**REACTJS-HOL**

**Objectives**

**1. Define SPA and its benefits**

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

**Benefits of SPA:**

* **Faster navigation** – Only data changes, not the whole page reload.
* **Smooth user experience** – Feels like a desktop app.
* **Reduced server load** – Fewer requests for full-page reloads.
* **Better caching** – SPA can cache local data using JavaScript.
* **Code reusability** – Easier to reuse components across the app.

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

React is an open-source JavaScript library developed by Facebook for building user interfaces, especially for SPAs. It uses **components** to build reusable UI elements.

**How React Works:**

* Uses JSX (JavaScript XML) to describe UI elements.
* Maintains a Virtual DOM to track changes.
* Updates only the changed part of the UI in the actual DOM using efficient diffing algorithms.

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

|  |  |  |
| --- | --- | --- |
| **Feature** | **SPA (Single-Page App)** | **MPA (Multi-Page App)** |
| Page Reload | No reload; content updates dynamically | Every interaction loads a new page |
| Speed | Fast and responsive | Slower due to full-page reloads |
| Development Complexity | Easier routing with tools like React | Complex due to multiple HTML pages |
| SEO | Challenging for SEO | Better SEO support |
| Examples | Gmail, Facebook | Amazon, Wikipedia |

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

**Pros:**

* Fast performance after the first load
* Better user experience
* Easier code reuse and component-based structure

**Cons:**

* Poor SEO by default (can be improved using SSR)
* May require extra security handling (like XSS)
* Initial load time can be large due to loading all JS at once

**5. Explain about React**

React is:

* A JavaScript library for building UI
* Focused only on the view layer (in MVC)
* Component-based: UI is split into small, reusable pieces
* Declarative: You describe what the UI should look like, not how to update it
* Powered by a virtual DOM for efficient UI rendering

**6. Define Virtual DOM**

**Virtual DOM (VDOM)** is a lightweight JavaScript representation of the real DOM.

**How it works:**

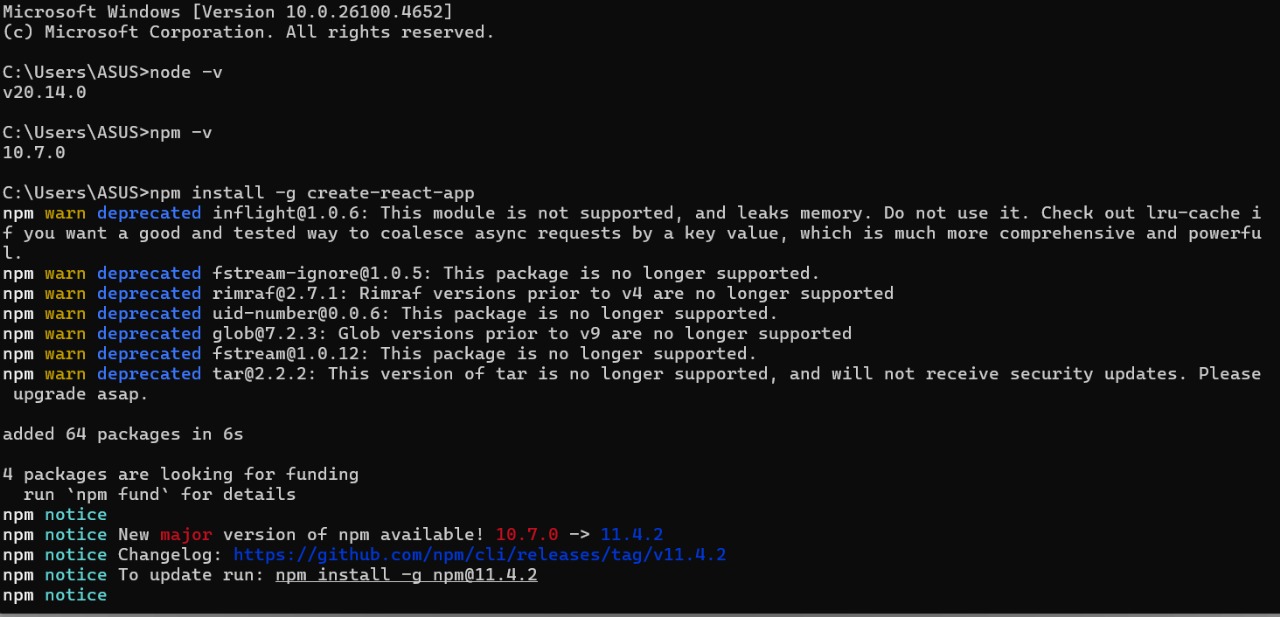
1. React creates a virtual DOM when the state changes.
2. It compares the new virtual DOM with the previous one **(diffing).**
3. Only the changed parts are updated in the actual DOM (**reconciliation**).

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.

