**Student Name:** KISHORE S

**Registration No:** 22CSR100

**Course/Batch:** KONGU ENGINEERING COLLEGE (B.E COMPUTER SCIENCE AND ENGINEERING)

**EXERCISE 1: CREATING REACT APP**

**Introduction:**

This project marks the beginning of a journey into React, a powerful JavaScript library for building dynamic user interfaces. The example demonstrates a simple React application that displays a welcome message on the screen.

**Objective:**

* To understand the structure of a basic React application.
* To learn how components work in React.
* To explore JSX (JavaScript XML) syntax used in React.
* To successfully run and view a simple React component in the browser.

**Implementation Breakdown:**

**Step 1:**

Install Node.js and npm

Download and install Node.js from https://nodejs.org. npm (Node Package Manager) comes bundled with it.

**Step 2: Installing a new React App**  
Open the terminal and run:



**Step 2: Create a new React App**Open the terminal and run:

npx create-react-app my-first-react-app



**Step 3: Modify App.js to include custom content**  
Replace the content of src/App.js with the following:

import './App.css';

function App() {

return (

<div className="App">

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

</div>

);

}

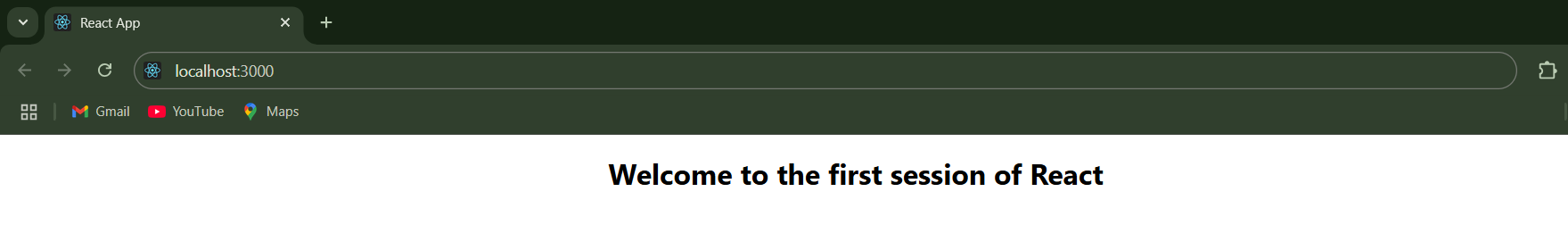
export default App;

**Step 4: Run the React App**  
Start the development server with:



The app will open in your default browser at <http://localhost:3000>.

**Output:**

****

**Conclusion:**

This exercise helps in getting started with React by setting up the development environment and creating a simple component. It builds the foundation for developing more complex and interactive user interfaces using React in future sessions.

**EXERCISE 2: STUDENT MANAGEMENT PORTAL**

**Introduction:**

This mini project introduces the concept of class components in React by building a simple Student Management Portal. The portal includes three separate components: Home, About, and Contact, each rendering specific information about the portal.

**Objective:**

* To understand how to create and use class-based components in React.
* To learn how to import and render multiple components in a parent component (App.js).
* To practice organizing React files using a Components folder.
* To apply basic CSS for layout and styling.

**Implementation Breakdown:**

**Step 1: Create a React App**If not already created, open the terminal and run:

****

****

**Step 2: Create Component Files**Inside src, make a new folder named Components. Then, create the following files inside it:

* Home.js
* About.js
* Contact.js

**Step 3: Write Class Components**  
In each file,

**Home.js**

import React from 'react';

class Home extends React.Component {

render() {

return (

<div>

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

</div>

);

}

}

export default Home;

**About.js**

import React from 'react';

class About extends React.Component {

render() {

return (

<div>

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

</div>

);

}

}

export default About;

**Contact.js**

import React from 'react';

class Contact extends React.Component {

render() {

return (

<div>

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

</div>

);

}

}

export default Contact;

**Step 4: Modify App.js**  
Import the components and render them inside the App component:

import './App.css';

import Home from './Components/Home';

import About from './Components/About';

import Contact from './Components/Contact';

function App() {

return (

<div className="Container">

<Home />

<About />

<Contact />

</div>

);

}

export default App;

**Step 5: Style the App using App.css**

body {

font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

background-color: #f4f6f8;

margin: 0;

padding: 0;

