**Week – 06**

**React**

**Exercise – 01**

**1. Define SPA and its benefits**

SPA (Single Page Application) is a web app that loads a single HTML page and dynamically updates content without refreshing the whole page.

Benefits:

* Faster navigation and loading
* Smoother user experience
* Reduces server load
* Easier to build responsive apps (e.g., Gmail, Facebook)

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

React is an open-source JavaScript library developed by Facebook for building user interfaces, especially SPAs.

How it works:

* React uses components (reusable UI blocks)
* It manages a virtual DOM, and updates the real DOM only when needed
* Uses JSX, a syntax that lets you write HTML in JavaScript

**3. Differences between SPA and MPA**

|  |  |  |
| --- | --- | --- |
| **Feature** | **SPA (Single Page App)** | **MPA (Multi Page App)** |
| Pages | Loads one HTML page | Loads multiple HTML pages |
| Navigation | Dynamic (via JavaScript) | Full page reload on navigation |
| Speed | Faster after initial load | Slower due to full reload |
| Example | Gmail, Facebook | Amazon, Wikipedia |
| Backend Calls | Uses APIs (AJAX/fetch) | Often reloads entire page |

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

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| Faster UI performance | SEO challenges (content not always indexable) |
| Better user experience | Initial load time may be large |
| Efficient bandwidth use | Requires JavaScript enabled in the browser |
| Can be built as a mobile-first PWA | Complex state management for large apps |

**5. Explain about React**

React is a component-based library used to create interactive UIs.

* Maintained by Meta (Facebook)
* Follows declarative programming
* Ideal for building large-scale apps with complex UIs
* Uses virtual DOM for efficient updates

**6. Define virtual DOM**

Virtual DOM (V-DOM) is a lightweight in-memory copy of the real DOM.

* React updates the virtual DOM first
* It compares the new and old versions (diffing)
* Then applies the minimal changes to the real DOM
* Result: faster rendering and better performance

**7. Features of React**

* JSX – Write HTML-like code in JavaScript
* Component-based – Reusable, independent UI pieces
* Virtual DOM – Efficient DOM updates
* One-way data binding – Predictable data flow
* NPM ecosystem – Large community support and libraries
* Hooks – Modern way to handle state and lifecycle (e.g., useState, useEffect)

**Code:**

**App.js**

import React from 'react';

import './App.css'

**function** App() {

  return (

    <div className="app-container">

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

    </div>

  );

}

export default App;

**App.css**

.app-container {

  display: flex;

  justify-content: center;

  align-items: center;

  text-align: center;

  background-color: #f9f9f9;

}

h1 {

  font-family: Arial, sans-serif;

  font-size: 2rem;

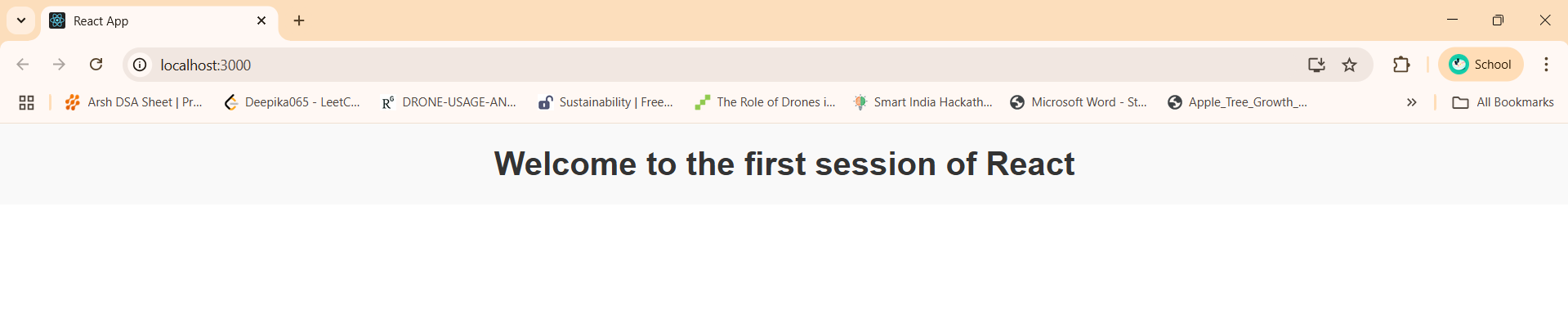
  color: #333;

}

**Explanation:**

* Installed **Node.js** and used npx create-react-app to automatically generate all the necessary files and configuration for a new React project named myfirstreact.
* Launched the folder in VS Code to access and edit the source files, especially the App.js file located in the src folder.
* Replaced the default content in App.js with a simple React component that returns a <h1> tag displaying: **"Welcome to the first session of React"**
* Applied CSS to center-align the heading both vertically and horizontally using flexbox inside .app-container.
* Used npm start in the terminal to run the development server. This automatically opened http://localhost:3000 in a browser, showing the custom React output.

