**ReactJS – Hands On – 1**

**1. Define SPA and its Benefits**

* SPA (Single Page Application) is a web application that dynamically rewrites the current page rather than loading entire new pages from the server.
* Benefits of SPA:
  + Faster user experience (no full-page reloads)
  + Smooth navigation
  + Reduced server load
  + Easier caching
  + Better performance after the initial load

**2. Define React and Identify Its Working**

* React is a JavaScript library for building user interfaces, especially SPAs.
* It works by breaking the UI into components and uses a virtual DOM to efficiently update and render only the components that change.

**3. Differences between SPA and MPA**

|  |  |  |
| --- | --- | --- |
| **Feature** | **SPA (Single Page App)** | **MPA (Multi Page App)** |
| **Page Load** | One initial load | Multiple full-page loads |
| **Speed** | Fast after first load | Slower due to multiple requests |
| **Server Requests** | Fewer | More frequent |
| **Development** | Ideal for dynamic platforms | Ideal for static content-heavy sites |
| **Examples** | Gmail, Facebook | Amazon, LinkedIn |

**4. Pros and Cons of SPA**

Pros:

* Faster interactions
* Better user experience
* Efficient data loading (via API)

Cons:

* SEO challenges
* Initial load may be heavy
* Requires JavaScript to function

**5. Explain About React**

React:

* Developed by Facebook
* Uses JSX (JavaScript XML) to design UI
* Encourages reusable components
* Has a large ecosystem (React Router, Redux, etc.)

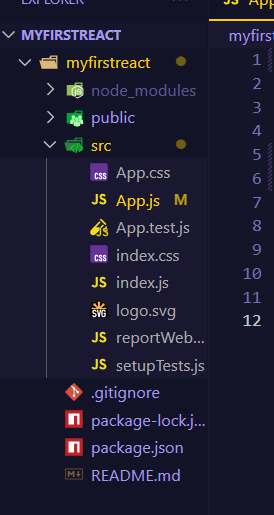
**6. Define Virtual DOM**

* Virtual DOM is a lightweight copy of the actual DOM.
* React compares the virtual DOM with a previous version (diffing) and updates only what has changed in the real DOM (called reconciliation), improving performance.

**7. Features of React**

* Component-Based Architecture
* Virtual DOM for fast rendering
* Unidirectional Data Flow
* JSX syntax
* Reusable Components
* Fast Performance
* Rich Developer Tools

**ReactJS Setup**

****

**App.js**

import React *from* 'react';

function App() {

  return (

    <div>

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

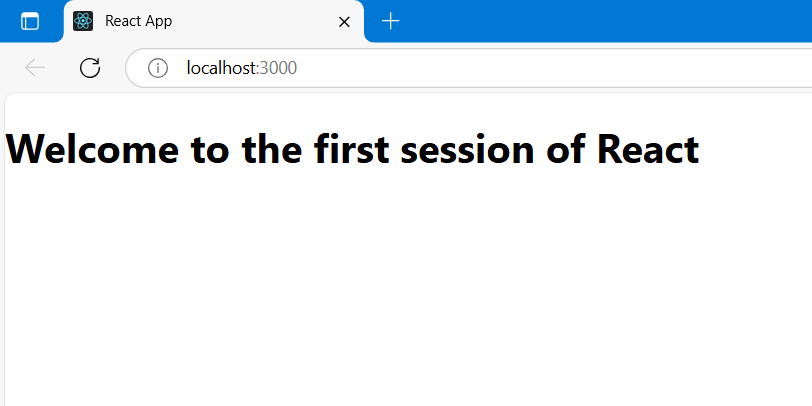
    </div>

  );

}

export default App;

**Output**

****

**ReactJS – Hands On – 2**

**1. React Components Explanation**

A React Component is a reusable, self-contained block of code that controls a part of the UI. Just like functions in programming break down logic into manageable parts, components break down the UI into reusable pieces.

**Example:**

When building a Student Management Portal, components could be:

* <Home /> – displays homepage
* <About /> – shows information about the app
* <StudentList /> – displays all students
* <StudentForm /> – form to add or edit students

Each of these is a React component.

