**WEEK – 6**

**REACT**

**1. REACTJS-HOL**

**Define SPA (Single Page Application) and its Benefits**

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

**Benefits of SPA:**

* Fast Performance: Only data is updated, not the entire page.
* Better User Experience: Smooth transitions and instant feedback.
* Reduced Server Load: Fewer requests to the server after the initial load.
* Offline Support: Easy to implement offline capabilities with caching.

**Define React and Identify Its Working**

React is a JavaScript library developed by Facebook for building fast, interactive user interfaces, especially SPAs.

**How React Works:**

* React uses components to build the UI.
* It uses a virtual DOM to efficiently update the UI.
* When data changes, React updates only the changed parts in the DOM, not the entire page.

**Differences between SPA and MPA (Multi-Page Application)**

|  |  |  |
| --- | --- | --- |
| **Feature** | **SPA** | **MPA** |
| Page Reloads | No (dynamic updates) | Yes (full reload on navigation) |
| Speed | Faster (after initial load) | Slower due to full reload |
| User Experience | Seamless and app-like | Traditional website feel |
| Initial Load Time | Slightly longer | Usually quicker |
| SEO Support | Needs extra configuration (SSR) | Better SEO out-of-the-box |

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

**Pros:**

* Faster navigation
* Smooth user experience
* Reusable frontend code
* Efficient performance using Virtual DOM

**Cons:**

* Poor SEO without server-side rendering
* Initial load time can be longer
* Requires JavaScript to function
* Browser back/forward buttons need manual handling

**Explain About React**

React is:

* A **declarative**, **component-based** library for building user interfaces.
* Used to create **dynamic**, **responsive** web applications.
* Maintained by **Meta (Facebook)** and a strong open-source community.

**Define Virtual DOM**

The **Virtual DOM** is a lightweight copy of the actual DOM.

**How It Works:**

* React keeps a virtual DOM in memory.
* When the state changes, a **diffing algorithm** compares the new virtual DOM with the old one.
* Only the differences are updated in the real DOM (more efficient).

**Features of React**

* **Component-Based:** Breaks UI into reusable components.
* **Virtual DOM:** Enhances performance with minimal DOM manipulation.
* **JSX Syntax:** Allows writing HTML-like code in JavaScript.
* **One-Way Data Binding:** Data flows in a single direction (top-down).
* **Declarative UI:** Simplifies UI design with clear code.
* **Unidirectional Data Flow:** Helps in maintaining code predictability.
* **React Hooks:** Manage state and side-effects in functional components.

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.

**APPROACH**

**** Install Node.js and use npx to create a new React app named "myfirstreact".

 Navigate into the "myfirstreact" project folder using the terminal.

 Open the project in Visual Studio Code.

 In the src folder, open and edit App.js to display a welcome message as a heading.

 Start the application using the terminal and view it in the browser at localhost:3000.

**CODE**

**App.js**

import React from 'react';

