**WEEK 7 Additional Hands On**

**14. React JS - HOL**

**Objectives**

* Explain the need and Benefits of React Context API

The React Context API is used to manage and share global data across components without the need to pass props manually at every level. This solves the problem of "prop drilling", where data has to be passed down through multiple nested components, even if only a deeply nested component needs it. Context provides a cleaner and more efficient way to manage state or data such as user authentication, themes, or language settings. It helps in building scalable applications by keeping the codebase more organized, reducing redundancy, and improving maintainability.

* Working with createContext()

The createContext() function in React is used to create a new Context object. This object contains two main components: a Provider and a Consumer. The Provider component is used to wrap parts of the component tree that need access to the shared data. It accepts a value prop, which holds the data you want to share

* List the types of Router Components

React Router provides different types of router components for handling navigation. The BrowserRouter is the most commonly used, which uses the HTML5 history API to manage routing through clean URLs. The HashRouter uses the hash portion of the URL (e.g., example.com/#/home) and is suitable for static websites where the server cannot handle dynamic routes. The MemoryRouter keeps the routing information in memory and is typically used for testing or in non-browser environments.

In this hands-on lab, you will learn how to:

* Create a context to be used by child components
* Create a provider and consumer of the context

## **Prerequisites**

The following is required to complete this hands-on lab:

* Node.js
* NPM
* Visual Studio Code

## **Notes**

Estimated time to complete this lab: **30 minutes.**

Developers of Apps Centric Solutions have created an employee management application which supports light and dark themes for the buttons. The current solution uses the react state and props to provide the theme name to be used from App component to Employee List component and from there to Employee Card component. Quality assurance team analyzed the solutions and found the technique being used to be a substandard one. React architect suggested to use the react context API to share the theme name with nested child components instead of passing them down using props from the parent component.

You are assigned the task of converting the application form props only to React Context API.

Application can be downloaded from below



1. Unzip the application and open it using VS Code
2. Go to terminal and execute *npm install* command to restore all the node modules



Figure 1: Restore node modules

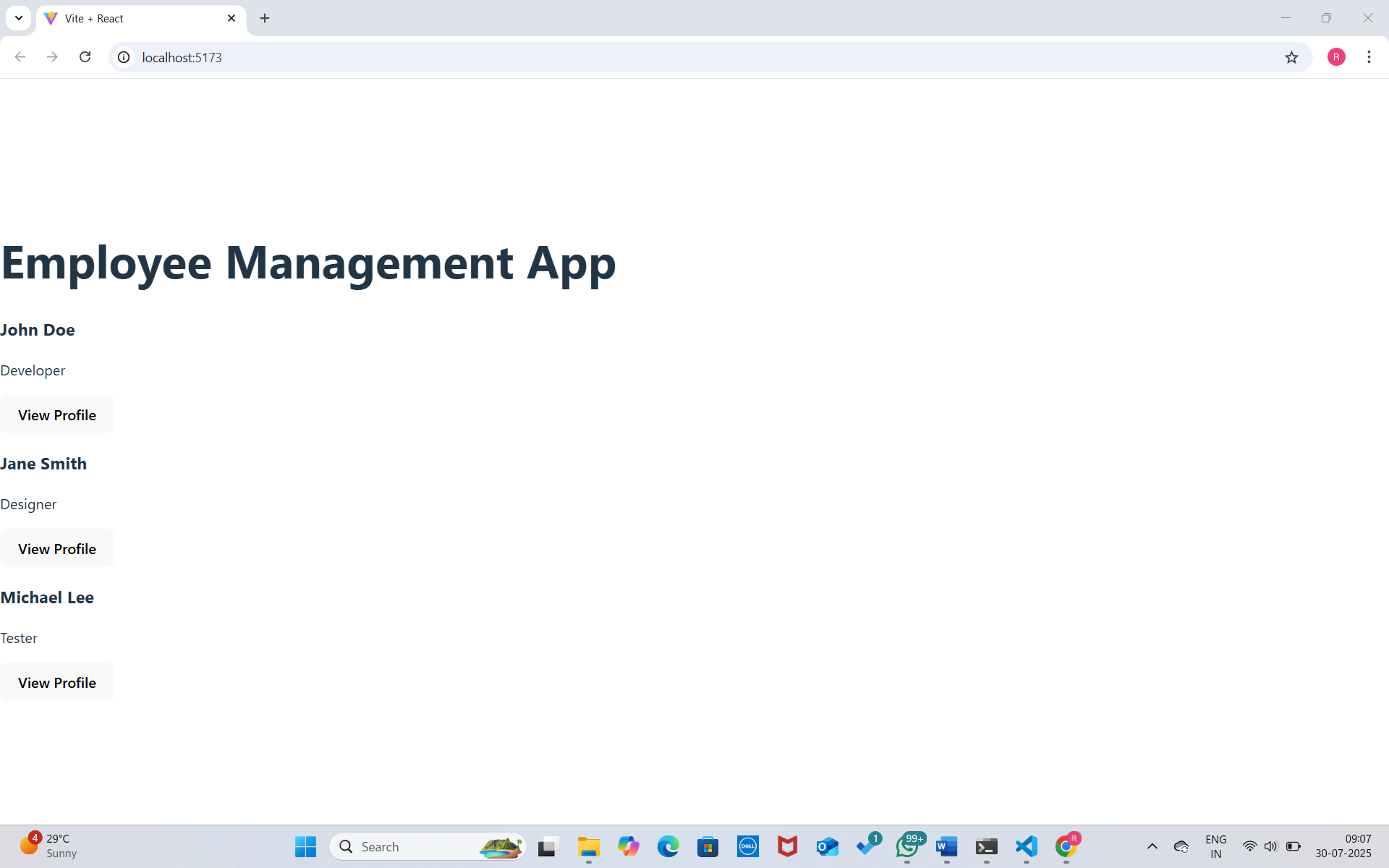
1. Run the application once to see the output. Use npm start command.