}

.ComponentsContainer {

display: flex;

flex-direction: column;

align-items: center;

gap: 30px;

}

.Container {

text-align: center;

padding: 40px;

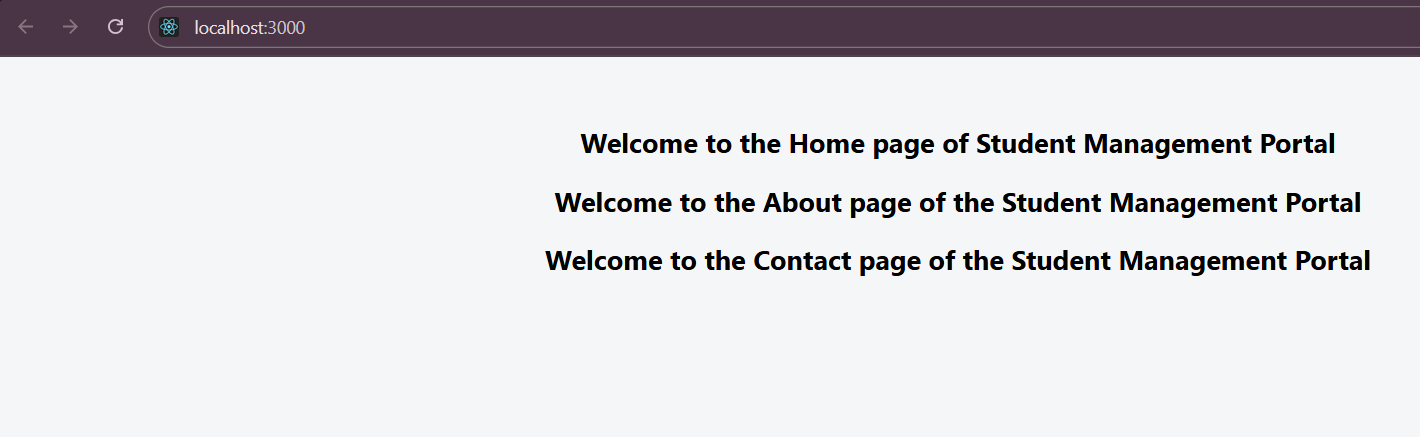
}

**Step 6: Run the App**  
Start the development server:



The app will open in the browser showing Home, About, and Contact messages.

**Output:**

****

**Conclusion:**

This basic React project demonstrates how to structure an application using class-based components and render them inside a parent component. It also shows how to separate concerns by creating individual files for each component, making the project more modular and maintainable.

**EXERCISE 3: STUDENT SCORE CALCULATOR**

**Introduction:**

This project is a simple React application that displays a student’s details and calculates their score percentage based on total marks and a goal. It also demonstrates how to use inline variables and basic arithmetic within JSX and apply external CSS for styling.

**Objective:**

* To understand how to pass and display data using variables in a React component.
* To perform calculations (like percentage) dynamically in JSX.
* To learn how to apply custom CSS styles using an external stylesheet.
* To create modular and readable React code using a separate component (CalculateScore).

**Implementation:**

**Step 1: Create a New React App**

****

Navigate



**Step 2: Create Component and CSS Files**  
Inside src, create:

* **A folder:** Components
* **A file:** CalculateScore.js inside Components
* **A file:** mystyle.css in src for custom styling

**Step 3: Implement the CalculateScore Component**

Components/CalculateScore.js

import './mystyle.css';

function CalculateScore() {

const name = "Steeve";

const school = "DNV Public School";

const total = 284;

const goal = 300;

const average = (total / goal) \* 100;

return (

<div className="container">

<div className="heading">Student Details:</div>

<p className="label name-label">

Name: <span className="name-value">{name}</span>

</p>

<p className="label school-label">

School: <span className="school-value">{school}</span>

</p>

<p className="label total-label">

Total: <span className="total-value">{total} Marks</span>

</p>

<p className="label score-label">

Score: <span className="score-value">{average.toFixed(2)}%</span>

</p>

</div>

);

}

export default CalculateScore;

**Step 4: Update App.js to Use the Component**

**App.js**

import './App.css';

import CalculateScore from './Components/CalculateScore';

function App() {

return (

<div className="App">

<CalculateScore />

</div>

);