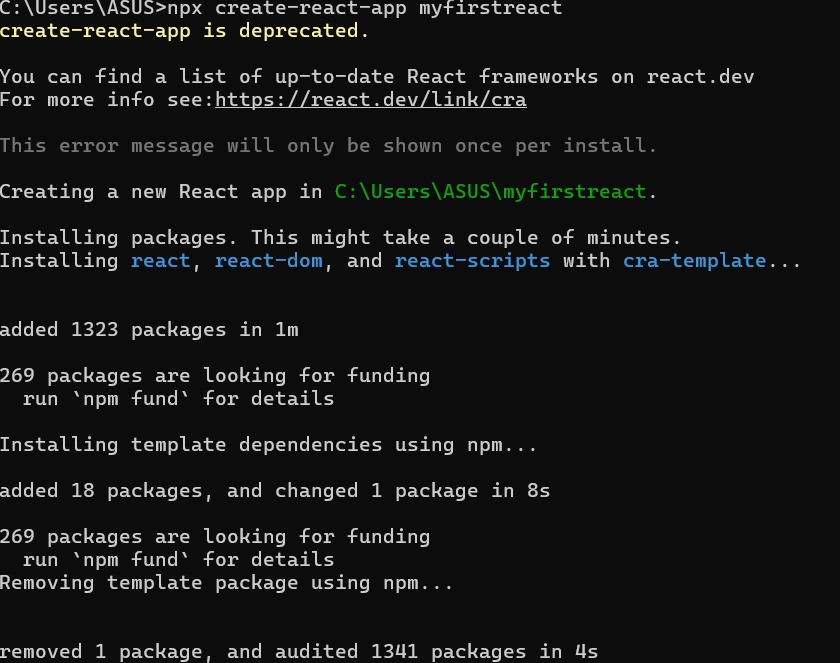
1.To create a new React app, Install Nodejs and Npm from the following link:

<https://nodejs.org/en/download/>

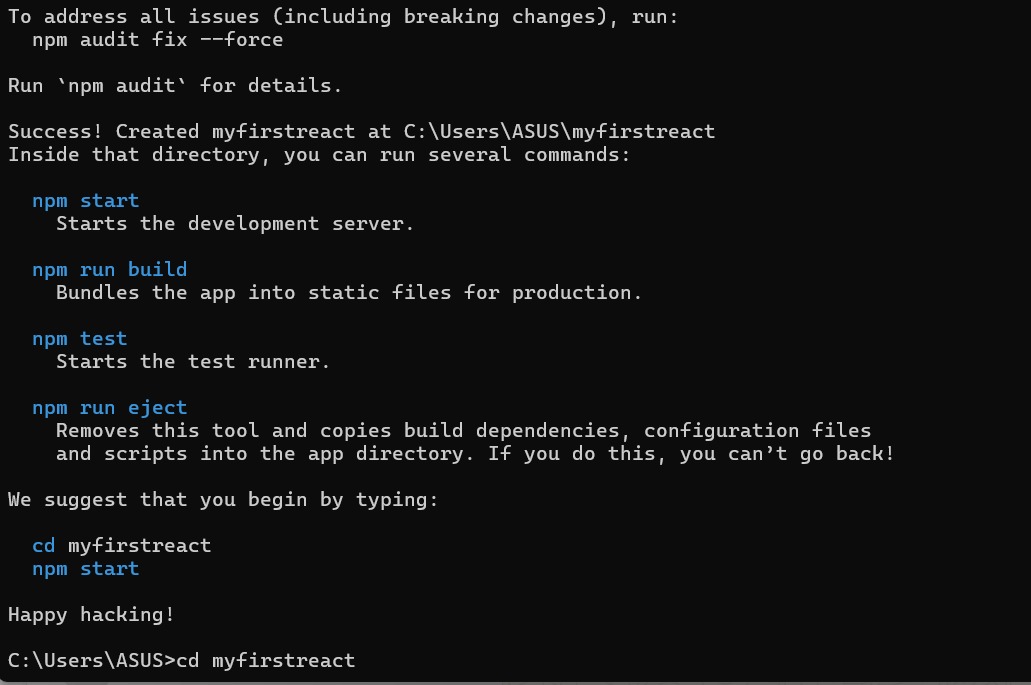
2.Install Create-react-app by running the following command in the command prompt:



3.To create a React Application with the name of “myfirstreact”, type the following command:



4.Once the App is created, navigate into the folder of myfirstreact by typing the following command:



5.Open the folder of myfirstreact in Visual Studio Code

6.Open the App.js file in Src Folder of myfirstreact

7.Remove the current content of “App.js”

8.Replace it with the following:

App.js

function App() {

  return (

    <div>

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

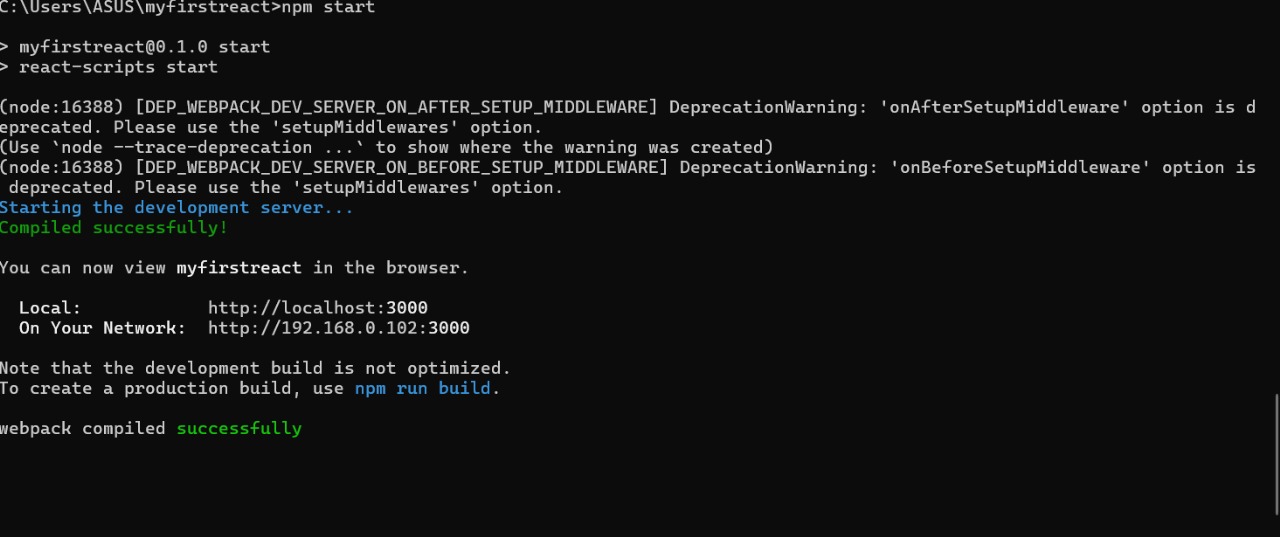
    </div>

  );

