficiently

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

| **Feature** | **React Component** | **JavaScript Function** |
| --- | --- | --- |
| Purpose | Builds UI in React | Executes logic or computation |
| Return Value | Returns JSX (UI) | Returns values (number, string, object) |
| React Lifecycle Methods | Available (like componentDidMount) | Not available |
| Integration with State | Can use state and props | No built-in support for state or props |
| JSX Support | Returns HTML-like JSX | Does not support JSX directly |

**3. Types of Components**

React components are mainly of two types:

* Function Components
* Class Components

Newer React projects mostly use function components with hooks, but class components are still valid and useful.

**4. Explain Class Component**

A class component is a component defined using ES6 class syntax. It can hold internal state and use lifecycle methods like componentDidMount.

Example:

class Welcome extends React.Component {

render() {

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

}

}

**Key features:**

* Uses render() method to return JSX
* Can use this.state and this.props
* Has access to lifecycle methods

**5. Explain Function Component**

A function component is a simpler way to write components using JavaScript functions. It can now manage state and side effects using React Hooks (e.g., useState, useEffect).

**Example:**

function Welcome() {

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

}

**Key features:**

* Simpler and cleaner syntax
* No this keyword needed
* Uses Hooks to handle state and lifecycle

**6. Define Component Constructor**

In class components, the constructor is a special method used to:

* Initialize state
* Bind event handlers

Syntax:

constructor(props) {

super(props);

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

}

Always call super(props) first to use this.props inside the constructor.

**7. Define render() Function**

The render() function is a required method in every class component. It tells React what to display on the screen. It returns JSX (the UI elements).

Syntax:

render() {

return (

<div>

<h1>Hello World</h1>

</div>

);

}

React calls render() automatically when the component needs to update.

**CODE:**

**Home.js (File created in Folder “Component” in src):**

import React from 'react';

function Home() {

  return (

    <div>

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

    </div>

  );

}

export default Home;

**About.js (File created in Folder “Component” in src):**

import React from 'react';

function About() {

  return (

    <div>

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

    </div>

  );

}

export default About;

**Contact.js (File created in Folder “Component” in src):**

import React from 'react';

function Contact() {

  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 Home from './Components/Home';

import About from './Components/About';

import Contact from './Components/Contact';