}

export default App;

**Step 5: Style the Component**

**mystyle.css**

.container {

text-align: center;

margin-top: 50px;

font-family: Arial, sans-serif;

}

.heading {

color: brown;

font-size: 32px;

font-weight: bold;

margin-bottom: 30px;

}

.label {

font-weight: bold;

margin: 5px 0;

font-size: 20px;

}

.name-label {

color: blue;

}

.name-value {

color: slateblue;

}

.school-label {

color: red;

}

.school-value {

color: hotpink;

}

.total-label {

color: darkred;

}

.total-value {

color: black;

}

.score-label {

color: green;

}

.score-value {

color: lightgreen;

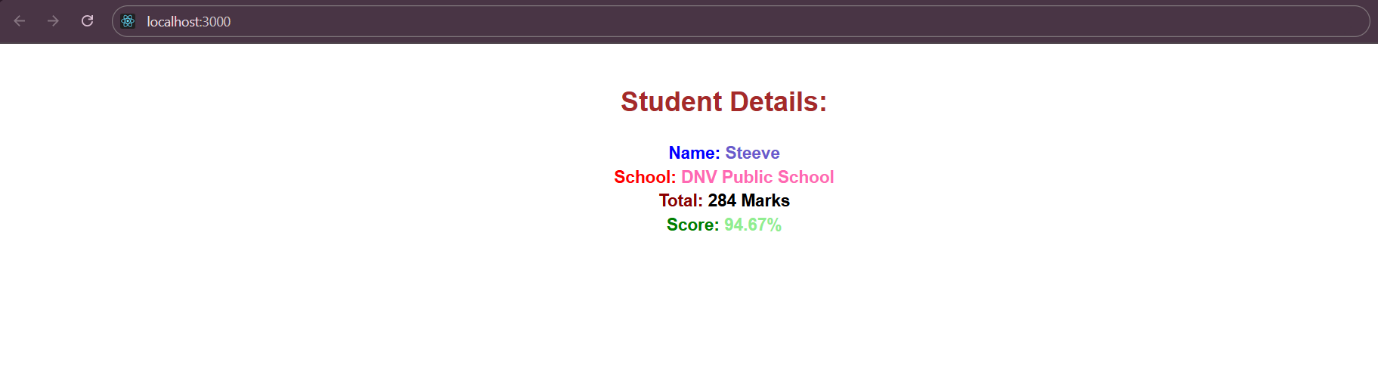
}

**Step 6: Run the App**

****

This will open the browser and display the student’s name, school, total marks, and calculated score percentage with colorful styling.

**Output:**

****

**Conclusion:**

This project demonstrates how to use variables and perform calculations in React, as well as how to style components using an external CSS file. It’s a great exercise to practice JSX, component creation, and CSS integration in a React project.

**EXERCISE 4 : REACT BLOG POSTS APP**

**Introduction:**

This React application fetches and displays blog posts from a public API using class-based components. It demonstrates fetching data with fetch(), managing component state, rendering dynamic lists, and handling errors gracefully.

**Objective:**

* To learn how to fetch data from an API in a React class component.
* To use **componentDidMount()** and **componentDidCatch()** lifecycle methods.
* To dynamically render multiple components **using .map().**
* To apply CSS styling to present the blog posts cleanly.

**Implementation:**

**Step 1: Create a new React app**

****

Navigate



**Step 2: File Structure**

Inside src, create:

* Posts.js
* Post.js
* App.css

**Step 3: Code Setup**

**Post.js**

import React from 'react';

class Post extends React.Component {

render() {

const { title, body } = this.props;

return (

<div className="post-card">

<h3>{title}</h3>

<p>{body}</p>

</div>

);

}

}

export default Post;

**Posts.js**

import React from 'react';

import Post from './Post';

class Posts extends React.Component {

constructor(props) {

super(props);

this.state = {

posts: [],

error: null

};

}

componentDidMount() {

this.loadPosts();

}

loadPosts() {

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

.then(response => {

if (!response.ok) throw new Error('Failed to fetch posts');

return response.json();

})

.then(data => this.setState({ posts: data }))

.catch(error => this.setState({ error }));

}

componentDidCatch(error, info) {

alert('An error occurred: ' + error.message);

