## **HANDS ON\_01**

## **Objectives**

* Define SPA and its benefits
* Define React and identify its working
* Identify the differences between SPA and MPA
* Explain Pros & Cons of Single-Page Application
* Explain about React
* Define virtual DOM
* Explain Features of React

In this hands-on lab, you will learn how to:

* Set up a react environment
* Use create-react-app

## **Prerequisites**

The following is required to complete this hands-on lab:

* Node.js
* NPM
* Visual Studio Code

## **Notes**

Estimated time to complete this lab: **30 minutes.**

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/>

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



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



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



1. Open the folder of myfirstreact in Visual Studio Code
2. Open the App.js file in Src Folder of myfirstreact
3. Remove the current content of “App.js”
4. Replace it with the following:



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



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



**What is an SPA (Single Page Application) and Its Benefits**

1. A Single Page Application (SPA) loads one HTML page and dynamically updates content as you interact—without refreshing the whole page
2. Benefits include: Smooth, fast user experience Reduced data usage—only necessary parts are updated
3. Better state management and UI continuity via modular front‑end logic

**What Is React and Its Working?**

1. React (also known as React.js) is an open‑source JavaScript library by Meta (formerly Facebook), launched in 2013, for building UIs—especially SPAs
2. Component‑based: UI is built using reusable components; functional components with Hooks are now the modern standard
3. JSX syntax: write HTML‑style code within JavaScript, which gets converted into React.createElement calls
4. Virtual DOM & diff/reconciliation: React creates a Virtual DOM, diffs changes, and updates only what's necessary in the real DOM
5. One‑way data flow & Hooks: data flows via props; Hooks like useState and useEffect let functional components manage state and side effects

**MPA vs SPA: Difference?**

| **Feature** | **SPA** | **MPA (Multi Page App)** |
| --- | --- | --- |
| Page reloads | None—page fragments update dynamically | Full browser reload on each navigation |
| Speed | Very fast after initial load | Slower—new page load adds overhead |
| URL & navigation | Managed via history API (pushState, etc.) | Standard navigation and browser history |
| SEO friendliness | Harder—needs SSR or pre‑rendering | SEO works naturally |
| Use case | Web apps like dashboards, email, interactive UIs | Content sites—blogs, e‑commerce, marketing pages |

**Pros & Cons of SPAs**

* Pros:  
  • Seamless, app‑like interactivity   
  • Faster updates after initial load   
  • Efficient client‑side rendering and caching
* Cons:  
  • Initial load may be slower (large JS bundle)   
  • SEO needs extra setup (SSR, hydration, static rendering)

