**ReactJS – HOL 1**

**Step 1:** Define SPA and its benefits

A Single Page Application (SPA) is a type of web application that loads and updates content dynamically without refreshing the entire page. Unlike traditional websites, SPAs use modern technologies to enhance the user experience by minimizing interruptions and providing a smoother interface. Users can interact with the application seamlessly, similar to using desktop software.

Benefits:

* Faster User Experience
* Caching for Offline Access
* Improved Performance
* Reduced Bandwidth Usage
* Enhanced Responsiveness
* Seamless User Navigation
* Cross-Platform Compatibility
* Scalability

**Step 2:** Define React and identify its working

React is a powerful JavaScript library for building fast, scalable front-end applications. Created by Facebook, it's known for its component-based structure, single-page applications (SPAs), and virtual DOM, enabling efficient UI updates and a seamless user experience.

Working:

* Virtual DOM: React updates only the changed parts of the DOM, resulting in faster rendering.
* One-Way Data Binding: Ensures predictable and easy to debug data flow.
* Component-Based Architecture: Breaks UI into reusable pieces, improving the code reusability and scalability.

**Step 3:** Identify the differences between SPA and MPA

**SPA (Single Page Application)**:

* Loads a single HTML page and updates content dynamically.
* Faster navigation, no full page reloads.
* Uses AJAX/JavaScript to fetch data.

**MPA (Multi Page Application)**:

* Loads a new HTML page from the server for each action.
* Slower navigation, full page reloads.
* Better for large sites with lots of content (e.g., e-commerce).

**Step 4:** Explain Pros & Cons of Single-Page Application

Pros:

* Faster User Experience: SPAs load once, and only fetch the necessary data, reducing page reloads and providing a smoother, more responsive experience.
* Caching for Offline Access: SPAs can implement caching strategies, allowing users to access certain parts of the application even when offline.
* Improved Performance: By minimizing server requests and only updating the required components, SPAs significantly reduce the load on servers, resulting in faster load times and better overall performance.
* Reduced Bandwidth Usage: Since SPAs only fetch the data needed for specific interactions, they minimize the amount of data transferred between the client and server, reducing bandwidth usage and improving efficiency.
* Enhanced Responsiveness: It enable dynamic content updates without requiring full page reloads.
* Seamless User Navigation: SPAs use client-side routing, enabling seamless navigation between sections of the application without the need for full page reloads.
* Cross-Platform Compatibility: SPAs are inherently compatible with various devices and platforms, promoting a consistent user experience across desktops, tablets, and mobile devices.
* Scalability: SPAs support the scalability of web applications by efficiently managing client-server interactions.

Cons:

* Slower Initial Load: Can be slower initially, affecting users with slower internet.
* SEO Challenges: SEO can be tricky due to heavy reliance on JavaScript.
* Limited Browser Support: Advanced features may not work well on older browsers.
* Security Risks: Vulnerable to security issues like Cross-Site Scripting (XSS).
* Client-Side Resource Intensity: Places a heavy load on the client side, impacting older devices.
* Dependency on JavaScript: Essential functionality may break if users disable JavaScript.
* Browser History Management: Handling navigation dynamically poses challenges with browser history.
* Complex Development: Developing SPAs is more complex, requiring a learning curve.

**Step 5**: Explain about React

ReactJS is a component-based JavaScript library used to build dynamic and interactive user interfaces. It simplifies the creation of single-page applications (SPAs) with a focus on performance and maintainability.

* It is developed and maintained by Facebook.
* The latest version of React is React 19.
* Uses a virtual DOM for faster updates.
* Supports a declarative approach to designing UI components.
* Ensures better application control with one-way data binding.

**Step 6:** Define virtual DOM

The Virtual DOM (VDOM) is a lightweight, in-memory representation of the real DOM (Document Object Model). It helps React manage UI updates more efficiently by keeping a virtual version of the UI in memory. When changes occur, React updates only the necessary parts of the real DOM, instead of re-rendering everything.