**2. Difference between React Components and JavaScript Functions**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Javascript Function** | **React Component** |
| **Purpose** | Performs logic, returns value | Renders UI (HTML via JSX) |
| **Output** | Data or computation | JSX (HTML-like code in JavaScript) |
| **Reusability in UI** | Cannot be directly used in JSX | Used like <ComponentName /> |
| **State Management** | No state handling | Can handle state (useState, this.state) |
| **Lifecycle Management** | Not applicable | Has lifecycle methods/hooks |

React components are special JavaScript functions that return JSX and may manage state, effects, and lifecycle.

**3. Types of React Components**

**React supports two types of components:**

**a) Class Components**

* Uses ES6 class syntax.
* Must extend React.Component.
* Has a render() method that returns JSX.
* Can have state and lifecycle methods.

**b) Function Components**

* Simpler, written as JavaScript functions.
* Before React 16.8, couldn’t manage state.
* Now supports Hooks like useState, useEffect.

**4. What is a Class Component?**

A class component is a React component defined using a class, which:

* Extends the React.Component class
* Contains a render() method to output JSX
* Can maintain its own state
* Can use lifecycle methods (componentDidMount, componentDidUpdate, etc.)

Example:

import React from 'react';

class Home extends React.Component {

render() {

return <h1>Welcome to the Home page</h1>;

}

}

Useful when your component needs to store and manage state, or use lifecycle methods.

**5. What is a Function Component?**

A function component is a simpler and modern way to write components using plain JavaScript functions. With the introduction of React Hooks, function components can now also have state and lifecycle features.

**Example:**

function About() {

return <h1>This is the About page</h1>;

}

Preferred for simplicity and readability, especially with Hooks like useState() and useEffect().

**6. What is a Component Constructor?**

* The constructor() is a special method in class components.
* It is called before the component is mounted.
* It is used to initialize state and bind methods.

**Example:**

class Welcome extends React.Component {

constructor(props) {

super(props);

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

}

render() {

return <h1>Welcome to {this.state.name}</h1>;

}

}

**7. What is the render() Method?**

The render() method is mandatory in class components.  
It tells React what to display on the screen.

* Always returns a single JSX element (can wrap multiple in a <div> or <React.Fragment>).
* Called automatically by React when the component mounts or updates.

**Example:**

render() {

return (

<div>

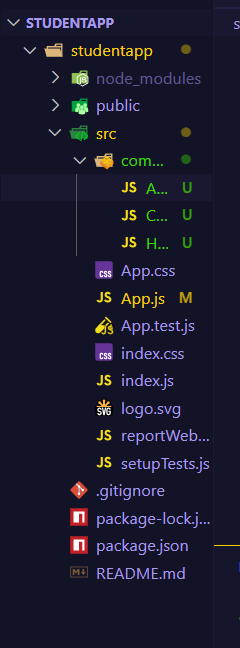
<h1>Hello, React!</h1>

</div>

);

}

**ReactJS Setup**

****

**Home.js**

import React from 'react';

class Home extends React.Component {

  render() {

    return (

      <div>

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

      </div>

    );

  }

}

export default Home;

**About.js**

import React from 'react';

class About extends React.Component {

  render() {

    return (

      <div>

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

      </div>

    );

  }

}

export default About;

**Contact.js**

import React from 'react';

class Contact extends React.Component {

  render() {

    return (

      <div>

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

      </div>

    );

  }

}

export default Contact;

**App.js**

import React from 'react';

import './App.css';

import Home from './components/Home';

import About from './components/About';

import Contact from './components/Contact';