function App() {

  return (

    <div>

      <Home />

      <About />

      <Contact />

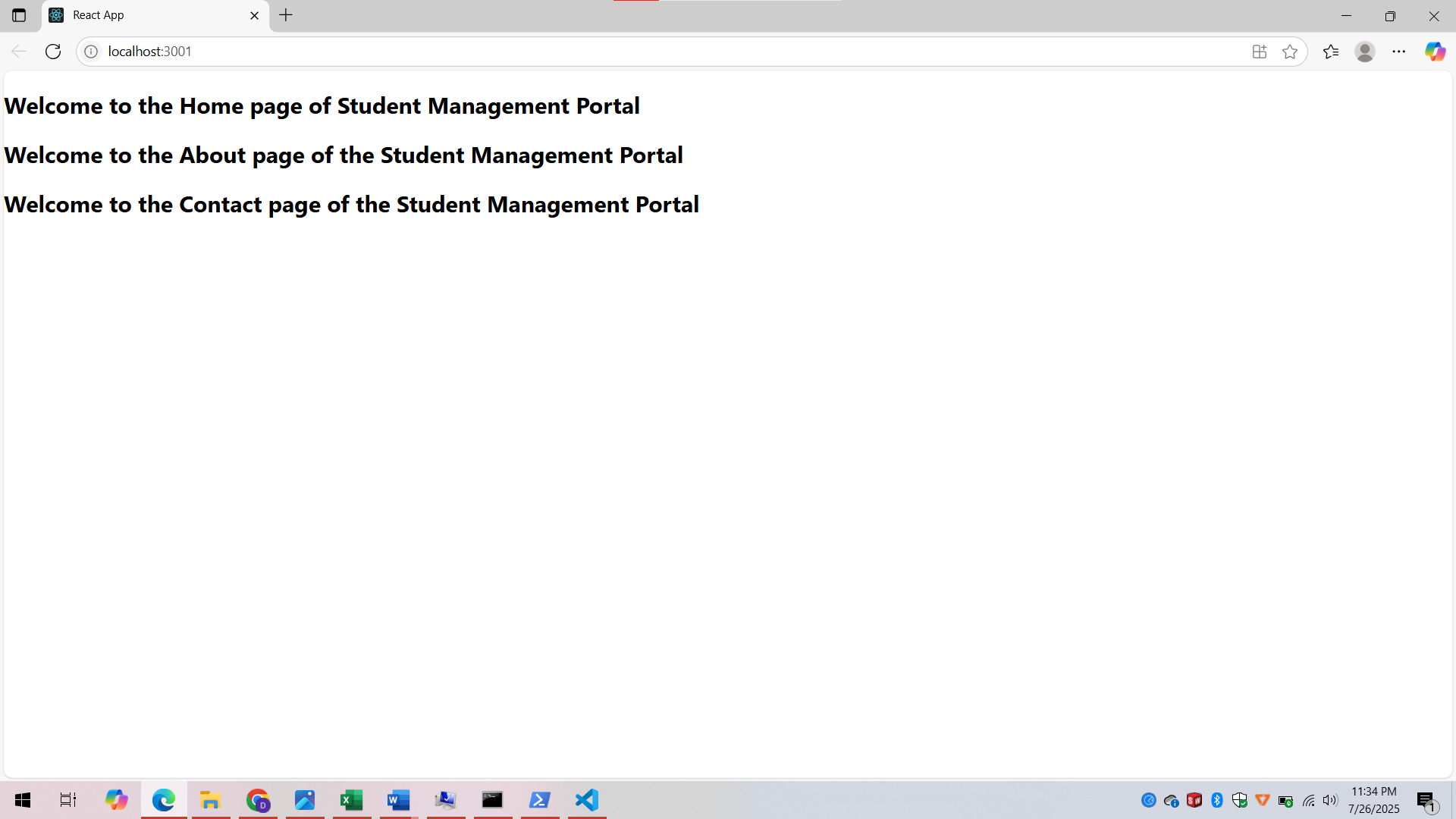
    </div>

  );

}

export default App;

**Output:**

****

**Explanation:**

* A folder named Components was created under the src directory to organize all component files.
* Three functional components (Home.js, About.js, and Contact.js) were created using JavaScript functions that return JSX to display welcome messages.
* Each component was exported using export default and then imported into App.js to be used inside the main application.
* Inside App.js, the components were rendered using JSX syntax like <Home />, <About />, and <Contact /> inside the App function.
* The project was run using the npm start command, and the output displayed all three messages on the browser at <http://localhost:3000>.

**Exercise 03:**

**1. Explain React Components**

* React components are the core building blocks of a React application.
* They are reusable, self-contained pieces of UI that define how a particular part of the application should look and behave.
* Components help in breaking down the UI into smaller, manageable sections, making development easier and more efficient.

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

| **Feature** | **React Component** | **JavaScript Function** |
| --- | --- | --- |
| Purpose | Builds and renders UI in a React app | Performs a specific task or calculation |
| Output | Returns JSX (UI code) | Returns values like numbers, strings, etc. |
| State and Props Support | Can use state and props | No support for state/props |
| Lifecycle Methods | Class components have lifecycle methods | No lifecycle features |
| JSX Compatibility | Built to handle and return JSX | Not designed to use JSX |

**3. Identify the Types of Components**

React supports two main types of components:

* Class Components – Use ES6 class syntax, support state and lifecycle methods.
* Function Components – Use JavaScript functions, support hooks for state and side effects.

**4. Explain Class Component**

A class component is a component that extends from

“React.Component”. It has access to React lifecycle methods like componentDidMount, and it uses a render() method to return JSX.