**Step 7:** Explain Features of React

1. **Virtual DOM**

React uses a Virtual DOM to optimize UI rendering. Instead of updating the entire real DOM directly, React:

* Creates a lightweight copy of the DOM (Virtual DOM).
* Compares it with the previous version to detect changes (diffing).
* Updates only the changed parts in the actual DOM (reconciliation), improving performance.

1. **Component-Based Architecture**

React follows a component-based approach, where the UI is broken down into reusable components. These components:

* Can be functional or class-based.
* It allows code reusability, maintainability, and scalability.

1. **JSX (JavaScript XML)**

React usesJSX, a syntax extension that allows developers to write HTML inside JavaScript. JSX makes the code:

* More readable and expressive.
* Easier to understand and debug.

1. **One-Way Data Binding**

React uses one-way data binding, meaning data flows in a single direction from parent components to child components via props. This provides better control over data and helps maintain predictable behavior.

1. **State Management**

React manages component state efficiently using the useState hook (for functional components) or this.state (for class components). State allows dynamic updates without reloading the page.

1. **React Hooks**

Hooks allow functional components to use state and lifecycle features without needing class components. Common hooks include:

* useState: for managing local state.
* useEffect: for handling side effects like API calls.
* useContext: for global state management.

1. **React Router**

React provides React Router for managing navigation in single-page applications (SPAs). It enables dynamic routing without requiring full-page reloads.

**Step 8:** Install Create-react-app by running the following command in the command prompt:

Code:

npm install -g create-react-app

**Step 9:** To create a React Application with the name of “myfirstreact”, type the following command:

Code:

npx create-react-app myfirstreact

**Step 10:** App.js

Code:

**function** App() {

return (

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

);

}

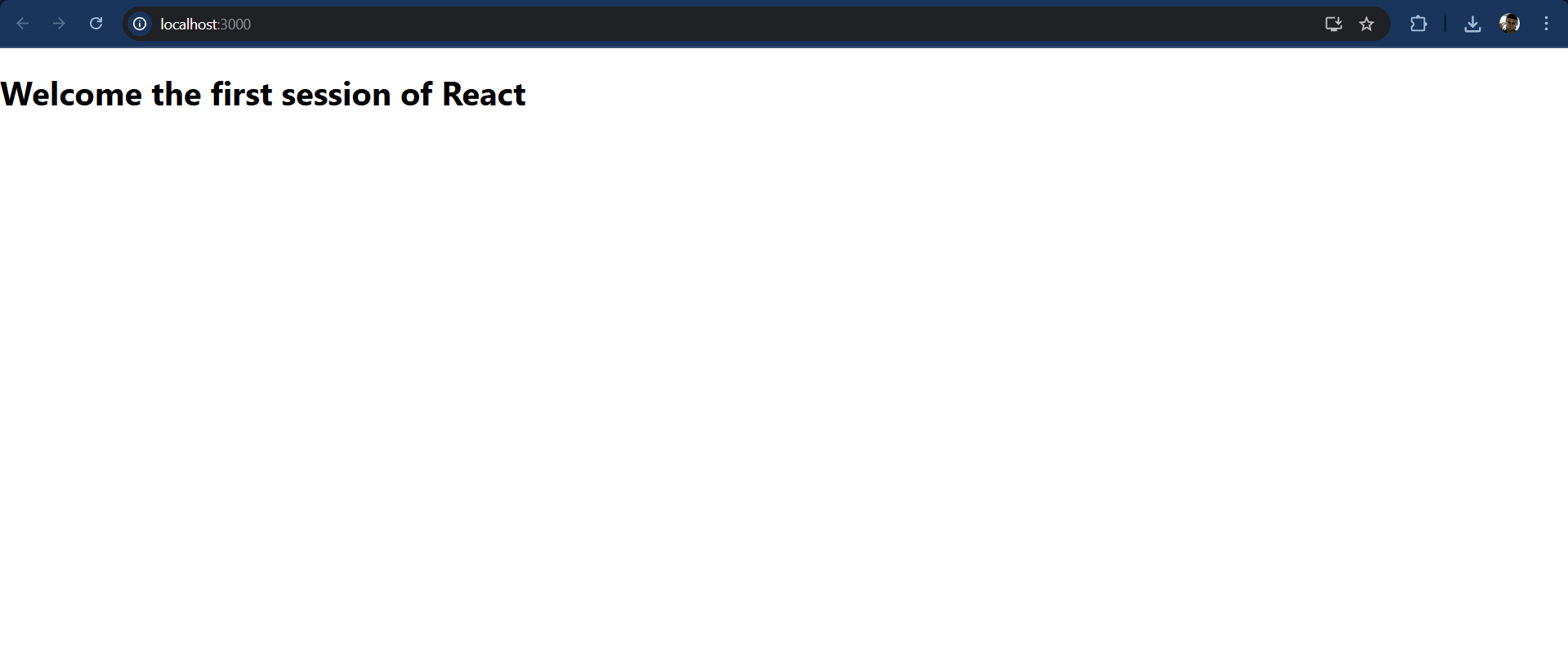
export default App;