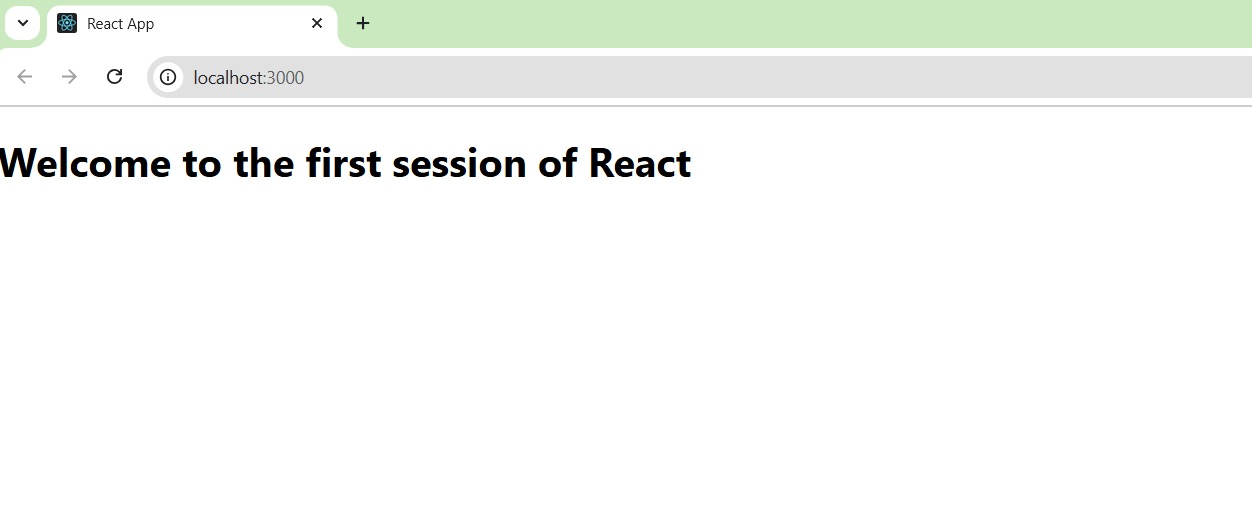
}

export default App;

9. Run the following command to execute the React application:



10.Open a new browser window and type “localhost:3000” in the address bar



**Objectives:**

**1. Explain React Components**

React components are the building blocks of a React application. They represent parts of the user interface (UI), such as a button, form, or entire page section. Components can be reused, nested, and managed independently.

* A component takes input and returns JSX which describes what should appear on the screen.
* React components make the UI modular, maintainable, and testable.

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

| **Feature** | **JavaScript Function** | **React Component** |
| --- | --- | --- |
| Purpose | Performs logic/calculations | Defines UI structure and behavior |
| Return | Any data (number, string, object) | JSX (UI representation) |
| Naming Convention | Any case (usually camelCase) | PascalCase (e.g., MyComponent) |
| Used in React? | Not directly | Yes, used as building blocks |
| Lifecycle Methods | No lifecycle | React components (especially class) have them |

**3. Identify the Types of Components**

React has two main types of components:

| **Type** | **Description** |
| --- | --- |
| **Class Component** | A JavaScript class that extends React.Component. Has lifecycle methods and state. |
| **Function Component** | A plain JavaScript function. Can use Hooks to manage state and side effects. |

**4. Explain Class Component**

A Class Component is a React component defined using a JavaScript class that extends React.Component.

**Example:**

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 define a component using a JavaScript function.

**Example:**

function Welcome(props) {

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

}

**6. Define Component Constructor**

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

* Initialize state
* Bind event handlers

**Syntax:**

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = {

count: 0

};

}

}

**7. Define render() Function**

The render() function is a required method in class components. It returns the JSX that defines what the UI should display.