**Example:**

class Welcome extends React.Component {

render() {

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

}

}

**5. Explain Function Component**

A function component is a way to write components using regular JavaScript functions. It returns JSX and can use React Hooks (like useState, useEffect) for managing state and side effects.

**Example:**

function Welcome() {

return <h1>Hello, Function Component</h1>;}

**6. Define Component Constructor**

In class components, the constructor is a special method used for:

* Initializing state
* Binding event handlers

It’s the first method called when a component is created.

**Example:**

constructor(props) {

super(props);

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

}

super(props) is required to use this.props inside the constructor.

**7. Define render() Function**

The render() function is a mandatory method in class components. It returns the JSX code that defines what the component will display on the screen.

**Example:**

render() {

return (

<div>

<h2>Hello from render()</h2>

</div>

);

}

React automatically calls render() when the component needs to update or re-render.

**CODE:**

**CalculateScore.js (File created in Folder “Components” inside src):**

import React from 'react';

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

function CalculateScore() {

  const name = "Divyadharshiny";

  const school = "SKCET";

  const total = 550;

  const goal = 600;

  const average = (total / goal) \* 100;

  return (

    <div className="score-card">

      <h2>Student Score Calculator</h2>

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

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

      <p><strong>Total Score:</strong> {total}</p>

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

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

    </div>

  );

}

export default CalculateScore;

**mystyle.css (File created in Folder “Stylesheets” Folder inside src):**

.score-card {

  margin: 40px auto;

  width: 50%;

  padding: 20px;

  border: 2px solid #007bff;

  border-radius: 10px;

  background-color: #f5f5f5;

  font-family: Arial, sans-serif;

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

}

.score-card h2 {

  color: #007bff;

  text-align: center;

}

.score-card p {

  font-size: 16px;

  line-height: 1.6;

}

**App.js:**

import React from 'react';

import CalculateScore from './Components/CalculateScore';

