**REACT**

* **Define SPA and its benefits**

A Single Page Application (SPA) loads a single HTML page and dynamically updates content without reloading the page.

Benefits:

Faster experience for users

No full page reloads

Uses AJAX/Fetch to update content

Good for modern web apps

* **Define React and identify its working**

**React** is a **JavaScript library** created by Facebook for building fast and interactive **User Interfaces (UI)**.

It uses **Components** – reusable pieces of UI (e.g., button, form, header).

React updates the page efficiently using the **Virtual DOM**.

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

| **SPA** | **MPA** |
| --- | --- |
| Loads a single HTML file | Loads multiple HTML files |
| Faster performance | Slower due to full reloads |
| Good for dynamic apps | Good for large sites like blogs |
| Uses client-side routing | Uses server-side routing |

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

**Pros:**

Fast & smooth user experience

Easy to debug (with React DevTools)

Reusable components

**Cons:**

SEO is harder (though solvable with tools like Next.js)

Initial loading can be slow

Requires JavaScript enabled in browser

* **Explain about React**

ReactJS is a **component-based** JavaScript library used to build dynamic and interactive user interfaces. It simplifies the creation of single-page applications (SPAs) with a focus on performance and maintainability.

It is developed and maintained by Facebook.

The latest version of React is React 19.

Uses a virtual DOM for faster updates.

Supports a declarative approach to designing UI components.

Ensures better application control with one-way data binding.

* **Define virtual DOM**

The **Virtual DOM** is a lightweight copy of the real DOM.

React compares the virtual DOM and real DOM and only updates the changed part (called **reconciliation**).

This makes the UI very fast

* **Explain Features of React**

Component-Based Architecture

JSX (JavaScript + HTML)