**Example:**

class Greeting extends React.Component {

render() {

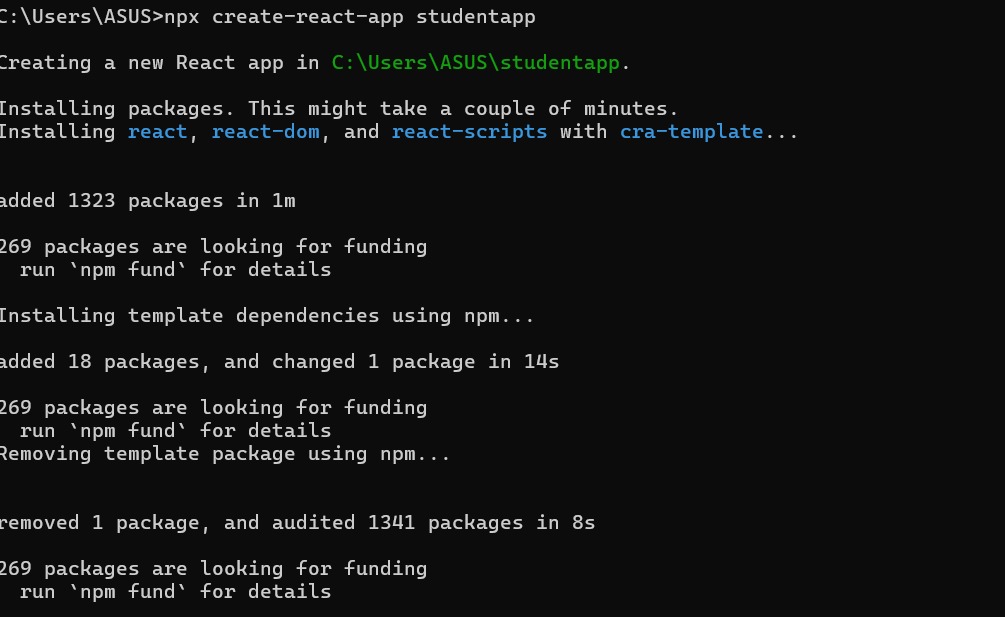
return <h1>Hello, world!</h1>;

}

}

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.

1. Create a React project named “StudentApp” type the following command in terminal of Visual studio:



2. Create a new folder under Src folder with the name “Components”. Add a new file named “Home.js”

3. 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;

4. Under Src folder add another file named “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;

5. Repeat the same steps for Creating “About” and “Contact” component by adding a new file as ”About.js”, “Contact.js” under “Src” folder and edit the code as mentioned for “Home” Component.

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;

6. Edit the App.js to invoke the Home, About and Contact component as follows:

import React from 'react';

import Home from './Components/Home';

import About from './Components/About';

import Contact from './Components/Contact';

import './App.css'

function App() {

  return (

    <div className='container'>

      <Home />

      <About />

      <Contact />

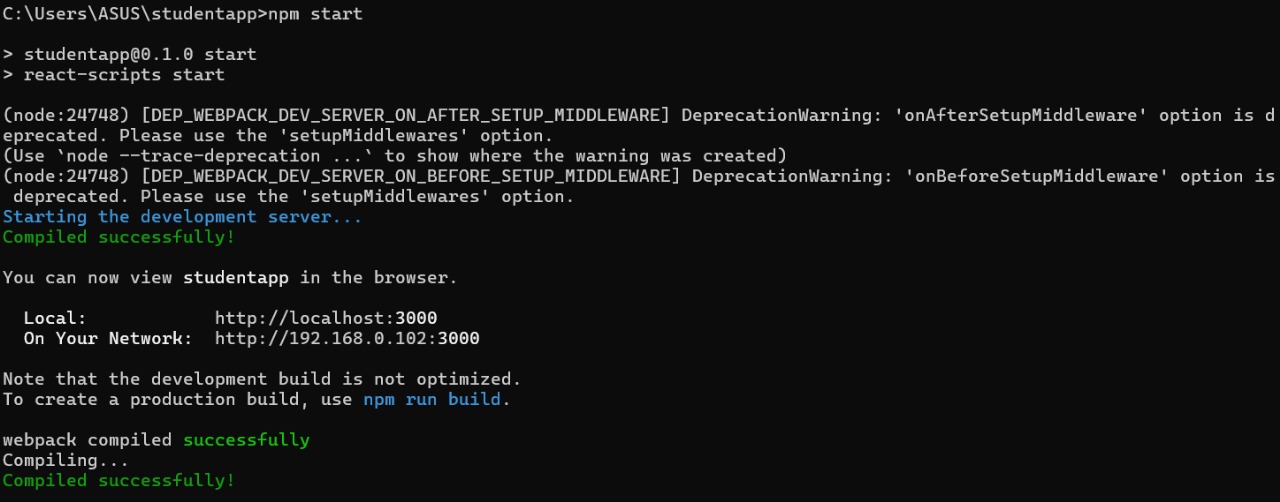
    </div>

  );