function App() {

  return (

    <div className="App">

      <CalculateScore />

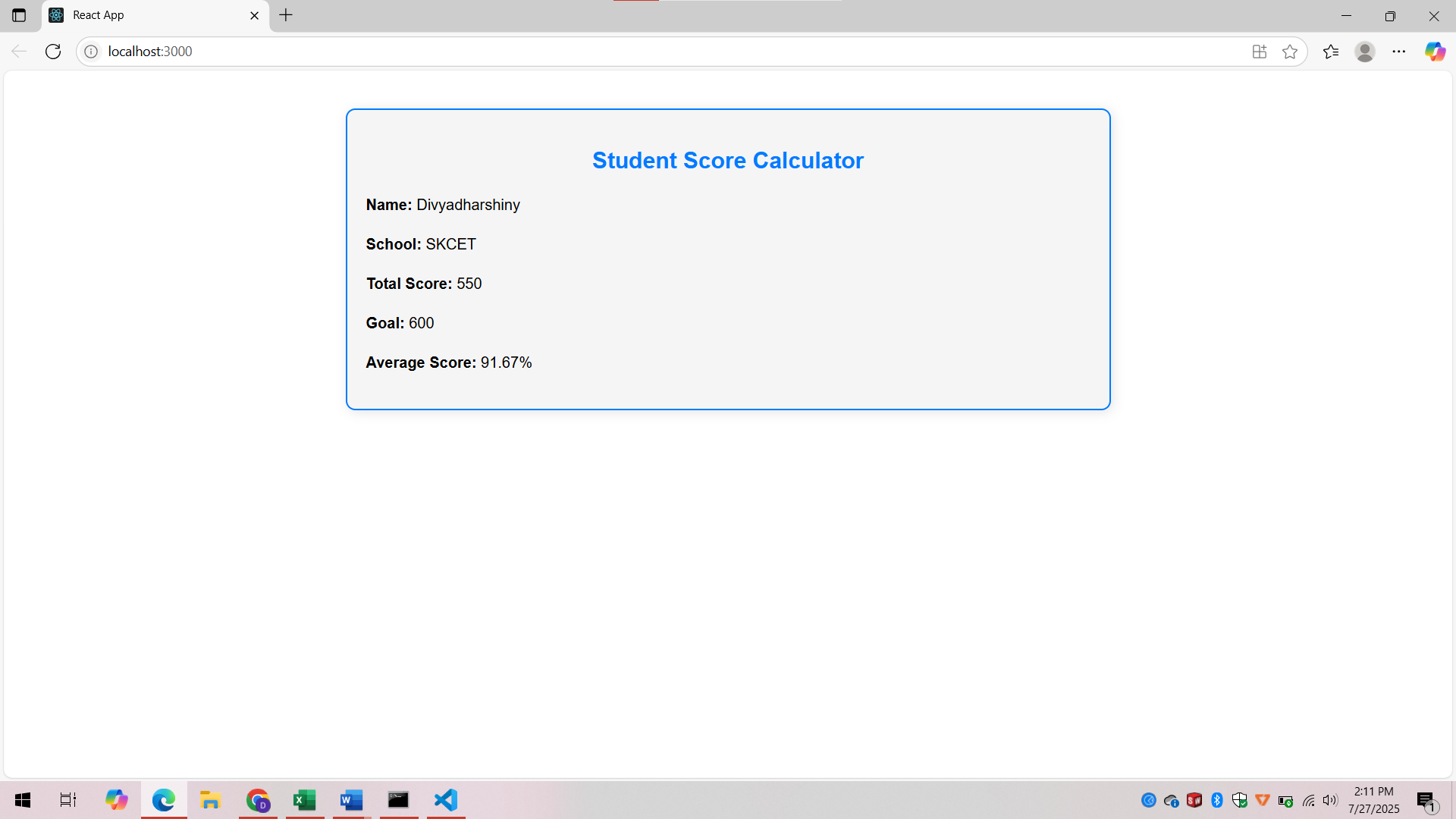
    </div>

  );

}

export default App;

**Output:**



**Explanation:**

* A React project named scorecalculatorapp was created using create-react-app to build a student score calculator.
* A functional component named CalculateScore was created inside the Components folder to calculate and display a student’s average score using static values for Name, School, Total, and Goal.
* The average was calculated using the formula

(total / goal) \* 100 and displayed using JSX inside the component.

* A separate CSS file named mystyle.css was created inside a Stylesheets folder to style the output with borders, background color, and font settings.
* The component was imported and rendered in App.js, and the application was run using npm start, displaying the result at <http://localhost:3000>.

**Exercise-04**

**1. Explain the Need and Benefits of Component Lifecycle**

The React component lifecycle refers to the different phases a component goes through from creation to removal. Each phase has special methods (called lifecycle hooks) that allow developers to run specific code during those moments.

**Need:**

* To control how a component behaves during mounting, updating, and unmounting.
* To handle tasks such as API calls, subscriptions, timers, or cleanup processes at the right time.

**Benefits:**

* Efficient resource management through controlled mounting and unmounting.
* Enables code execution at precise moments (e.g., after component loads).
* Improves performance by minimizing unnecessary re-renders.
* Allows better error handling with lifecycle error boundaries.

**2. Identify Various Lifecycle Hook Methods**

React class components include multiple lifecycle methods, divided into different phases:

**Mounting Phase (Component is created and inserted into DOM):**

* constructor() – Initializes state and props.
* static getDerivedStateFromProps() – Syncs state with props (rarely used).
* render() – Returns JSX to display UI.
* componentDidMount() – Invoked after the component is rendered for the first time (ideal for API calls).

**Updating Phase (Component updates due to state/props change):**

* static getDerivedStateFromProps()
* shouldComponentUpdate() – Optimizes rendering by checking if a re-render is needed.
* render()
* getSnapshotBeforeUpdate() – Captures info (like scroll position) before update.
* componentDidUpdate() – Executes after re-render.