Figure 2: Starting application

1. Explore the components present in **App.js**, **EmployeesList.js** and **EmployeeCard.js** files.
2. Create a new file with the name as **ThemeContext.js**. Define a new context in the file with the name as ThemeContext and assign it a default value of ‘light’ and export it as default form the module.
3. Open App component present in **App.js** file.
   1. Import the ThemeContext in App component.
   2. Define the theme context provider to be the entire JSX of the App component.
   3. Assign the value for the theme provider from the state of the component.
   4. Modify the call to EmployeeList component so that theme name is no longer passed as props.
4. Go to EmployeeList component present in **EmployeeList.js** file and modify it so that theme name is not passed explicitly to its child component.
5. Go to **EmployeeCard** component inside **EmployeeCard.js** file
   1. Import the ThemeContext into the component file
   2. Retrieve the value of the context with the help of **useContext()** and store it in a variable
   3. Use the variable to pass the className for the buttons.

**OUTPUT:**



**15. React JS - HOL**

**Objectives**

* **Explain about React forms**

React Forms are used to collect user input in a web application built with React. Unlike traditional HTML forms, React forms are typically managed using the component's state. This allows the form data to be controlled and dynamically updated as the user interacts with the form fields. React provides flexibility and control by enabling real-time validation, dynamic input rendering, and more predictable form behavior.

* **Define controlled components**

Controlled Components in React are input elements (like <input>, <textarea>, or <select>) whose values are controlled by the React state. In this approach, the value of each form element is bound to the state variable, and any user input is handled by updating the state via event handlers like onChange. This gives React full control over the form data, making it easier to validate, conditionally render, or manipulate inputs.

* **Explain about various input controls**

Various Control Inputs in React include standard HTML elements such as text boxes (<input type="text">), password fields (<input type="password">), radio buttons (<input type="radio">), checkboxes (<input type="checkbox">), dropdowns (<select>), and text areas (<textarea>). Each of these inputs can be made into controlled components by binding them with state and handling their respective change events.

* **Explain about handling forms**

Handling Forms in React typically involves managing form inputs via state and defining functions to handle input events. The onChange event is commonly used to capture user input and update the component’s state accordingly. Handling forms also includes managing validation logic, enabling or disabling buttons, and providing error messages based on the current state of the form.

