**Exercise 1:**

**1. Define SPA and its Benefits**

SPA (Single Page Application) is a web application that loads a single HTML page and dynamically updates content as the user interacts with the app, without reloading the entire page.

**Benefits of SPA:**

* Faster navigation since only necessary content updates.
* Smooth user experience similar to desktop or mobile apps.
* Reduces server load as most resources are loaded once.
* Efficient data handling using APIs (like REST or GraphQL).

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

React is a JavaScript library developed by Facebook for building fast, interactive user interfaces, especially for SPAs.

**How React Works:**

* React uses a component-based architecture.
* Components are reusable and encapsulated units of UI.
* It maintains a Virtual DOM to update only the changed parts of the real DOM efficiently.
* Uses JSX (JavaScript XML) for writing HTML-like code in JavaScript.

**3. Differences Between SPA and MPA**

| **Feature** | **SPA (Single Page App)** | **MPA (Multi Page App)** |
| --- | --- | --- |
| Page Loading | Loads a single page, updates dynamically | Loads a new page from server on every request |
| Speed | Faster interactions after initial load | Slower due to full page reloads |
| User Experience | Seamless and smooth | May feel slower and less dynamic |
| Server Communication | API calls for data | Full page reload on every request |
| Example | Gmail, Facebook, React Apps | Amazon, Wikipedia, traditional websites |

**4. Pros & Cons of Single Page Applications**

**Pros:**

* Smooth, app-like experience
* Faster response after first load
* Easier to debug with developer tools
* Good for mobile performance

**Cons:**

* Initial loading time can be higher
* SEO challenges (can be solved with SSR)
* JavaScript-dependent – may not work well if JS is disabled
* Security considerations (like XSS)

**5. Explain About React**

React is a powerful JavaScript library that allows developers to:

* Build reusable UI components.
* Use a virtual DOM for efficient rendering.
* Manage state and props to control how components behave.

**6. Define Virtual DOM**

The Virtual DOM (VDOM) is a lightweight copy of the real DOM maintained by React.

**How It Works:**

1. When a component state changes, React creates a new Virtual DOM tree.
2. It compares the new VDOM with the previous one (diffing).
3. Only the differences are updated in the real DOM (patching), improving performance.

**7. Features of React**

* **Component-Based**: UI is broken into independent, reusable components.
* **JSX**: JavaScript + XML syntax for creating UI elements.
* **Virtual DOM**: Efficient rendering by minimizing direct DOM updates.
* **Unidirectional Data Flow**: Data flows from parent to child using props.
* **Hooks**: Functions like useState, useEffect to manage state and side effects.
* **Declarative UI**: Describe how UI should look for a given state.
* **Strong Ecosystem**: Tools like Redux, React Router, etc.

**CODE:**

**App.js:**

import logo from './logo.svg';

import './App.css';