**Unmounting Phase (Component is removed):**

* componentWillUnmount() – Used to clean up (e.g., stop timers, remove listeners).

**Error Handling Phase:**

* componentDidCatch() – Catches errors in rendering or lifecycle and handles them gracefully.

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

When a class component is mounted (rendered for the first time), React follows this sequence:

* constructor() – Initializes component state and binds methods.
* getDerivedStateFromProps() – Syncs state with props if needed.
* render() – Returns JSX to display the UI.
* componentDidMount() – Called after the component is rendered; suitable for API calls and DOM manipulation.

**CODE:**

**Post.js:**

class Post {

  constructor(userId, id, title, body) {

    this.userId = userId;

    this.id = id;

    this.title = title;

    this.body = body;

  }

}

export default Post;

**Posts.js:**

import React, { Component } from 'react';

import Post from './Post'; // Make sure you create Post.js as a model class (optional)

class Posts extends Component {

  constructor(props) {

    super(props);

    this.state = {

      posts: [],

      hasError: false

    };

  }

  loadPosts() {

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

      .then(response => response.json())

      .then(data => {

        const postList = data.map(post => new Post(post.userId, post.id, post.title, post.body));

        this.setState({ posts: postList });

      })

      .catch(error => {

        this.setState({ hasError: true });

        console.error('Error loading posts:', error);

      });

  }

componentDidMount() {

    this.loadPosts();

  }

componentDidCatch(error, info) {

    alert("Something went wrong while rendering posts.");

    console.error(error, info);

  }

  render() {

    if (this.state.hasError) {

      return <h2>Failed to load blog posts.</h2>;

    }

    return (

      <div className="post-container">

        <h1>Blog Posts</h1>

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

          <div key={post.id} className="post-card">

            <h2>{post.title}</h2>

            <p>{post.body}</p>

            <hr />

          </div>

        ))}

      </div>

    );

  }

}

export default Posts;

**App.js:**

import React from 'react';

import Posts from './Posts';

