**Student Name:** KARTHYAYENI P

**Registration No:** 22CSR089

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

**EXERCISE 1: CREATING REACT APP**

**Introduction:**

React is a popular JavaScript library developed by Facebook for building user interfaces, especially single-page applications. It enables developers to create reusable UI components, making web development efficient and scalable.This simple React project demonstrates the basic structure of a React application by displaying a welcome message.

**Objective:**

* Understand how to set up a basic React application.
* Learn the structure of a React component.
* Render a message on the browser using JSX.
* Gain hands-on experience with the **create-react-app** tool.

**Implementation Breakdown:**

**Step 1: Installing a React App**



**Step 2: Creating a React App**

npx create-react-app myfirstreact

**Step 3: Navigate to the Project Folder**

****

**Step 4: Open the Project in Your Code Editor**

code .

**Step 5: Modify the App Component**

import './App.css';

function App() {

return (

<div className="App">

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

</div>

);

}

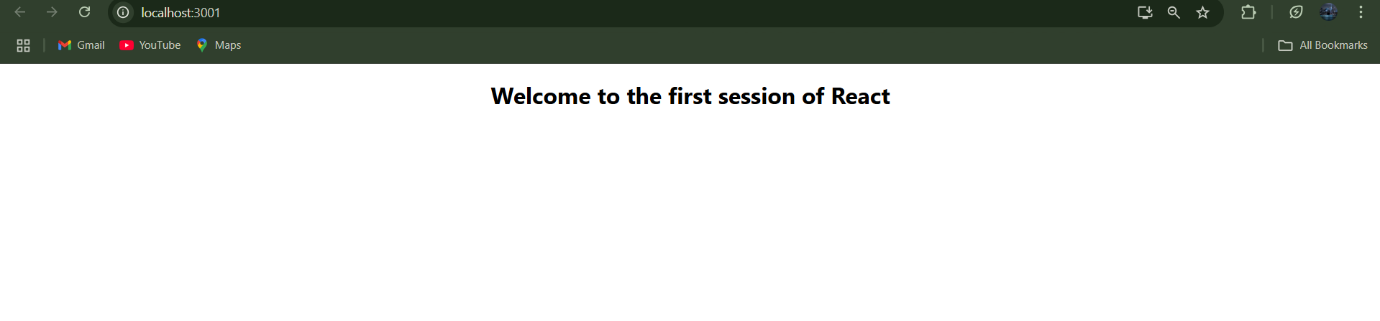
export default App;

**Step 6: Run the Application**

****

This will open your browser at http://localhost:3000, and you’ll see the welcome message rendered.

**Output:**

****

**Conclusion:**

In this session, we successfully created a basic React application using create-react-app. We explored the structure of a React component and rendered a simple message to the screen. This foundational knowledge is crucial for building more complex and interactive web applications using React in future sessions.

**EXERCISE 2: STUDENT MANAGEMENT PORTAL**

**Introduction:**

This project demonstrates a basic React application that includes multiple components: **Home**, **About**, and **Contact**. It is a simple starting point for building a **Student Management Portal**, showcasing how components work together in a React app.

**Objective:**

* Understand how to create and use class-based components in React.
* Learn how to structure a multi-page interface using React components.
* Display different sections (Home, About, Contact) within a single page.
* Apply basic styling using CSS.

**Implementation Breakdown:**

**Step 1: Create a React App**

Open the terminal and create a new React app:



Navigate



**Step 2: Create Component Files**

Inside the src folder, create a Components folder and add three files:

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

**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 3: Update App Component**

Open App.js and replace it with:

import './App.css';

import About from './Components/About';

import Contact from './Components/Contact';

import Home from './Components/Home';

function App() {

return (

<div className="Container">

<Home />

<About />

<Contact />

</div>

);

}

export default App;

**Step 4: Add CSS Styling**

**App.css**

body {

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

background-color: #f4f6f8;

margin: 0;

padding: 0;

}

.Container {

text-align: center;

padding: 40px;

}

.ComponentsContainer {

display: flex;

flex-direction: column;

align-items: center;

gap: 30px;