• Browser back‑button, analytics, and security require custom handling

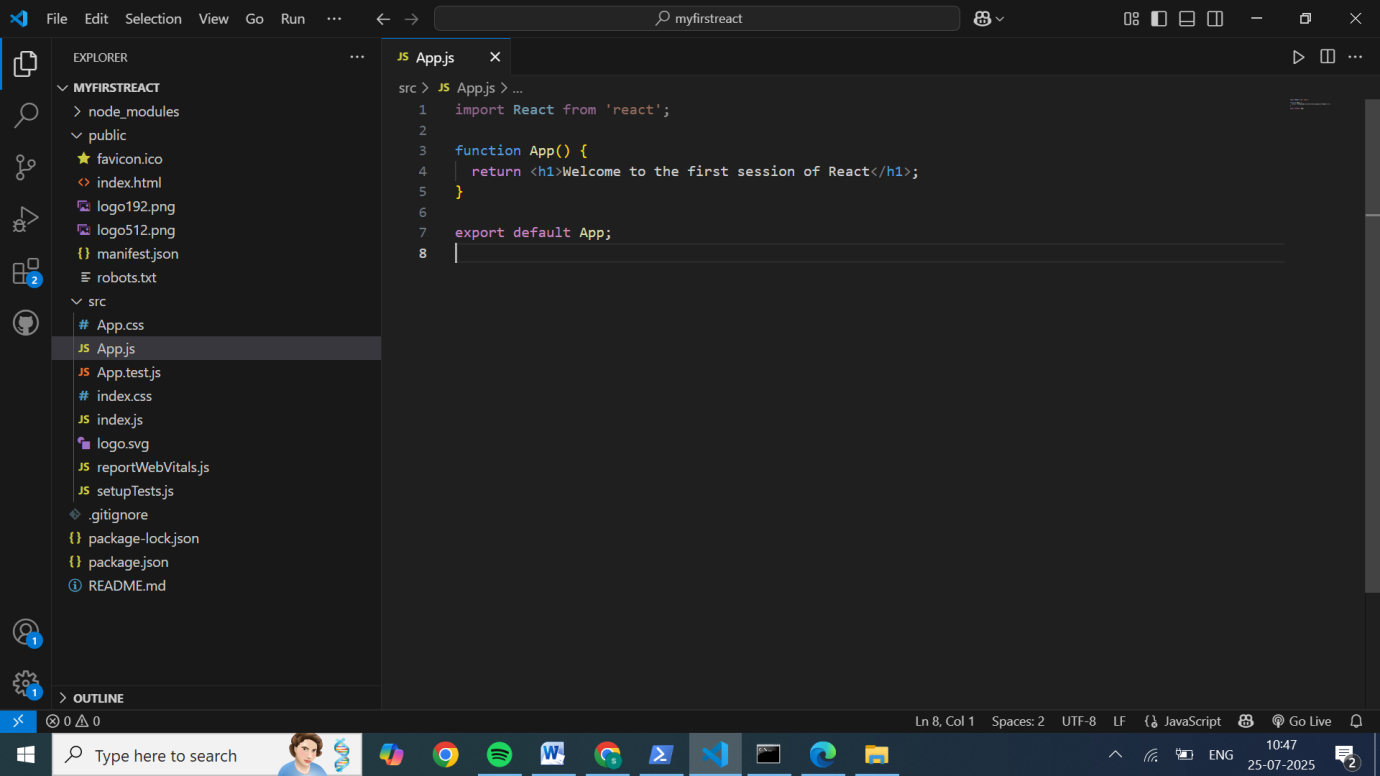
**What Is the Virtual DOM?**

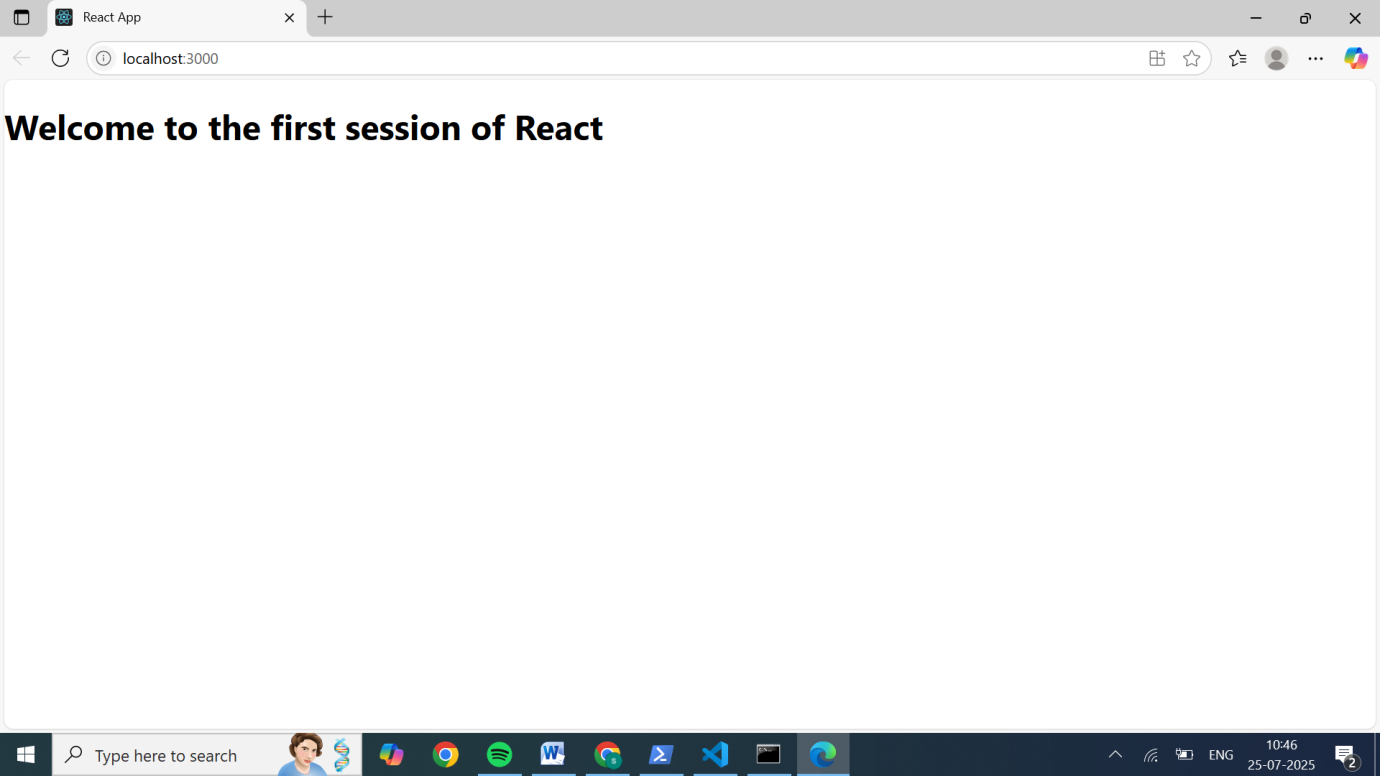
1. The Virtual DOM is React’s in‑memory representation of the UI interface
2. When state or props change, React diffs the new Virtual DOM vs the old one, then applies just those changes to the real DOM for efficient updates

