**ReactJs-HOL-1**

**1.Define SPA and its benefits**

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

**Benefits of SPA:**

* Faster user experience because only necessary data is fetched and updated.
* Reduces server load as only one page is served.
* Smooth transitions between pages without full reload.
* Easier to develop as a modular, component-based architecture.

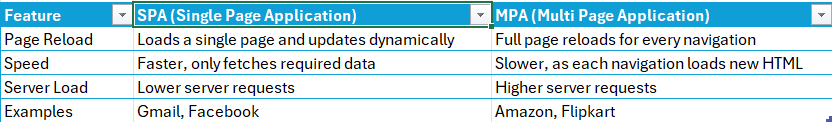
**2. Define React and identify its working**

**React** is a **JavaScript library** developed by Facebook for building user interfaces, especially for single-page applications. It allows developers to create reusable UI components and manage the view layer efficiently.

**How React works:**

* React uses a **Virtual DOM** to track changes in the UI.
* When data changes, React calculates the difference between the current and previous Virtual DOM (diffing) and updates only the changed parts in the real DOM.
* This makes React highly performant.

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

****

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

**Pros:**  
Faster performance due to reduced server interaction.  
Better user experience with seamless navigation.  
Simplified backend APIs as the frontend handles routing.

**Cons:**  
SEO optimization is more challenging compared to MPA.  
Initial loading time can be higher.  
Browser history and navigation might need additional handling.

**5. Explain about React**

React is an open-source JavaScript library maintained by Facebook and a community of developers. It is widely used for building dynamic, interactive, and fast web applications with a component-based architecture.

Key reasons for using React:

* **Component-Based Architecture:** Allows reusability and modularity.
* **Virtual DOM:** Optimizes updates and renders efficiently.
* **Declarative Syntax:** Makes code easier to read and debug.

**6. Define Virtual DOM**

The **Virtual DOM (Document Object Model)** is a lightweight copy of the real DOM used by React. When a component’s state changes:

* React updates the Virtual DOM first.
* It compares the new Virtual DOM with the previous one (diffing).
* Only the changed elements are updated in the real DOM (reconciliation).

This improves application performance and ensures faster updates.

**7. Explain Features of React**

* **JSX (JavaScript XML):** Lets you write HTML-like syntax inside JavaScript.
* **Components:** Encapsulated, reusable UI building blocks.
* **Virtual DOM:** Improves rendering efficiency.
* **Unidirectional Data Flow:** Data flows in a single direction, simplifying debugging.
* **React Hooks:** Allows state and lifecycle management in functional components.
* **High Performance:** Only necessary DOM parts are updated.

**Welcome to the first session of React**

Run the following command in the command prompt:

* npm install -g create-react-app

Create a new project folder named **myfirstreact**:

* npx create-react-app myfirstreact
* cd myfirstreact

**App.js:**

import React from 'react';

function App() {

return (

<div>

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

</div>

);

}

export default App;

**Run the React Application:**

* npm start

**OUTPUT:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**ReactJs-HOL-2**

**1. Explain React Components**

React components are the building blocks of a React application. A component is a reusable, independent, and encapsulated piece of UI that can accept input (props) and return React elements that define what should appear on the screen.

* Components help in dividing the UI into smaller, manageable parts.
* They allow code reuse and better organization.

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

| **Aspect** | **React Components** | **JavaScript Functions** |
| --- | --- | --- |
| **Purpose** | Used to define UI and behavior in React apps. | Used to perform a specific task or calculation. |
| **Return Type** | Returns JSX (React elements) | Returns a value (number, string, object, etc.). |
| **Usage in UI** | Describes how the UI should look. | Cannot be directly used to define UI structure. |
| **Reactivity** | Supports reactivity with state and props. | Does not have built-in reactivity. |
| **Lifecycle Methods** | Supports lifecycle methods (in class components). | No lifecycle methods. |

**3. Types of Components**

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

1. **Class Components**
   * Defined using ES6 classes.
   * Can use lifecycle methods like componentDidMount(), componentWillUnmount().
   * Can maintain their own state using this.state.
   * Example:

class Welcome extends React.Component {

render() {

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

}

}

**2.Function Components**

Defined as JavaScript functions.