* **Explain about submitting forms**

Submitting Forms in React involves capturing the form's submit event using the onSubmit handler. In the handler function, the default form submission (which refreshes the page) is usually prevented using event.preventDefault(). The collected data from the form's state can then be processed, sent to a server, or used in other application logic.

**In this hands-on lab, you will learn how to:**

* **Implement React forms**
* **Use various input controls like textbox, button and textarea**

**Prerequisites**

**The following is required to complete this hands-on lab:**

* **Node.js**
* **NPM**
* **Visual Studio Code**

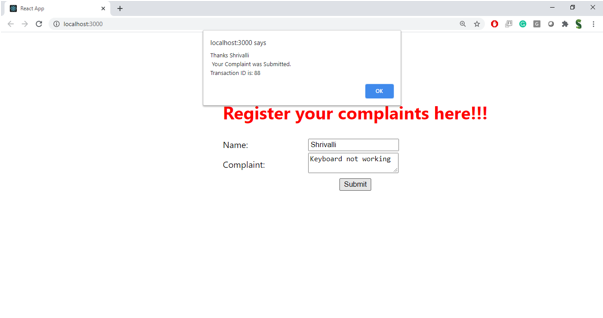
**Notes**

**Estimated time to complete this lab: 60 minutes.**

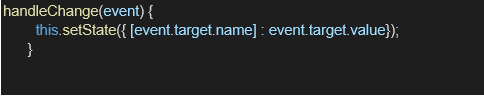
**Create a React App named “ticketraisingapp” which will help to raise a complaint and get it resolved.**

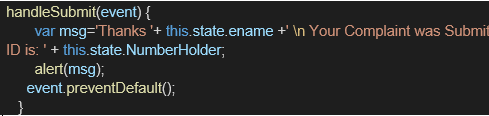
**Create a component named “ComplaintRegister” with a form containing a textbox to enter the employee name and a textarea to enter the complaint. Use “handleSubmit” event of the button to submit the complaint and generate a Reference number for further follow ups in the alert box.**