}

render() {

const { posts, error } = this.state;

if (error) {

return <h2>Error loading posts.</h2>;

}

return (

<div className="posts-container">

<h2>Blog Posts</h2>

{posts.slice(0, 10).map(post => (

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

))}

</div>

);

}

}

export default Posts;

**App.js**

import './App.css';

import Posts from './Posts';

function App() {

return (

<div className="App">

<Posts />

</div>

);

}

export default App;

**Step 4: Add Styling**

**App.css**

body {

font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

background-color: #f2f2f2;

margin: 0;

padding: 0;

}

.App {

text-align: center;

}

.posts-container {

padding: 30px;

max-width: 800px;

margin: auto;

}

.post-card {

background-color: #ffffff;

border-radius: 8px;

box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1);

padding: 20px;

margin-bottom: 20px;

text-align: left;

}

.post-card h3 {

color: #333333;

margin-bottom: 10px;

}

.post-card p {

color: #555555;

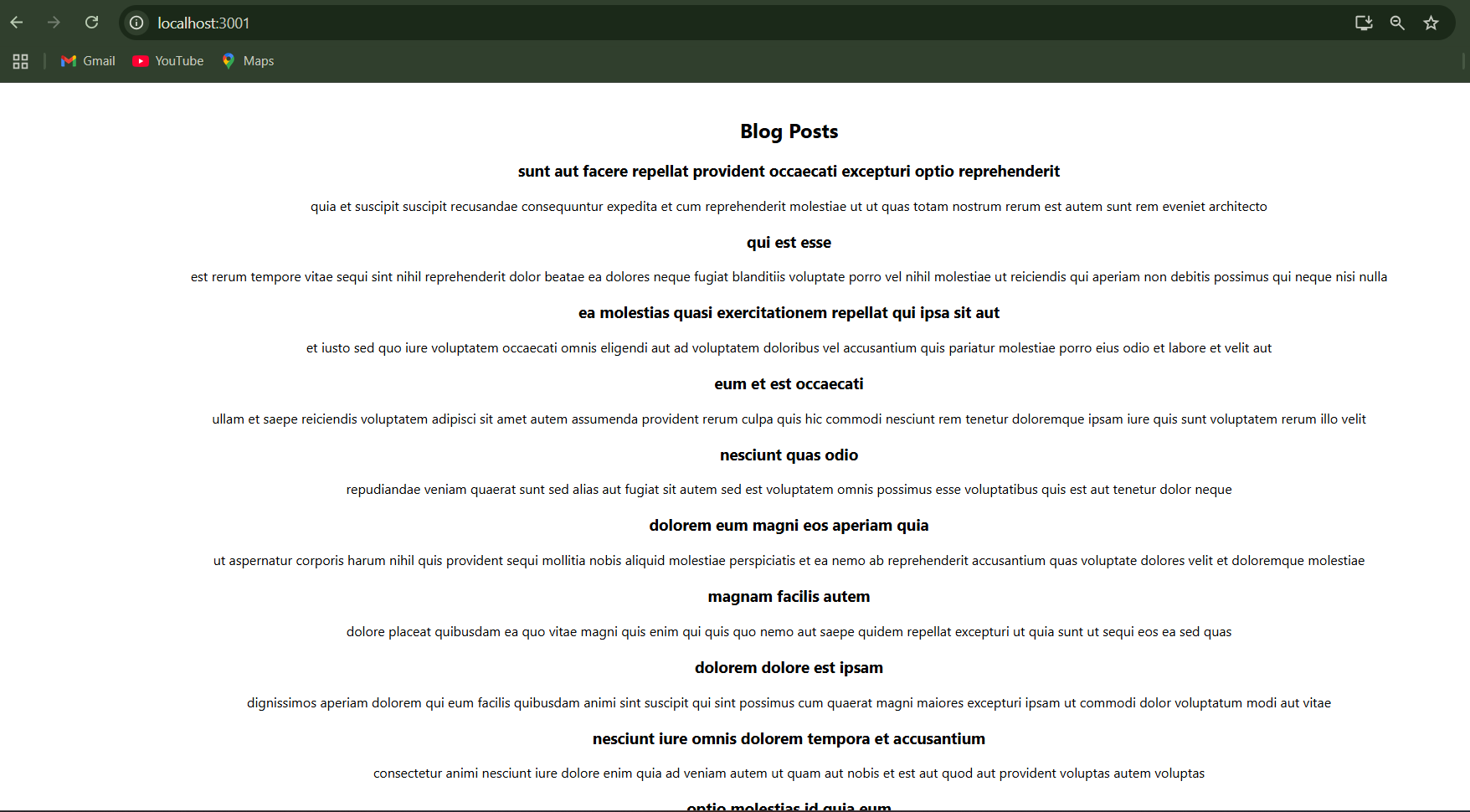
line-height: 1.5;