Earlier, they were stateless, but with React Hooks, they can now use state and lifecycle features.

Example:

function Welcome() {

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

}

**4. Explain Class Components**

* A **class component** is a React component created using ES6 classes.
* It must include a **render()** method that returns the JSX to display.
* Class components can hold and manage their own state and have access to lifecycle methods.

import React, { Component } from 'react';

class Welcome extends Component {

render() {

return <h1>Hello, this is a Class Component!</h1>;

}

}

export default Welcome;

**5. Explain Function Components**

* A **function component** is a plain JavaScript function that returns JSX.
* They are simpler and more concise than class components.
* With **React Hooks** like useState() and useEffect(), function components can now manage state and side effects.

import React from 'react';

function Welcome() {

return <h1>Hello, this is a Function Component!</h1>;

}

export default Welcome;

**6. Define Component Constructor**

* The **constructor()** method is a special function used in class components.
* It initializes the component’s state and binds event handler methods.
* It is called automatically when the component is created.

class Welcome extends React.Component {

constructor(props) {

super(props);

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

}

render() {

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

}

}

**7. Define render() Function**

* The **render()** method is mandatory in every class component.
* It tells React what to display on the screen.
* It must return a single parent element (usually JSX).

render() {

return (

<div>

<h1>Hello from Render Function</h1>

</div>

);

}

**About.js:**

import React from 'react';

function About() {

  return (

    <div>

      <h1>Welcome to the About page of the Student Management Portal</h1>

    </div>

  );

}

export default About;

**Contact.js:**

import React from 'react';

function Contact() {

  return (

    <div>

      <h1>Welcome to the Contact page of the Student Management Portal</h1>

    </div>

  );

}

export default Contact;

**Home.js:**

import React from 'react';

function Home() {

  return (

    <div>

      <h1>Welcome to the Home page of the Student Management Portal</h1>

    </div>

  );

}

export default Home;

**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>

      <h1>Student Management Portal</h1>

      <Home />

      <About />

      <Contact />

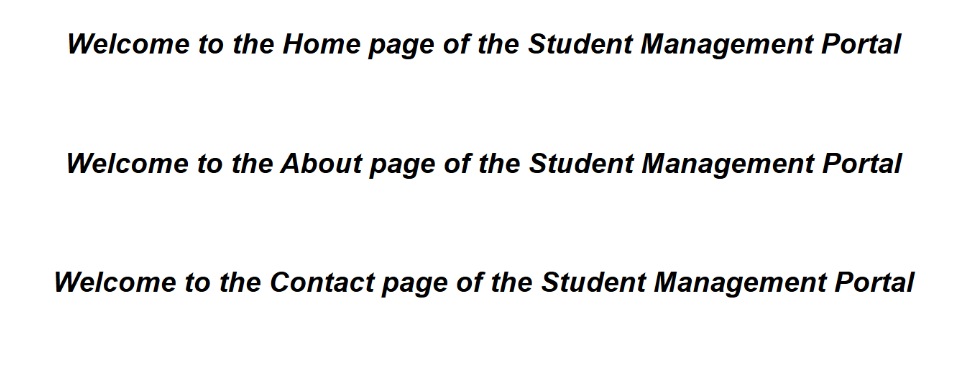
    </div>

  );

}

export default App;

**OUTPUT:**



**ReactJs-HOL-3**

**Student Details Display**

**App.js:**

import './App.css';

import React from "react";

import CalculateScore from "./Components/CalculateScore";

function App() {

return (

<div>

<CalculateScore

Name="Steeve"

School="DNV Public School"

total={284}

goal={300}

/>

</div>

);

}

export default App;

**CalculateScore.js:**

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

const percentToDecimal = (decimal) => {

return (decimal \* 100).toFixed(2) + '%';

};

const calcScore = (total, goal) => {

return percentToDecimal(total / goal);

};

const CalculateScore = ({ Name, School, total, goal }) => (

<div className="formatstyle">

<h1 style={{ color: "brown" }}>Student Details:</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} Marks</span>

</div>

<div className="Score">

<b>Score: </b>

<span>{calcScore(total, goal)}</span>

</div>

</div>

);

export default CalculateScore;

**mystyle.css:**

.Name {

font-weight: 300;

color: blue;

}

.School {

color: crimson;

}

.Total {

color: darkmagenta;

}

.Score {

color: forestgreen;

}

.formatstyle {

text-align: center;

font-size: large;

}

**OUTPUT:**

A screen shot of a computer

AI-generated content may be incorrect.

**ReactJs-HOL-4**

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

In frameworks like Angular or React, a **component life cycle** refers to the series of phases a component goes through from creation to destruction. Understanding this life cycle is crucial for writing efficient, maintainable code.

**Need for Component Life Cycle:**

* To **manage the behavior** of components at different stages (creation, update, and destruction).
* To **optimize performance** by running code at appropriate times.
* To handle **data fetching, event subscriptions, and resource cleanup** effectively.

**Benefits:**

* **Efficient Resource Management**: Enables initialization and cleanup (e.g., closing timers, unsubscribing from observables).
* **Code Organization**: Separates logic for different stages of a component’s existence.
* **Improved Debugging**: Developers can pinpoint issues to specific life cycle stages.
* **Reusability**: Allows reuse of logic across components (e.g., setting default states during initialization).

**2. Life Cycle Hook Methods in React**

In **class components**, React provides lifecycle methods that let you run code at specific points during a component’s lifetime. These are grouped into **Mounting**, **Updating**, and **Unmounting** phases.

**Mounting (When component is created & inserted into the DOM)**

* constructor() → Initialize state and bind methods.
* static getDerivedStateFromProps(props, state) → Sync state with props. Rarely used.
* render() → Returns JSX to render UI.
* componentDidMount() → Called after the component is rendered in the DOM. (Good for API calls, subscriptions).

**Updating (When props or state changes)**

* static getDerivedStateFromProps() → Again called before re-rendering.
* shouldComponentUpdate(nextProps, nextState) → Determines whether to re-render or not.
* render() → Re-renders the component.
* getSnapshotBeforeUpdate(prevProps, prevState) → Capture some info (e.g., scroll position) before DOM updates.
* componentDidUpdate(prevProps, prevState, snapshot) → Called after DOM updates.

**Unmounting (When component is removed from the DOM)**

* componentWillUnmount() → Cleanup work like removing timers or listeners.

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

**Mounting Sequence**

1. constructor()
2. getDerivedStateFromProps()
3. render()
4. (DOM updates)
5. componentDidMount()

**Updating Sequence**

1. getDerivedStateFromProps()
2. shouldComponentUpdate()
3. render()
4. getSnapshotBeforeUpdate()
5. (DOM updates)
6. componentDidUpdate()

**Unmounting Sequence**

1. componentWillUnmount()

**BlogApp:**

**Post.js:**

import React, { Component } from 'react';

class Posts extends Component {

  constructor(props) {

    super(props);

    this.state = {

      posts: []

    };

  }

  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));

  };

  componentDidMount() {

    this.loadPosts();

  }

  componentDidCatch(error, info) {

    alert("An error occurred: " + error);

  }

  render() {

    return (

      <div>

        <h1>Blog Posts</h1>

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

          <div key={post.id}>

            <h2>{post.title}</h2>

            <p>{post.body}</p>

          </div>

        ))}

      </div>

    );

  }

}

export default Posts;

**App.js:**

import logo from './logo.svg';

import './App.css';

import React from 'react';

import Posts from './Post';