**Output:**

****

**Exercise – 02**

**1. Explain React components**

React components are independent, reusable blocks of code that represent parts of a UI (e.g., buttons, forms, sections). They can be written as functions or classes.

**2. Differences between components and JavaScript functions**

|  |  |  |
| --- | --- | --- |
| **Feature** | **React Component** | **JavaScript Function** |
| Purpose | UI building block | General-purpose logic |
| JSX Support | Yes | No |
| Lifecycle methods | Yes (class components) | No |
| Can return UI | Yes | No (unless React component) |

**3. Types of components**

* Function Component – Uses a simple function to return JSX.
* Class Component – Uses ES6 classes and must define render() method.

**4. Explain class component**

A class component is a React component defined using the class keyword. It extends React.Component and must include a render() method to return JSX.

class Example extends React.Component {

render() {

return <h1>Hello</h1>;

}}

**5. Explain function component**

A function component is a simpler component written using a JavaScript function. It returns JSX and supports hooks like useState and useEffect.

function Example() {

return <h1>Hello</h1>;}

**6. Define component constructor**

In class components, the constructor initializes state and binds methods. It runs before the component is mounted.

constructor(props) {

super(props);

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

}

**7. Define render() function**

The render() function is required in class components. It defines what UI to display. It returns JSX (similar to HTML).

render() {

return <h1>Welcome</h1>;

}

**Code:**

**Home.js**

import React from 'react';

**class** Home **extends** React.Component {

  render() {

    return (

      <div>

        <h3>Welcome to the Home page of Student Management Portal</h3>

      </div>

    );

  }

}

export default Home;

**About.js**

import React from 'react';

**class** About **extends** React.Component {

  render() {

    return (

      <div>

        <h3>Welcome to the About page of the Student Management Portal</h3>

      </div>

    );

  }

}

export default About;

**Contact.js**

import React from 'react';

**class** Contact **extends** React.Component {

  render() {

    return (

      <div>

        <h3>Welcome to the Contact page of the Student Management Portal</h3>

      </div>

    );

  }

}

export default Contact;

**App.js**

import React from 'react';

import logo from './logo.svg';

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;

**App.css**

.container {

  display: flex;

  flex-direction: column;

  align-items: center;

  justify-content: flex-start;

  height: 100vh;

  background-color: #f5f5f5;

  text-align: center;

}

h3 {

  margin: 15px 0;

  font-family: Arial, sans-serif;

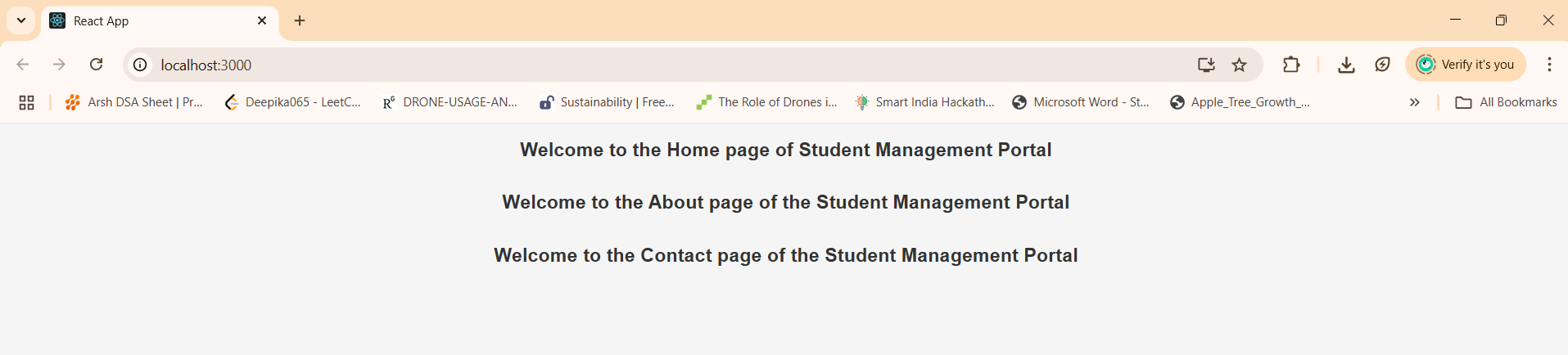
  color: #333;

}

**Explanation:**

* **Created a React project** using npx create-react-app . inside a folder named StudentApp.
* **Organized components** by creating a new Components folder inside src, and added three component files: Home.js, About.js, and Contact.js.
* **Used class components** for each file, each returning a heading message inside the render() method.
* **Modified** App.js to import and render all three components using JSX.
* **Ran the project** using npm start, which opened the app at http://localhost:3000, displaying all messages in the browser.