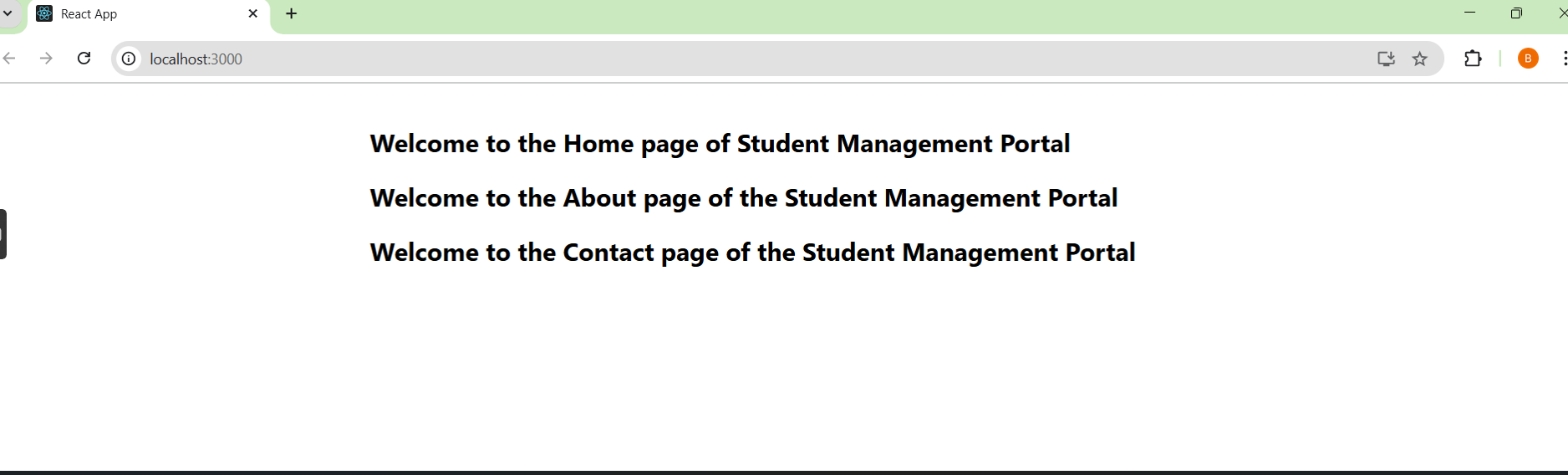
}

export default App;

7. In command Prompt, navigate into StudentApp and execute the code by typing the following command:

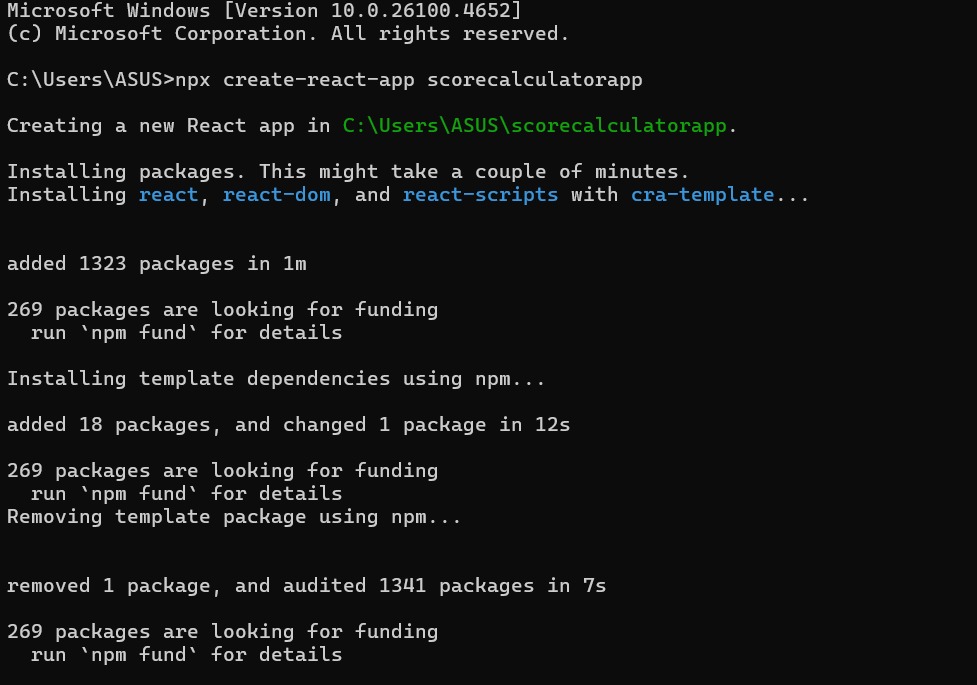


8. Open browser and type “localhost:3000” in the address bar:



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.

1. Create a React project named “scorecalculatorapp” type the following command in terminal of Visual studio:



2. Create a new folder under Src folder with the name “Components”. Add a new file named “CalculateScore.js”

3. **CalculateScore.js**

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

const percentToDecimal = (decimal) => {

    return (decimal.toFixed(2) + '%');

};

const calcScore = (total, goal) => {

    return percentToDecimal(total / goal);

};

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 a Folder named Stylesheets and add a file named “mystyle.css” in order to add some styles to the components:

.Name {

    font-weight: 300;

    color: blue;

}

.School {

    color: crimson;

}

.Total {

    color: darkmagenta;

}

.formatstyle {

    text-align: center;

    font-size: large;

}

.Score {

    color: forestgreen;

}

5. Edit the App.js to invoke the CalculateScore functional component as follows:

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

function App() {

    return (

        <div>

            <CalculateScore

                Name="Steeve"

                School="DNV Public School"

                total={284}

                goal={3}

            />

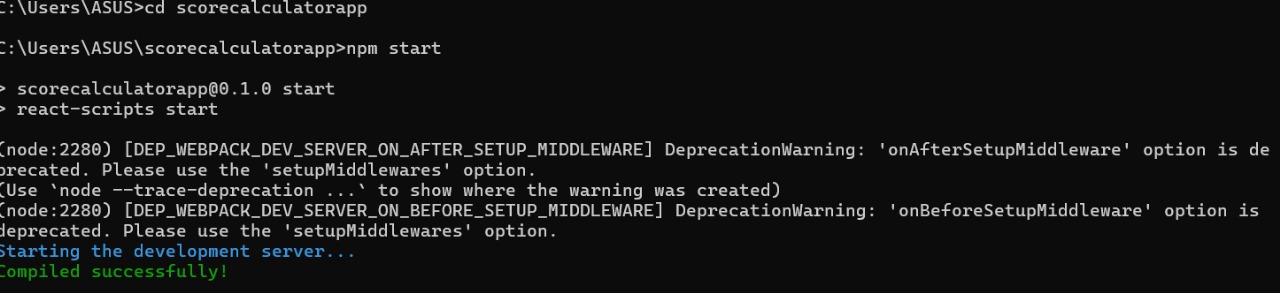
        </div>

    );