function App() {

  return (

    <div className="App">

      <header className="App-header">

        <h1>This is my React project</h1>

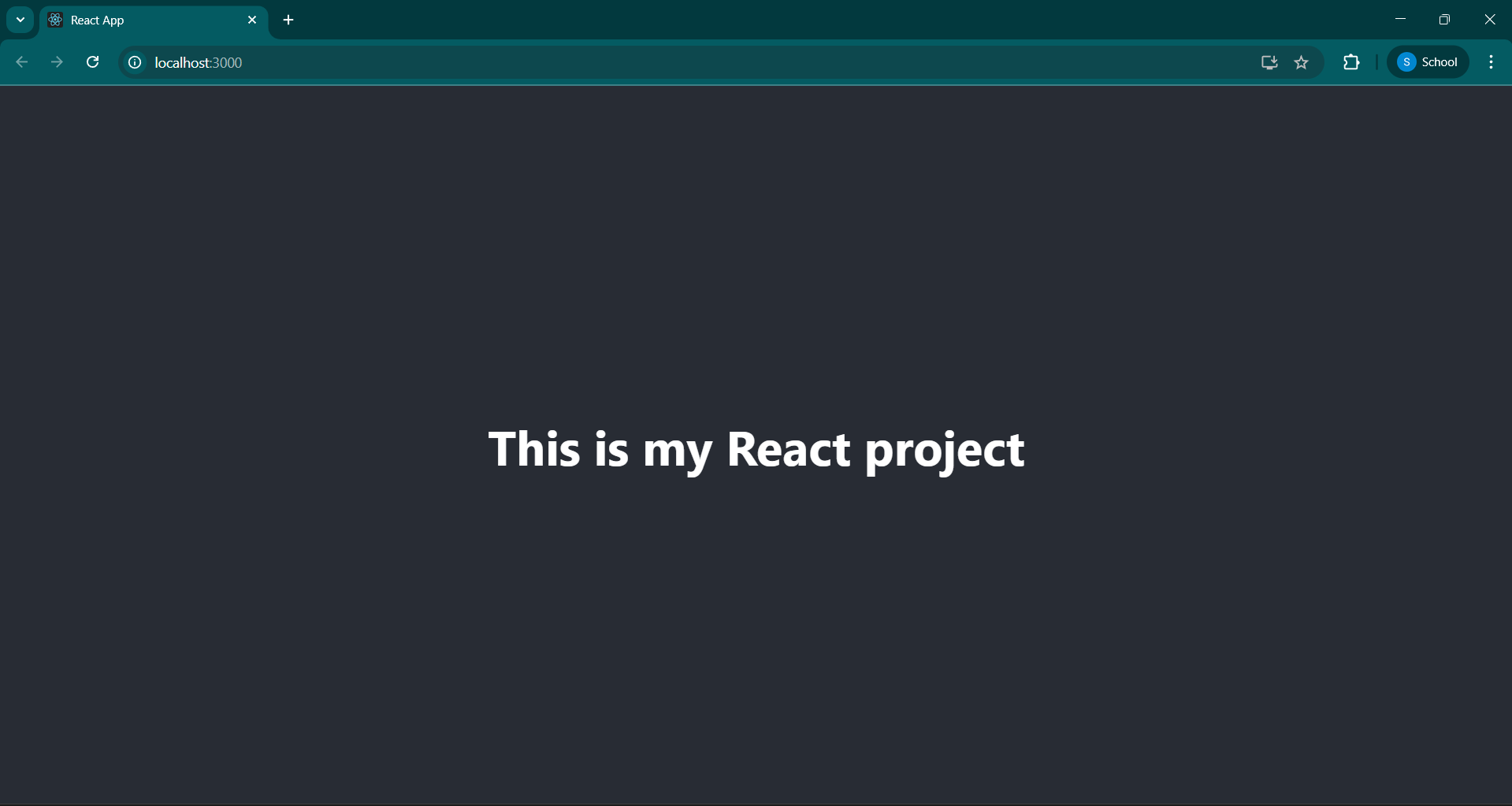
      </header>

    </div>

  );

}

export default App;

**OUTPUT:** ****

**EXERCISE 2:**

**1. Explain React Components**

**React components** are the building blocks of a React application. They are self-contained, reusable pieces of code that return React elements (usually JSX) to be rendered on the screen.

Think of components as **custom HTML elements** that describe parts of the user interface.

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

| **Aspect** | **React Component** | **JavaScript Function** |
| --- | --- | --- |
| Purpose | Used to define UI | Used to perform logic or return values |
| Output | Returns JSX or React elements | Returns primitive types or objects |
| Lifecycle Methods | Yes (e.g., componentDidMount in class) | No lifecycle methods |
| Used in JSX | Yes (e.g., <MyComponent />) | No direct use in JSX |
| React-Specific | Yes | No |

**3. Types of React Components**

There are **two main types** of React components:

1. **Class Components**
2. **Function Components**

**4. Explain Class Component**

A **class component** is a JavaScript ES6 class that extends React.Component. It can hold its own **state** and use **lifecycle methods**.

**Syntax:**

import React, { Component } from 'react';

class Welcome extends Component {

render() {

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

}

}

**5. Explain Function Component**

A **function component** is a simpler way to write components. They are **JavaScript functions** that return JSX. With **Hooks**, they can also manage state and lifecycle.

**Syntax:**

function Welcome(props) {

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

}

Or using arrow function:

const Welcome = (props) => <h1>Hello, {props.name}</h1>;

**6. Define Component Constructor**

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

* Initializing **state**
* Binding event handlers

**Syntax:**

constructor(props) {

super(props);

this.state = {

count: 0

};

}

super(props) must be called to access this.props inside the constructor.

**7. Define render() Function**

The render() method is a **mandatory** method in class components. It returns the JSX that should be displayed in the UI.

**Example:**

render() {

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

}

It is called automatically by React whenever the component’s state or props change.

**App.js:**

import React from 'react';

import './App.css';

import Home from './Home';

import About from './About';

import Contact from './Contact';

function App() {

  return (

    <div className="App">

      <h1>Student Management Portal</h1>

      <Home />

      <About />

      <Contact />

    </div>

  );

}

export default App;

**Contact.js:**

import React from 'react';

function Contact() {

  return (

    <div>

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

    </div>

  );

}

export default Contact;