}

**Step 5: Run the App**

****

**Output:**

****

**Conclusion:**

This project demonstrates how to build a simple yet functional blog viewer using React class components and public APIs. It provides practical experience with data fetching, state management, rendering lists dynamically, and styling React components using CSS.

**EXERCISE 5 : COHORT DETAILS REACT APP**

**Introduction:**

This React app displays detailed information about various learning cohorts in a card format. It uses a class-based model (Cohort), passes data via props to reusable components, and applies **CSS Modules** for scoped styling based on cohort status like Ongoing or Scheduled.

**Objective:**

* To learn how to structure and render an array of custom class objects using React.
* To create a reusable, styled component **(CohortDetails)** that accepts props.
* To dynamically assign styles based on data values (e.g., status).
* To modularize styles using CSS Modules for better maintainability.

**Implementation:**

**Step 1: Extracting the zip file cohorttracker.zip**

**Step 2: Navigating to the folder in VS Code**

****

**Step 3: Reinstalling the packages**

****

**Step 4: Reviewing Cohort.js**

**Cohort.js**

class Cohort {

constructor(cohortCode, startDate, technology, trainerName, coachName, currentStatus) {

this.cohortCode = cohortCode;

this.coachName = coachName;

this.trainerName = trainerName;

this.technology = technology;

this.startDate = startDate;

this.currentStatus = currentStatus;

}

}

const CohortsData = [

new Cohort('INTADMDF10', '22-Feb-2022', '.NET FSD', 'Jojo Jose', 'Aathma', 'Scheduled'),

new Cohort('ADM21JF014', '10-Sep-2021', 'Java FSD', 'Elisa Smith', 'Apoorv', 'Ongoing'),

new Cohort('CDBJF21025', '24-Dec-2021', 'Java FSD', 'John Doe', 'Aathma', 'Ongoing'),

new Cohort('INTADMJF12', '22-Feb-2022', 'Java FSD', 'To Be Assigned', 'Ibrahim', 'Scheduled'),

new Cohort('CDE22JF011', '24-Dec-2021', 'Java FSD', 'Emma Swan', 'Apoorv', 'Ongoing'),

new Cohort('INTADMDF09', '22-Feb-2022', 'Dataware Housing', 'Babjee Rao', 'Aathma', 'Scheduled'),

new Cohort('ADM22DF001', '10-Sep-2021', '.NET FSD', 'Marie Curie', 'Ibrahim', 'Ongoing'),

];

export { Cohort, CohortsData };

**Step 5: Modifying CohortDetails.js**

**CohortDetails.js**

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

function CohortDetails(props) {

const { cohortCode, technology, startDate, currentStatus, coachName, trainerName } = props.cohort;

const titleClass = currentStatus === 'Ongoing' ? styles['title-ongoing'] : styles['title-scheduled'];

return (

<div className={styles.box}>

<h3 className={titleClass}>

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

</h3>

<dl>

<dt>Started On</dt>

<dd>{startDate}</dd>

<dt>Current Status</dt>

<dd>{currentStatus}</dd>

<dt>Coach</dt>

<dd>{coachName}</dd>

<dt>Trainer</dt>

<dd>{trainerName}</dd>

</dl>

</div>

);

}

export default CohortDetails;

**Step 6: Adding styles according to the requirements**

**CohortDetails.module.css**

.box {

width: 300px;

display: inline-block;

margin: 10px;

padding: 10px 20px;

border: 1px solid black;

border-radius: 10px;

background-color: #fff;

box-shadow: 2px 2px 8px rgba(0,0,0,0.1);

}

dt {

font-weight: 500;

margin-top: 10px;

}

dd {

margin: 0 0 10px 0;

}

.title-ongoing {

color: green;