function App() {

  return (

    <div className="App">

      <Posts />

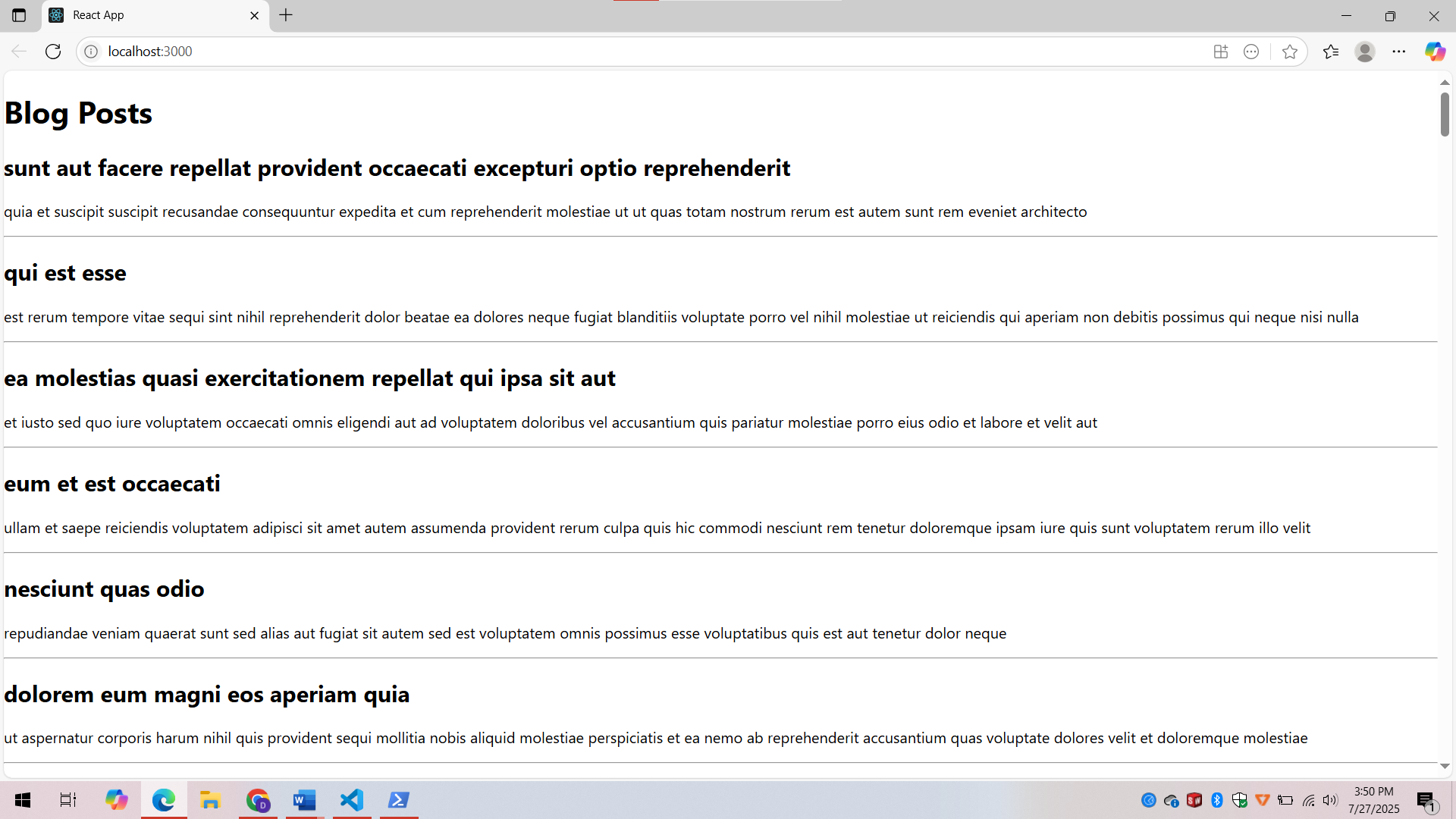
    </div>

  );

}

export default App;

**Output:**

****

**Explanation:**

* A class component named Posts is created to manage blog post data and lifecycle events.
* The loadPosts() method uses the Fetch API to retrieve post data from an external URL and updates the component’s state.
* The componentDidMount() lifecycle method is used to call loadPosts() when the component first mounts, ensuring data is loaded automatically.
* The componentDidCatch() lifecycle method handles errors during rendering and displays alerts if something goes wrong.
* The render() method dynamically displays the fetched posts using JSX, formatting each post with a title and body.

**Exercise-05**

**CODE:**

**CohortDetails.js(File created in Folder “CohortDetails” inside src):**

import React from 'react';

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

function CohortDetails({ cohort }) {

  const statusStyle = {

    color: cohort.status === 'ongoing' ? 'green' : 'blue',

  };

  return (

    <div className={styles.box}>

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

      <dl>

        <dt>Start Date:</dt>

        <dd>{cohort.startDate}</dd>

        <dt>End Date:</dt>

        <dd>{cohort.endDate}</dd>

        <dt>Status:</dt>

        <dd>{cohort.status}</dd>

      </dl>

    </div>

  );

}

export default CohortDetails;

**CohortDetails.module.css:**

.box {

  width: 300px;

  display: inline-block;

  margin: 10px;

  padding: 10px 20px;

  border: 1px solid black;

  border-radius: 10px;