**Home.js:**

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:**

import React from 'react';

function About() {

  return (

    <div>

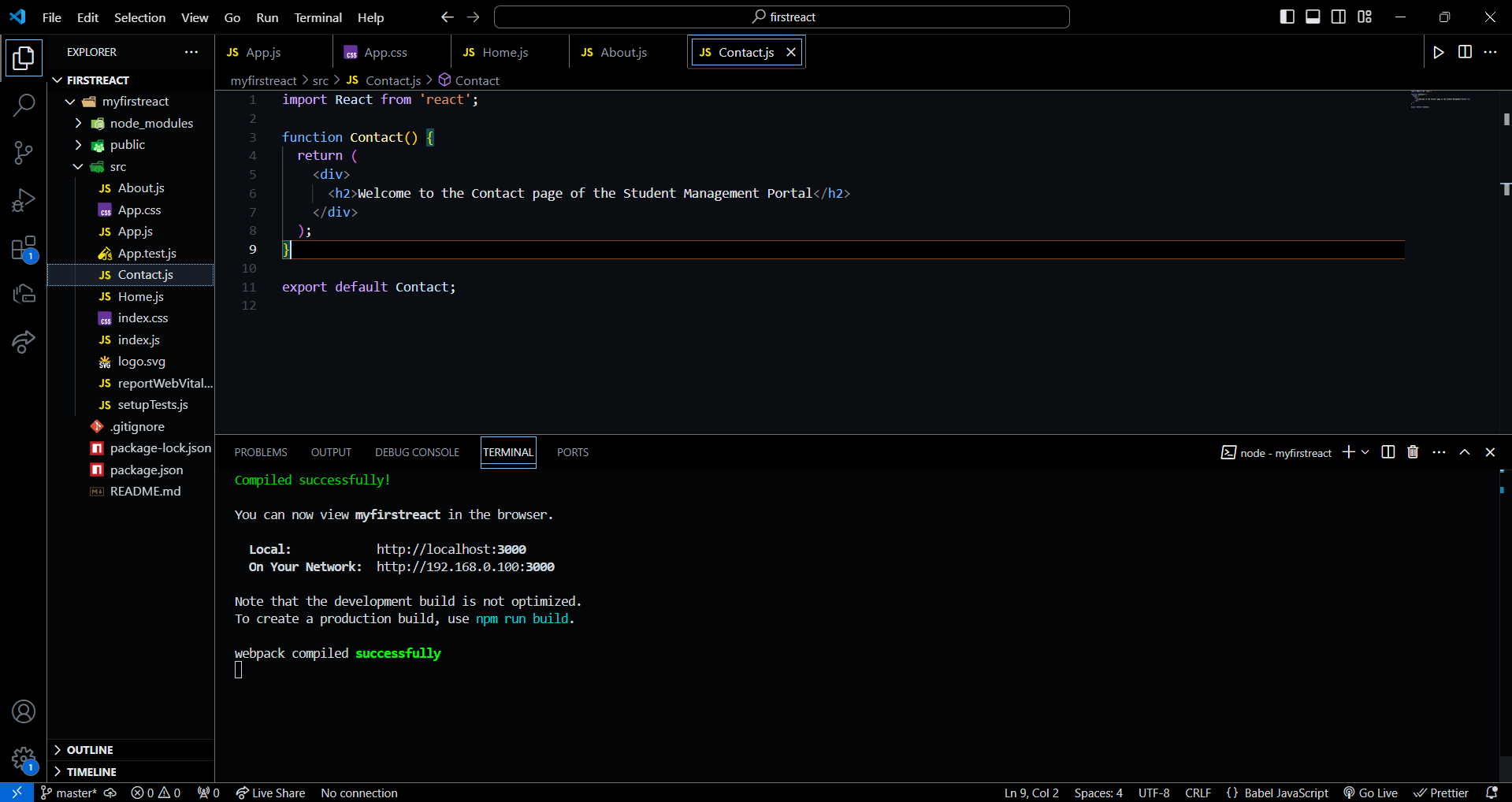
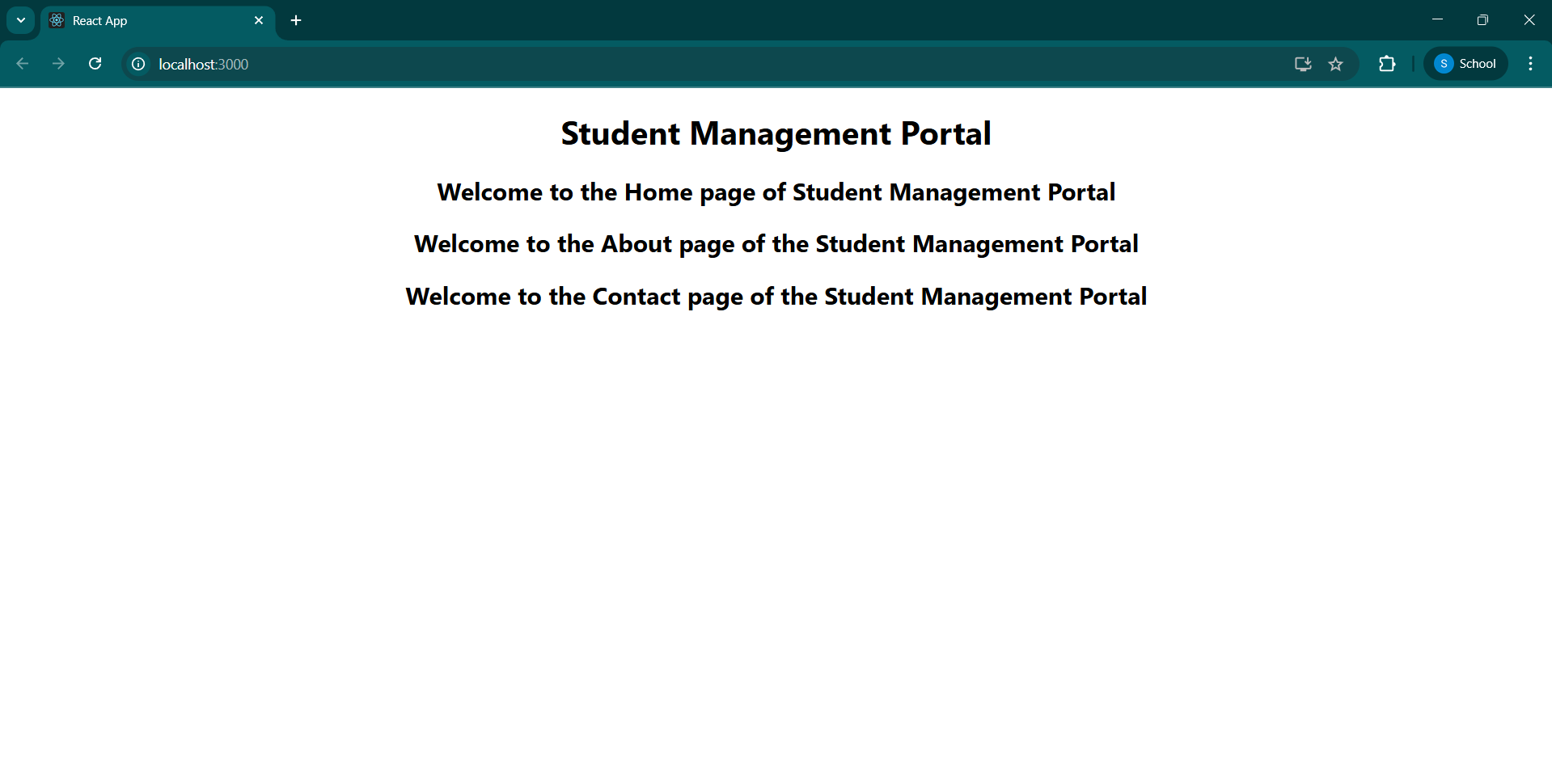
      <h2>Welcome to the About page of the Student Management Portal</h2>