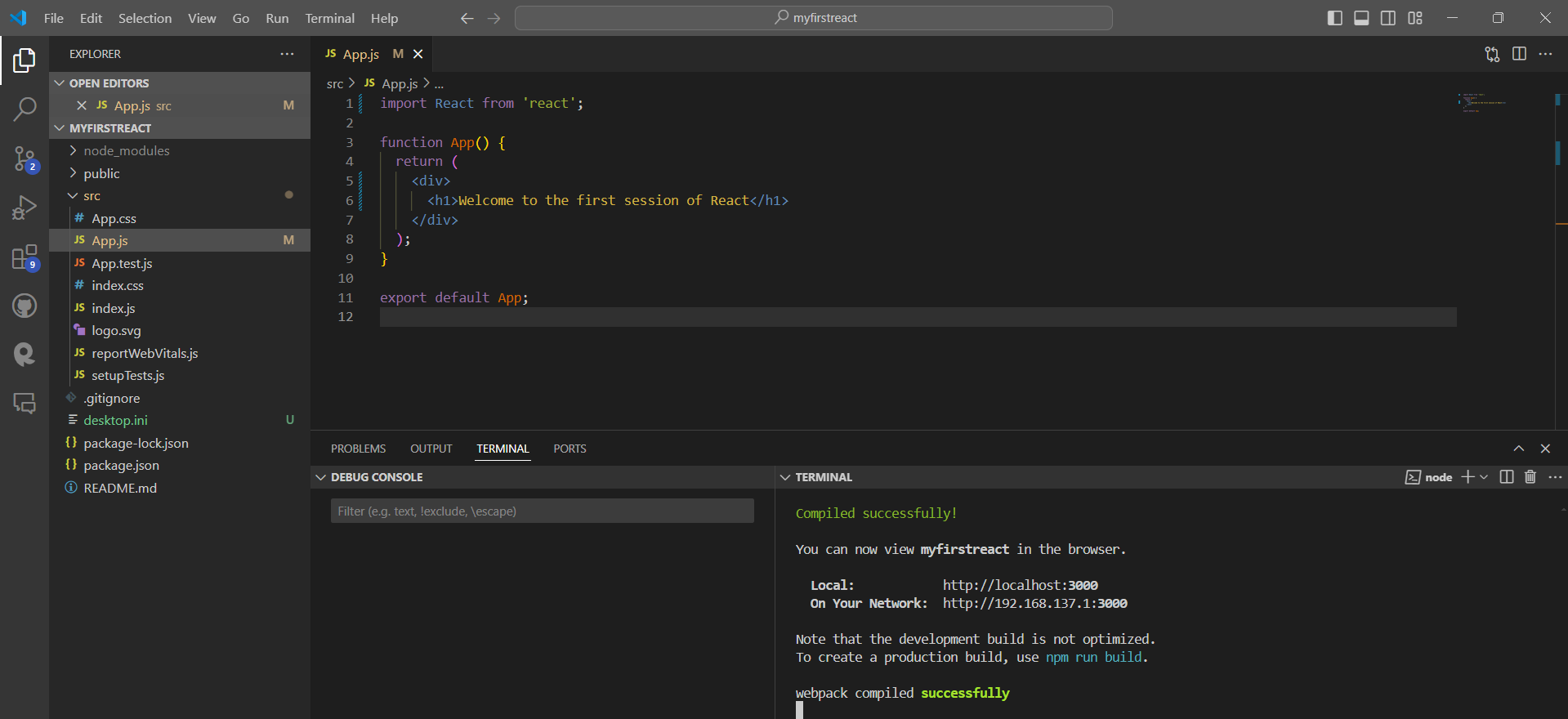
One-way Data Binding

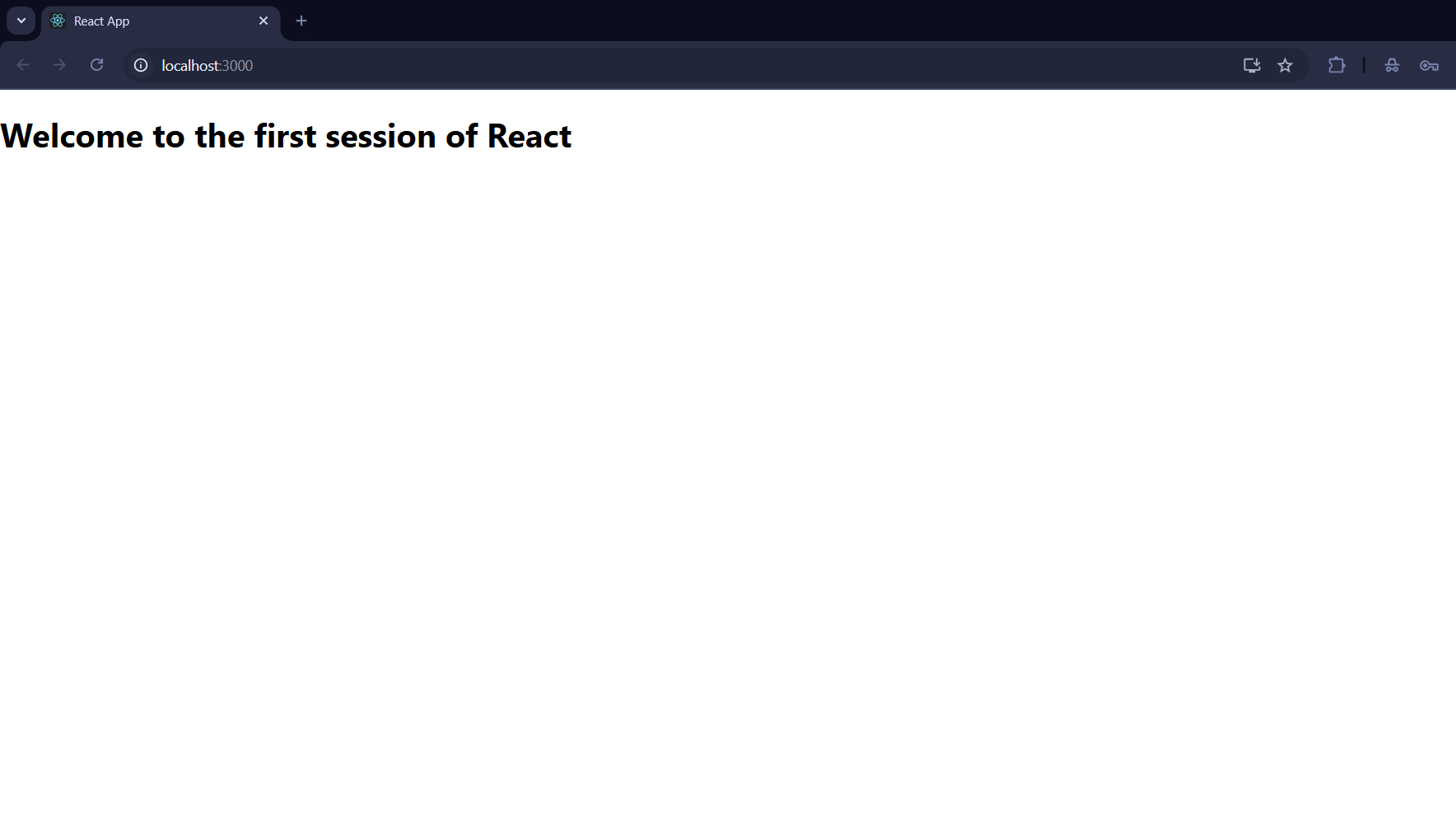
Virtual DOM

Fast Performance

Reusable Components

**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.**

****

****

* **Explain React components**

**Components** are the building blocks of a React application.

Think of them like **custom HTML tags** that you define.

* **Identify the differences between components and JavaScript functions**

| **JavaScript Function** | **React Component** |
| --- | --- |
| Returns a value | Returns JSX (UI) |
| No special structure | Follows React rules |
| Called directly | Rendered by React |

* **Identify the types of components**

**Class Components** (older style)

**Function Components** (modern and preferred)

* **Explain class component**

Uses a JavaScript class

Extends React.Component

Requires a render() method

import React, { Component } from 'react';

class Home extends Component {

render() {

return <h1>Welcome to the Home page</h1>;

}

}

export default Home;

* **Explain function component**

Just a simple function

Returns JSX

function About() {

return <h1>Welcome to the About page</h1>;

}

export default About;

* **Define component constructor**

Used in **class components**

Initializes state or binds methods

constructor(props) {

super(props);

this.state = {};