**Features of React**

* Component‑based architecture: UI built as independent, reusable blocks
* Unidirectional data flow: easier to track state and debug
* JSX: HTML‑like syntax within JavaScript for more intuitive code structure
* State and props management: local component state vs parent-provided props
* Lifecycle methods and Hooks: manage component behavior over time with ease Virtual DOM diffing: updates only what actually changed for performance benefits

**OUTPUT:**





## **HANDS ON\_02**

## **Objectives**

* Explain React components
* Identify the differences between components and JavaScript functions
* Identify the types of components
* Explain class component
* Explain function component
* Define component constructor
* Define render() function

In this hands-on lab, you will learn how to:

* Create a class component
* Create multiple components
* Render a component

## **Prerequisites**

The following is required to complete this hands-on lab:

* Node.js
* NPM
* Visual Studio Code

## **Notes**

Estimated time to complete this lab: **30 minutes.**

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:



1. Create a new folder under Src folder with the name “Components”. Add a new file named “Home.js”
2. Type the following code in Home.js



1. Under Src folder add another file named “About.js”
2. 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.
3. Edit the App.js to invoke the Home, About and Contact component as follows:



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



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

 **React Components**

**Class Components**  
Class components are ES6 classes that extend React.Component. They have access to lifecycle methods and can manage local state using this.state. The render() method returns JSX to describe the component's UI.

**Function Components**  
Function components are simpler and are written as JavaScript functions that return JSX. They can accept props and are often used for components that don't require their own state or lifecycle methods.

**Identify the differences between components and JavaScript functions**  
A React component is a JavaScript function (or class) that takes a single props object and returns JSX—React treats this as a virtual UI to render and manage (e.g., tracking updates, state, and effects). In contrast, a plain JavaScript function simply accepts any arguments and returns any values, without integration into React’s rendering or lifecycle system.

**Identify the types of components**  
There are two types of components:

* Class components
* Function components

**Class Components**  
Class components are ES6 classes that extend React.Component. They have access to lifecycle methods and can manage local state using this.state. The render() method returns JSX to describe the component's UI.

**Function Components**  
Function components are simpler and are written as JavaScript functions that return JSX. They can accept props and are often used for components that don't require their own state or lifecycle methods.

**Define component constructor**  
In a React class component, the constructor is an optional special method that runs before the component mounts. Inside it, you must call super(props) to properly initialize this.props. The constructor is typically used to initialize local state by directly assigning this.state = { … }, and to bind event handling methods to the component instance (e.g., this.handleClick = this.handleClick.bind(this)). It should not include side effects, subscriptions, or API calls—those belong in mounting lifecycle methods like componentDidMount().

**Define render() function**  
The render() method is the only required method in a React class component. It must return the UI description in the form of JSX (or React elements, arrays/fragments, portals, strings, numbers, and even booleans or null). render() should be pure: no side effects, no direct state mutations, just taking props and state and returning what should be displayed on screen.

CODES:

Home.js:

import React from 'react';

function Home() {

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

}

export default Home;

About.js:

import React from 'react';

function About() {

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

}

export default About;

Contact.js:

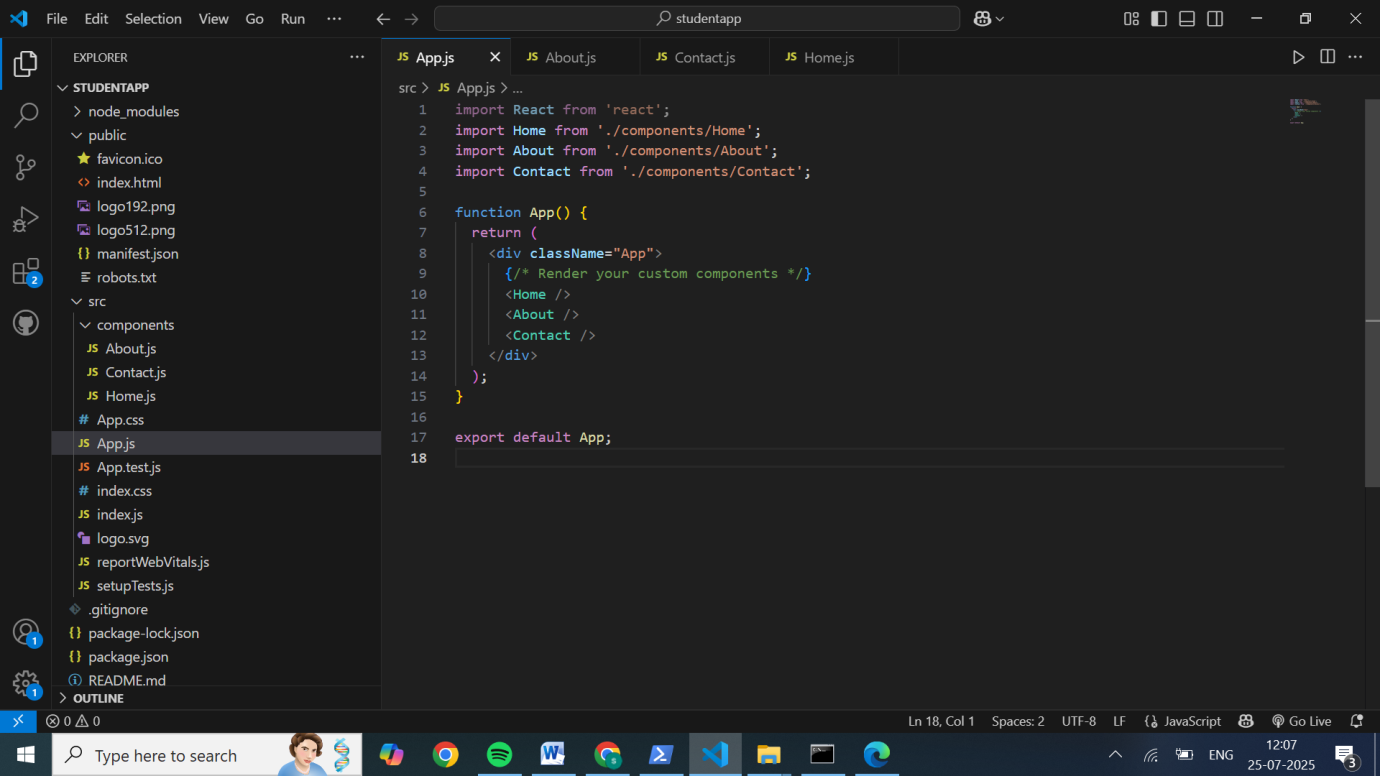
import React from 'react';