    </div>

  );

}

export default About;

**OUTPUT:** ****

**EXERCISE 3:**

**1. Explain React Components**

React components are **building blocks of a React application**. Each component is a **self-contained piece of UI**, like a button, form, or a whole page.

A React component returns JSX (HTML-like syntax) that tells React what to display on the screen.

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

| **Aspect** | **React Components** | **JavaScript Functions** |
| --- | --- | --- |
| Purpose | Create UI elements | Perform logic or calculations |
| Return type | Returns JSX (UI) | Returns values (number, string, object, etc.) |
| Reusability in UI | Used as <ComponentName /> | Called as functionName() |
| State & Lifecycle | Can use state and lifecycle (in class or hooks) | No built-in state or lifecycle handling |
| Integration with React | Integrated directly into the UI | Used inside logic, not for UI display |

**3. Types of React Components**

There are **two main types** of React components:

1. **Class Components**
2. **Function Components**

**4. Explain Class Component**

A **class component** is created using ES6 class syntax. It can hold **state**, use **lifecycle methods**, and is used when you need more control.

**Example:**

import React, { Component } from 'react';

class MyComponent extends Component {

render() {

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

}

}

export default MyComponent;

Use when state management and lifecycle methods (e.g. componentDidMount) are needed.

**5. Explain Function Component**

A **function component** is a simpler and modern way to create components using JavaScript functions. With **React Hooks**, function components can also manage state and lifecycle.

**Example:**

import React from 'react';

function MyComponent() {

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

}

export default MyComponent;

**6. Define Component Constructor**

The **constructor** in a class component is a special method used for:

* Initializing the component's **state**
* Binding event handlers

**Example:**

constructor(props) {

super(props);

this.state = {

name: 'Sudherson'

};

}

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

**7. Define render() Function**

The render() function is **required in class components** and is used to return the **JSX code** that defines the UI.

**Example:**

render() {

return <div>Hello World</div>;

}

It is automatically called by React whenever the component needs to display or update the UI.