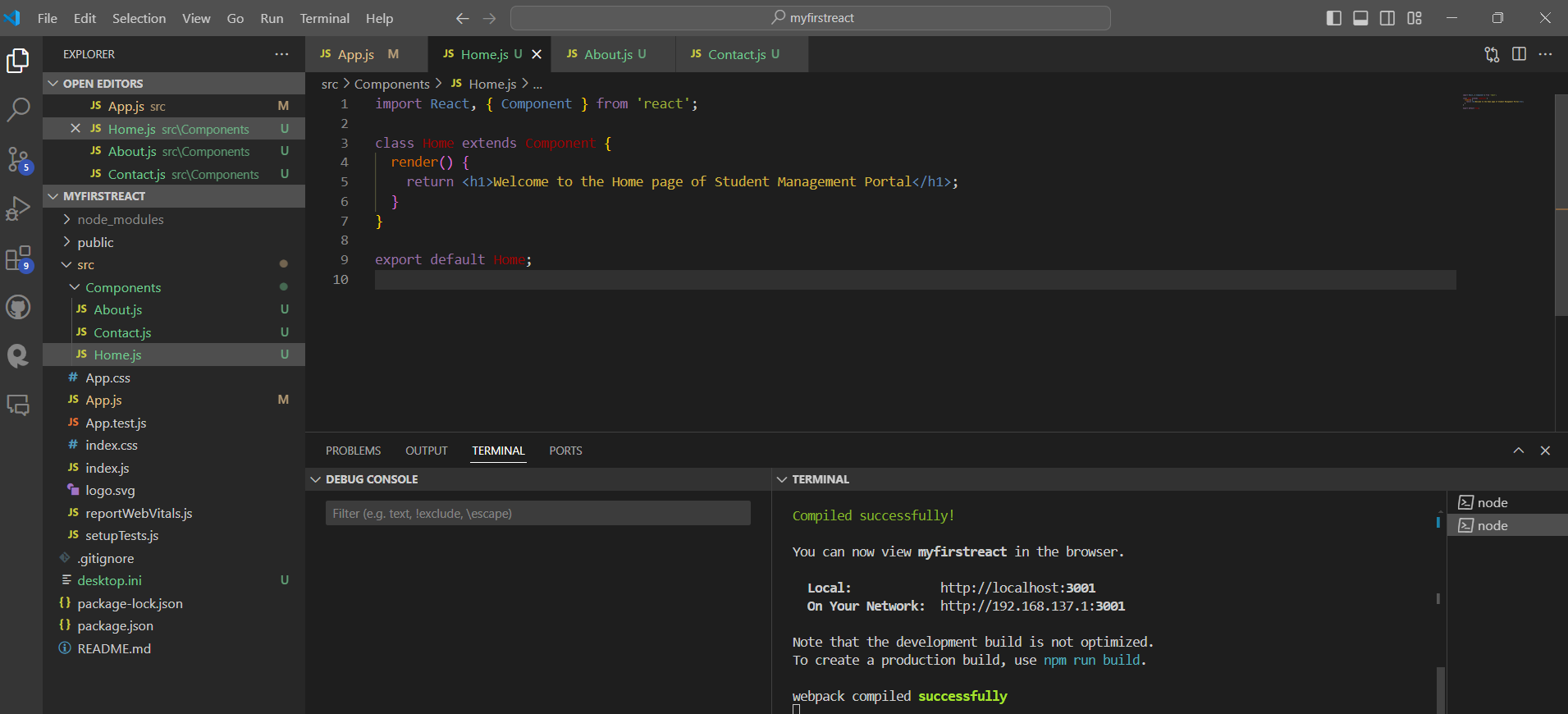
}

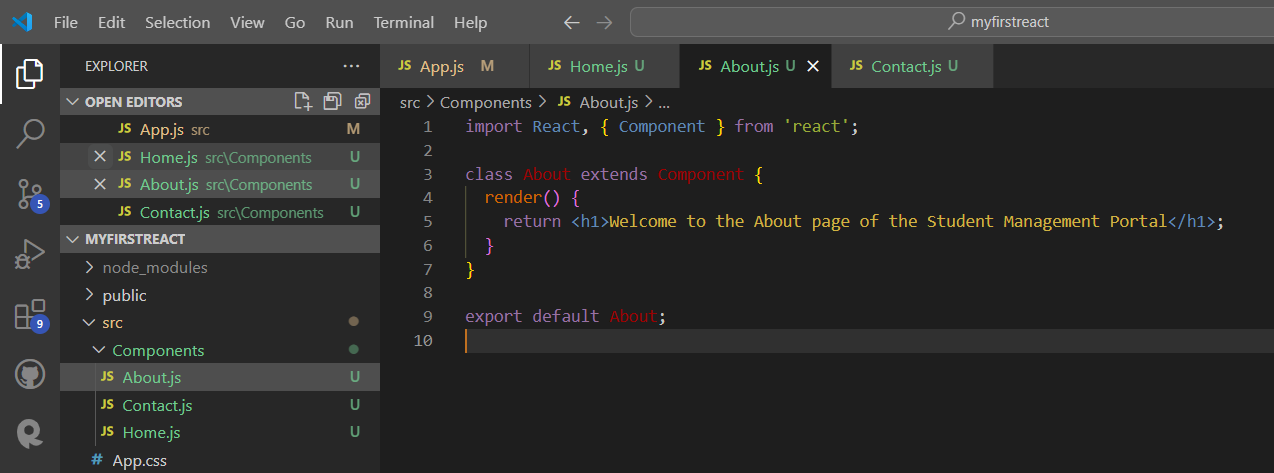
**Define render() function**

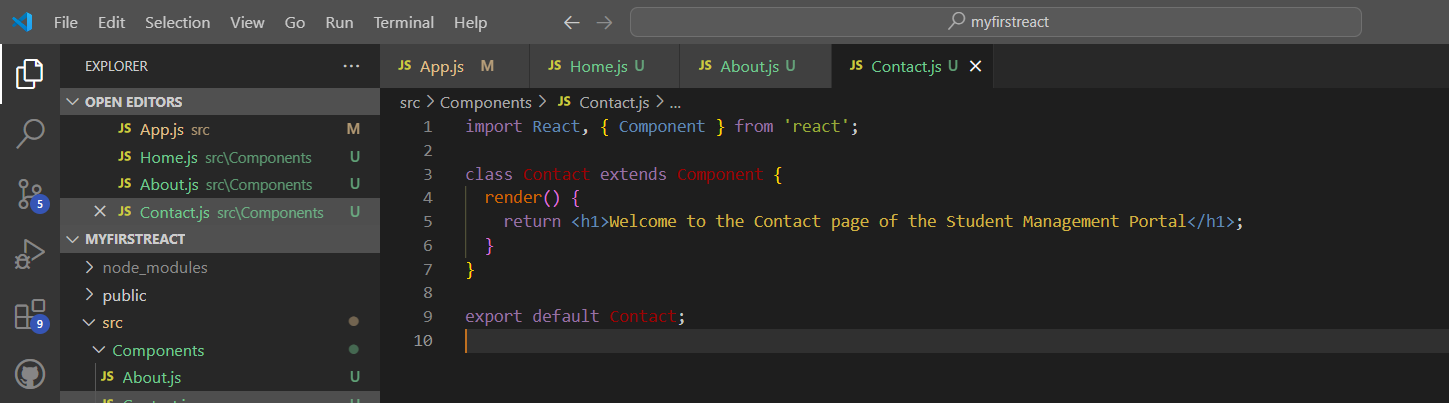
Must be defined in a class component

Outputs JSX

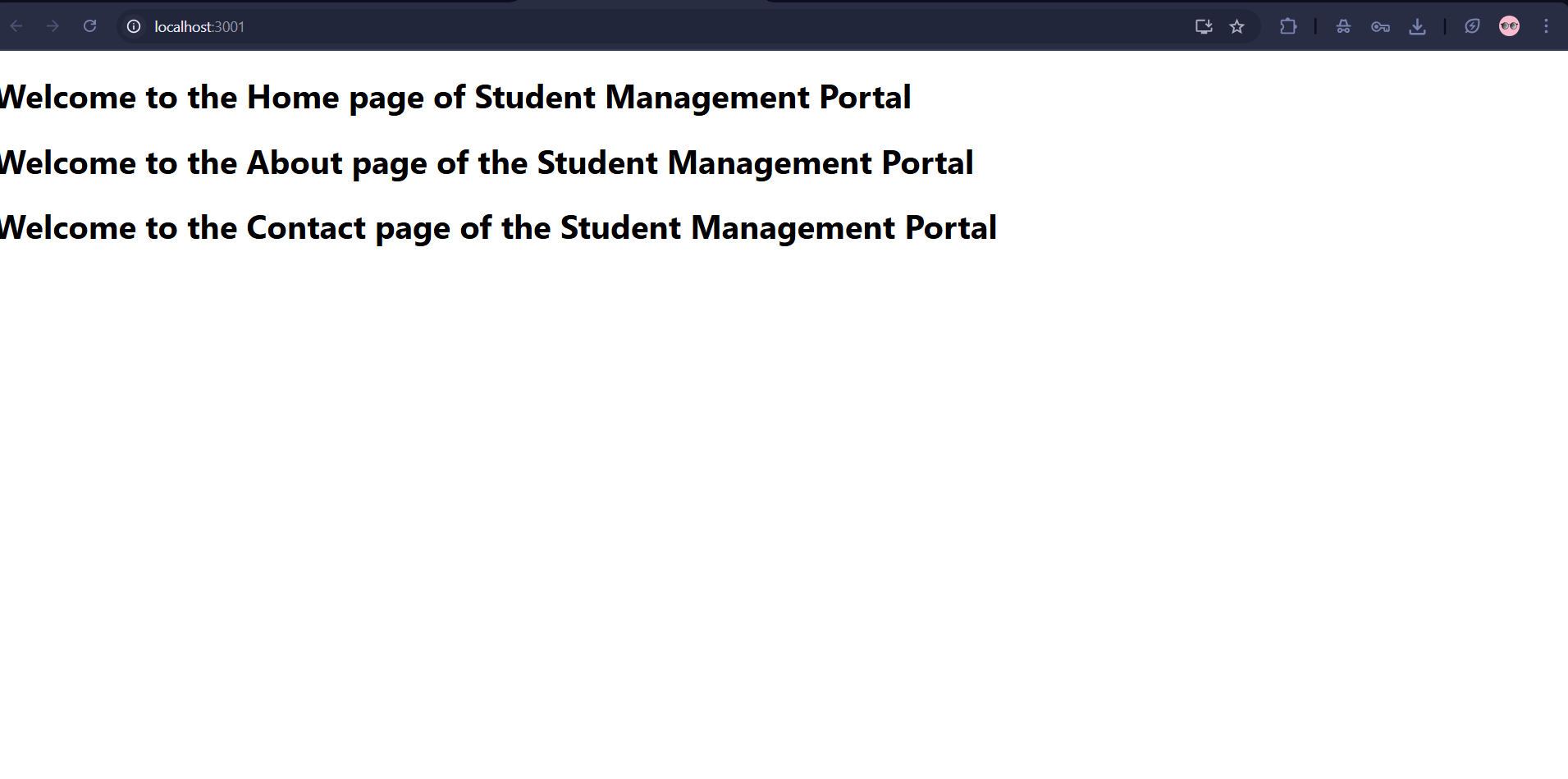
**Create a react app for Student Management Portal named StudentApp and create a component named Home which will display the Message “Welcome to the Home page of Student Management Portal”. Create another component named About and display the Message “Welcome to the About page of the Student Management 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.**

****

****

****

**Output:**

****

* **Explain React components**

A component is a reusable, self-contained piece of UI. It returns JSX (HTML-like syntax).