}

.title-scheduled {

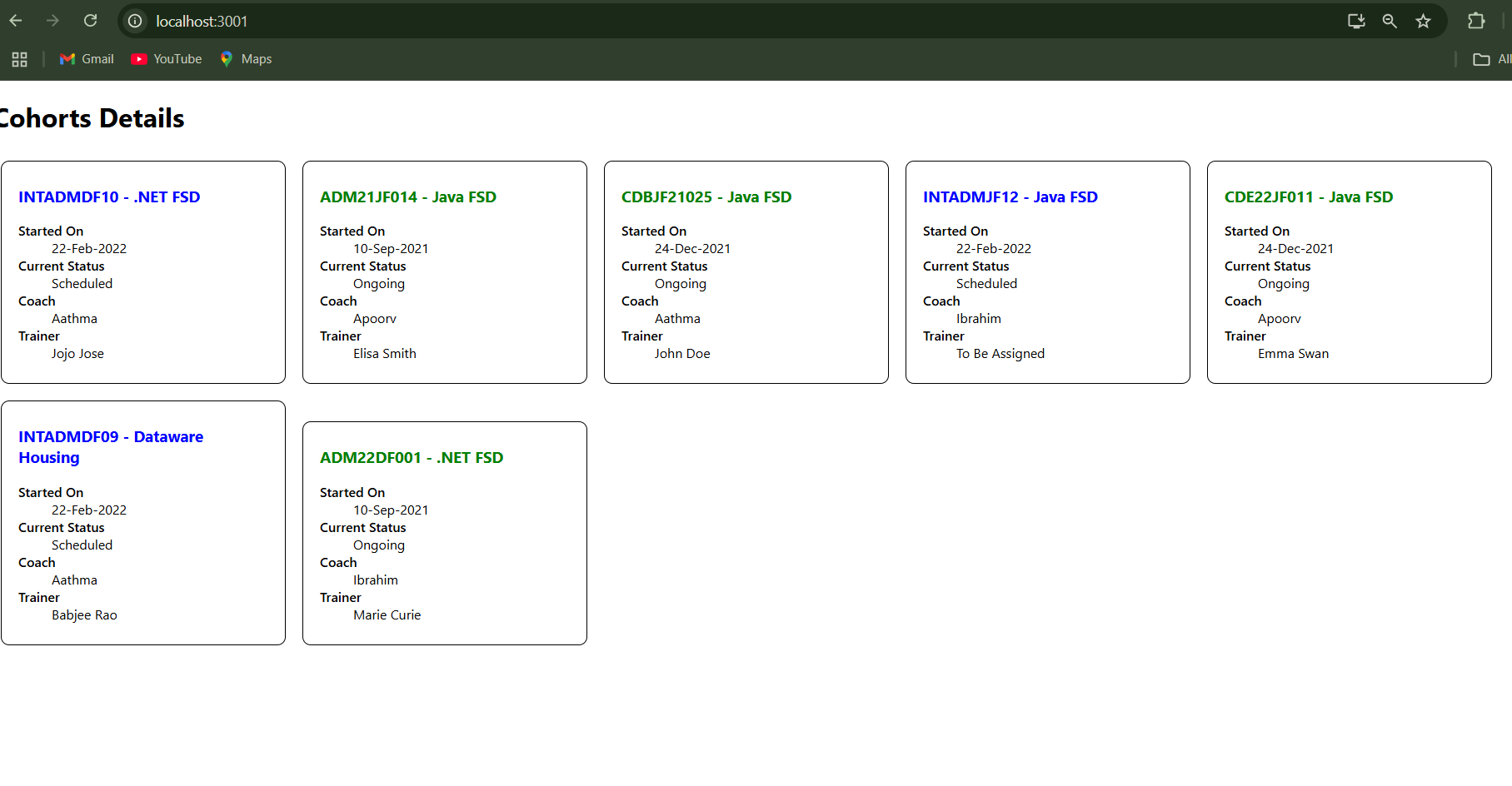
color: blue;

}

**Step 8: Run the App**

****

**Output:**

****

**Conclusion:**

This app showcases how to render detailed data from a model class in React using functional components, props, and CSS Modules for scoped and maintainable styling. It also introduces dynamic class selection based on component state (currentStatus), reinforcing conditional rendering and styling patterns in modern React.