**CODE:**

**App.js:**

import React from 'react';

import './App.css';

import CalculateScore from './Components/CalculateScore';

function App() {

return (

<div className="App">

<h1>Student Management Portal - Score Calculator</h1>

<CalculateScore name="Sudherson" school="SKCET" total={480} goal={6} />

</div>

);

}

export default App;

**CalculateScore.js:**

import React from 'react';

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

function CalculateScore(props) {

const { name, school, total, goal } = props;

const average = total / goal;

return (

<div className="score-card">

<h2>Student Score Details</h2>

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

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

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

<p><strong>Number of Subjects (Goal):</strong> {goal}</p>

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

</div>

);

}

export default CalculateScore;

**mystyle.css:**

.score-card {

background-color: #f0f8ff;

border: 2px solid #007bff;

padding: 20px;

margin: 30px auto;

width: 400px;

text-align: left;

font-family: Arial, sans-serif;

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

}

.score-card h2 {

color: #007bff;

text-align: center;

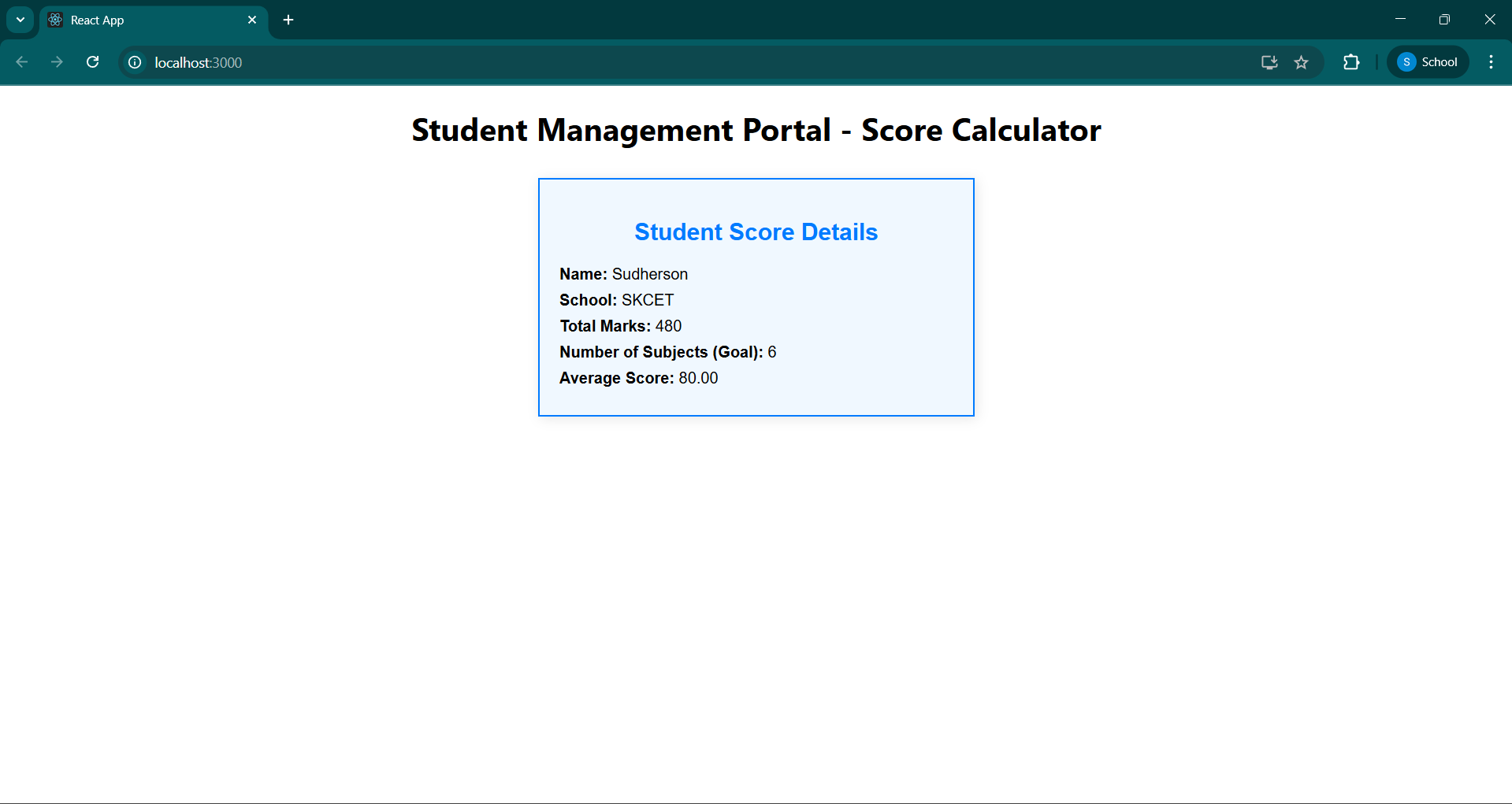
}

.score-card p {

font-size: 16px;

margin: 8px 0;

}

**OUTPUT:** 

**EXERCISE 4:**

**1. Explain the Need and Benefits of Component Lifecycle (3 Marks)**

Component lifecycle methods allow React to manage the creation, update, and removal of components.  
They help perform specific actions like data fetching, event binding, or cleanup at each stage of the component’s life.  
**Benefits** include better performance, clean code, and precise control over how and when things happen in a component.