function App() {

  return (

    <div>

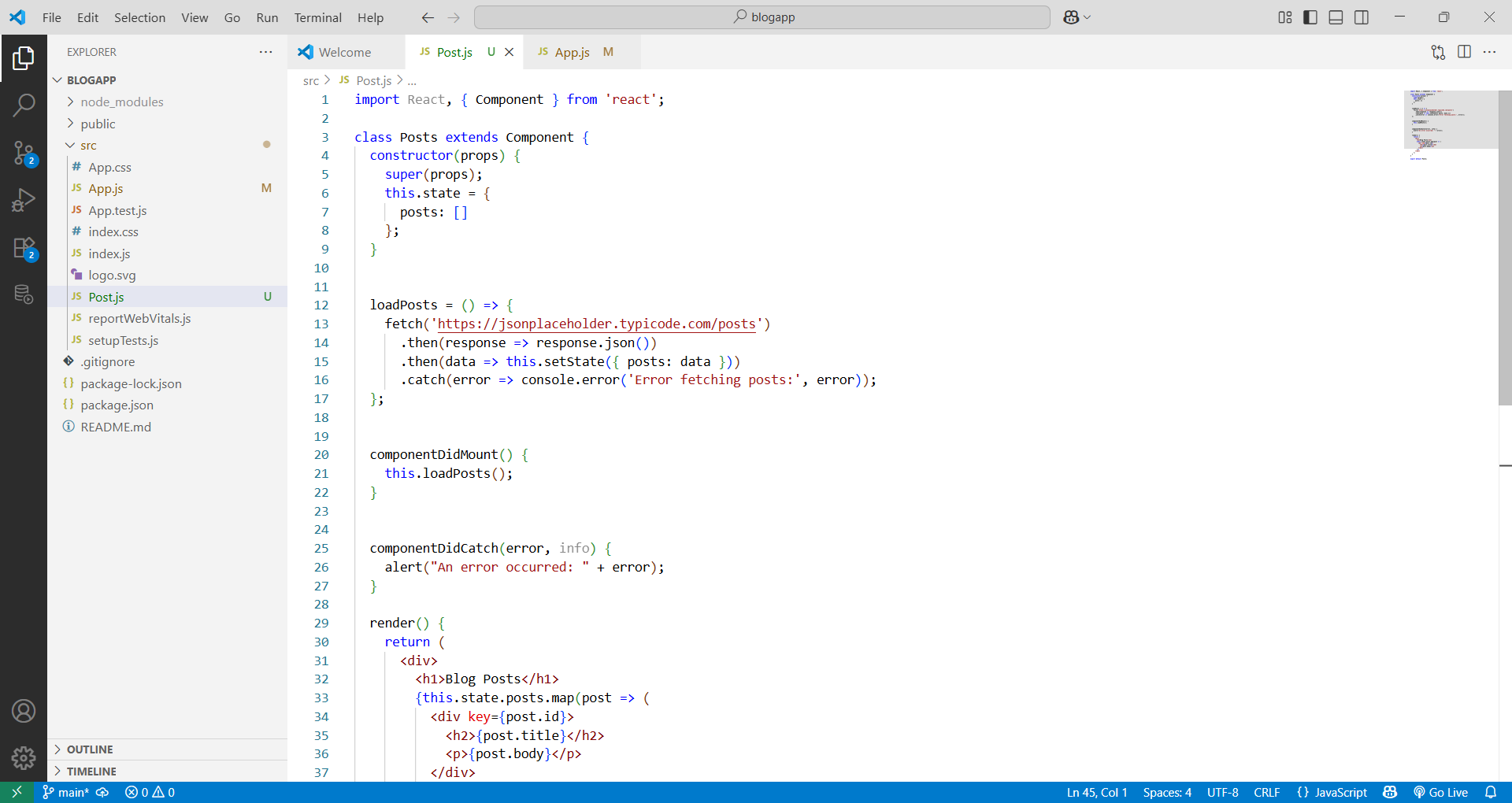
      <Posts />

    </div>

  );

}

export default App;



**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

**ReactJs-HOL-5**

**src/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;

}

**src/components/CohortDetails.js**

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

function CohortDetails(props) {

    const { cohort } = props;

    const titleColor = cohort.currentStatus === 'Ongoing' ? 'green' : 'blue';

    return (

        <div className={styles.box}>

            <h3 style={{ color: titleColor }}>

                {cohort.cohortCode} - <span>{cohort.technology}</span>

            </h3>

            <dl>

                <dt>Started On</dt>

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

                <dt>Current Status</dt>

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

                <dt>Coach</dt>

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

                <dt>Trainer</dt>

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

            </dl>

        </div>

    );

}

export default CohortDetails;

**src/App.js**

import './App.css';

import { CohortsData} from './Cohort'

import CohortDetails from './CohortDetails';

function App() {

  return (

  <div>

    <h1>Cohorts Details</h1>

    {CohortsData.map(cohort => <CohortDetails cohort={cohort}/>)}

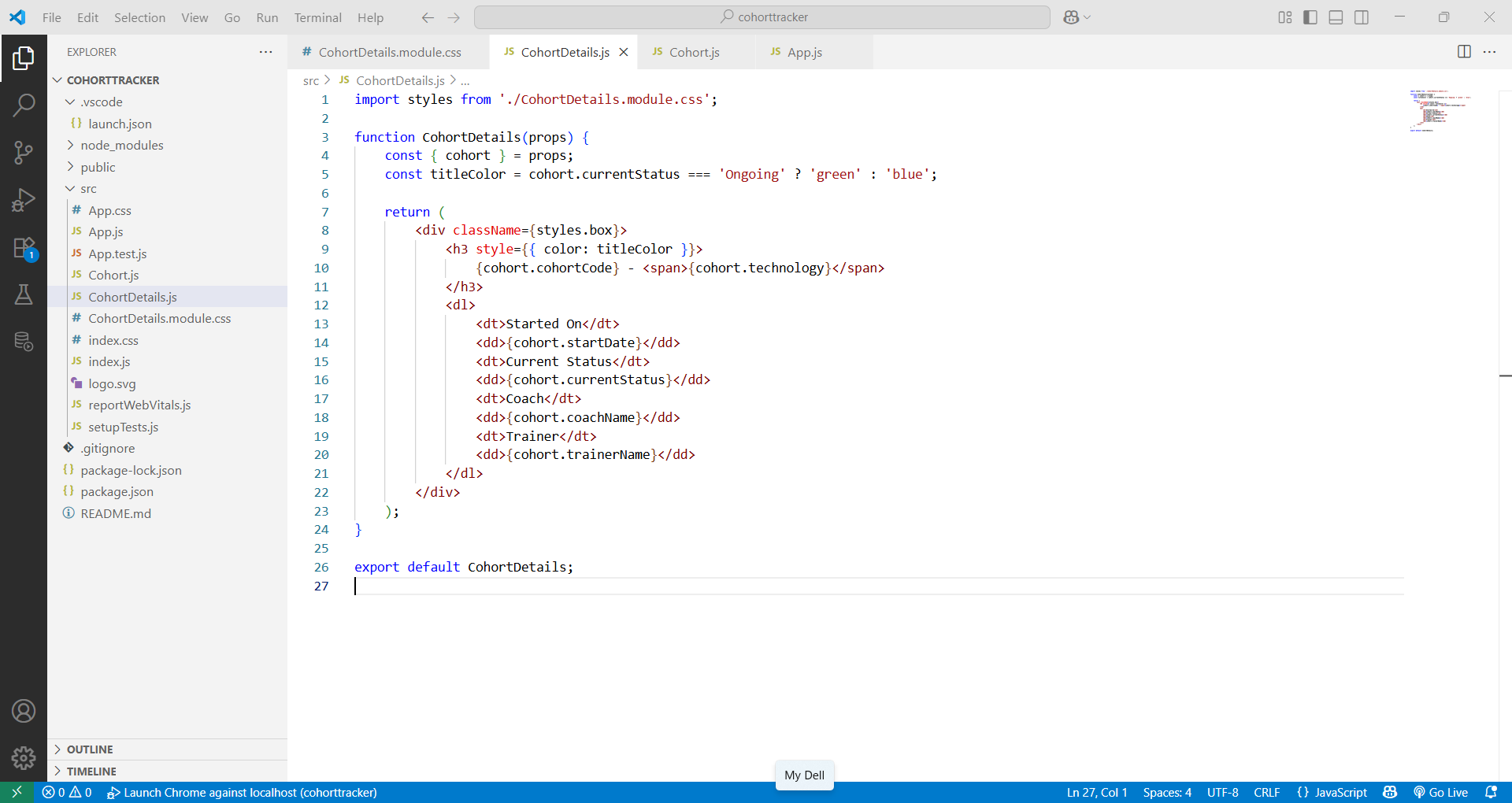
  </div>

  );

}

export default App;

**OUTPUT:**

****

A screenshot of a computer

AI-generated content may be incorrect.