function Contact() {

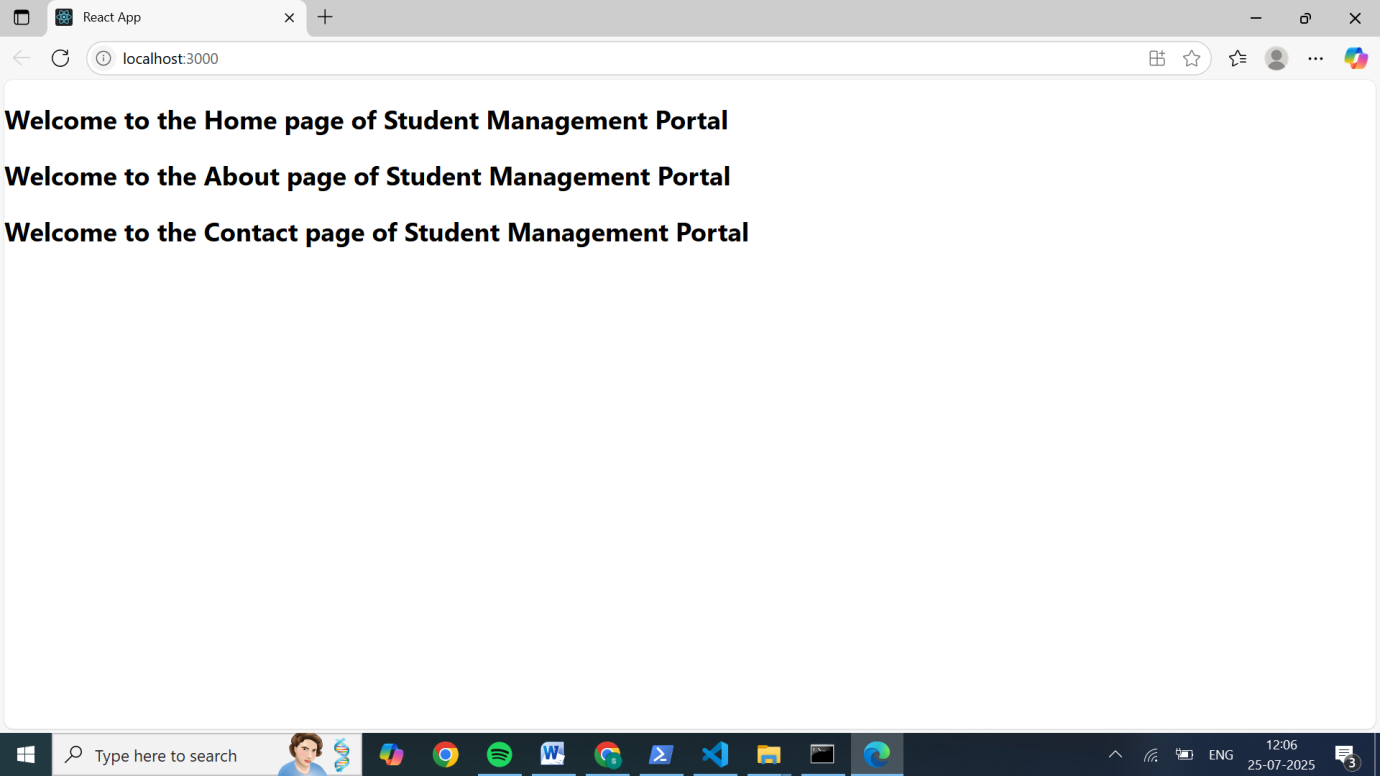
  return <h2>Welcome to the Contact page of Student Management Portal</h2>;