* **Identify the differences between components and JavaScript functions**

| **JavaScript Function** | **React Function Component** |
| --- | --- |
| General purpose logic | Returns JSX (UI elements) |
| Doesn’t use hooks | Can use hooks like useState, etc |
| Called explicitly | Rendered by React |

* **Identify the types of components**

**Class Components** (older, use class)

**Function Components** (modern, preferred)

* **Explain class component**

Uses a JavaScript class

Extends React.Component

Requires a render() method

import React, { Component } from 'react';

class Home extends Component {

render() {

return <h1>Welcome to the Home page</h1>;

}

}

export default Home;

* **Explain function component**

A function that returns JSX (HTML) and optionally accepts props.

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

* **Define component constructor**

Used in **class components**

Initializes state or binds methods

constructor(props) {

super(props);

this.state = {};

}

* **Define render() function**

Must be defined in a class component

Outputs JSX

**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.**

**App.js:**

import React from 'react';

import './App.css';

import CalculateScore from './Components/CalculateScore';

function App() {

  return (

    <div className="App">

      <CalculateScore />

    </div>

  );

}

export default App;

**CalculateScore.js:**

import React from 'react';

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

function CalculateScore() {

  const student = {

    name: "Sohana",

    school: "Springfield High School",

    totalMarks: 450,

    goal: 500

  };

  const average = (student.totalMarks / student.goal) \* 100;

  return (

    <div className="score-container">

      <h2>Student Score Calculator</h2>

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

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

      <p><strong>Total Marks:</strong> {student.totalMarks}</p>

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

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

    </div>

  );

}

export default CalculateScore;

**mystyle.css:**

.score-container {

    margin: 20px;

    padding: 20px;

    border: 2px solid #1e90ff;

    background-color: #f0f8ff;

    border-radius: 8px;

    font-family: Arial, sans-serif;

  }

  .score-container h2 {

    color: #1e90ff;

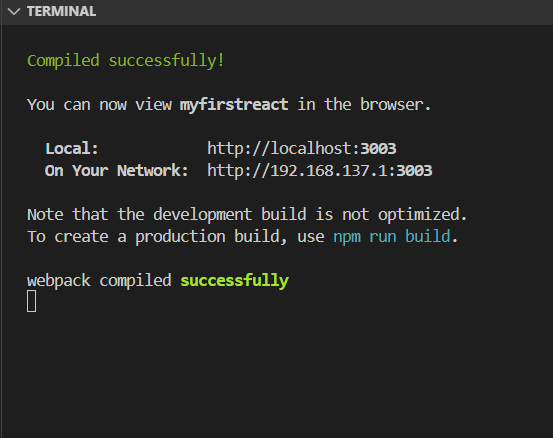
  }

  .score-container p {

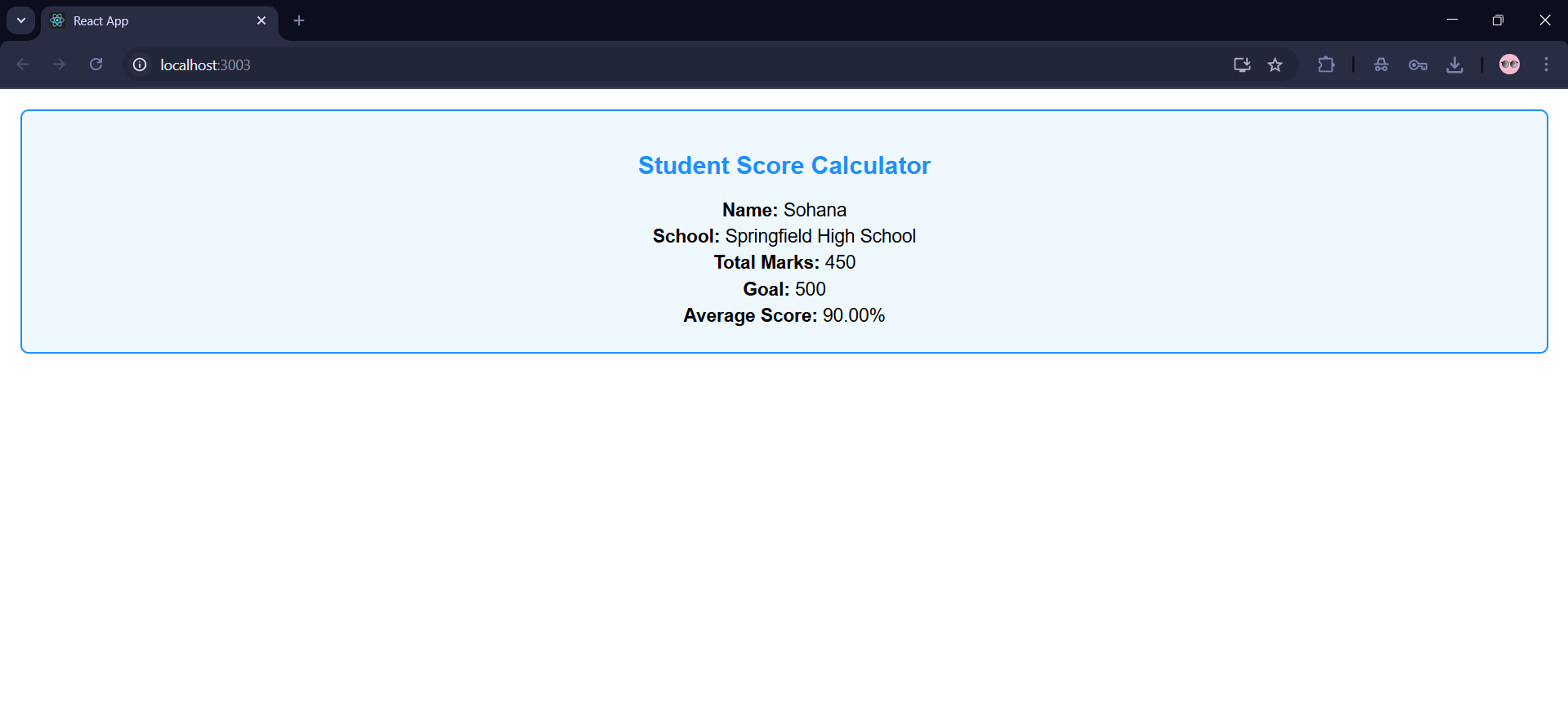
    font-size: 18px;