**Step 11:** Run the following command to execute the React application:

Code:

npm start

Output:



**React JS – HOL 2**

**Step 1:** Explain React components

In React, React components are independent, reusable building blocks in a React application that define what gets displayed on the UI. They accept inputs called props and return React elements describing the UI.

**Step 2:** Identify the differences between components and JavaScript functions

|  |  |  |
| --- | --- | --- |
| **Feature** | **Component** | **JavaScript Function** |
| **Purpose** | Builds and renders UI elements | Performs logic or calculations |
| **Return Value** | Returns JSX/UI elements | Returns data (e.g., numbers, strings) |
| **Framework Dependent** | Yes (e.g., React, Vue) | No |
| **State Management** | Can manage state (e.g., useState) | Cannot manage state directly |
| **Lifecycle Hooks** | Supports lifecycle methods | No lifecycle awareness |

**Step 3:** Identify the types of components

1. Functional Components
2. Class Components

**Step 4:** Explain class component

Class components are ES6 classes that extend React.Component. They include additional features like state management and lifecycle methods.

* State Management: State is managed using the this.state property.
* Lifecycle Methods: Includes methods like componentDidMount, componentDidUpdate, etc.

**Step 5:** Explain function component

Functional components are simpler and preferred for most use cases. They are JavaScript functions that return React elements. With the introduction of React Hooks, functional components can also manage state and lifecycle events.

* Stateless or Stateful: Can manage state using React Hooks.
* Simpler Syntax: Ideal for small and reusable components.
* Performance: Generally faster since they don’t require a 'this' keyword.

**Step 6:** Define component constructor

In class-based components (e.g., in React), a constructor is a special method used to:

* Initialize state
* Bind event handlers
* Set up the component before it mounts

**Step 7:** Define render() function

Rendering a component refers to displaying it on the browser. React components can be rendered using the ReactDOM.render() method or by embedding them inside other components.

* Ensure the component is imported before rendering.
* The ReactDOM.render method is generally used in the root file.

**Step 8:** Create a React project named “StudentApp” type the following command in terminal of Visual studio:

Code:

npx create-react-app studentapp

**Step 9:** Create a new folder under Src folder with the name “Components”. Add a new file named “Home.js”

Code:

**class** Home **extends** Component{

render() {

return (

<div>

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

</div>

);

}

}

export default Home;

**Step 10:** Repeat the same steps for Creating “About” and “Contact” component by adding a new file as ”About.js”, “Contact.js” under “Src” folder and edit the code as mentioned for “Home” Component.