}

export default Contact;



OUTPUT:



**HANDS ON\_03**

**Objectives**

* Explain React components
* Identify the differences between components and JavaScript functions
* Identify the types of components
* Explain class component
* Explain function component
* Define component constructor
* Define render() function

In this hands-on lab, you will learn how to:

* Create a function component
* Apply style to components
* Render a component

## **Prerequisites**

The following is required to complete this hands-on lab:

* Node.js
* NPM
* Visual Studio Code

## **Notes**

Estimated time to complete this lab: **30 minutes.**

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.Type the following code in CalculateScore.js





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



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



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:

CODES:

Calculator.js:

import React from 'react';

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

function CalculateScore({ name, school, total, goal }) {

  const average = total / goal;

  const avgFormatted = average.toFixed(2);

  return (

    <div className="formalstyle">

      <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} Marks</span>

      </div>

      <div className="Score">

        <b>Average Score: </b><span>{avgFormatted}</span>

      </div>

    </div>

  );

}

export default CalculateScore;

mystyle.css:

.Name  {

  font-weight: 300;

  color: blue;

}

.School {

  color: crimson;

}

.Total {

  color: darkmagenta;

}

.Score {

  color: forestgreen;

}

.formalstyle {

  text-align: center;

  font-size: large;

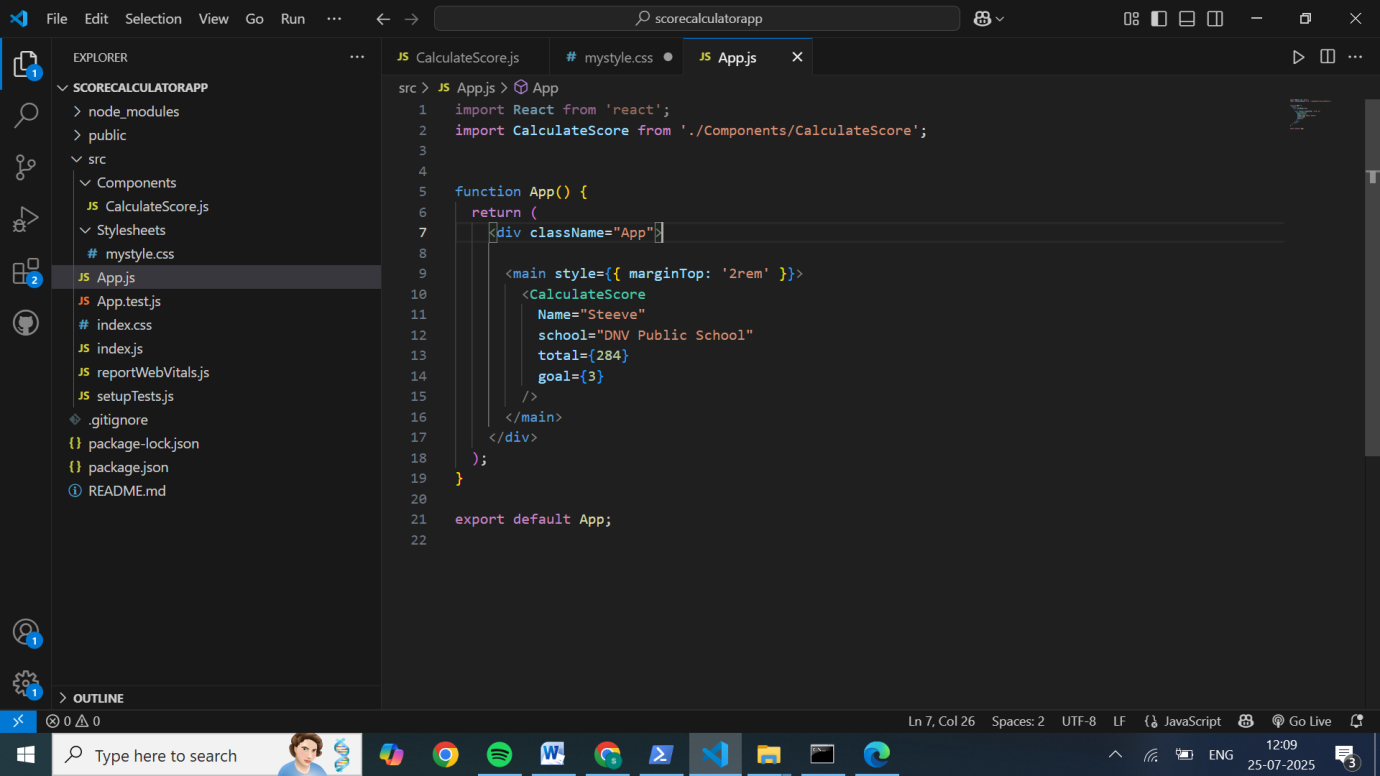
  margin: 16px;

  padding: 8px;

  border: 1px solid #aaa;

  background-color: #f5f5f5;

}



OUTPUT:

## **C:\Users\HOME\OneDrive\Pictures\Screenshots\Screenshot (50).png**

## **HANDS ON\_04**

**Objectives**

* Explain the need and Benefits of component life cycle
* Identify various life cycle hook methods
* List the sequence of steps in rendering a component

In this hands-on lab, you will learn how to:

* Implement componentDidMount() hook
* Implementing componentDidCatch() life cycle hook.