function App() {

  return (

    <div>

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

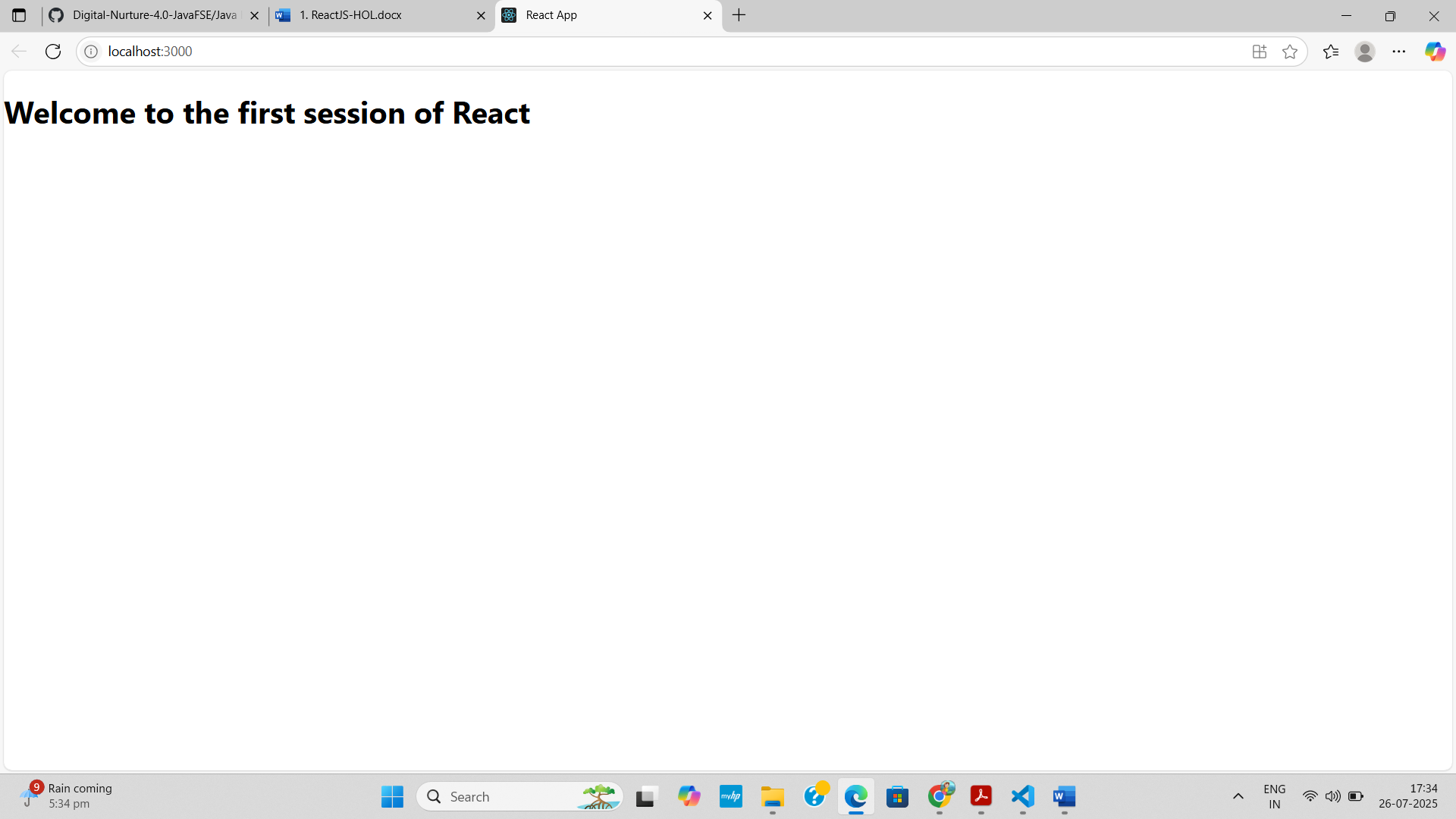
    </div>

  );

}

export default App;

**OUTPUT**



**2. REACTJS-HOL**

**Explain React Components**

React components are the building blocks of a React application. They are reusable pieces of code that return JSX (HTML-like syntax) to describe what the UI should look like.

* Each component represents a part of the UI, such as a button, header, or form.
* Components can be nested inside other components.
* They help in maintaining a clean, modular, and manageable codebase.

**Differences Between Components and JavaScript Functions**

|  |  |  |
| --- | --- | --- |
| Aspect | React Component | JavaScript Function |
| Return Type | Returns JSX (UI) | Returns any data type (string, number) |
| React Integration | Part of React framework | Generic, not tied to React |
| Lifecycle Methods | Present in class components | Not available in regular functions |
| Hooks Usage | Functional components can use Hooks | Not supported in regular JS functions |

**Types of React Components**

There are mainly two types:

1. Class Components
2. Function Components

**Explain Class Component**

* Class components are ES6 classes that **extend React.Component**.
* They must include a **render()** method that returns JSX.
* Can have **state**, **lifecycle methods**, and **props**.