function App() {

  return (

    <div className="App">

      <Home />

      <About />

      <Contact />

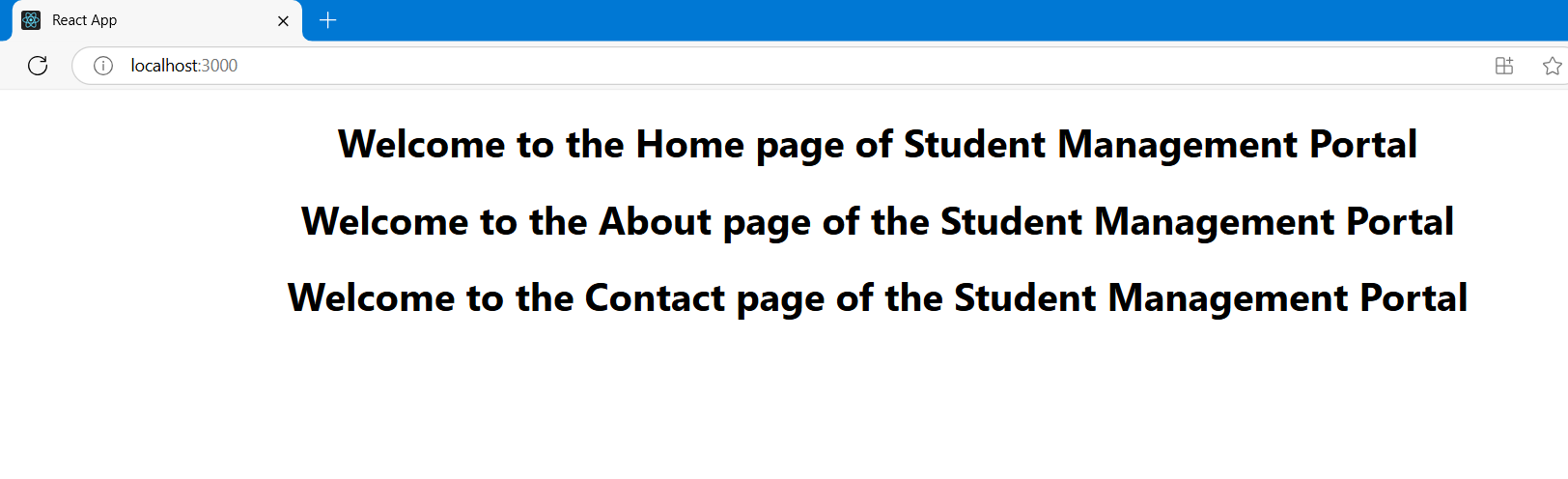
    </div>

  );

}

export default App;

**Output**

****

**ReactJS – Hands On – 3**

**Explain React Components**

React components are the building blocks of any React application. They are reusable, independent pieces of UI that can have their own logic and structure. Think of them as custom HTML tags that can render content dynamically.

**Identify Differences Between Components and JavaScript Functions**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **JavaScript Function** | **React Component (Function)** |
| **Purpose** | Performs logic, returns a value | Returns JSX (UI) |
| **Return Value** | Primitive value or object | JSX (HTML-like structure in JS) |
| **Invocation** | Called like myFunction() | Used like <MyComponent /> |
| **Use of Hooks/State** | Not allowed | Can use React features like useState |

**Identify Types of Components**

React has two main types of components:

1. Function Components
   * Simple JavaScript functions
   * Modern and commonly used
   * Can use React Hooks for state and side effects
2. Class Components
   * Use ES6 class syntax
   * Older approach (still used in legacy projects)
   * Can manage state and lifecycle using constructor, render, componentDidMount, etc.

**Explain Class and Function Components**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Feature** | **Syntax** | **State Management** | **Lifecycle Methods** | **Simplicity** |
| **Class Component** | Uses class keyword | Uses this.state and this.setState() | Has lifecycle methods like componentDidMount | More boilerplate code |
| **Function Component** | Uses function or arrow function | Uses useState hook | Uses useEffect instead | Simpler and cleaner syntax |

**Example – Class Component:**

class Hello extends React.Component {

render() {

return <h1>Hello from Class Component</h1>;

}

}

**Example – Function Component:**

function Hello() {

return <h1>Hello from Function Component</h1>;

}

**Define Component Constructor**

In class components, the constructor is a special method used to:

* Initialize state
* Bind event handlers
* Accept props using super(props)

constructor(props) {

super(props);

this.state = { count: 0 };

}

**Define render() Function**

The render() function in a class component:

* Returns JSX
* Describes what UI should appear

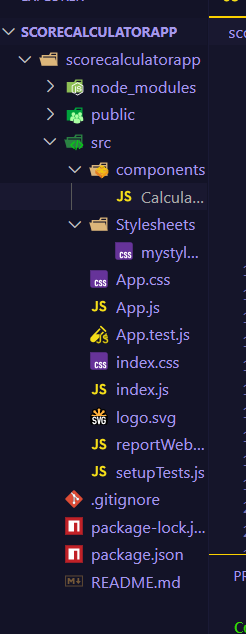
render() {

return <div>Hello</div>;

}

This method is mandatory in class components.