**2. Identify Various Lifecycle Hook Methods (3 Marks)**

The main **lifecycle methods** in class components are:

* **Mounting:** constructor(), render(), componentDidMount()
* **Updating:** shouldComponentUpdate(), componentDidUpdate()
* **Unmounting:** componentWillUnmount()
* **Error Handling:** componentDidCatch()

These help manage component behavior during different phases.

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

The sequence of rendering a React class component is:

1. constructor() – Initializes state.
2. render() – Returns JSX to display UI.
3. componentDidMount() – Runs after the component is added to the DOM.

This flow ensures proper setup and rendering of the component.

**Post.js:**

import React from 'react';

class Post extends React.Component {

render() {

const { title, body } = this.props;

return (

<div className="post">

<h2>{title}</h2>

<p>{body}</p>

<hr />

</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(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("An error occurred while loading the posts.");

console.error("Error boundary caught:", error, info);

this.setState({ hasError: true });

}

render() {

const { posts, hasError } = this.state;

if (hasError) {

return <h2>Something went wrong. Please try again later.</h2>;

}

return (

<div>

<h1>Blog Posts</h1>

{posts.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 (

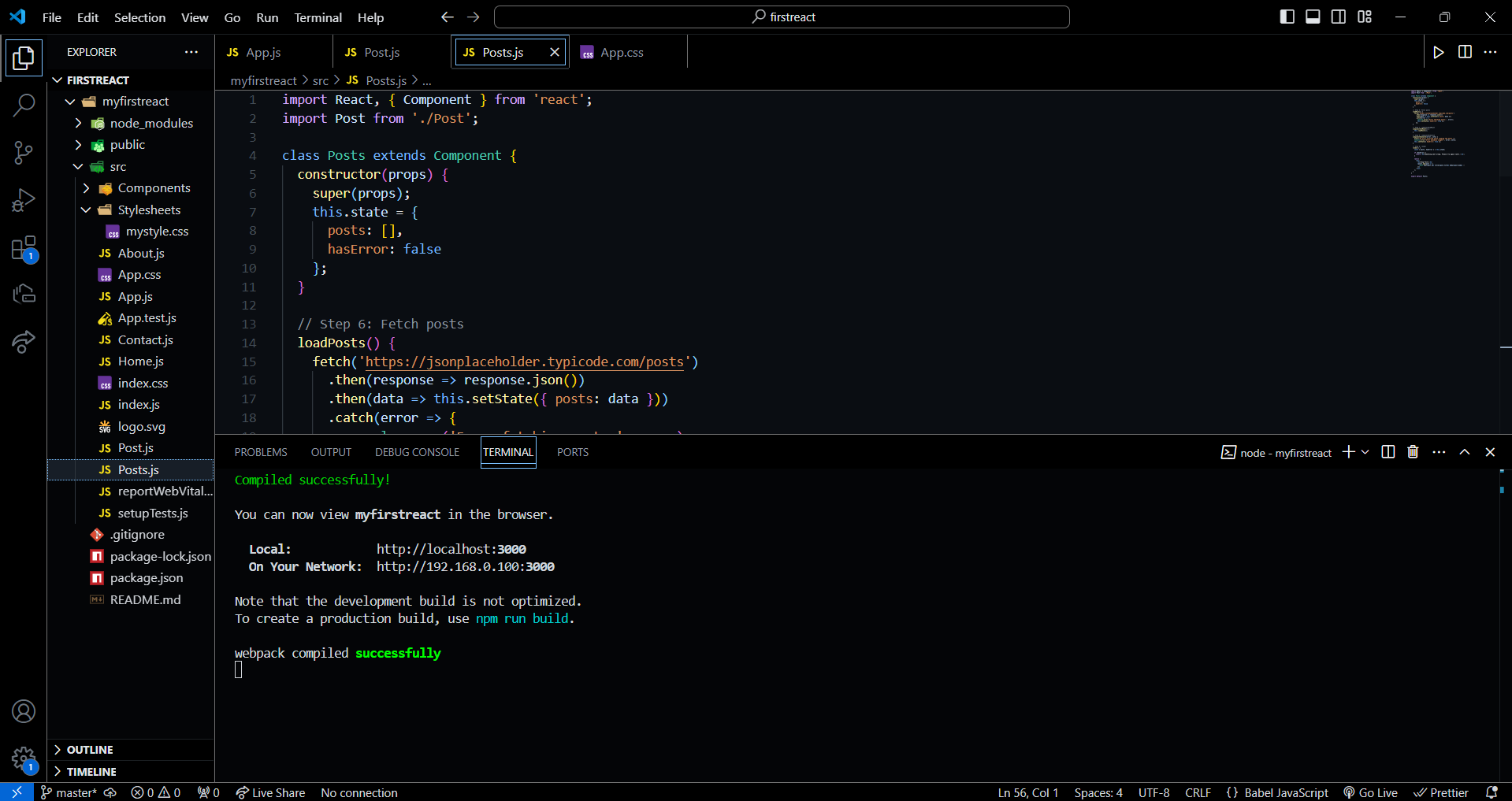
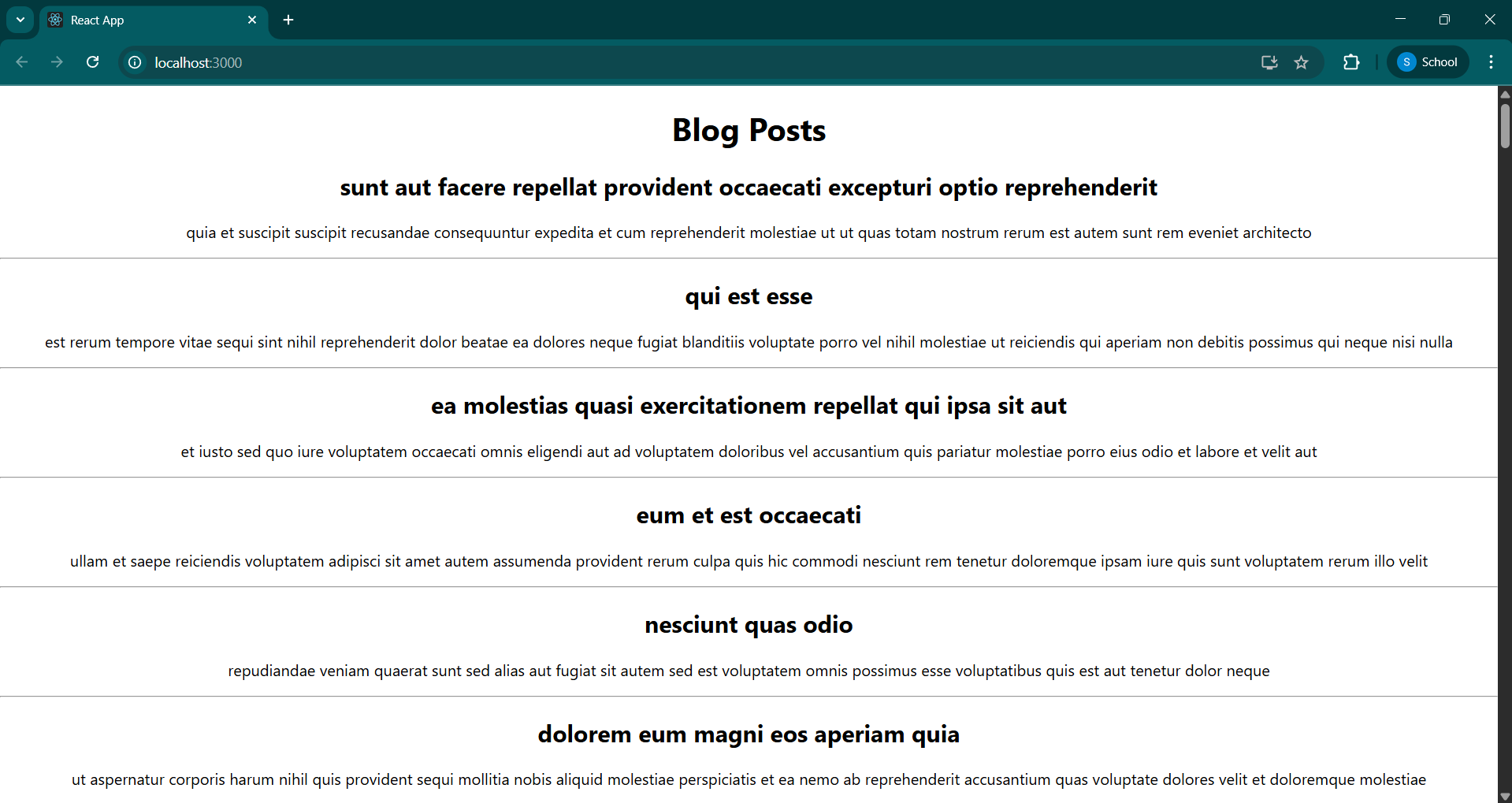
<div className="App">

<Posts />

</div>

);

}

export default App;  
  
  
**OUTPUT:** ****

**EXERCISE 5:**

**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';

class CohortDetails extends React.Component {

render() {

const { cohort } = this.props;

return (

<div className={styles.box}>

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

{cohort.status}

</h3>

<dl>

<dt>Name:</dt>

<dd>{cohort.name}</dd>

<dt>Start Date:</dt>

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

<dt>End Date:</dt>

<dd>{cohort.endDate}</dd>

</dl>

</div>

);

}

}

export default CohortDetails;

**App.js:**

import React from 'react';

import CohortDetails from './CohortDetails';

function App() {

const cohort = {

name: 'React Bootcamp',

startDate: '2024-01-01',

endDate: '2024-04-01',

status: 'ongoing'

};

return (

<div className="App">

<h1>Cohort Dashboard</h1>

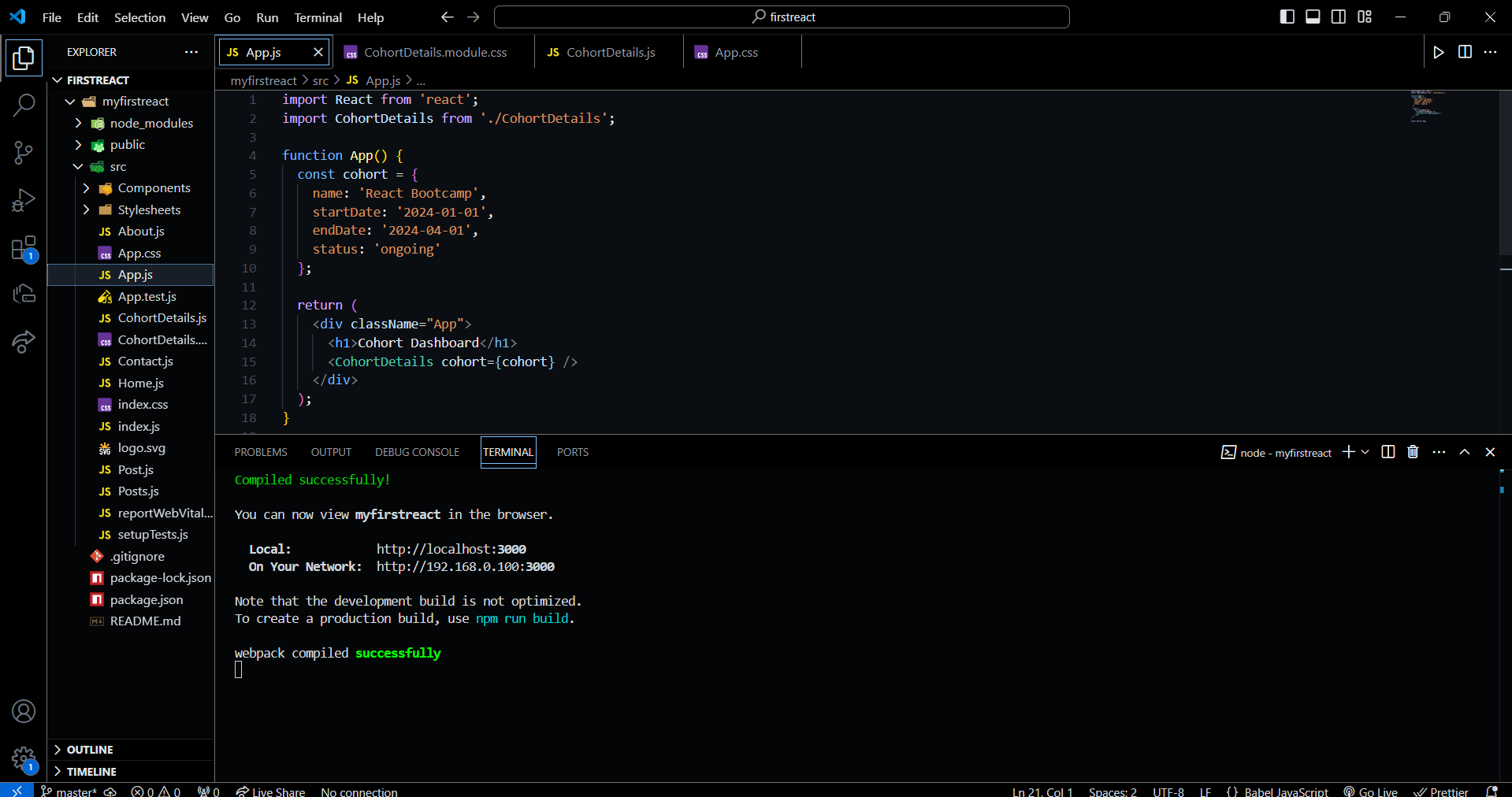
<CohortDetails cohort={cohort} />

</div>

);

}

export default App;

**OUTPUT:** 

****