**Output Expected:**

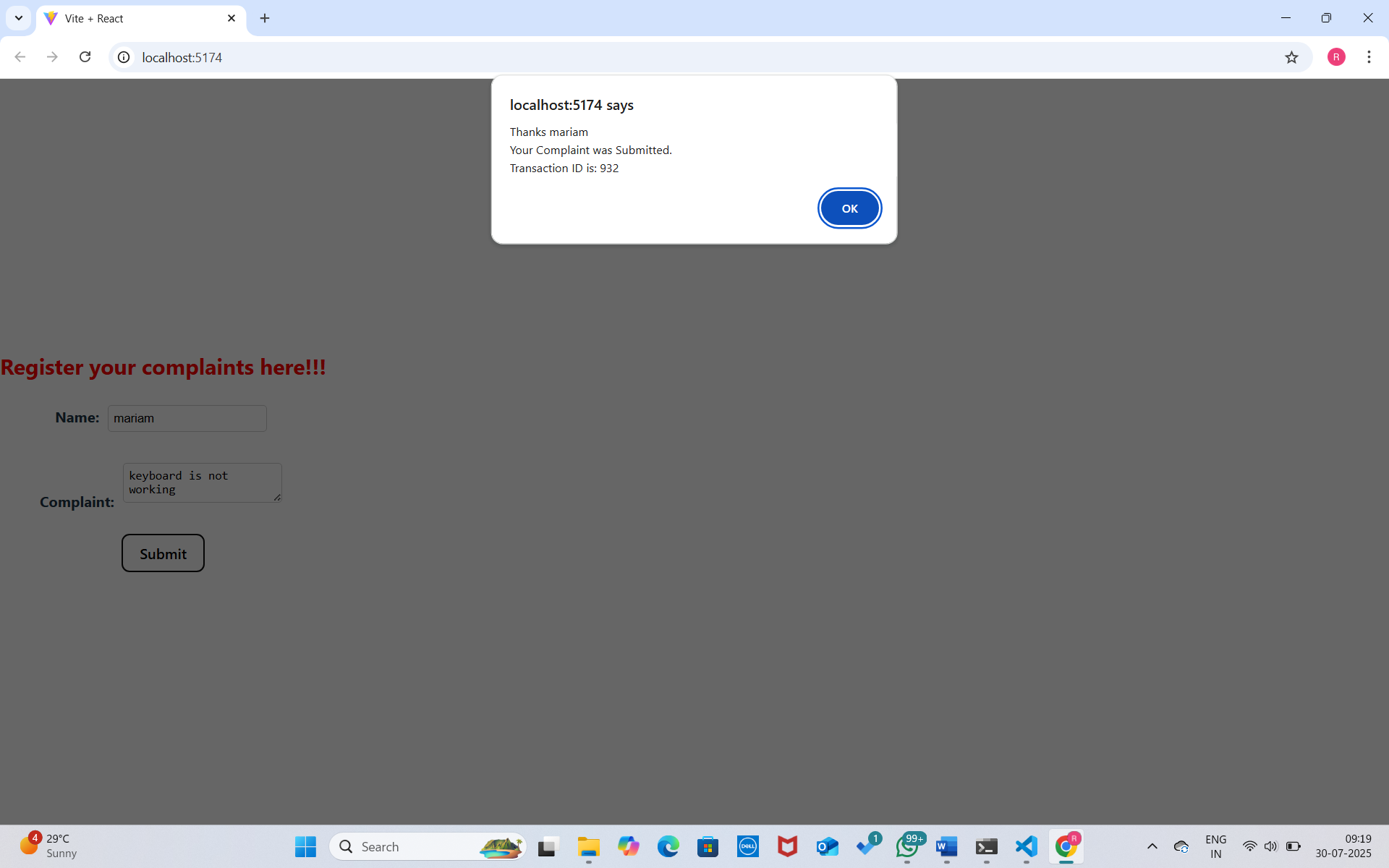
****

**Hint:**

****

****

**OUTPUT:**



**16. React JS - HOL**

**Objectives**

* **Explain React Forms validation**

React forms validation is the process of checking user inputs in a form to ensure that they meet specific criteria before the data is submitted or processed. Validation can be done using simple JavaScript checks or with external libraries like Formik or React Hook Form. In React, validation typically happens in the component’s state and is triggered during form events like onChange or onSubmit. Common validation tasks include checking if fields are empty, validating email formats, or enforcing password rules.

* **Identify the differences between React Form and HTML Form**

While HTML forms rely on the browser’s built-in form handling and validation, React forms are more dynamic and controlled through JavaScript and component state. In a traditional HTML form, inputs are managed by the DOM, whereas in React, form data is managed through the component’s state, allowing more flexible and controlled interactions. React forms provide better control over user input, custom validation, and integration with other app logic, while HTML forms are more limited in their customization and interactivity.

* **Explain about controlled components**

Controlled components in React are form elements like <input>, <textarea>, and <select> whose values are controlled by the React state. Instead of the form element managing its own state, React does this via the value attribute and onChange handler. This approach ensures that the form input values are always in sync with the component state, making it easier to implement validation, dynamic input behavior, and form submission logic.

* **Identify various React Form input controls**

React supports various form input controls, including text inputs (<input type="text">), password fields, checkboxes, radio buttons, select dropdowns, and textareas. These inputs can be controlled or uncontrolled and can be customized to handle different types of user data. Each control works with onChange events and state updates to capture and manage user interactions within a React component.

* **Explain how to handle React Forms**

Handling forms in React involves managing form state using React’s useState hook or class component state. The form data is captured through onChange events and stored in state variables. Developers can write handler functions to update state, validate inputs, and process form data as needed. This approach allows for better control over input values and a more dynamic user experience compared to static HTML forms.

* **Explain about submitting forms in React**

Form submission in React is typically managed through an onSubmit handler on the <form> element. Instead of allowing the default HTML form submission behavior (which reloads the page), React uses event.preventDefault() to stop the page refresh and then processes the form data manually. This allows developers to validate input, send data to a server using APIs, or perform other actions like showing success messages or clearing the form.