    margin: 5px 0;

  }

****

**Output:**

****

* **Explain the need and Benefits of component life cycle**

To **control what happens at each stage** of a component’s life.

To **fetch data**, **start animations**, **set timers**, or **clean up resources** at the right moment.

To **debug**, **optimize performance**, and **handle errors** systematically.

**Benefits:**

| **Benefit** | **Description** |
| --- | --- |
| **Control** | Gives full control over the UI component’s behavior |
| **Performance** | Fetch data only once (on mount), prevent unnecessary re-renders |
| **Cleanup** | Remove timers, subscriptions when component is removed |
| **Error Handling** | Catch runtime errors gracefully using lifecycle methods |

* **Identify various life cycle hook methods**

| **Phase** | **Lifecycle Method** | **Purpose** |
| --- | --- | --- |
| Mounting | constructor() | Initialize state and props |
|  | componentDidMount() | Fetch data or run side-effects after first render |
| Updating | shouldComponentUpdate() | Optimize performance by controlling re-renders |
|  | componentDidUpdate() | Run code after re-render (e.g., data sync, logging) |
| Unmounting | componentWillUnmount() | Cleanup tasks like timers, listeners |
| Error | componentDidCatch(error, info) | Handle errors in child components gracefully |

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

When a component is created and rendered for the first time (mounting), these steps happen in order:

1. constructor()  
   → Sets initial state and binds methods.
2. render()  
   → Returns the JSX to show on the screen.
3. componentDidMount()  
   → Runs after the component is added to the DOM. Ideal for API calls.

When the component updates (due to state or props change), this sequence runs:

1. render()
2. componentDidUpdate()

When the component is removed from the screen (unmounting):

1. componentWillUnmount()

When an error happens during rendering:

1. componentDidCatch(error, info)
2. Create a new react application using *create-react-app* tool with the name as “blogapp”
3. Open the application using VS Code
4. 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.