## **Prerequisites**

The following is required to complete this hands-on lab:

* Node.js
* NPM
* Visual Studio Code

## **Notes**

Estimated time to complete this lab: **60 minutes.**

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



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.

### Purpose & Benefits of the Component Lifecycle

React components undergo a structured lifecycle—Mounting, Updating, Unmounting, and Error Handling. This structured approach offers several advantages:

* **Predictable Stages**: Each phase allows developers to place logic at appropriate points, ensuring organized and maintainable code.
* **Side Effect Management**: Methods like componentDidMount() and componentWillUnmount() help in handling tasks such as data fetching and cleanup, ensuring memory safety.
* **Performance Optimization**: The shouldComponentUpdate() method enables developers to prevent unnecessary re-renders, enhancing performance.
* **Error Handling**: With methods like componentDidCatch(), components can gracefully handle errors and display fallback UIs, preventing the entire app from crashing.

### 🔄 Lifecycle Hook Methods in Class Components

React class components provide several lifecycle methods corresponding to different phases:

* **Mounting**: When the component is being created and inserted into the DOM.
  + constructor(): Initializes state and binds methods.
  + render(): Returns JSX to render the component.
  + componentDidMount(): Invoked after the component is mounted; ideal for data fetching.
* **Updating**: Occurs when a component's state or props change.
  + shouldComponentUpdate(): Determines if a re-render is necessary.
  + render(): Returns updated JSX.
  + componentDidUpdate(): Invoked after the component updates; useful for responding to prop or state changes.
* **Unmounting**: When the component is being removed from the DOM.
  + componentWillUnmount(): Invoked before the component is unmounted; used for cleanup tasks.
* **Error Handling**: Allows components to handle errors gracefully.
  + componentDidCatch(): Catches errors in child components and displays a fallback UI.

### 🔁 Sequence of Steps in Rendering a Class Component

Understanding the sequence of lifecycle methods helps in managing component behavior:

* **Mounting**:
  + constructor() → render() → componentDidMount()
  + The component is initialized, rendered to the DOM, and then side-effects run after mounting.
* **Updating**:
  + shouldComponentUpdate() → render() → componentDidUpdate()
  + Ensures only necessary updates run and allows for responding to prop or state changes.
* **Unmounting**:
  + componentWillUnmount()
  + The component is removed from the DOM, and cleanup tasks are performed.
* **Error Handling**:
  + componentDidCatch()
  + Catches errors in child components and displays a fallback UI.

Post.js code:

// src/Post.js

import React, { Component } from 'react';

class Post extends Component {

  render() {

    return (

      <div>

        <h2>{this.props.title}</h2>

        <p>{this.props.body}</p>

      </div>

    );

  }

}

export default Post;

posts.js code:

import React, { Component } from 'react';

import post from './Post';  // lowercase import retained

class Posts extends Component {

  constructor(props) {

    super(props);

    this.state = {

      posts: [],

      error: null,

    };

  }

  loadPosts = () => {

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

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

      .then((data) => {

        console.log(data);

        this.setState({ posts: data });

      })

      .catch((error) => {

        console.error(error);

        this.setState({ error: 'Failed to load posts' });

      });

  };

  componentDidMount() {

    this.loadPosts();

  }

  componentDidCatch(error, info) {

    this.setState({ error: 'Something went wrong!' });

  }

  render() {

    const { posts, error } = this.state;

    if (error) {

      return <h2>{error}</h2>;

    }

    // Rename lowercase 'post' import to a capitalized local variable

    const PostComponent = post;

    return (

      <div>

        <h1>Blog Posts</h1>

        {posts.map((p) => (

          <PostComponent key={p.id} title={p.title} body={p.body} />

        ))}

      </div>

    );

  }

}

export default Posts;

app.js:

// src/App.js

import React from 'react';

import Posts from './Posts';

function App() {

  return (

    <div className="App">

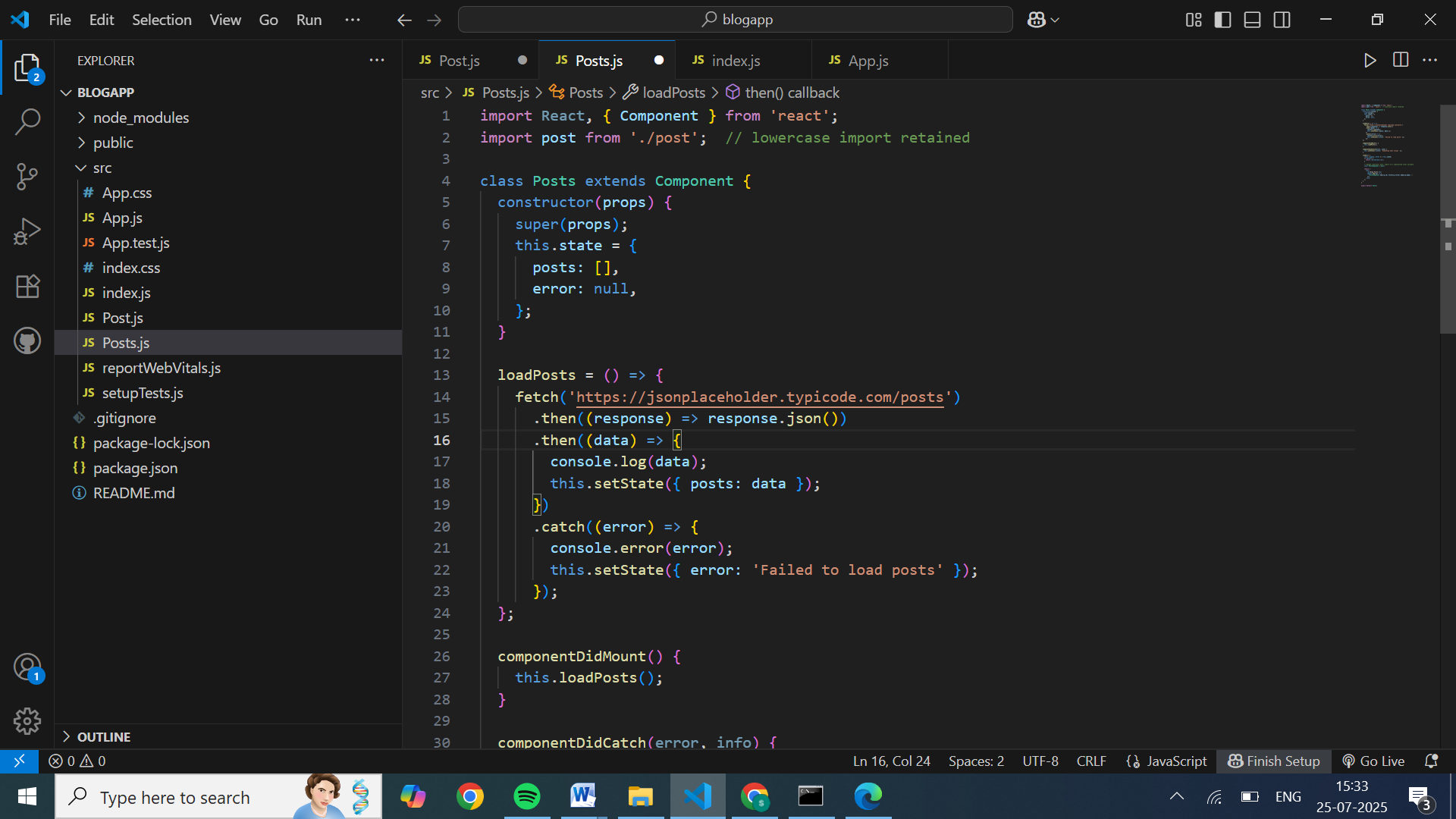
      <Posts />

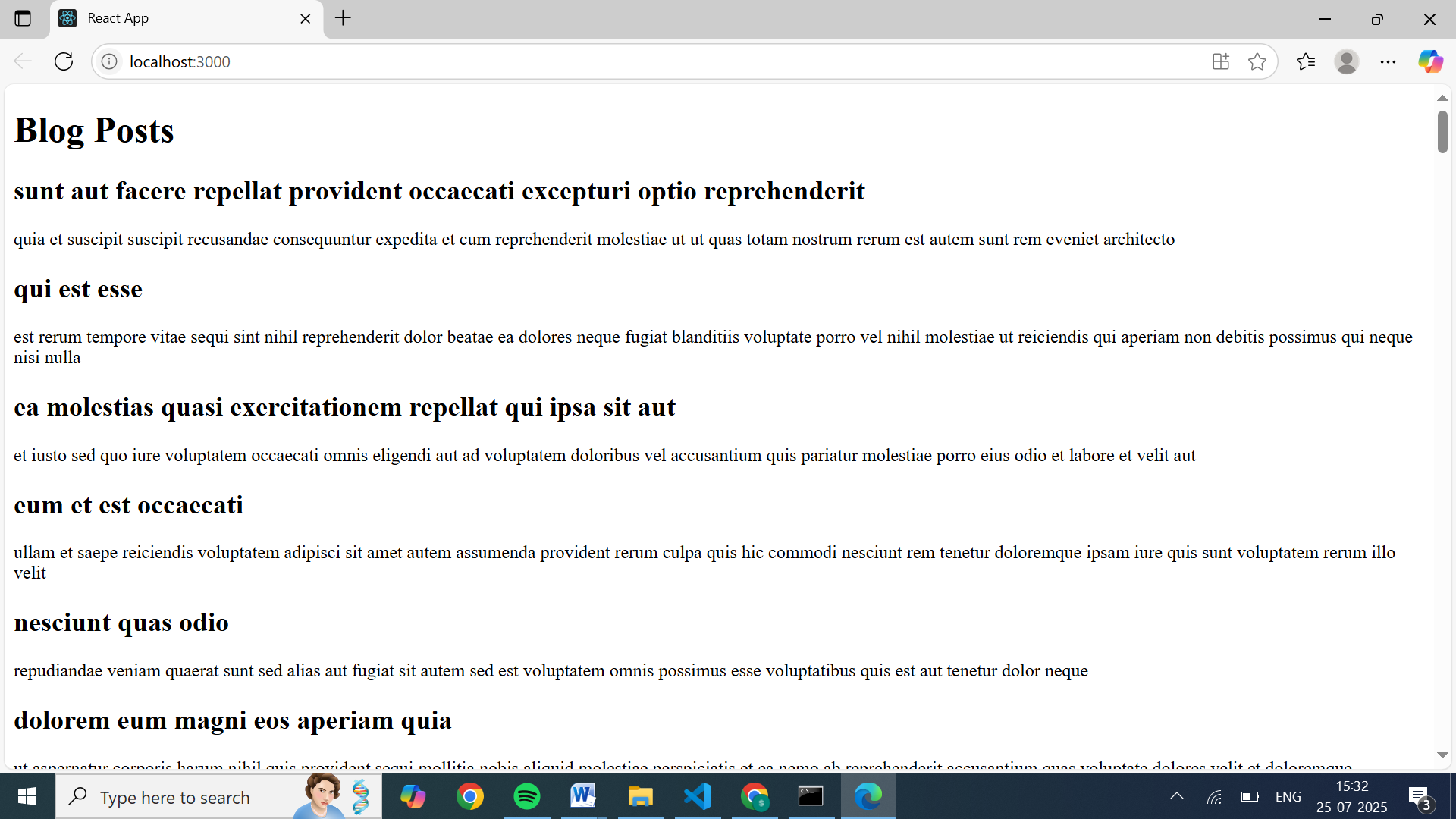
    </div>

  );

}

export default App;





## **HANDS ON\_05**

**Objectives**

* Understanding the need for styling react component
* Working with CSS Module and inline styles

In this hands-on lab, you will learn how to:

* Style a react component
* Define styles using the CSS Module
* Apply styles to components using className and style properties

## **Prerequisites**

The following is required to complete this hands-on lab:

* Node.js
* NPM
* Visual Studio Code

## **Notes**

Estimated time to complete this lab: **30 minutes.**

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

CODES:  
cohortDetails.js

import React from 'react';

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

const cohorts = [

  {

    id: 1,

    name: 'INTADMDF10 - .NET FSD',

    startDate: '22-Feb-2022',

    status: 'Scheduled',

    coach: 'Aathma',

    trainer: 'Jojo Jose'

  },

  {

    id: 2,

    name: 'ADM21JF014 - Java FSD',

    startDate: '10-Sep-2021',

    status: 'Ongoing',

    coach: 'Apoorv',

    trainer: 'Elisa Smith'

  },

  {

    id: 3,

    name: 'CDBJF21025 - Java FSD',

    startDate: '24-Dec-2021',