Code:

**class** About **extends** Component {

render() {

return (

<div>

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

</div>

)

}

}

export default About;

**class** Contact **extends** Component {

render() {

return (

<div>

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

</div>

);

}

}

export default Contact;

**Step 11:** Edit the App.js to invoke the Home, About and Contact component as follows:

Code:

**function** App() {

return (

<div className="container">

<Home />

<About />

<Contact />

</div>

);

}

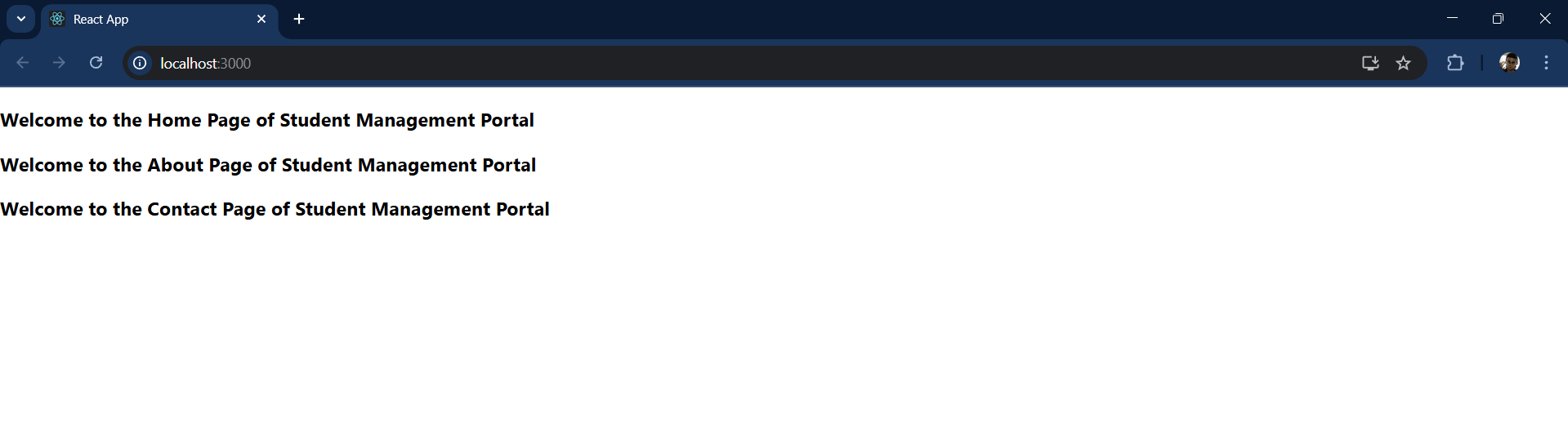
**Step 12:** In command Prompt, navigate into StudentApp and execute the code by typing the following command:

Code:

npm start

**Step 13:** Open browser and type “localhost:3000” in the address bar:

Output:



**ReactJS – HOL 3**

**Step 1:** Create a React project named “scorecalculatorapp” type the following command in terminal of Visual studio:

Code:

npx create-react-app scorecalculatorapp

**Step 2:** Create a new folder under Src folder with the name “Components”. Add a new file named “CalculateScore.js”

Code:

**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>

);

**Step 3:** Create a Folder named Stylesheets and add a file named “mystyle.css” in order to add some styles to the components:

Code:

.Name {

font-weight: 300;

color: blue;

}

.School {

color: crimson;

}

.Total {

color: darkmagenta;

}

.formatstyle {

text-align: center;

font-size: large;

}

.Score {

color: forestgreen;

}

**Step 4:** Edit the App.js to invoke the CalculateScore functional component as follows:

Code:

**function** App() {

return (

<div>

<CalculateScore

Name="Steeve"

School={"DNV Public School"}

total={284}

goal={3}

/>

</div>

);

}

export default App;