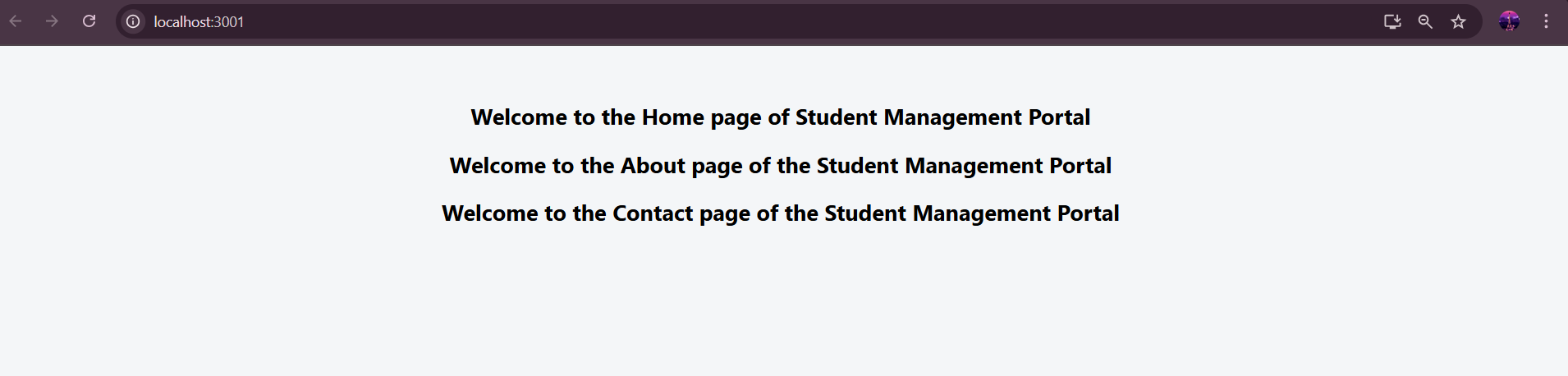
}

**Step 5: Run the App**

****

It will show all three components rendered on the same page, each displaying a welcome message for different sections of the Student Management Portal.

**Output:**

****

**Conclusion:**

In conclusion, this project introduced the basics of React using class-based components to build a simple Student Management Portal. It helped understand component structure, reusability, and basic styling. This forms the foundation for building more advanced React features.

**EXERCISE 3: STUDENT SCORE CALCULATOR**

**Introduction:**

This project is a simple React application that displays student details and calculates their performance percentage based on total marks and a goal. It demonstrates the use of functional components and CSS styling for a clean UI.

**Objective:**

* To create a functional component that calculates and displays a student’s score.
* To apply custom styles using an external CSS file.
* To practice JSX syntax and basic arithmetic operations in React.

**Implementation:**

**Step 1: Create a React App**

****

Navigate

****

**Step 2: Create Component and CSS File**

Inside the src/Components/ folder:

**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;

**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 3: Use the Component in App.js**

import './App.css';

import CalculateScore from './Components/CalculateScore';

function App() {

return (

<div className="App">

<CalculateScore />

</div>

);