**Output:**



**Exercise – 03**

**1. Explain React components**

* A React component is a self-contained and reusable piece of code that represents part of a user interface.
* Components can be defined using JavaScript functions or ES6 classes and help build complex UIs by composing them from smaller, manageable parts.

**2. Identify the differences between components and JavaScript functions**

* While both are functions, a React component is specifically designed to return JSX (JavaScript XML), which describes the UI structure.
* React components can also manage state and handle side effects using hooks (in functional components) or lifecycle methods (in class components), unlike regular JavaScript functions.

**3. Identify the types of components**

* There are two main types of React components: **Functional components** and **Class components.**
* Functional components are more concise and modern, while class components were traditionally used for more complex logic before React hooks were introduced.

**4. Explain class component**

* A class component in React is built using ES6 class syntax and must extend React.Component.
* It contains a render() method to return JSX and can hold its own state, use lifecycle methods, and manage more complex logic and interactions.

**5. Explain function component**

* A function component is a simpler way to write components using JavaScript functions that return JSX.
* With the introduction of React Hooks (like useState, useEffect), functional components can now handle state, side effects, and more, making them equally powerful as class components.

**6. Define component constructor**

* In class components, the constructor is a special method used to initialize the component’s internal state and bind methods.
* It is called automatically when a component instance is created and is useful for setting up initial configurations.

**7. Define render() function**

* The render() function is a required method in class components that returns the JSX to be rendered on the UI.
* It tells React what the component should display, and it is called automatically whenever the component’s state or props change.

**Code:**

**CalculateScore.js**

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

**const** percentToDecimal = (decimal) **=>** {

  return (decimal \* 100).toFixed(2) + '%';

};

**const** calcScore = (total, goal) **=>** {

  return percentToDecimal(total / goal);

};

export **const** CalculateScore = ({ Name, School, total, goal }) **=>** {

  return (

    <div className="formatstyle">

      <h1 style={{ color: 'brown' }}>Student Details:</h1>

      <div className="Name">

        <b>Name:</b> <span>{Name}</span>

      </div>

      <div className="School">

        <b>School:</b> <span>{School}</span>

      </div>

      <div className="Total">

        <b>Total:</b> <span>{total} Marks</span>

      </div>

      <div className="Score">

        <b>Score:</b> <span>{calcScore(total, goal)}</span>

      </div>

    </div>

  );

};

**mystyle.css**

.formatstyle {

  text-align: center;

  font-size: large;

  margin-top: 30px;

}

.Name {

  font-weight: 300;

  color: blue;

}

.School {

  color: crimson;

}

.Total {

  color: darkmagenta;

}

.Score {

  color: forestgreen;

}

**App.js**

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

**function** App() {

  return (

    <div>

      <CalculateScore

        Name="Steeve"

        School="DNV Public School"

        total={284}

        goal={300}

      />

    </div>

  );