**Example:**

jsx

Copy code

import React, { Component } from 'react';

class Welcome extends Component {

render() {

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

}

}

**Explain Function Component**

* Function components are **JavaScript functions** that return JSX.
* With **React Hooks**, function components can now use state and lifecycle features.

**Example:**

jsx

Copy code

function Welcome(props) {

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

}

**Or using arrow function:**

jsx

Copy code

const Welcome = (props) => <h1>Hello, {props.name}</h1>;

**Define Component Constructor**

The constructor() method is used in **class components** to:

* Initialize **state**
* Bind methods
* Call the parent class (super(props))

**Example:**

jsx

Copy code

constructor(props) {

super(props);

this.state = { count: 0 };

}

**Define render() Function**

* The render() method is **mandatory** in class components.
* It returns the **JSX** that represents the UI of the component.
* React calls this method to **display the component** on the screen.

**Example:**

jsx

Copy code

render() {

return <div>Welcome to React</div>;

}

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.

**APPROACH**

**** Create a React app named studentapp using npx create-react-app studentapp (use only lowercase).

 Inside the src folder, create a Components folder and add Home.js, About.js, and Contact.js files.

 In each file, define a functional component that displays a welcome message for Home, About, and Contact pages.

 In App.js, import all three components and render them inside the main component.

 Run the app using npm start and open localhost:3000 to view the output.

**CODE**

**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;

**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;

**App.js**

import React from 'react';

import Home from './Components/Home';

import About from './Components/About';

import Contact from './Components/Contact';