**In this hands-on lab, you will learn how to:**

* **Implement React forms validation**
* **Use various input controls like textbox, button and textarea**

**Prerequisites**

**The following is required to complete this hands-on lab:**

* **Node.js**
* **NPM**
* **Visual Studio Code**

**Notes**

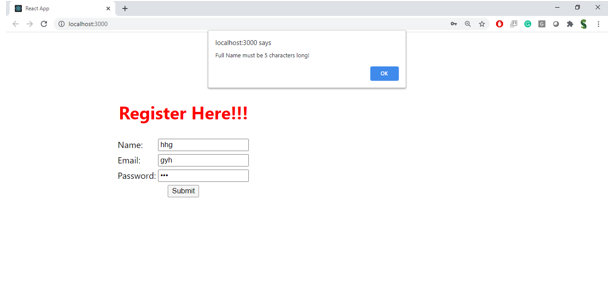
**Estimated time to complete this lab: 60 minutes.**

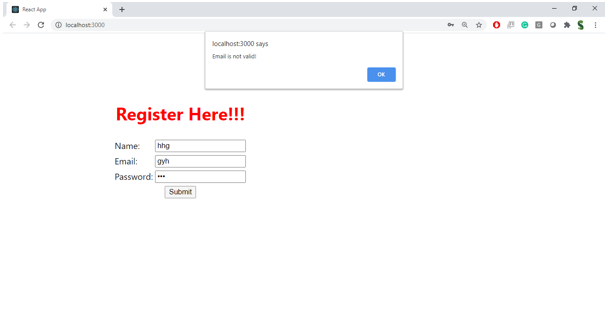
**Create a React App named “mailregisterapp” which will have a component named “register.js”. Create a form which accepts the name, email and password and validate the fields as per the following:**

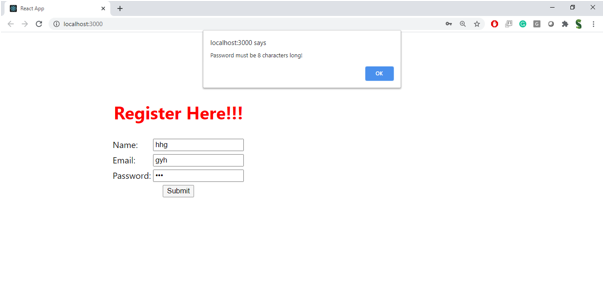
1. **Name should have atleast 5 characters**
2. **Email should have @ and .**
3. **Password should have atleast 8 characters.**