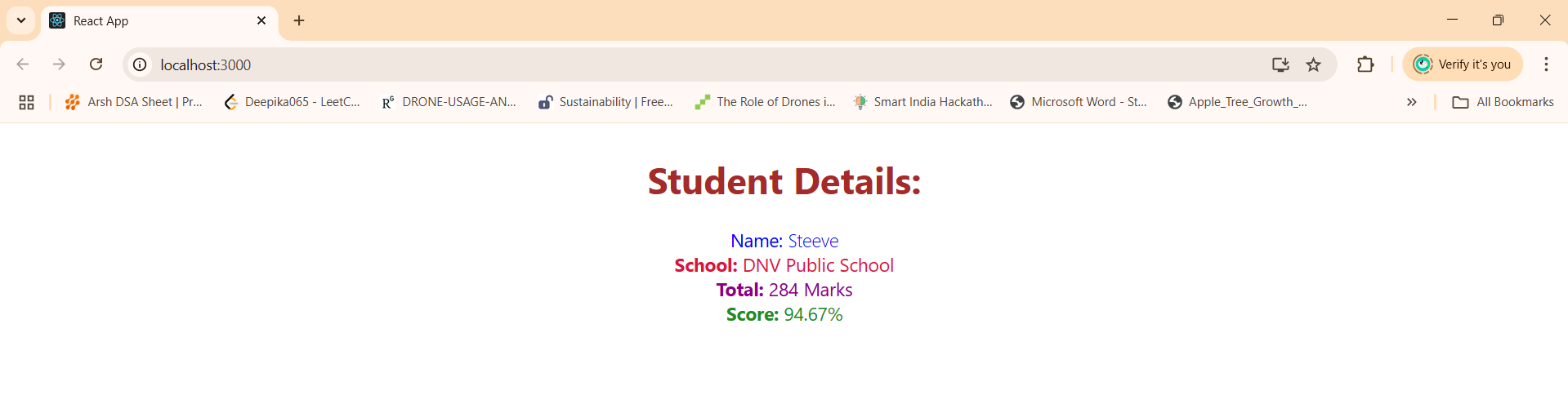
}

export default App;

**Explanation:**

* **Created React app using terminal** with npx create-react-app scorecalculatorapp.
* **Structured folders** by adding Components and Stylesheets under src.
* **Developed a functional component** CalculateScore to accept props and display student score.
* **Created and applied external CSS** using mystyle.css to center and style content.
* **Rendered the component in** App.js and verified output in localhost:3000.

**Output:**

****

**Exercise – 04**

**1. Explain the need and Benefits of component life cycle**

* React components go through different phases (mounting, updating, unmounting).
* Lifecycle methods help in controlling and optimizing behavior during these phases.
* Useful for API calls, error handling, memory management, and performance tuning.

**2. Identify various life cycle hook methods**

* componentDidMount(): Invoked after component mounts, good for API calls.
* componentDidCatch(): Captures and handles errors in child components.
* componentDidUpdate(): Called after component updates.
* componentWillUnmount(): Called before component is removed from DOM.

**3. List the sequence of steps in rendering a component**

* Constructor → render() → componentDidMount() → (Update/State change) → render() → componentDidUpdate()

**Code:**

**Post.js**

**class** Post {

  constructor(id, title, body) {

    this.id = id;

    this.title = title;

    this.body = body;

  }

}

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

    };

  }

  loadPosts() {

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

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

      .then((data) **=>** {

**const** postList = data.slice(0, 5).map(

          (item) **=>** new Post(item.id, item.title, item.body)

        );

        this.setState({ posts: postList });

      })

      .catch((error) **=>** {

        console.error("Error loading posts:", error);

      });

  }

  componentDidMount() {

    this.loadPosts();

  }

  componentDidCatch(error, info) {

    this.setState({ hasError: true });

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

  }

  render() {

    if (this.state.hasError) {

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

    }

    return (

      <div style={{ textAlign: "center" }}>

        <h1>Blog Posts</h1>

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

          <div key={post.id}>

            <h2>{post.title}</h2>

            <p>{post.body}</p>

          </div>

        ))}

      </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;

**Explanation:**

* **Created a React App** using create-react-app to scaffold a full React development environment.
* **Created a** Post **class** to define a data model for each blog post.
* **Built a class component** Posts, initialized state and added lifecycle methods.
* **Used** componentDidMount() to load posts from API using Fetch and update state.
* **Rendered posts in JSX,** handling errors using componentDidCatch() with alerts.

**Output:**



**Exercise – 05**

**1. Understanding the Need for Styling React Components**

* Styling React components is essential to create visually appealing, user-friendly interfaces.
* It ensures components are consistent, readable, and adaptable across different devices and screen sizes.
* Proper styling also improves user experience and reflects branding and design standards.

**2. Working with CSS Module and Inline Styles**

* CSS Modules allow you to write CSS that is scoped locally to a specific component, preventing class name clashes and making styles easier to manage in large applications.
* Inline Styles in React use JavaScript objects to apply styles directly to elements. They are useful for dynamic or conditional styling but are limited compared to traditional CSS (e.g., no pseudo-classes).