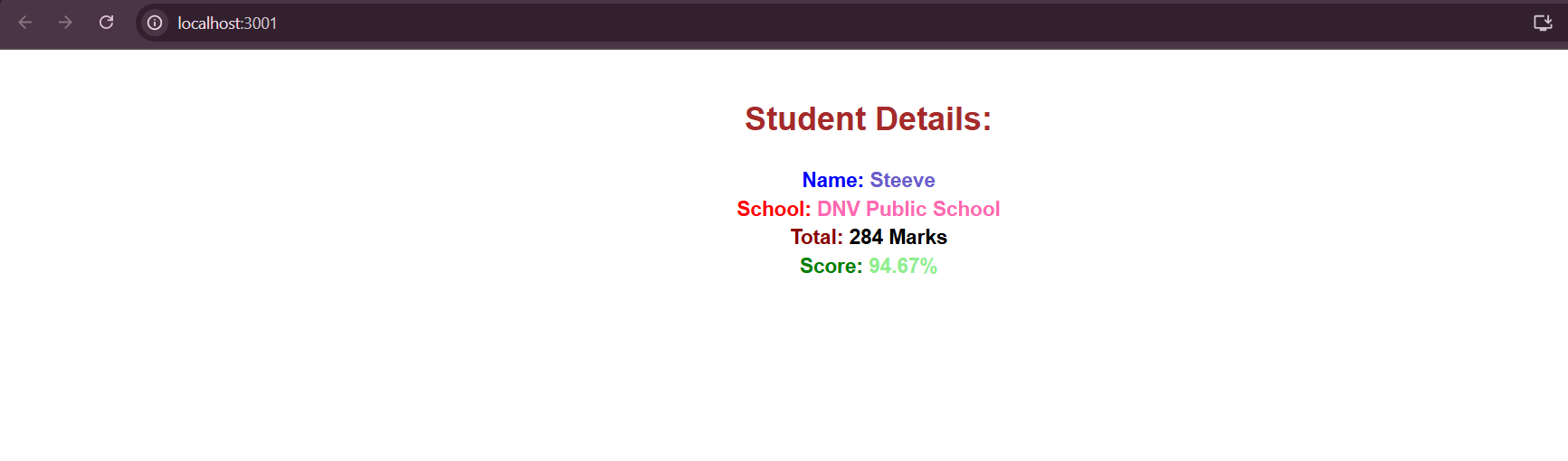
}

export default App;

**Step 4: Run the App**

****

**Output:**

****

**Conclusion:**

This project helped understand how to build a functional component in React and apply dynamic calculations using JSX. It also reinforced using external CSS for component styling and building a visually appealing UI.

**EXERCISE 4 : REACT BLOG POSTS APP**

**Introduction:**

This React application demonstrates how to fetch and display data from an external API using class-based components. It uses fetch() to load blog posts from the JSONPlaceholder API and renders each post using reusable components.

**Objective:**

* To fetch data from a public API using fetch().
* To manage component state and lifecycle using class-based components.
* To display blog posts dynamically using reusable components (Post).
* To implement basic error handling with componentDidCatch.

**Implementation:**

**Step 1: Create a React App**



Navigate



**Step 2: Create Components**

**Post,js**

import React from 'react';

class Post extends React.Component {

render() {

const { title, body } = this.props;

return (

<div style={{ marginBottom: '20px' }}>

<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 style={{ padding: '20px' }}>

<h2>Blog Posts</h2>

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

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

))}

</div>

);

}

}

export default Posts;

**Step 3: Update App.js**

import './App.css';

import Posts from './Posts';

function App() {

return (

<div className="App">

<Posts />

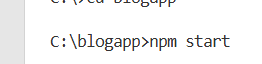
</div>

);