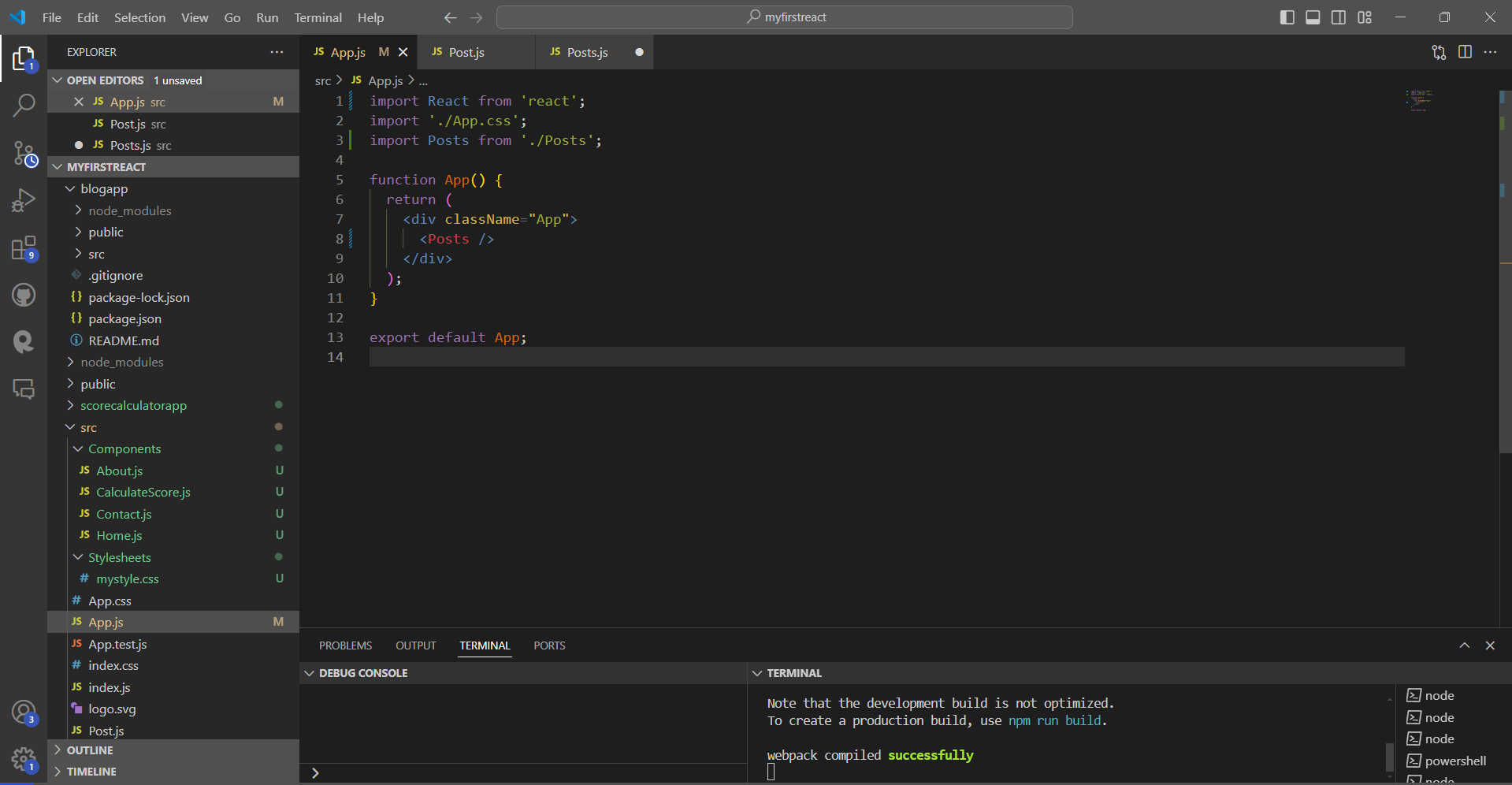
Figure 6: render() method

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.



**Post.js:**

import React from 'react';

class Post extends React.Component {

  render() {

    const { title, body } = this.props;

    return (

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

        <h2>{title}</h2>

        <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: [],

      hasError: false

    };

  }

  loadPosts = () => {

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

      .then((res) => res.json())

      .then((data) => {

        console.log("Fetched posts:", data);

        this.setState({ posts: data });

      })

      .catch((err) => {

        console.error('Error:', err);

        this.setState({ hasError: true });

      });

  };

  componentDidMount() {

    console.log("Component mounted!");

  }

  componentDidCatch(error, info) {

    alert('Something went wrong!');

    console.log(error, info);

  }

  render() {

    return (

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

        <h1>Blog Posts</h1>

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

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

        ))}

      </div>

    );

  }

}

export default Posts;