**ReactJS Setup**

****

**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>

);

**mystyle.css**

.Name {

  font-weight: 300;

  color: blue;

}

.School {

  color: crimson;

}

.Total {

  color: darkmagenta;

}

.formatstyle {

  text-align: center;

  font-size: large;

}

.Score {

  color: forestgreen;

}

**App.js**

import React from 'react';

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

function App() {

  return (

    <div>

      <CalculateScore

        Name="Steeve"

        School="DNV Public School"

        total={284}

        goal={3}

      />

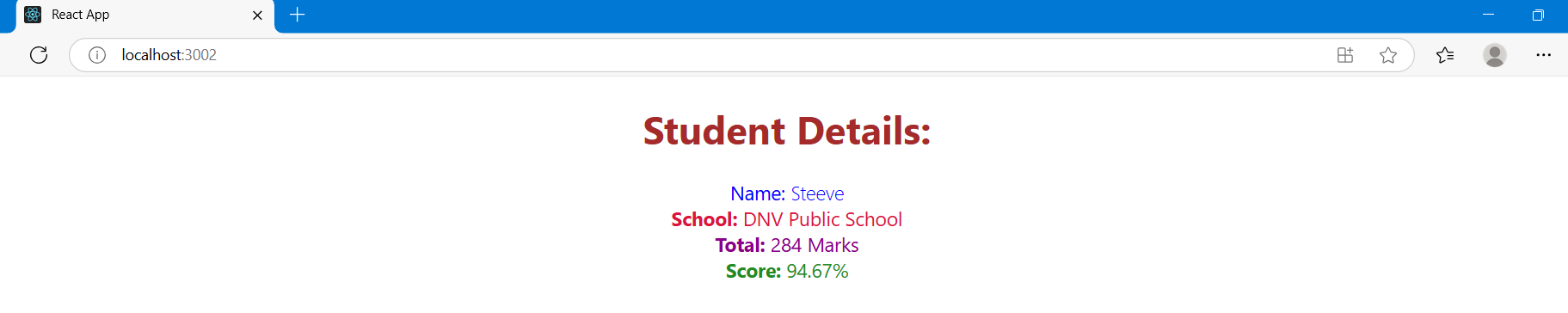
    </div>

  );

}

export default App;

**Output**

****

**ReactJS – Hands On – 4**

**Objective**

**1.Component Lifecycle**

In React, every component goes through a lifecycle: from being created, mounted on the DOM, updated, and finally unmounted or removed.

**Need For Lifecycle Methods**

* To control the behavior of components at specific points (e.g., after mounting or updating).
* To load data, initialize settings, or clean up resources.
* To handle errors gracefully.

**Benefits:**

* Efficient data fetching (componentDidMount)
* Prevent memory leaks (componentWillUnmount)
* Improved performance through controlled re-renders
* Error handling using lifecycle hooks
* Debugging made easier with lifecycle awareness

**2. Lifecycle Hook Methods**

Lifecycle methods are grouped by phases:

**Mounting Phase (When component is added to the DOM):**

* constructor() – Initialize state and bind methods.
* static getDerivedStateFromProps() – Sync props to state (rarely used).
* render() – Return JSX (UI).
* componentDidMount() – Perform API calls or DOM operations.

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

* static getDerivedStateFromProps() – Used for syncing props.
* shouldComponentUpdate() – Optimize performance by controlling re-rendering.
* render()
* getSnapshotBeforeUpdate() – Capture scroll position or other pre-update info.
* componentDidUpdate() – Use for DOM updates or additional data fetching.

**Unmounting Phase (When component is removed):**

* componentWillUnmount() – Clean up timers, listeners, etc.

**Error Handling Phase:**

* componentDidCatch() – Catch errors during rendering or in lifecycle methods.
* static getDerivedStateFromError() – Update state after an error.