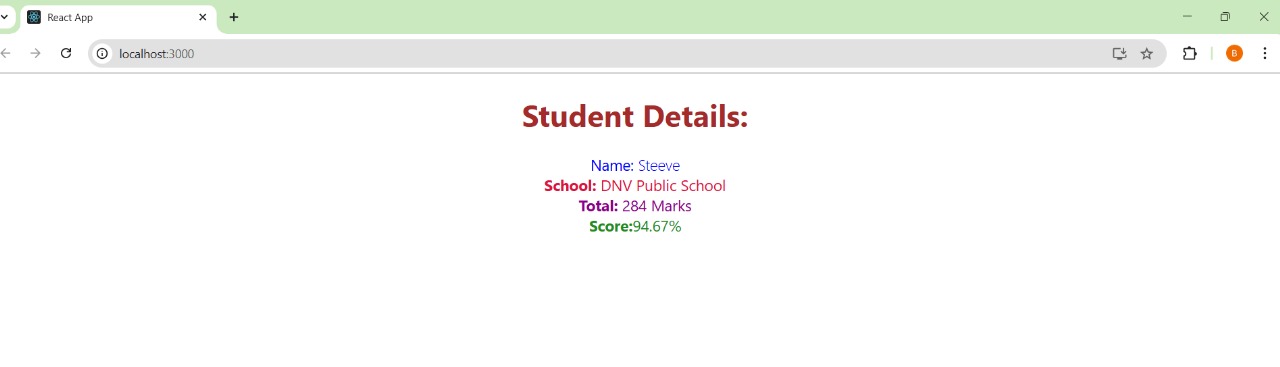
}

export default App;

6. In command Prompt, navigate into scorecalculatorapp and execute the code by typing the following command:



7. Open browser and type “localhost:3000” in the address bar:



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

Need for Component Lifecycle:  
React applications are dynamic and interactive. The component lifecycle helps manage how a component:

* Mounts (is created and added to the DOM)
* Updates (changes due to props or state)
* Unmounts (is removed from the DOM)

**Benefits:**

* Control over each phase of the component’s existence
* Efficient resource management, such as setting or clearing timers, listeners, or API calls
* Improved performance by controlling when updates occur
* Debugging and logging by tracking component behavior during its lifecycle

**2. Identify Various Lifecycle Hook Methods**

Lifecycle methods differ between class components and function components:

**For Class Components:**

| **Phase** | **Lifecycle Method** | **Purpose** |
| --- | --- | --- |
| Mounting | constructor() | Initialize state and bind methods |
|  | static getDerivedStateFromProps() | Update state from props before rendering |
|  | render() | Returns JSX to be rendered |
|  | componentDidMount() | Called after component is added to DOM |
| Updating | shouldComponentUpdate() | Determines if component should re-render |
|  | getSnapshotBeforeUpdate() | Captures info before the DOM is updated |
|  | componentDidUpdate() | Called after the component re-renders |
| Unmounting | componentWillUnmount() | Cleanup before the component is removed |

**For Function Components (with Hooks):**

| **Hook** | **Purpose** |
| --- | --- |
| useEffect() | Acts like componentDidMount, DidUpdate, and WillUnmount combined |
| useState() | Manages state in functional components |
| useRef(), useMemo(), useCallback() | For optimization and DOM access |

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

Here’s the typical sequence for **class components**:

**Mounting Phase (Initial Render):**

1. constructor()
2. getDerivedStateFromProps()
3. render()
4. componentDidMount()

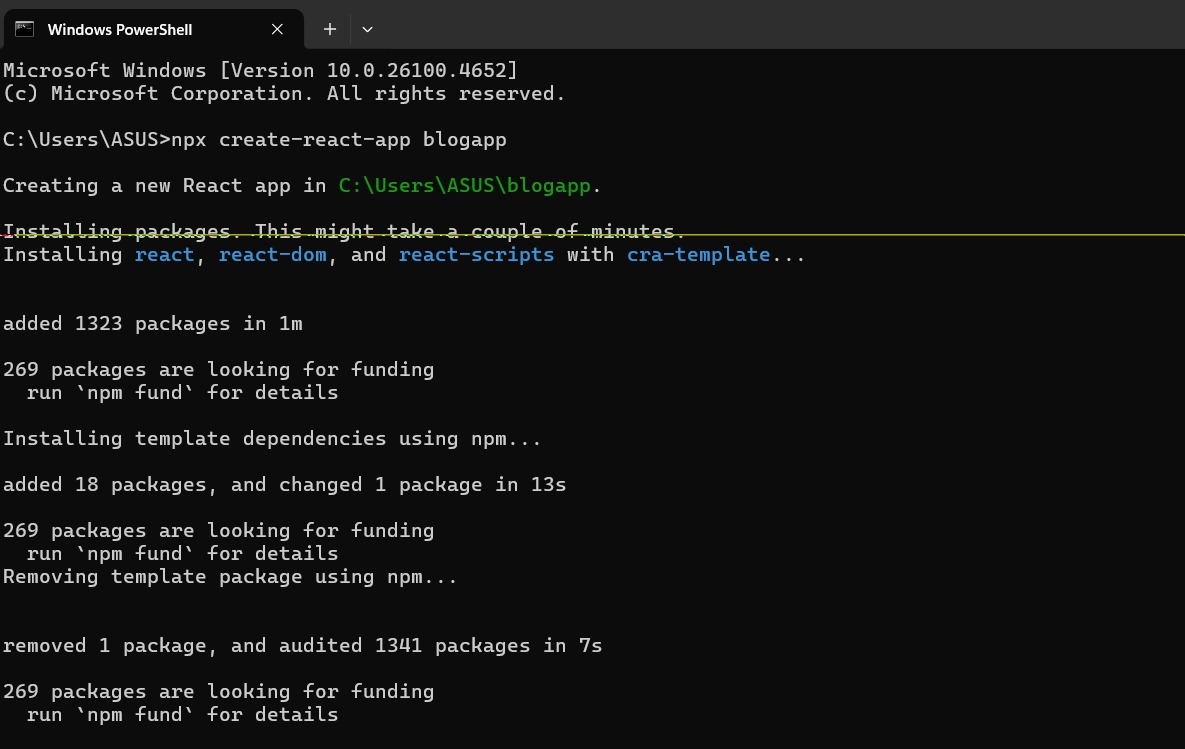
**Updating Phase (Re-render due to state/props change):**