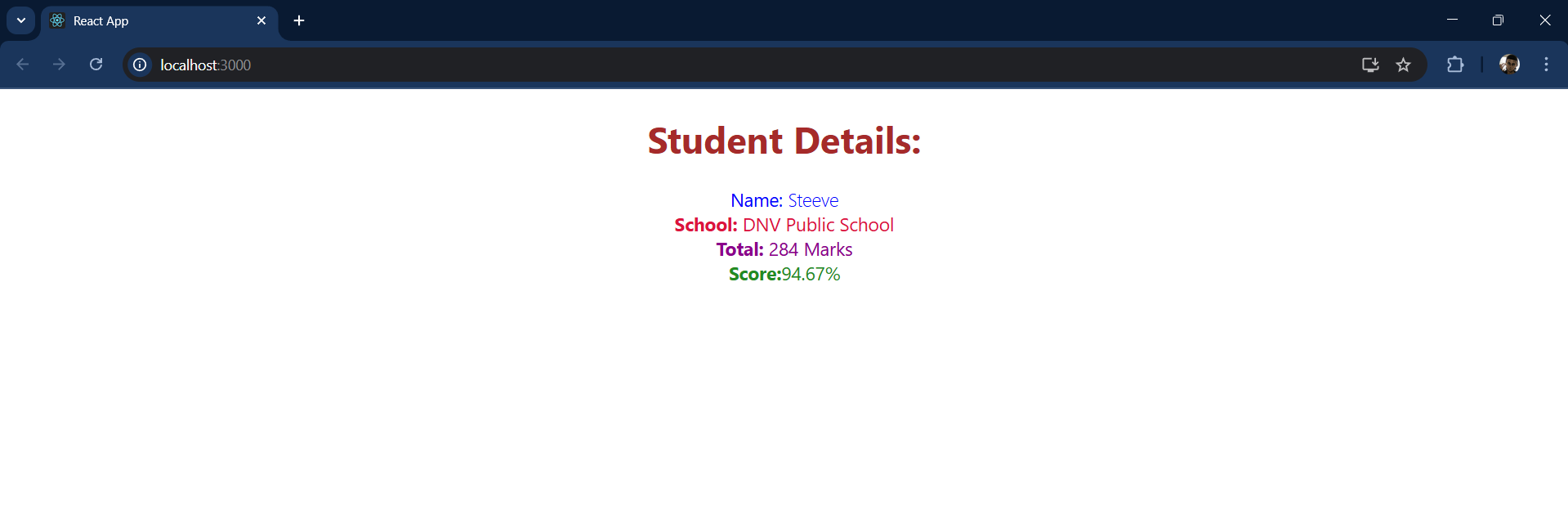
**Step 5:** In command Prompt, navigate into scorecalculatorapp and execute the code by typing the following command:

Code:

npm start

**Step 6:** Open browser and type “localhost:3000” in the address bar:

Output:



**ReactJS – HOL 4**

**Step 1:** Explain the need and Benefits of component life cycle

Need: React components go through a predictable cycle (mounting, updating, unmounting), and we often need to run code at specific stages (e.g., data fetching after mount).

Benefits: Allows better control over when to fetch data, handle errors, optimize performance, and clean up resources.

**Step 2:** Identify various life cycle hook methods

|  |  |
| --- | --- |
| **Mounting** | constructor(), componentDidMount() |
| **Updating** | shouldComponentUpdate(), componentDidUpdate() |
| **Unmounting** | componentWillUnmount() |
| **Error Handling** | componentDidCatch() |

**Step 3:** List the sequence of steps in rendering a component

* constructor()
* render()
* componentDidMount() (after component is rendered to the DOM)
* On update: shouldComponentUpdate() → render() → componentDidUpdate()
* On error: componentDidCatch()

**Step 4:** Create a new react application using create-react-app tool with the name as “blogapp”

Code:  
npx create-react-app blogapp

**Step 5:** Create a new file named as Post.js in src folder with following properties

Code:

**class** Post {

**constructor**(userId, id, title, body) {

this.userId = userId;

this.id = id;

this.title = title;

this.body = body;

}

}

export default Post;

**Step 6:** Create a new class based component named as Posts inside Posts.js file

**class** Posts **extends** Component {

**constructor**(props) {

};

}

}

**Step 7:** Initialize the component with a list of Post in state of the component using the constructor

Code:

**class** Posts **extends** Component {

**constructor**(props) {

super(props);

this.state = {

posts: [],

hasError: false

};

}

}

**Step 8:** Create a new method in component with the name as loadPosts() which will be responsible for using Fetch API and assign it to the component state created earlier. To get the posts use the url (<https://jsonplaceholder.typicode.com/posts>)  
  
Code:

loadPosts = () **=>** {

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

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

.then(data **=>** {

**const** postList = data.map(post **=>** new Post(post.userId, post.id, post.title, post.body));

this.setState({ posts: postList });

})

.catch(error **=>** {

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

this.setState({ hasError: true });

});

};

**Step 9:** Implement the componentDidMount() hook to make calls to loadPosts() which will fetch the posts  
  
Code:

componentDidMount() {

this.loadPosts();

}

**Step 10:** Implement the render() which will display the title and post of posts in html page using heading and paragraphs respectively.

Code:

render() {

if (this.state.hasError) {

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

}

return (

<div>

<h1>Blog Posts</h1>

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

<div key={post.id} style={{ marginBottom: '20px', borderBottom: '1px solid #ccc', paddingBottom: '10px' }}>

<p><strong>Post ID:</strong> {post.id}</p>

<p><strong>User ID:</strong> {post.userId}</p>

<h3>{post.title}</h3>

<p>{post.body}</p>

</div>

))}

</div>

);

}

**Step 11:** Define a componentDidCatch() method which will be responsible for displaying any error happing in the component as alert messages.

Code:

componentDidCatch(error, info) {

alert("An error occurred while rendering posts.");

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

this.setState({ hasError: true });

}

**Step 12:** Add the Posts component to App component.

Code:

**function** App() {

return (

<div>

<Posts />

</div>

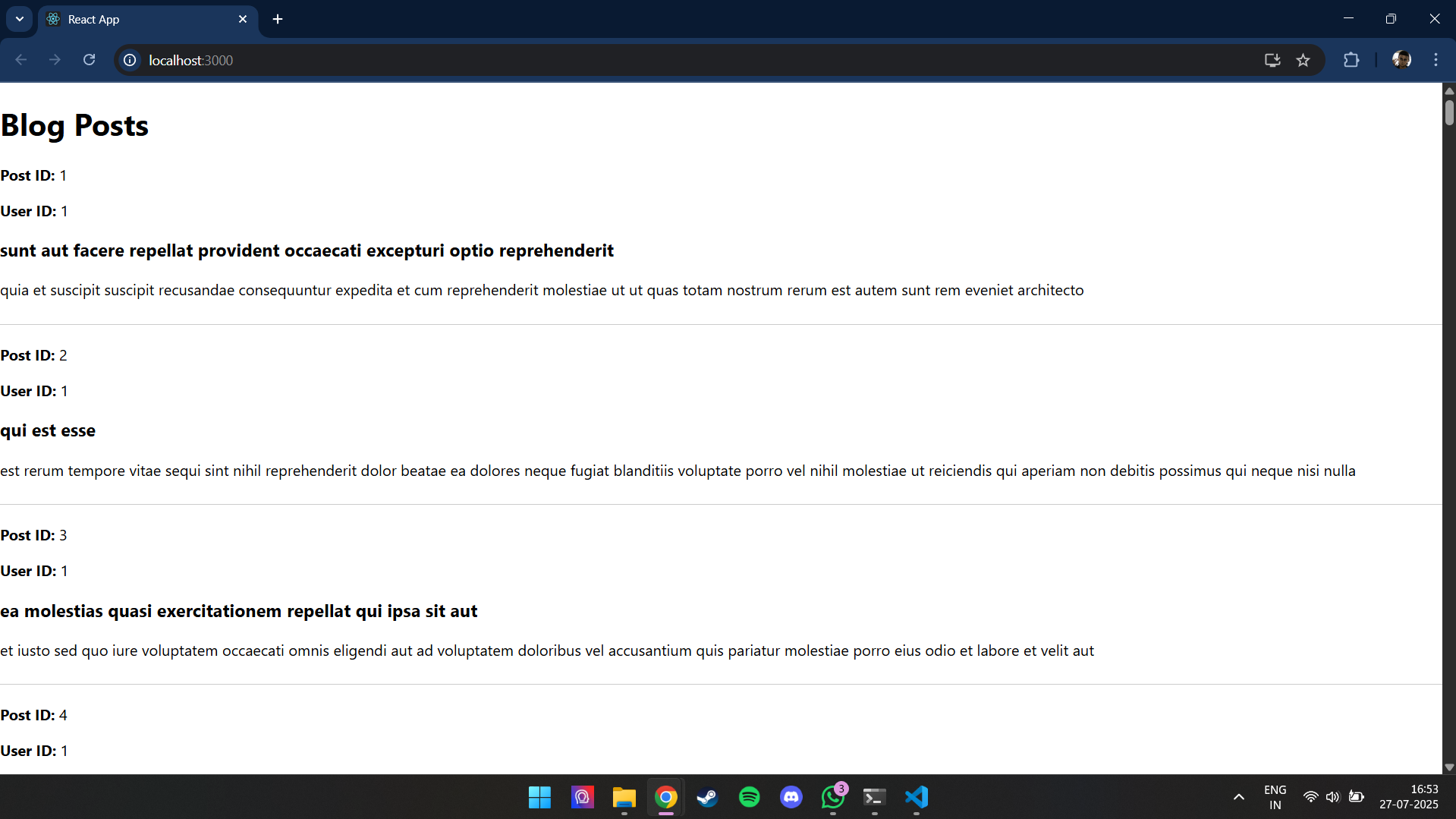
);

}

export default App;

**Step 13:** Build and Run the application using npm start command.

Output:



**ReactJS – HOL 5**

**Step 1:** Open command prompt and switch to the react application folder. Restore the node packages using the following commands.

Code:  
npm install

**Step 2:** Create a new CSS Module in a file called “CohortDetails.module.css”. Define a css class with the name as “box” with following properties.

Code:

.box {

width: 300px;

display: inline-block;

margin: 10px;

padding: 10px 20px;

border: 1px solid black;

border-radius: 10px;

}

**Step 3:** Define a css style for html <dt> element using tag selector. Set the font weight to 500.

Code:

dt {

font-weight: 500;

}

**Step 4:** Open the cohort details component and import the CSS Module.  
  
Code:

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

**Step 5:** Apply the box class to the container div.

Code:

<div className={styles.box}>

**Step 6:** Define the style for <h3> element to use “green” color font when cohort status is “ongoing” and “blue” color in all other scenarios.

Code:

**function** CohortDetails(props) {

**const** statusColor = props.cohort.currentStatus.toLowerCase() === 'ongoing' ? 'green' : 'blue';

return (

<div className={styles.box}>

<h3 style={{ color: statusColor }}>

{props.cohort.cohortCode} -

<span>{props.cohort.technology}</span>

</h3>

<dl>

<dt>Started On</dt>

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

<dt>Current Status</dt>

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

<dt>Coach</dt>

<dd>{props.cohort.coachName}</dd>

<dt>Trainer</dt>

<dd>{props.cohort.trainerName}</dd>

</dl>

</div>

);

}

export default CohortDetails;

**Step 7:** Final Result  
  
Output:

