**WEEK – 05**

**ReactJS-HOL**

**Superset ID: 6262264**

**EXERCISE 1:**

**OBJECTIVES :**

**1. Define SPA and its Benefits**

SPA (Single-Page Application) is a type of web application that loads a single HTML page and dynamically updates the content as the user interacts with the app, without reloading the entire page.

Benefits of SPA:

* Faster performance: Only the required data is updated, not the whole page.
* Better user experience: Smooth navigation without page refresh.
* Reduced server load: Only necessary data is fetched from the server via APIs (like AJAX or Fetch).
* Reusable components: Common sections like header/footer stay constant while the main content changes.
* Offline support: SPAs can work offline using service workers.

Examples: Gmail, Google Maps, Facebook, Twitter.

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

React is a JavaScript library developed by Facebook for building user interfaces, mainly SPAs. It follows a component-based architecture, meaning the UI is divided into reusable components.

**How React Works:**

* Uses Virtual DOM for efficient updates.
* React compares the Virtual DOM with the previous state (Diffing Algorithm) and updates only the changed elements (Reconciliation).
* It renders components declaratively (we describe what UI should look like).
* Supports one-way data binding, meaning data flows from parent to child components.

**3. Identify the Differences Between SPA and MPA**

| **Feature** | **SPA (Single-Page Application)** | **MPA (Multi-Page Application)** |
| --- | --- | --- |
| **Reload** | No full-page reload | Reloads entire page on each request |
| **Speed** | Faster after first load | Slower due to frequent reloads |
| **Server Calls** | Only API calls for data | HTML page reloads for every navigation |
| **Development** | Easier to manage once set up | More complex due to multiple pages |
| **SEO** | Harder (requires SSR for SEO) | Easier because each page is separate |
| **Examples** | Gmail, Facebook | Amazon, Flipkart |

**4. Explain Pros & Cons of SPA**

**Pros:**

* Smooth user experience (no page reload).
* Fast navigation.
* Works well with APIs.
* Good for mobile-friendly apps.
* Easy to maintain with component-based architecture.

**Cons:**

* SEO issues (requires SSR or prerendering for better SEO).
* Initial loading time can be high.
* Browser history handling is tricky.
* Security concerns (XSS attacks).
* Requires JavaScript to work; if disabled, app fails.

**5. Explain about React**

React is a **JavaScript library for building UI components**. Key points:

* Developed by Facebook in 2013.
* Used for Single-Page Applications**.**
* Component-based: Build UI by combining independent pieces.
* Declarative: We describe what the UI should look like, React updates it efficiently.
* Supports JSX (JavaScript XML) to write HTML inside JavaScript.
* Unidirectional data flow for predictable state management.
* Uses Virtual DOM for faster rendering.

**6. Define Virtual DOM**

* Virtual DOM is a lightweight copy of the actual DOM (Document Object Model).
* React uses Virtual DOM to update only the changed elements instead of re-rendering the entire page.

**How it works:**

* When the UI changes, React creates a Virtual DOM representation.
* It compares the new Virtual DOM with the previous one using a process called diffing.
* It updates only the changed parts in the actual DOM (reconciliation).
* Benefit: Improves performance because real DOM manipulation is expensive.

**7. Explain Features of React**

* **Component-Based Architecture:** Break UI into reusable pieces.
* **Declarative:** Easy to predict and debug.
* **Virtual DOM:** Faster updates and rendering.
* **JSX:** Write HTML inside JavaScript for cleaner code.
* **Unidirectional Data Flow:** Data flows from parent to child, making apps predictable.
* **Hooks:** Functions like useState, useEffect for state and lifecycle management.
* **Rich Ecosystem:** Large community, lots of libraries and tools.

**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.**

**App.Js :**

import logo from './logo.svg';

import './App.css';

function App() {

  return (

     <div className="center">

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

    </div>

  );

}

export default App;

**App.css:**

.App {

  text-align: center;

}

.App-logo {

  height: 40vmin;

  pointer-events: none;

}

@media (prefers-reduced-motion: no-preference)

{

  .App-logo {

    animation: App-logo-spin infinite 20s linear;

  }

}

.App-header {

  background-color: #282c34;

  min-height: 100vh;

  display: flex;

  flex-direction: column;

  align-items: center;

  justify-content: center;

  font-size: calc(10px + 2vmin);

  color: white;

}

.App-link {

  color: #61dafb;

}

.center {

  display: flex;

  justify-content: center;

  height: 100vh;

@keyframes App-logo-spin {

  from {

    transform: rotate(0deg);

  }

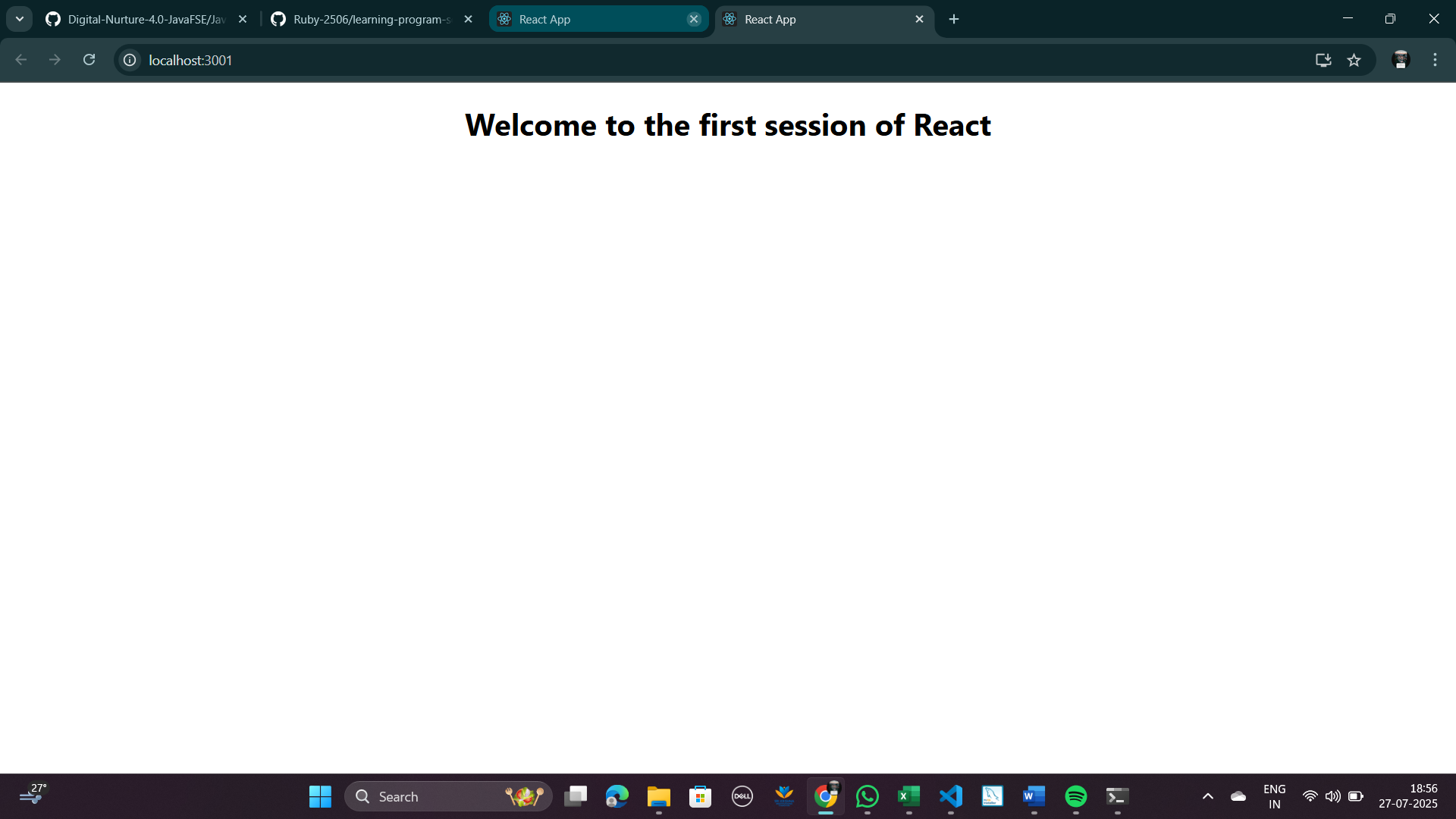
  to {

    transform: rotate(360deg);

  }

}

**OUTPUT:**



**EXERCISE 2:**

**Objectives:**

**1. Explain React Components**

React components are the building blocks of a React application. A component is a reusable piece of UI that can be composed together to form a complete interface. Each component:

* Encapsulates logic, structure (HTML/JSX), and styling.
* Accepts props (inputs) and maintains state (internal data).
* Can be nested inside other components to create complex UIs.

**Example:**

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

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

| **Aspect** | **React Component** | **JavaScript Function** |
| --- | --- | --- |
| **Purpose** | Used to render UI and manage state in React. | Performs a specific task or calculation. |
| **Return Value** | Returns **JSX** (UI representation). | Returns a value (number, string, etc.). |
| **Special Features** | Can manage **state** and **lifecycle methods**. | No lifecycle management. |
| **Execution** | Called by React during rendering. | Called manually by the programmer. |

**3. Identify the types of components**

React has two main types of components:

1. **Class Components**
   * ES6 classes that extend React.Component.
   * Use render() method to return JSX.
   * Support state and lifecycle methods.
2. **Function Components**
   * Simple JavaScript functions.
   * Accept props and return JSX.
   * Use Hooks (like useState, useEffect) for state and lifecycle behaviour.

**4. Explain Class Component**

A class component:

* Is declared using ES6 class syntax.
* Must extend React.Component.
* Must have a render() method that returns JSX.
* Can hold state and use lifecycle methods like componentDidMount().

**Example:**

class Welcome extends React.Component {

constructor(props) {

super(props);

this.state = { message: "Hello" };

}

render() {

return <h1>{this.state.message}, {this.props.name}</h1>;

}

}

**5. Explain Function Component**

A function component:

* Is a plain JavaScript function.
* Accepts props as arguments.
* Returns JSX directly.
* With React Hooks, function components can now manage state and lifecycle.

**Example:**

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

**With Hooks:**

function Counter() {

const [count, setCount] = React.useState(0);

return (

<div>

<p>{count}</p>

<button onClick={() => setCount(count + 1)}>Increase</button>

</div>

);

}

**6. Define Component Constructor**

* A constructor in a class component:
  + Initializes state.
  + Binds methods.
  + Called once when the component is created.
* Must call super(props) to access this.props.

**Example:**

constructor(props) {

super(props);

this.state = { count: 0 };

}

**7. Define render() function**

* The render() function:
  + Is mandatory in class components.
  + Returns the JSX (UI) for the component.
  + Gets called whenever:
    - State changes (setState()).
    - Props change.
  + Pure function: It should not modify state or interact with DOM directly.

**Key points:**

* It must return one root element (use a div or <> fragment if multiple elements).
* Should be idempotent (same output for same input).
* Executes on every re-render.

**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**.

**App.js:**

import React from 'react';

import './App.css';

import Home from './Home';

import About from './About';

import Contact from './Contact';

function App() {

  return (

    <div className="App">

      <h1>Student Management Portal</h1>

      <Home />

      <About />

      <Contact />

    </div>

  );

}

export default App;

**Contact.js:**

import React from 'react';

function Contact() {

  return (

    <div>

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

    </div>

  );

}

export default Contact;

**Home.js:**

import React from 'react';

function Home() {

  return (

    <div>

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

    </div>

  );

}

export default Home;

**About.js:**

import React from 'react';

function About() {

  return (

    <div>

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

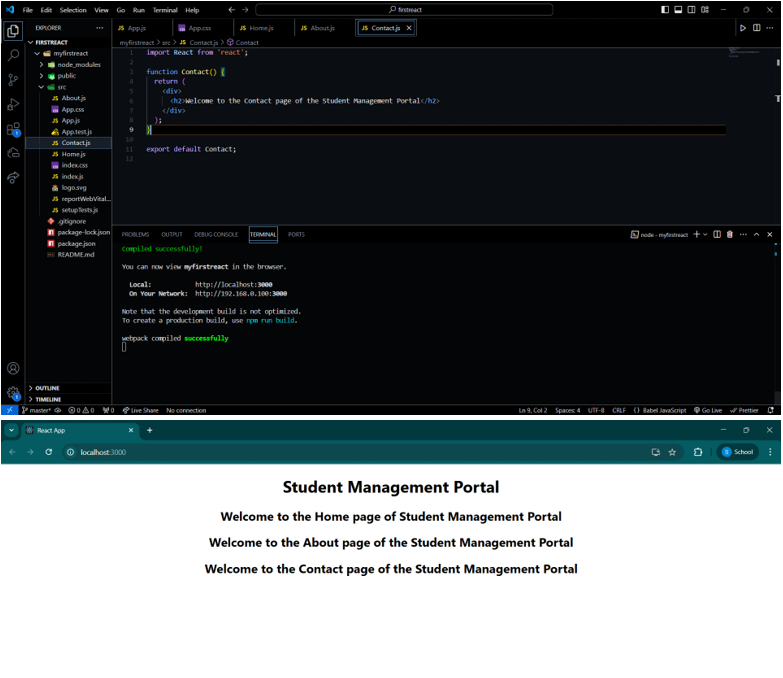
    </div>

  );

}

export default About;

**OUTPUT:**

****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**EXERCISE 3:**

**Objectives:**

**1. Explain React Components**

React components are the building blocks of a React application. A component is a reusable piece of UI that can be composed together to form a complete interface. Each component:

* Encapsulates logic, structure (HTML/JSX), and styling.
* Accepts props (inputs) and maintains state (internal data).
* Can be nested inside other components to create complex UIs.

**Example:**

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

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

| **Aspect** | **React Component** | **JavaScript Function** |
| --- | --- | --- |
| **Purpose** | Used to render UI and manage state in React. | Performs a specific task or calculation. |
| **Return Value** | Returns **JSX** (UI representation). | Returns a value (number, string, etc.). |
| **Special Features** | Can manage **state** and **lifecycle methods**. | No lifecycle management. |
| **Execution** | Called by React during rendering. | Called manually by the programmer. |

**3. Identify the types of components**

React has two main types of components:

1. **Class Components**
   * ES6 classes that extend React.Component.
   * Use render() method to return JSX.
   * Support state and lifecycle methods.
2. **Function Components**
   * Simple JavaScript functions.
   * Accept props and return JSX.
   * Use Hooks (like useState, useEffect) for state and lifecycle behaviour.

**4. Explain Class Component**

A class component:

* Is declared using ES6 class syntax.
* Must extend React.Component.
* Must have a render() method that returns JSX.
* Can hold state and use lifecycle methods like componentDidMount().

**Example:**

class Welcome extends React.Component {

constructor(props) {

super(props);

this.state = { message: "Hello" };

}

render() {

return <h1>{this.state.message}, {this.props.name}</h1>;

}

}

**5. Explain Function Component**

A function component:

* Is a plain JavaScript function.
* Accepts props as arguments.
* Returns JSX directly.
* With React Hooks, function components can now manage state and lifecycle.

**Example:**

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

**With Hooks:**

function Counter() {

const [count, setCount] = React.useState(0);

return (

<div>

<p>{count}</p>

<button onClick={() => setCount(count + 1)}>Increase</button>

</div>

);

}

**6. Define Component Constructor**

* A constructor in a class component:
  + Initializes state.
  + Binds methods.
  + Called once when the component is created.
* Must call super(props) to access this.props.

**Example:**

constructor(props) {

super(props);

this.state = { count: 0 };

}

**7. Define render() function**

* The render() function:
  + Is mandatory in class components.
  + Returns the JSX (UI) for the component.
  + Gets called whenever:
    - State changes (setState()).
    - Props change.
  + Pure function: It should not modify state or interact with DOM directly.

**Key points:**

* It must return one root element (use a div or <> fragment if multiple elements).
* Should be idempotent (same output for same input).
* Executes on every re-render.

**App.js:**

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>

);

**CalculateScore.js:**

import {

CalculateScore

} from './src/components/CalculateScore';

function App() {

return ( <

div >

<

CalculateScore Name = {

"Ruby M"

}

School = {

"DNV Public School"

}

total = {

284

}

goal = {

3

}

/> <

/div>

);

}

export default App;

**mystyle.css:**

.Name {

font-weight: 300;

color: blue;

}

.School {

color: crimson;

}

.Total {

color: darkmagenta;

}

.formatstyle {

text-align: center;

font-size: large;

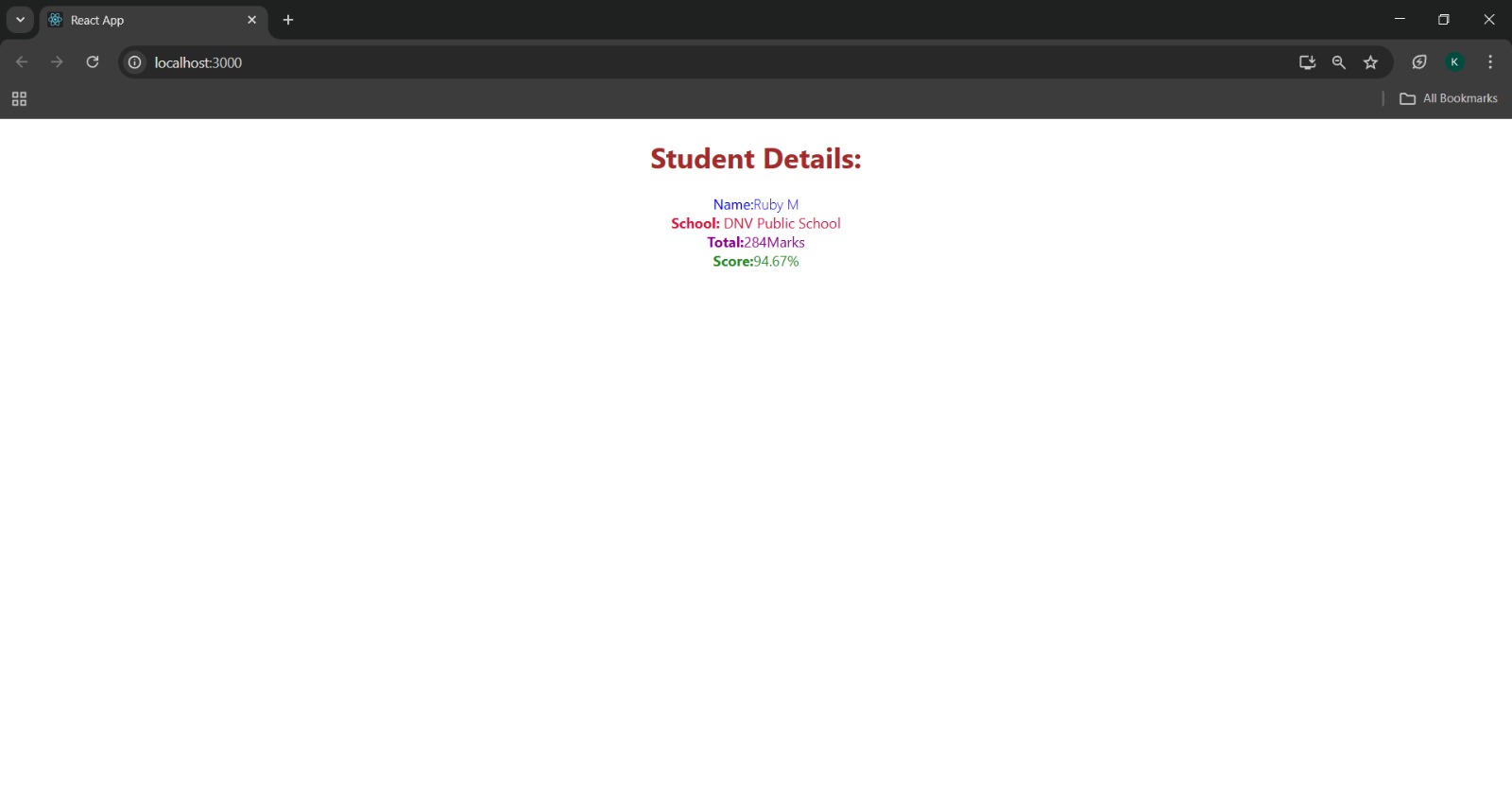
}

.Score {

color: forestgreen;

}

**OUTPUT:**



**EXERCISE 4:**

**Objectives:**

**1. Explain the Need and Benefits of Component Life Cycle**

The Component Life Cycle in React refers to the series of phases a component goes through from creation (mounting) to updating and finally unmounting.

Need for Life Cycle:

* To control component behaviour at different stages.
* To perform actions automatically like:
  + Fetching data when a component mounts.
  + Updating the DOM after changes.
  + Cleaning up resources when a component unmounts.

Benefits:

* Predictable behaviour – We know what happens at each stage.
* Optimized performance – Update only when necessary.
* Separation of concerns – Different tasks (fetch, update, cleanup) are handled in appropriate methods.
* Easier debugging & maintenance – We can track what happens when.

**2. Identify Various Life Cycle Hook Methods**

React Class Components have built-in life cycle methods, while Functional Components use Hooks (useEffect, useLayoutEffect, etc.).

**Phases & Methods**

**(A) Mounting Phase** *(when component is created and inserted into DOM)*

* constructor()
* static getDerivedStateFromProps()
* render()
* componentDidMount()

**(B) Updating Phase** *(when props/state changes)*

* static getDerivedStateFromProps()
* shouldComponentUpdate()
* render()
* getSnapshotBeforeUpdate()
* componentDidUpdate()

**(C) Unmounting Phase** *(when component is removed from DOM)*

* componentWillUnmount()

**Functional Component Hooks:**

* useEffect(() => { ... }, []) → Acts like componentDidMount
* useEffect(() => { ... }, [dependencies]) → Acts like componentDidUpdate
* useEffect(() => { return cleanup }, []) → Acts like componentWillUnmount

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

**In Class Components (First Render):**

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

**On Update (state/props change):**

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

**On Unmount:**

* componentWillUnmount()

**Create a new react application using *create-react-app* tool with the name as “blogapp”**

**Post.js:**

import React from 'react';

class Post extends React.Component {

render() {

const { title, body } = this.props;

return (

<div className="post">

<h2>{title}</h2>

<p>{body}</p>

<hr />

</div>

);

}

}

export default Post;

**Posts.js:**

import React, { Component } from 'react';

import Post from './Post';

class Posts extends Component {

constructor(props) {

super(props);

this.state = {

posts: [],

hasError: false

};

}

// Step 6: Fetch posts

loadPosts() {

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

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

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

.catch(error => {

console.error('Error fetching posts:', error);

this.setState({ hasError: true });

});

}

// Step 7: componentDidMount

componentDidMount() {

this.loadPosts();

}

// Step 9: componentDidCatch

componentDidCatch(error, info) {

alert("An error occurred while loading the posts.");

console.error("Error boundary caught:", error, info);

this.setState({ hasError: true });

}

// Step 8: render

render() {

const { posts, hasError } = this.state;

if (hasError) {

return <h2>Something went wrong. Please try again later.</h2>;

}

return (

<div>

<h1>Blog Posts</h1>

{posts.map(post => (

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

))}

</div>

);

}

}

export default Posts;

**App.js:**

import React from 'react';

import './App.css';

import Posts from './Posts';

function App() {

return (

<div className="App">

<Posts />

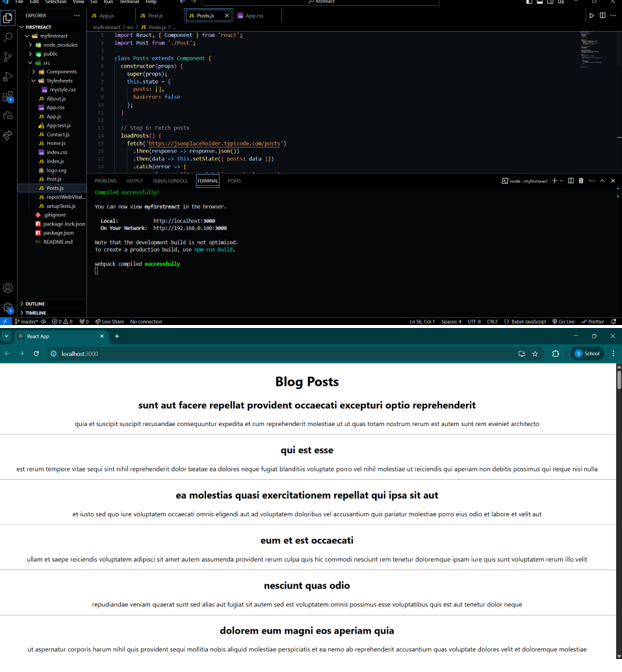
</div>

);

}

export default App;

**OUTPUT:**

****

**EXERCISE 5:**

**CODE:**

**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;

}

**CohortDetails.js**

import React from 'react';

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

class CohortDetails extends React.Component {

render() {

const { cohort } = this.props;

return (

<div className={styles.box}>

<h3 style={{ color: cohort.status === 'ongoing' ? 'green' : 'blue' }}>

{cohort.status}

</h3>

<dl>

<dt>Name:</dt>

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

<dt>Start Date:</dt>

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

<dt>End Date:</dt>

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

</dl>

</div>

);

}

}

export default CohortDetails;

**App.js:**

import React from 'react';

import CohortDetails from './CohortDetails';

function App() {

const cohort = {

name: 'React Bootcamp',

startDate: '2024-01-01',

endDate: '2024-04-01',

status: 'ongoing'

};

return (

<div className="App">

<h1>Cohort Dashboard</h1>

<CohortDetails cohort={cohort} />

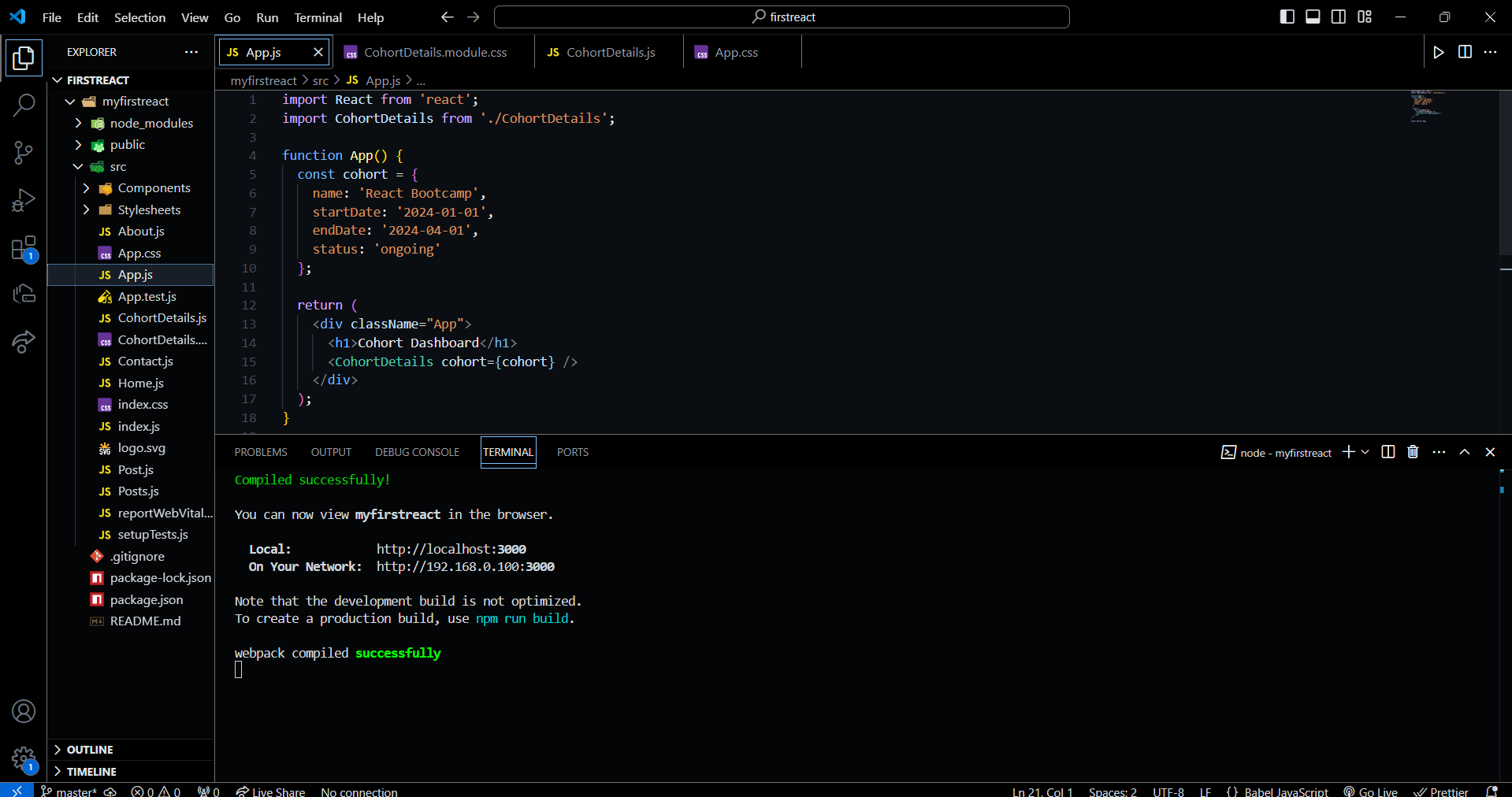
</div>

);

}

export default App;

**OUTPUT:**



****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**