function App() {

  return (

    <div>

      <Home />

      <About />

      <Contact />

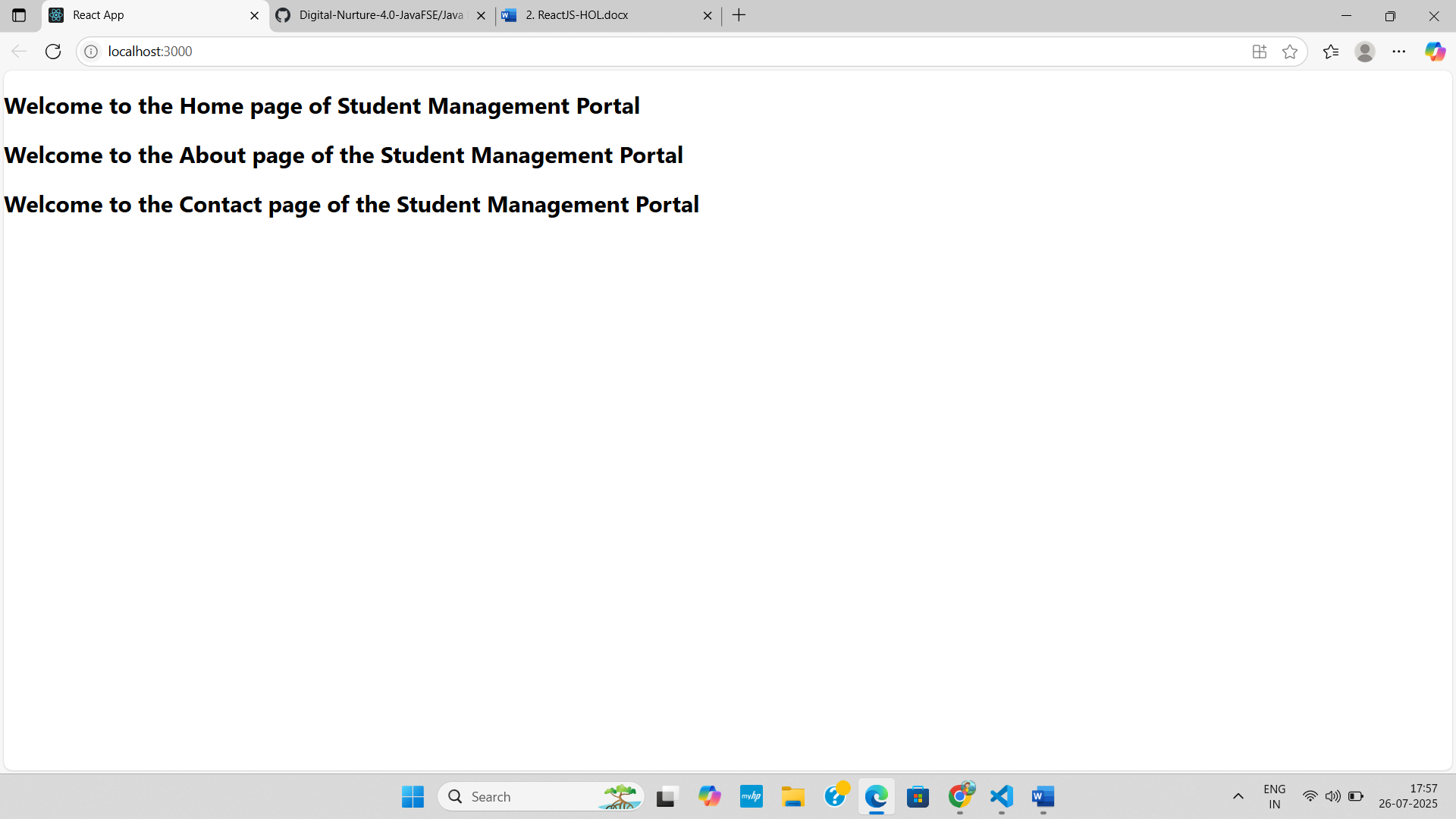
    </div>

  );

}

export default App;

**OUTPUT**



**3. REACTJS-HOL**

Create a react app for Student Management Portal named scorecalculatorapp and create a function component named “CalculateScore” which will accept Name, School, Total and goal in order to calculate the average score of a student and display the same.

**APPROACH**

**** Create a React app named scorecalculatorapp using npx create-react-app scorecalculatorapp.

 Inside the src folder, create a Components folder and add a CalculateScore.js file to define a functional component that calculates and displays average score using props.

 Create a Stylesheets folder and add mystyle.css to style the component.

 Edit App.js to import and render the CalculateScore component with appropriate props.

 Run the app using npm start and open localhost:3000 in your browser to view the result.

**CODE**

**CalculateScore.js**

import React from 'react';

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

function CalculateScore(props) {

  const average = props.total / props.goal;

  return (

    <div className="score-box">

      <h2>Student Score Details</h2>

      <p><strong>Name:</strong> {props.name}</p>

      <p><strong>School:</strong> {props.school}</p>

      <p><strong>Total Score:</strong> {props.total}</p>

      <p><strong>Goal:</strong> {props.goal}</p>

      <p><strong>Average Score:</strong> {average.toFixed(2)}</p>

    </div>

  );

}

export default CalculateScore;

**mystyle.css**

.score-box {

    background-color: #f0f8ff;

    border: 2px solid #007acc;

    border-radius: 10px;

    padding: 20px;

    width: 400px;

    margin: 30px auto;

    text-align: left;

    font-family: Arial, sans-serif;

}

**App.js**

import React from 'react';

import './App.css';

import CalculateScore from './Components/CalculateScore';

function App() {

  return (

    <div className="App">

      <CalculateScore

        name="John Doe"

        school="Greenwood High"