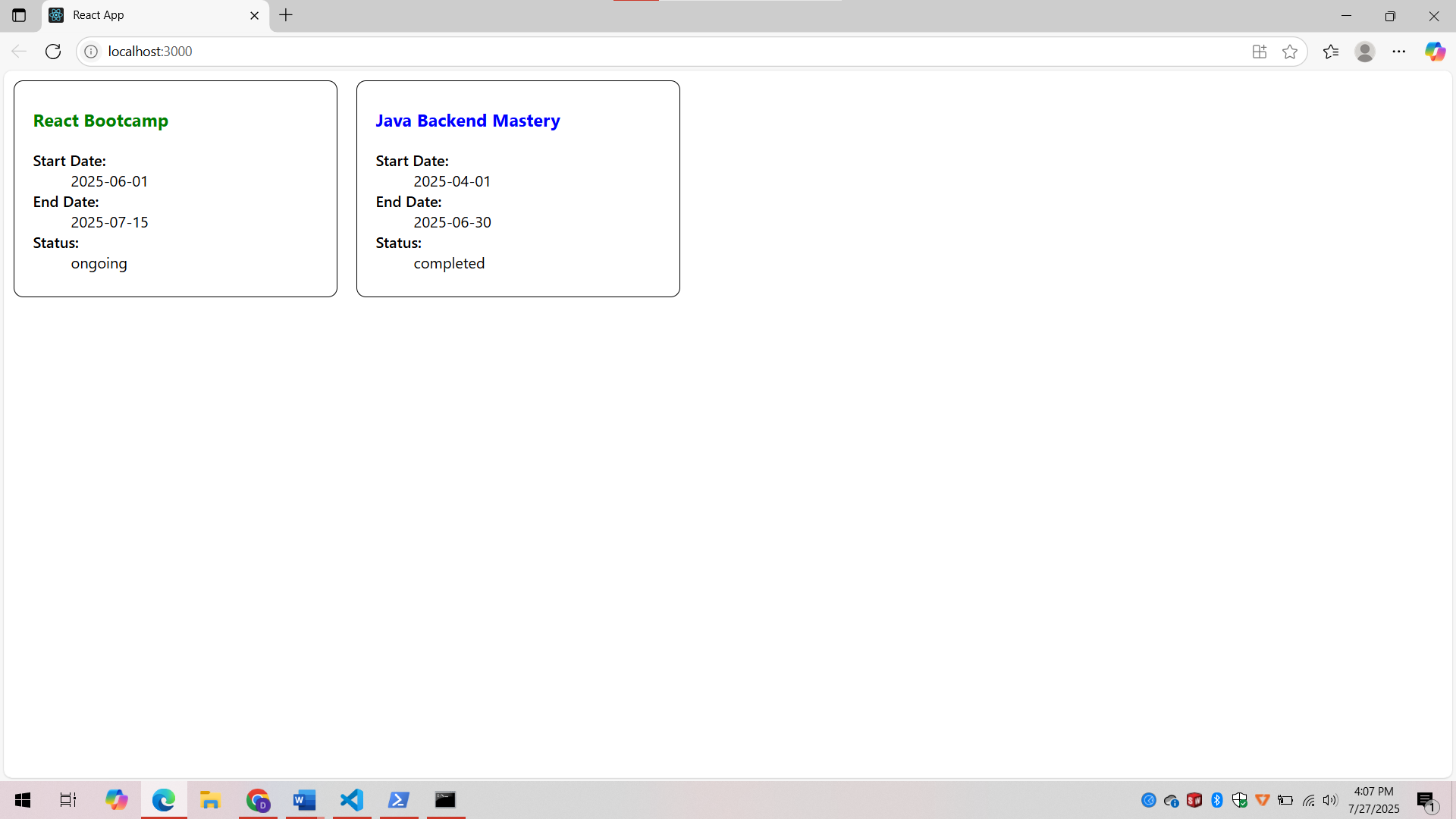
}

dt {

  font-weight: 500;

}

**Output:**



**Explanation:**

* A React component named CohortDetails is created to display information about each cohort, including its name, start and end dates, and status.
* The component uses a CSS Module (CohortDetails.module.css) to apply styles, including a bordered box layout with padding, margin, and rounded corners.
* The box class is applied using className={styles.box} to style the container div for each cohort.
* The status color is styled inline, using a conditional check: green for “ongoing” and blue for other statuses.
* A <dl> HTML element is used to semantically structure the cohort information, and the <dt> tag is styled with increased font weight for emphasis.