}

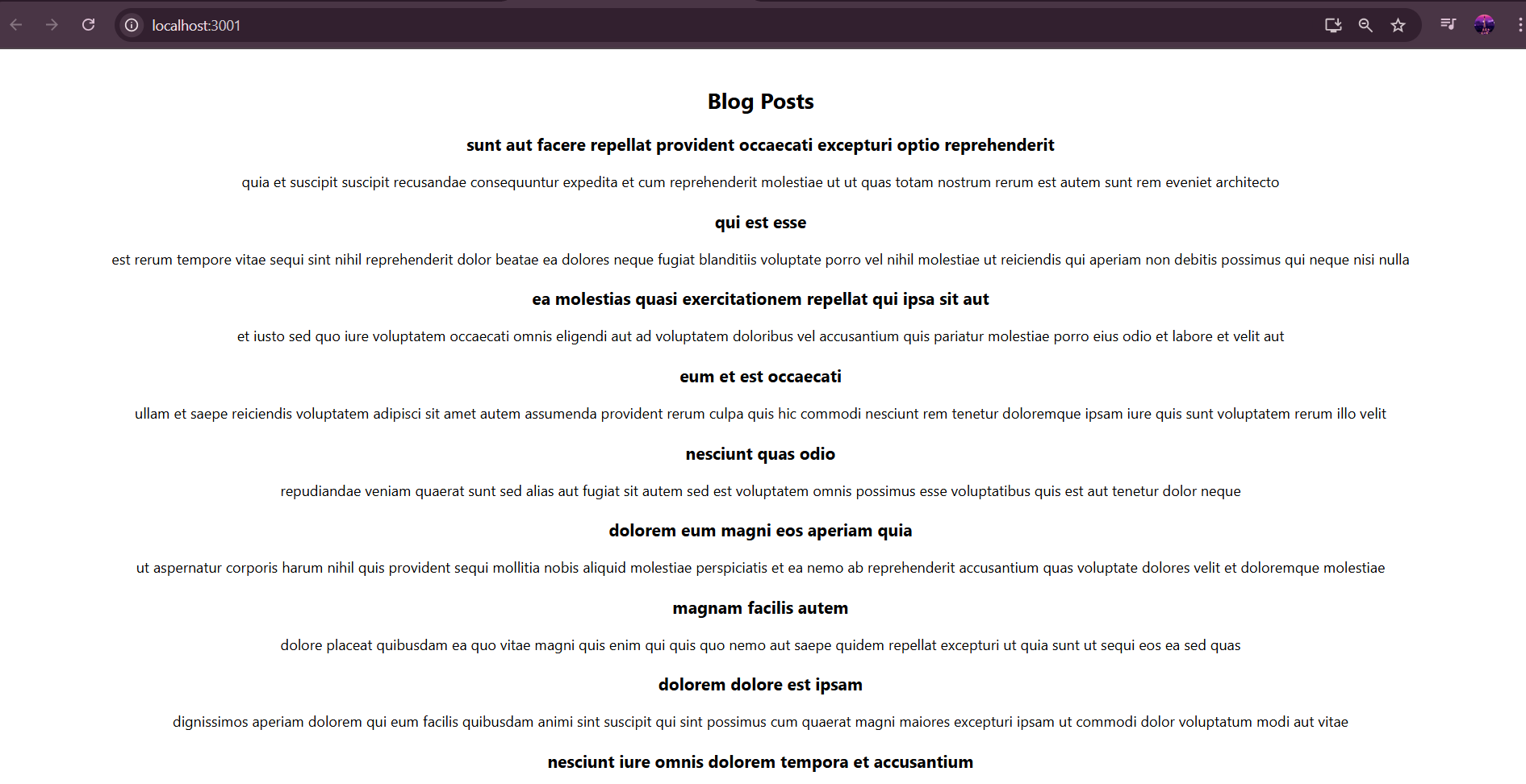
export default App;

**Step 4: Run the App**



You will see the first 10 blog posts loaded dynamically from the API and rendered to the screen.

**Output:**

****

**Conclusion:**

This project helped demonstrate fetching data from an external API and rendering it dynamically using class-based components in React. It also introduced basic error handling and reusable component design, forming the foundation for building dynamic data-driven UIs.

**EXERCISE 5 : COHORT DETAILS REACT APP**

**Introduction:**

This React application displays detailed information about training cohorts, including their technology, start date, coach, trainer, and current status. It showcases how to work with custom classes, modular CSS (.module.css), and data rendering using props and object-oriented JavaScript.

**Objective:**

* To dynamically display cohort data using React components and props.
* To implement conditional styling based on cohort status (e.g., Ongoing or Scheduled).
* To apply modular and reusable styles using CSS modules.
* To use class-based data modeling for better organization.

**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: Create the CSS Module**

**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;

}

.title-ongoing {

color: green;

}

.title-scheduled {

color: blue;

}

**Step 5: Create the CohortDetails Component**

**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: Define the Data Structure**

**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 7: Render the Cohorts in App.js**

**App.js**

import { CohortsData } from './Cohort';

import CohortDetails from './CohortDetails';

function App() {

return (

<div className="container">

<h1 className="title">Cohorts Details</h1>

<div>

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

</div>

</div>

);