**Output:**

****

**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***

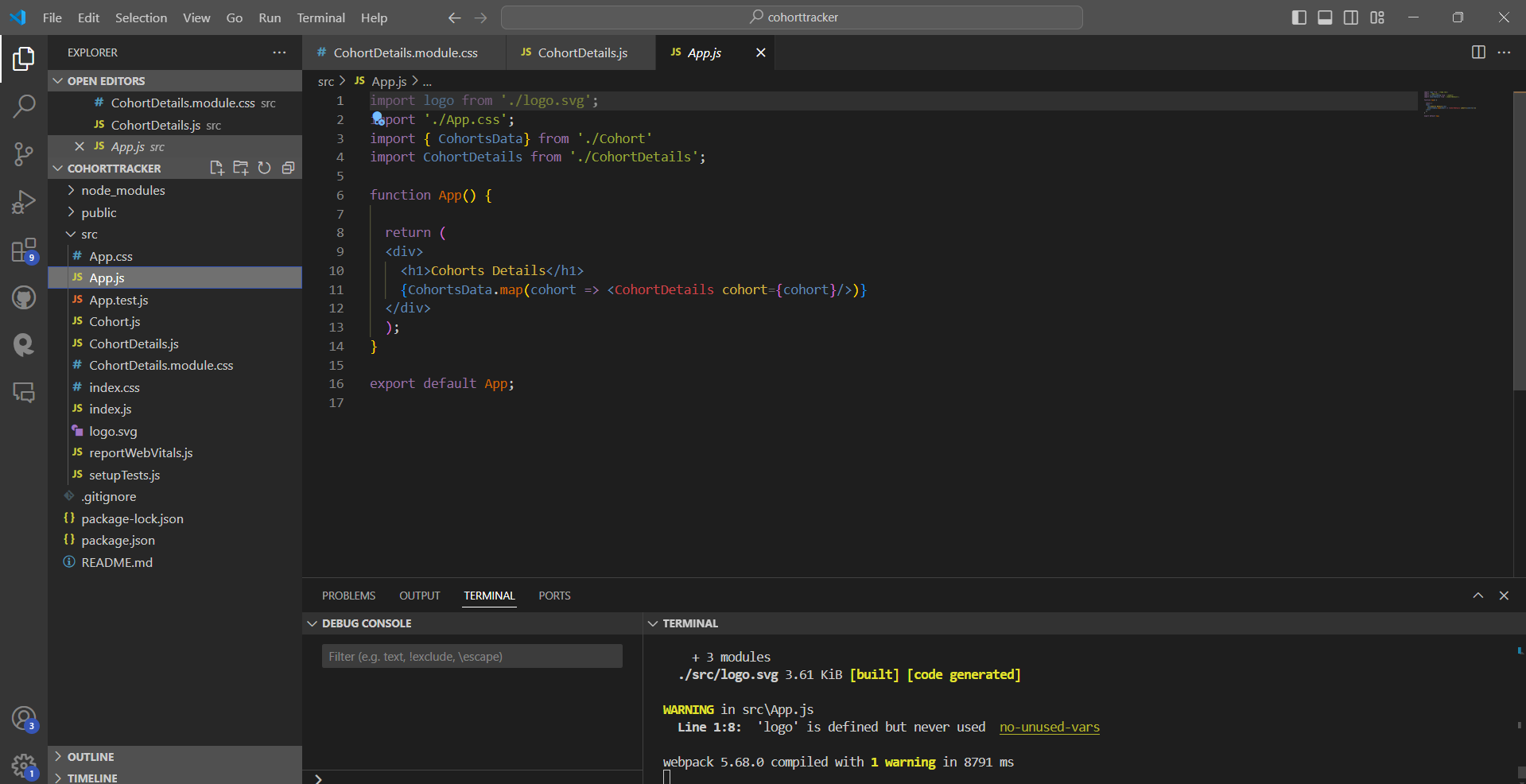
***1 px border in black color***

***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;

  }

**CohortDetails.js:**

import React from 'react';

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

function CohortDetails(props) {

  const { name, status, duration, trainer } = props;

  return (

    <div className={styles.box}>

      <h2>{name}</h2>

      <h3 style={{ color: status === 'ongoing' ? 'green' : 'blue' }}>

        {status}

      </h3>

      <dl>

        <dt>Duration</dt>

        <dd>{duration}</dd>

        <dt>Trainer</dt>

        <dd>{trainer}</dd>

      </dl>

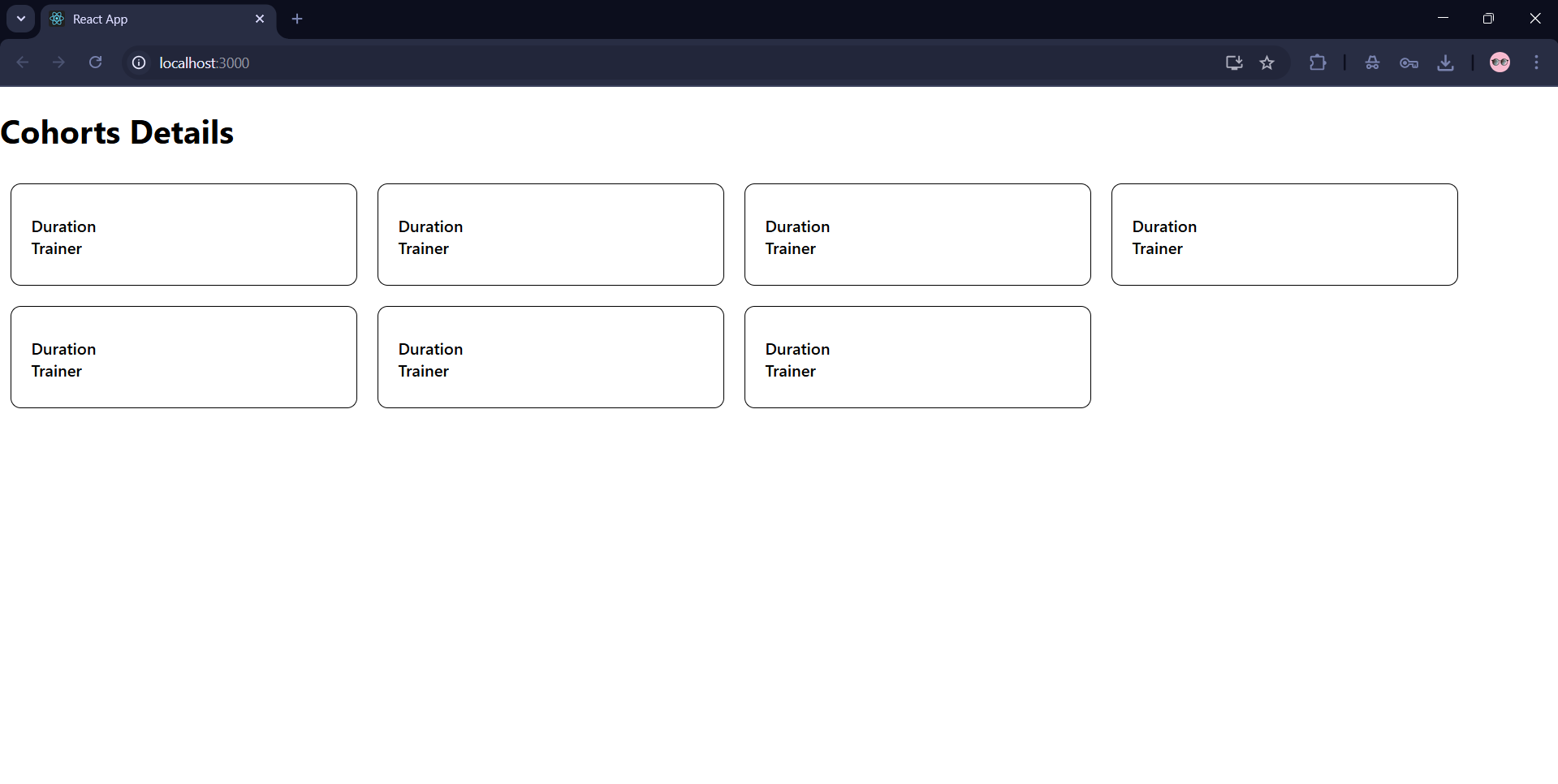
    </div>

  );

}

export default CohortDetails;

**Output:**

****