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

**Unmounting Phase (Component removal):**

1. componentWillUnmount()

1. Create a new react application using *create-react-app* tool with the name as “blogapp”



2. Open the application using VS Code

3. Create a new file named as **Post.js** in **src folder** with following properties

import React from 'react';

class Post extends React.Component {

  render() {

    const { title, body } = this.props;

    return (

      <div>

        <h3>{title}</h3>

        <p>{body}</p>

        <hr />

      </div>

    );

  }

}

export default Post;

4. Create a new class based component named as **Posts** inside **Posts.js** file

5. Initialize the component with a list of Post in state of the component using the constructor

6. 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>)

7. Implement the **componentDidMount()** hook to make calls to **loadPosts()** which will fetch the posts

8. Implement the **render()** which will display the title and post of posts in html page using heading and paragraphs respectively.

9*.*Define a **componentDidCatch()** method which will be responsible for displaying any error happing in the component as alert messages.

import React from 'react';

import Post from './Post';

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);

      });

  }

  componentDidMount() {

    this.loadPosts();

  }

  componentDidCatch(error, info) {

    alert('An error occurred while rendering the component.');

    this.setState({ hasError: true });

  }

  render() {

    if (this.state.hasError) {

      return <h2>Something went wrong!</h2>;

    }

    return (

      <div>

        <h1>Blog Posts</h1>

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

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

        ))}

      </div>

    );

  }

}

export default Posts;

10. Add the Posts component to App component.

import React from 'react';

import Posts from './Posts';

import './App.css'