    status: 'Ongoing',

    coach: 'Aathma',

    trainer: 'John Doe'

  }

];

function CohortDetails() {

  return (

    <div className={styles.container}>

      <h2>Cohorts Details</h2>

      {cohorts.map(cohort => (

        <div key={cohort.id} className={styles.box}>

          <h3

            style={{

              color: cohort.status === 'Ongoing' ? 'green' :

                     cohort.status === 'Scheduled' ? 'blue' : 'black'

            }}

          >

            {cohort.name}

          </h3>

          <dl>

            <dt>Started On</dt>

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

            <dt>Current Status</dt>

            <dd>{cohort.status}</dd>

            <dt>Coach</dt>

            <dd>{cohort.coach}</dd>

            <dt>Trainer</dt>

            <dd>{cohort.trainer}</dd>

          </dl>

        </div>

      ))}

    </div>

  );

}

export default CohortDetails;

cohertDetails.module.css

.box {

  width: 300px;

  display: inline-block;

  margin: 10px;

  padding: 10px 20px;

  border: 1px solid black;

  border-radius: 10px;

  vertical-align: top;

}

dt {

  font-weight: 500;

}

APP.js:

import React from 'react';

import CohortDetails from './CohortDetails';

function App() {

  return (

    <div className="App">

      <CohortDetails />

    </div>

  );

}

export default App;