        total={450}

        goal={5}

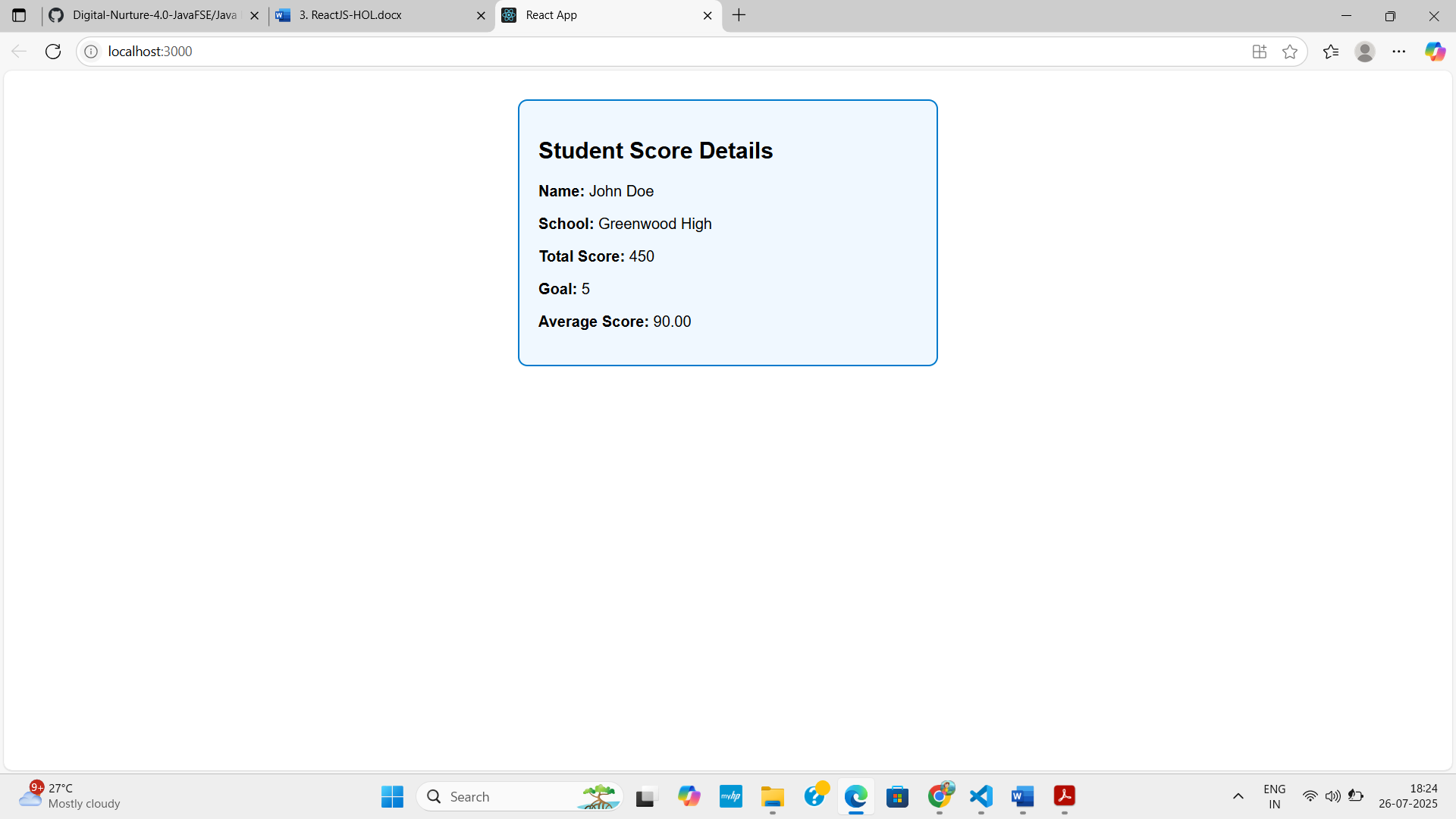
      />

    </div>

  );

}

export default App;

**OUTPUT**

**4. REACTJS-HOL**

**Need and Benefits of Component Lifecycle in React**

**Why is it needed?**The component lifecycle in React refers to the different stages a component goes through from creation to removal. It allows developers to hook into specific moments of a component's life and perform necessary actions.

Benefits:

* Initialization: Set initial state or fetch required data before the component appears.
* Update Control: Perform actions when the component updates (e.g., fetching new data or re-rendering).
* Cleanup: Free up memory/resources when the component is removed (like clearing timers or unsubscribing listeners).
* Optimization: Helps in performance tuning by avoiding unnecessary renders.
* Data Sync: Sync state or props changes smoothly at the right stages.