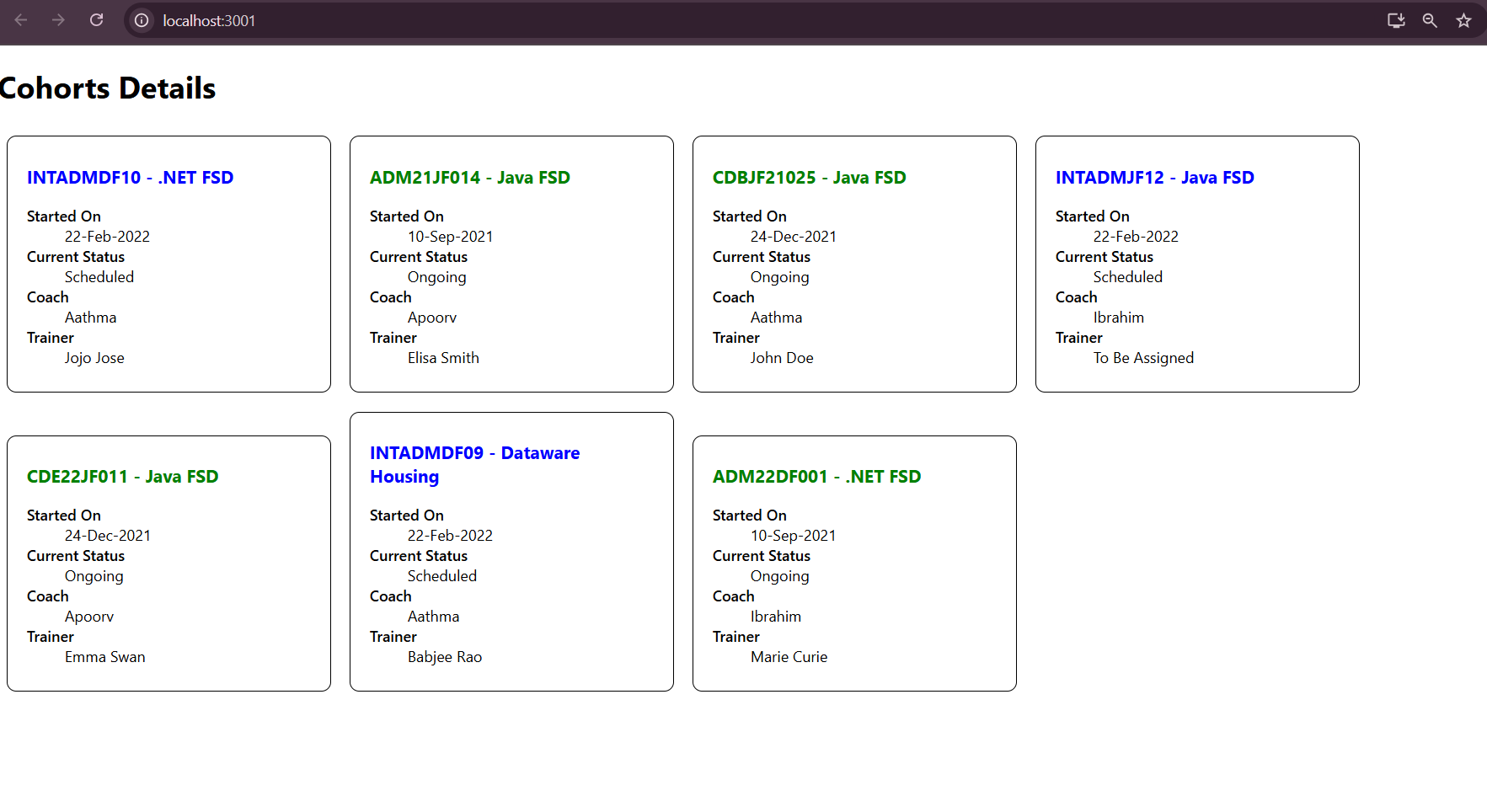
}

export default App;

**Step 8: Run the App**

****

**Output:**

****

**Conclusion:**

This project demonstrates how to use React components with props and class-based data models to dynamically render and style a list of training cohorts. It also shows how to apply conditional styling using CSS modules based on cohort status, making the UI clean and informative.