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

**Mounting (First Render)**

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

**Updating (When state/props change)**

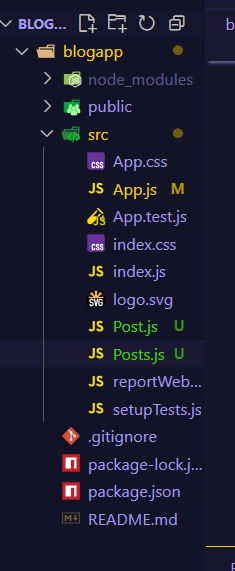
1. getDerivedStateFromProps()
2. shouldComponentUpdate() (optional)
3. render()
4. getSnapshotBeforeUpdate() (optional)
5. componentDidUpdate()

**Unmounting**

1. componentWillUnmount()

**Error Handling**

* If an error occurs in any of the above:
  + getDerivedStateFromError()
  + componentDidCatch()

**ReactJS Setup**

**Post.js**

import React from 'react';

class Post extends React.Component {

  render() {

    const { title, body } = this.props;

    return (

      <div style={{ border: "1px solid gray", padding: "10px", margin: "10px" }}>

        <h2>{title}</h2>

        <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: [],

      hasError: false,

      error: null,

    };

  }

  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({ hasError: true, error: error.message });

      });

  };

  componentDidMount() {

    this.loadPosts();

  }

  componentDidCatch(error, info) {

    this.setState({ hasError: true, error: error.toString() });

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

  }

  render() {

    if (this.state.hasError) {

      return <h3>Error occurred: {this.state.error}</h3>;

    }

    return (

      <div style={{ padding: "20px", fontFamily: "Arial" }}>

        <h1 style={{ color: "#2c3e50" }}>Welcome to BlogApp</h1>

        <p style={{ fontSize: "16px", color: "#555" }}>

          Explore a collection of posts fetched from a public API. Each post includes a title and content preview.

        </p>

        <h2 style={{ marginTop: "30px", color: "#34495e" }}>Recent Posts</h2>

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

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

        ))}

      </div>

    );

  }

}

export default Posts;

**App.js**

import React from 'react';

import Posts from './Posts';

function App() {

  return (

    <div className="App">

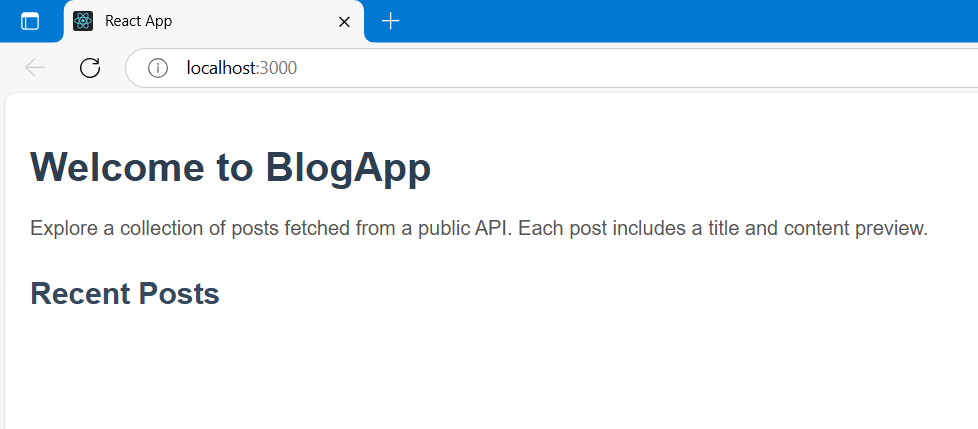
      <Posts />

    </div>

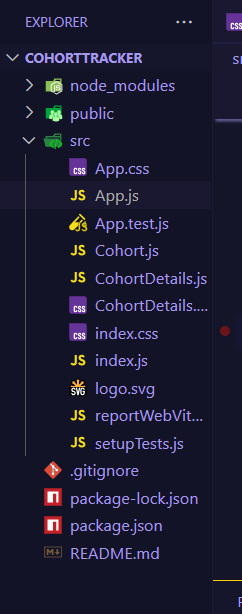
  );

}

export default App;

**Output**

**ReactJS – Hands On – 5**

**ReactJs Setup**

**CohortDetails.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;

}

**CohortDetails.js**

import React from 'react';

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

const CohortDetails = ({ cohort }) => {

  const headingStyle = {

    color: cohort.currentStatus.toLowerCase() === 'ongoing' ? 'green' : 'blue',

  };

  return (

    <div className={styles.box}>

      <h3 style={headingStyle}>

        {cohort.id} - {cohort.name}

      </h3>

      <dl>

        <dt>Started On</dt>

        <dd>{cohort.startDate}</dd>

        <dt>Current Status</dt>

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

        <dt>Coach</dt>

        <dd>{cohort.coach}</dd>

        <dt>Trainer</dt>

        <dd>{cohort.trainer}</dd>

      </dl>

    </div>

  );

};

export default CohortDetails;

**App.js**

import React from 'react';

import CohortDetails from './CohortDetails';

function App() {

  const cohorts = [

    {

      id: 'INTADMDF10',

      name: '.NET FSD',

      startDate: '22-Feb-2022',

      currentStatus: 'Scheduled',

      coach: 'Aathma',

      trainer: 'Jojo Jose',

    },

    {

      id: 'ADM21JF014',

      name: 'Java FSD',

      startDate: '10-Sep-2021',

      currentStatus: 'Ongoing',

      coach: 'Apoorv',

      trainer: 'Elisa Smith',

    },

    {

      id: 'CDBJF21025',

      name: 'Java FSD',

      startDate: '24-Dec-2021',

      currentStatus: 'Ongoing',

      coach: 'Aathma',

      trainer: 'John Doe',

    },

  ];

  return (

    <div style={{ padding: '20px' }}>

      <h2>Cohorts Details</h2>

      {cohorts.map((cohort) => (

        <CohortDetails key={cohort.id} cohort={cohort} />

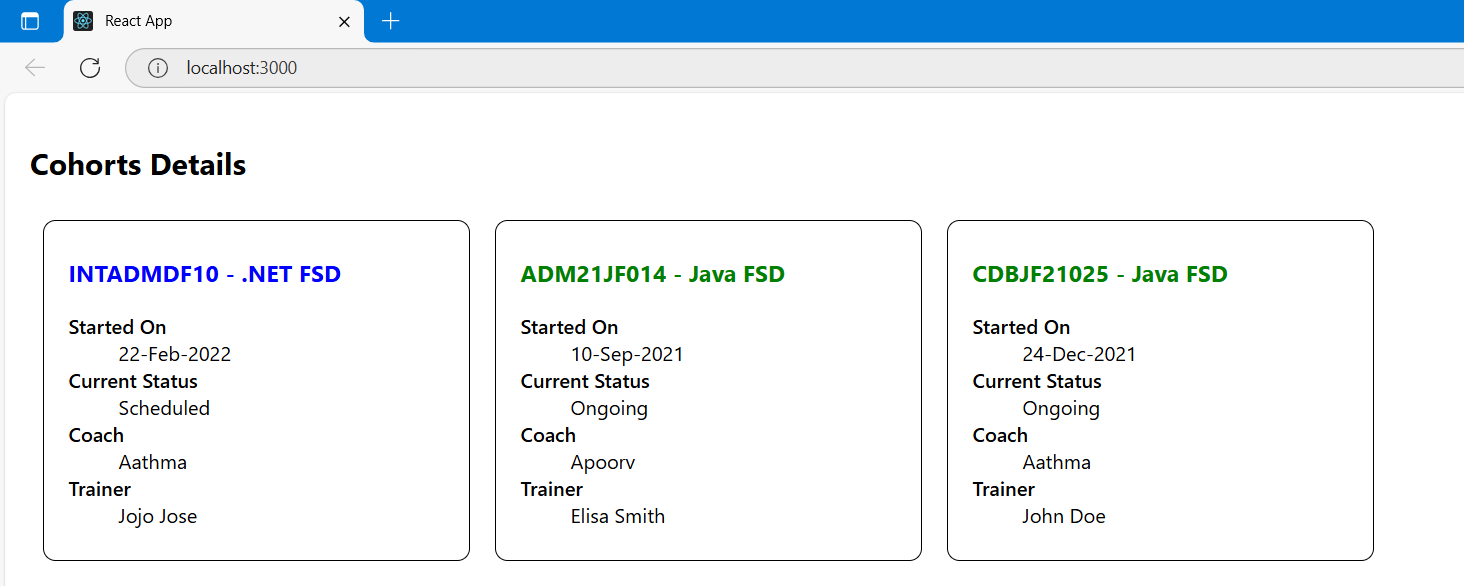
      ))}

    </div>

  );

}

export default App;

**Output**