**Identify Various Lifecycle Hook Methods**

React class components have lifecycle methods grouped into 3 phases:

1. Mounting (when the component is created and inserted into the DOM):

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

2. Updating (when the component is re-rendered due to changes in props or state):

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

3. Unmounting (when the component is removed from the DOM):

* componentWillUnmount()

4. Error Handling (when an error occurs in rendering or lifecycle):

* componentDidCatch()
* static getDerivedStateFromError()

**Sequence of Steps in Rendering a Component (Class Component)**

Mounting Phase:

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

Updating Phase:

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

Unmounting Phase:

1. componentWillUnmount()

**APPROACH**

**** Create a React app named blogapp using create-react-app and open it in VS Code.

 Create a reusable Post functional component to display post title and body.

 Create a class-based Posts component, initialize state with an empty list, and define a loadPosts() method to fetch data from an API.

 Use componentDidMount() to call loadPosts() and render posts using the Post component; implement componentDidCatch() for error alerts.

 Import and use the Posts component in App.js, then run the app with npm start and view it on localhost:3000.

**CODE**

**Post.js**

import React from 'react';

function Post(props) {

  return (

    <div style={{ marginBottom: "20px", border: "1px solid #ddd", padding: "10px" }}>

      <h3>{props.title}</h3>

      <p>{props.body}</p>

    </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: [],

      error: null

    };

  }

  loadPosts() {

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

      .then(response => {

        if (!response.ok) {

          throw new Error('Network response was not OK');

        }

        return response.json();

      })

      .then(data => {

        this.setState({ posts: data });

      })

      .catch(error => {

        this.setState({ error: error.message });

      });

  }

  componentDidMount() {

    this.loadPosts();