**Code:**

**CohortDetails.js**

import React from 'react';

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

**const** cohortList = [

  {

    name: "INTADMDF 10-NET FSD",

    startDate: "12-Aug-2021",

    status: "Scheduled",

    coach: "Arun",

    trainer: "Sonia"

  },

  {

    name: "ADM21JF014 Java FSD",

    startDate: "10-Sep-2021",

    status: "Scheduled",

    coach: "Apoorv",

    trainer: "Elsa"

  },

  {

    name: "CDRJF21025-JFSD",

    startDate: "01-Oct-2021",

    status: "Ongoing",

    coach: "Chaitanya",

    trainer: "Sami"

  }

];

**const** CohortDetails = () **=>** {

  return (

    <div>

      <h1>Cohorts Details</h1>

      {cohortList.map((cohort, index) **=>** (

        <div key={index} className={styles.box}>

          <h3 style={{ color: cohort.status === "Ongoing" ? "green" : "blue" }}>

            {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;

**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;

}

**App.js**

import React from 'react';

import './App.css';

import CohortDetails from './components/CohortDetails';

**function** App() {

  return (

    <div className="App">

      <CohortDetails />

    </div>

  );

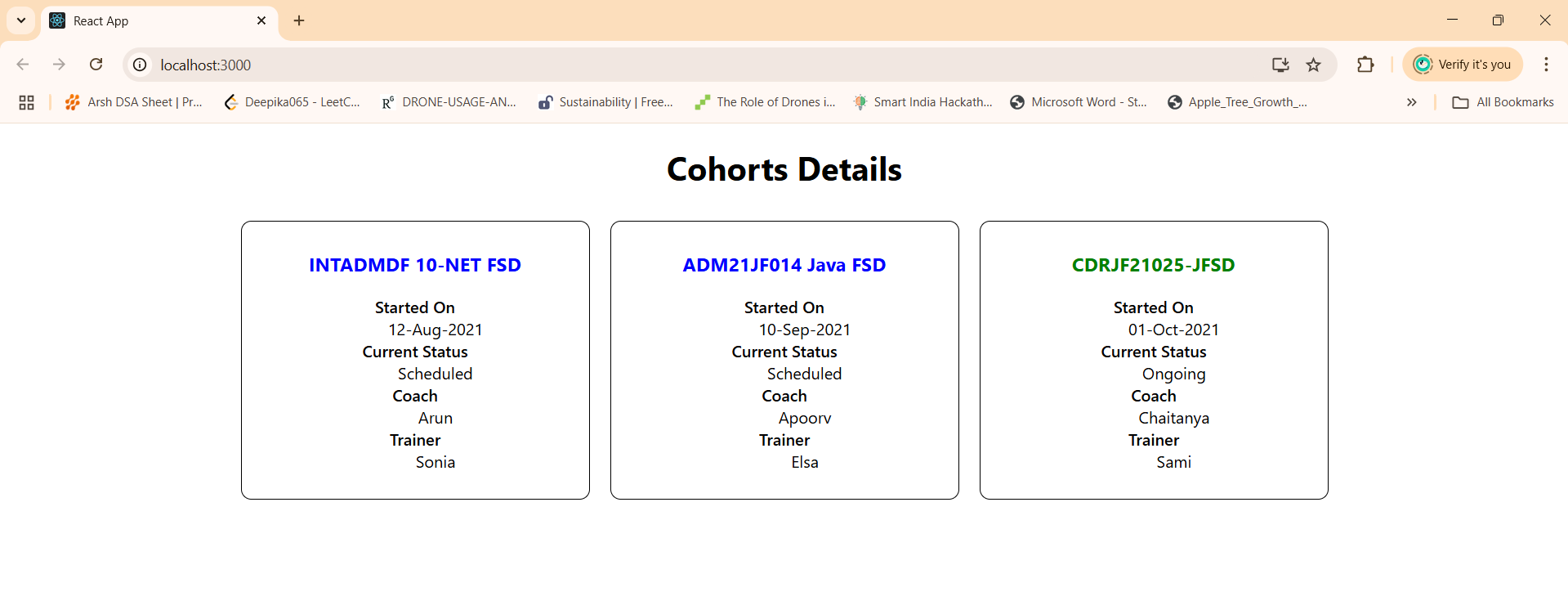
}

export default App;

**Explanation:**

* Used create-react-app cohort-dashboard to scaffold a new React project and opened it in VS Code for development.
* Inside the src/components folder, created two files: CohortDetails.js for the functional component and CohortDetails.module.css for component-specific styles.
* In CohortDetails.js, we defined a functional component that maps through a list of cohort objects and displays details like name, status, coach, and trainer.
* Styled the component using both **CSS Modules** (box class in CohortDetails.module.css) and **inline styles** to dynamically color the <h3> heading based on the cohort status (green for "Ongoing", blue otherwise).
* Imported and rendered the CohortDetails component inside App.js, and used npm start to launch the app in the browser at http://localhost:3000, where the styled cohort cards were displayed.

**Output:**

****