function App() {

  return (

    <div className="App">

      <Posts />

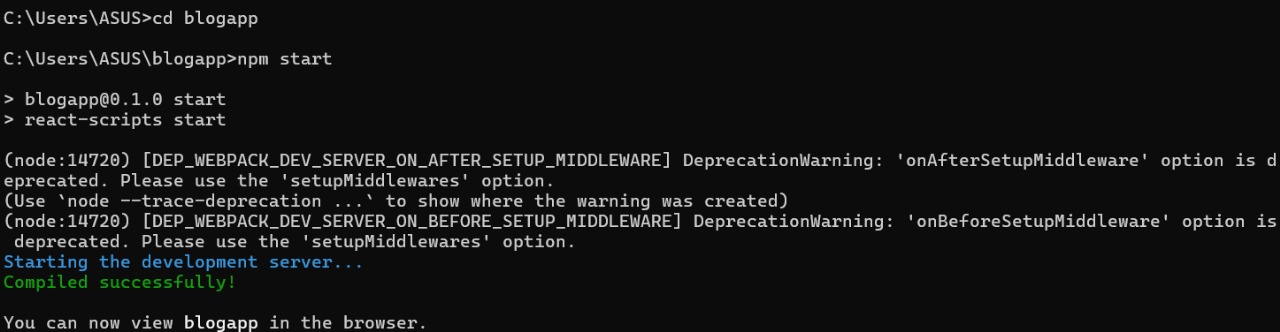
    </div>

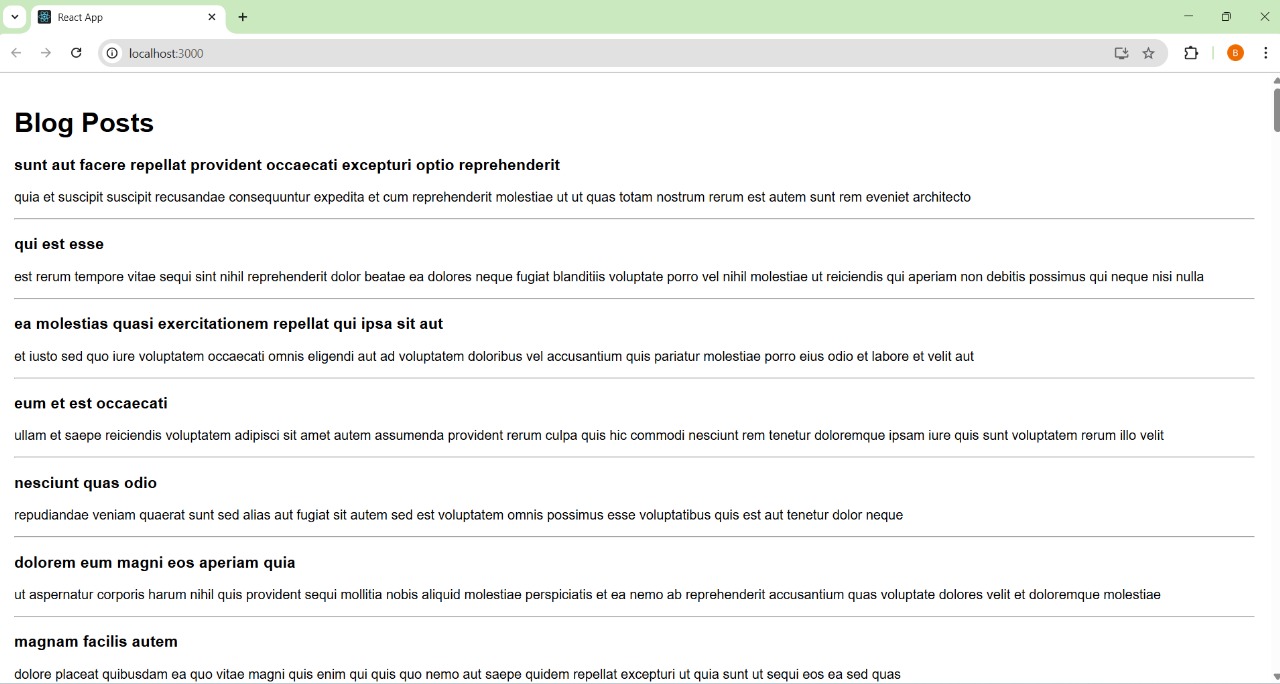
  );

}

export default App;

11. Build and Run the application using *npm start* command.



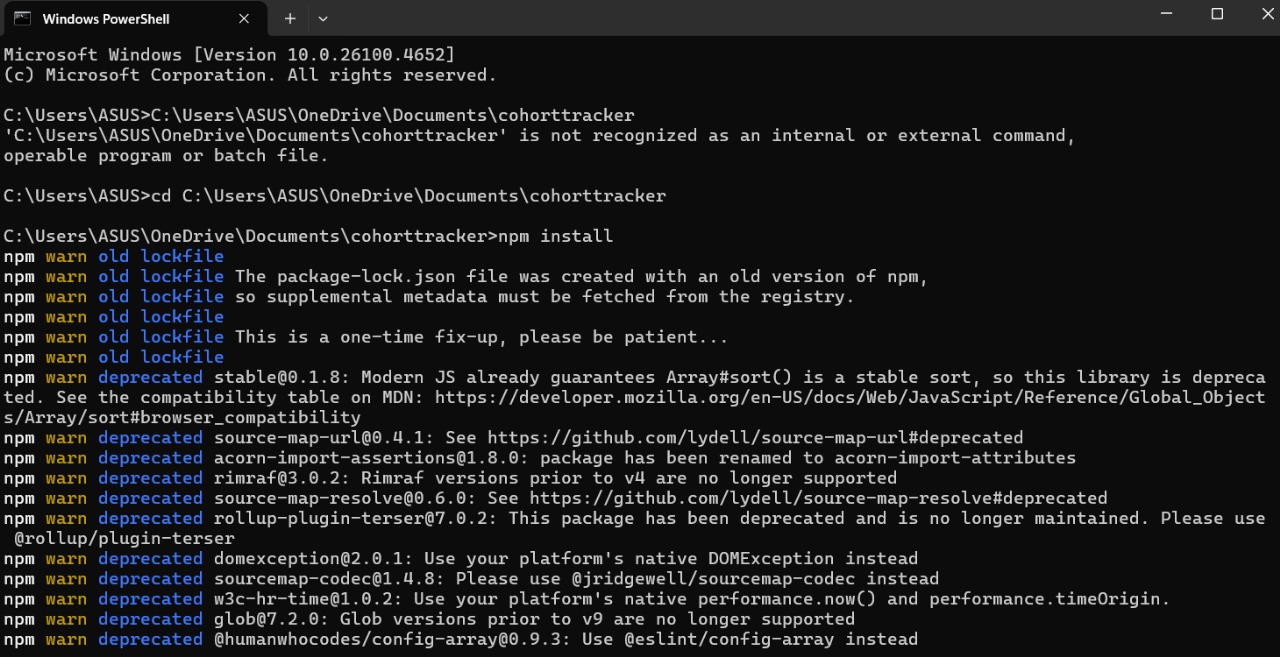


My Academy team at Cognizant want to create a dashboard containing the details of ongoing and completed cohorts. A react application is created which displays the detail of the cohorts using react component. You are assigned the task of styling these react components.

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.



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
4. Define a css style for html <dt> element using tag selector. Set the font weight to 500.

.box {

  width: 300px;

  display: inline-block;

  margin: 10px;

  padding: 10px 20px;

  border: 1px solid black;

  border-radius: 10px;

}

dt {

  font-weight: 500;

}

1. Open the cohort details component and import the CSS Module
2. Apply the box class to the container div
3. Define the style for <h3> element to use “green” color font when cohort status is “ongoing” and “blue” color in all other scenarios.

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

function CohortDetails(props) {

  return (

    <div className={styles.box}>

            <h3 style={{ color: props.cohort.currentStatus === 'Ongoing' ? 'green' : 'blue' }}>

                {props.cohort.cohortCode} -

                <span>{props.cohort.technology}</span>

            </h3>

            <dl>

                <dt>Started On</dt>

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

                <dt>Current Status</dt>

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

                <dt>Coach</dt>

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

                <dt>Trainer</dt>

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

            </dl>

        </div>

    );

}

export default CohortDetails;