  }

  componentDidCatch(error, info) {

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

  }

  render() {

    return (

      <div>

        <h2>Blog Posts</h2>

        {this.state.error && <p style={{ color: 'red' }}>Error: {this.state.error}</p>}

        {this.state.posts.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 />

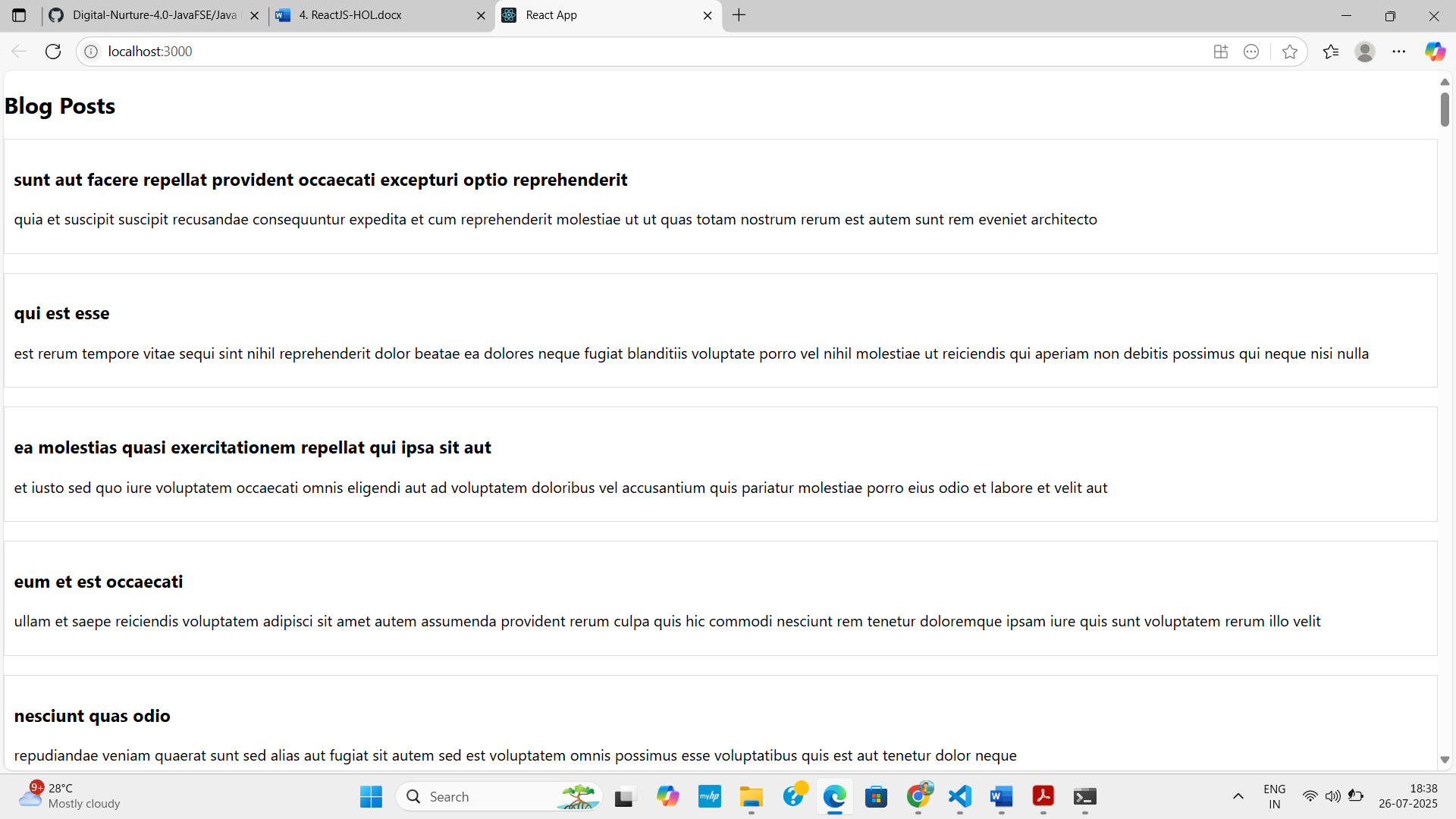
    </div>

  );

}

export default App;

**OUTPUT**



**5. REACTJS-HOL**

**APPROACH**

**** Create a new React app named cohort-dashboard using create-react-app and open it in VS Code.

 Inside the src/components folder, create CohortDetails.js and CohortDetails.module.css files.

 In CohortDetails.js, create a functional component that maps over a cohort list and displays details like name, status, coach, and trainer.

 In the CSS module, define a .box class for styling and apply dynamic color to <h3> based on the cohort's status (green for "Ongoing", blue otherwise).

 Import and render the CohortDetails component in App.js, then run the app with npm start to view the styled dashboard at localhost:3000.

**CODE**

**CohortDetails.js**

import React from 'react';

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

const cohortList = [

  {

    id: 1,

    name: "INTADMDF10 - .NET FSD",

    started: "22-Feb-2022",

    status: "Scheduled",

    coach: "Aarthna",

    trainer: "Jojo Jose"

  },

  {

    id: 2,

    name: "ADM21J1F014 - Java FSD",

    started: "10-Sep-2021",

    status: "Ongoing",

    coach: "Apoorv",

    trainer: "Elisa Smith"

  },

  {

    id: 3,

    name: "CDBJFZ1025 - Java FSD",

    started: "24-Dec-2021",

    status: "Ongoing",

    coach: "Aarthna",

    trainer: "John Doe"

  }

];

function CohortDetails() {

  return (

    <div>

      <h2>Cohorts Details</h2>

      {cohortList.map(cohort => (

        <div key={cohort.id} className={styles.box}>

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

          <dl>

            <dt>Started On</dt>

            <dd>{cohort.started}</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;

}

dt {

    font-weight: 500;

}

**App.js**

import React from 'react';

import CohortDetails from './components/CohortDetails';

function App() {

  return (

    <div className="App">

      <CohortDetails />

    </div>

  );

}

export default App;

**OUTPUT**