**Ensure that validations are implemented through eventhandle and eventsubmit of a form.**

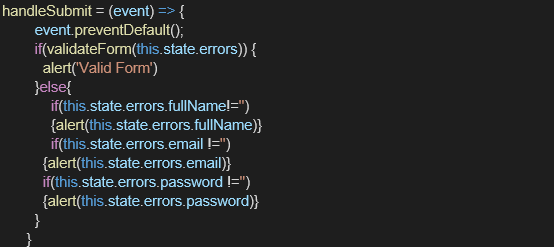
**Output Expected:**

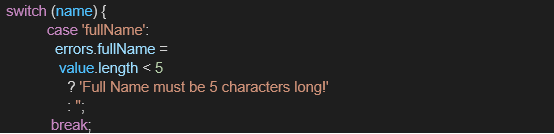
****

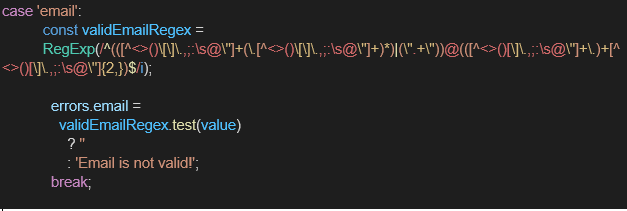
****

****

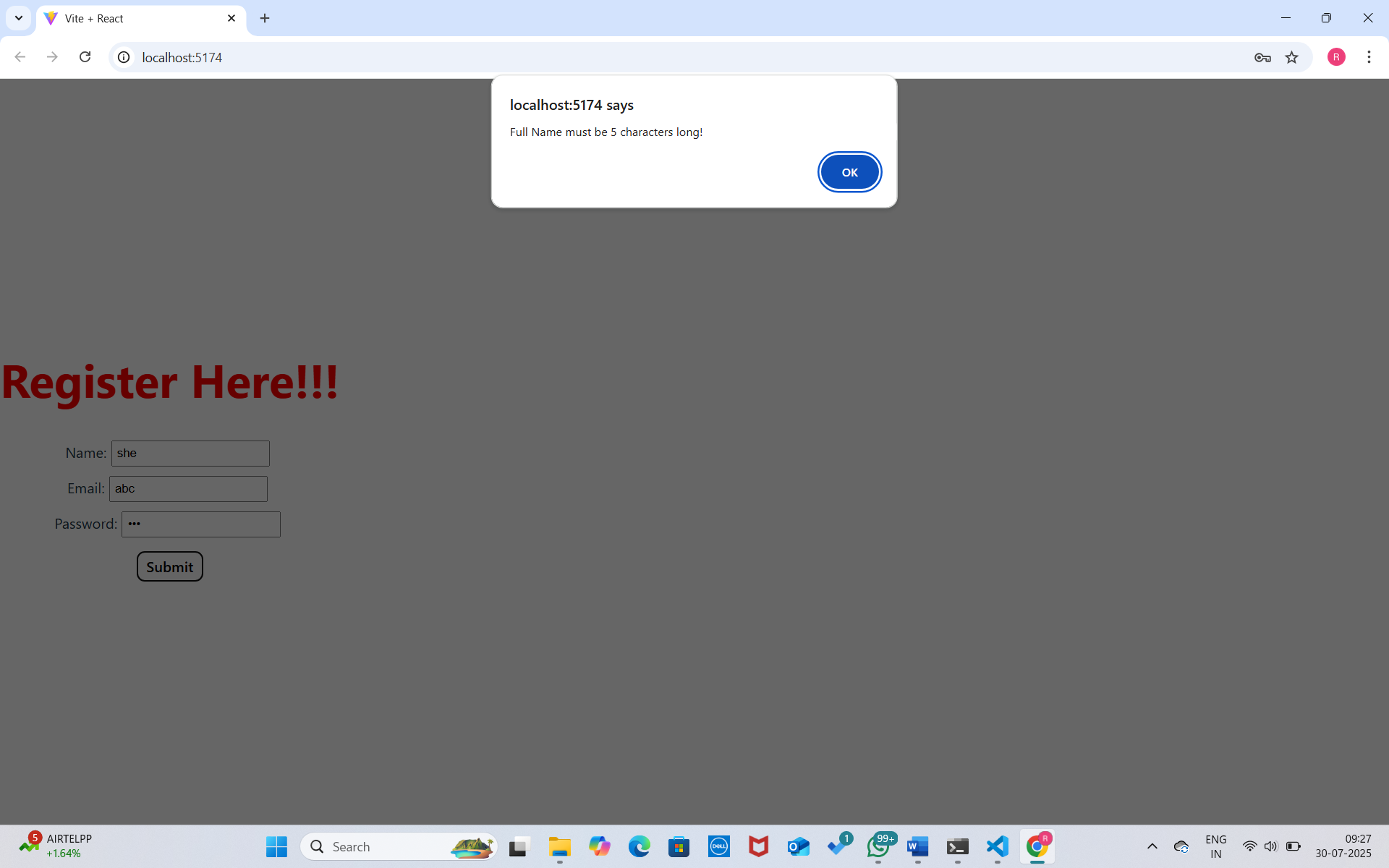
**Hint:**

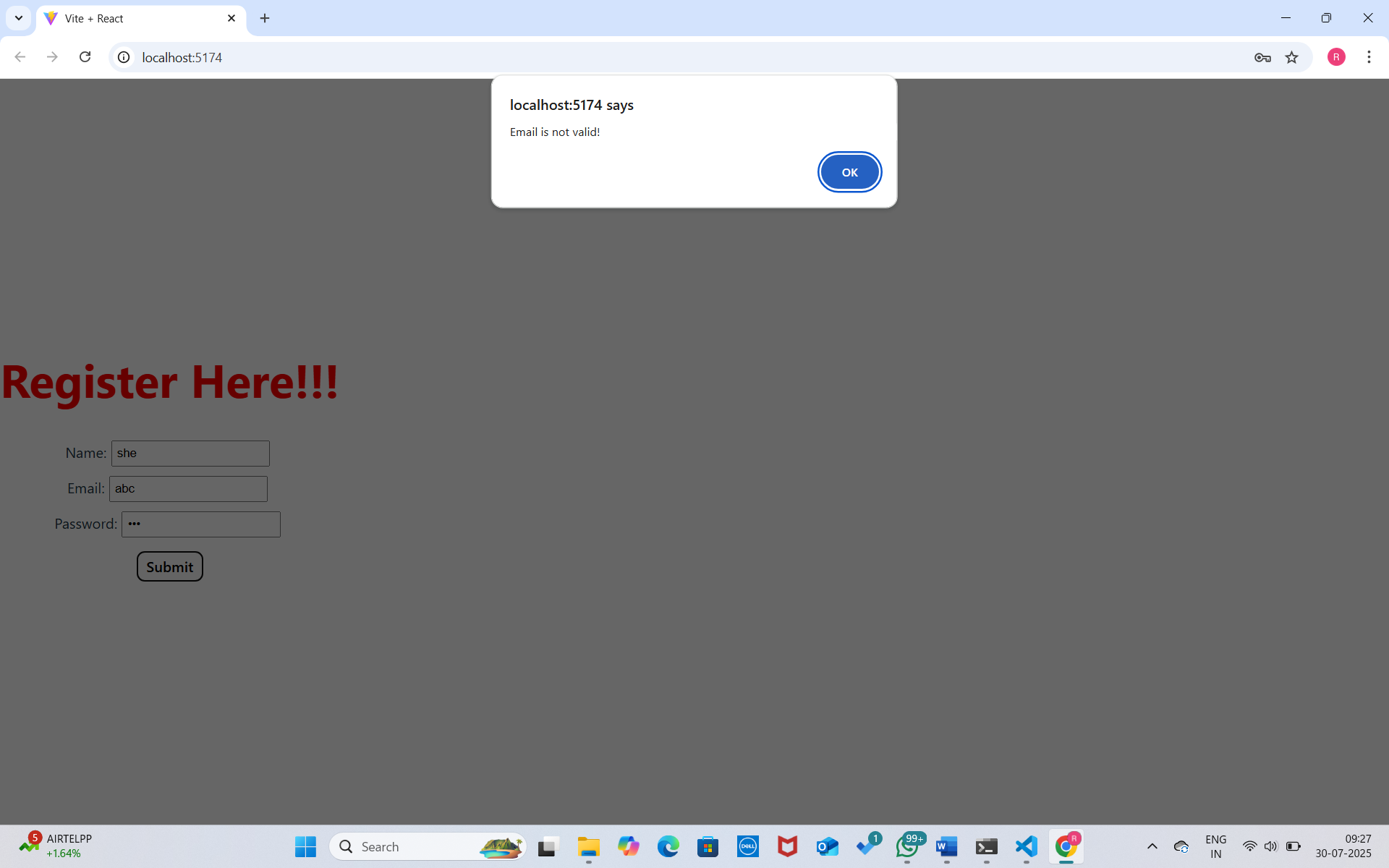
****

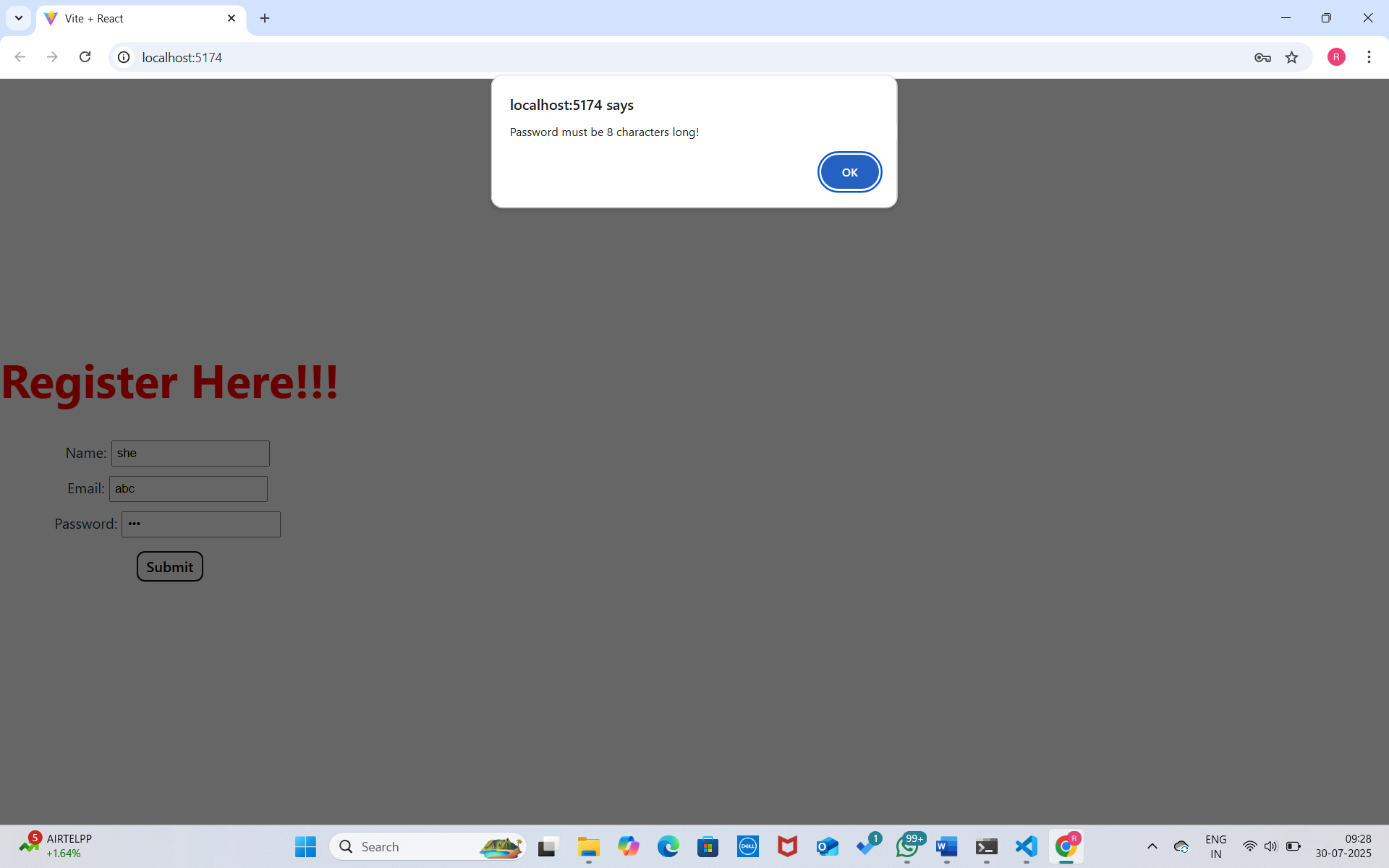
****

****

**OUTPUT:**







**17. React JS - HOL**

**Objectives**

* **Explain how to consume REST APIs from React applications**

To consume REST APIs from React applications, developers typically use JavaScript's fetch API or third-party libraries like Axios to make HTTP requests to external services. The process generally involves calling an API endpoint from a React component—often within lifecycle methods such as useEffect (for functional components) or componentDidMount (for class components)—and then updating the component's state with the received data. This allows the UI to render dynamically based on real-time information from the backend. It's important to handle asynchronous operations using promises or async/await syntax and to implement proper error handling for a smooth user experience. Additionally, tools like React Query or SWR can simplify data fetching by managing caching, revalidation, and background updates, making the application more efficient and responsive.

**In this hands-on lab, you will learn how to:**

* **Construct a React application that invokes the REST API and fetch data from the API**

**Prerequisites**

**The following is required to complete this hands-on lab:**

* **Node.js**
* **NPM**
* **Visual Studio Code**

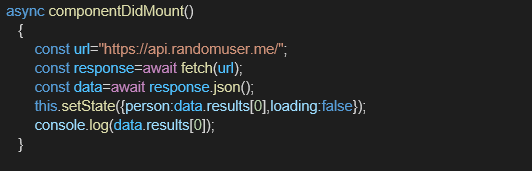
**Notes**

**Estimated time to complete this lab: 60 minutes.**

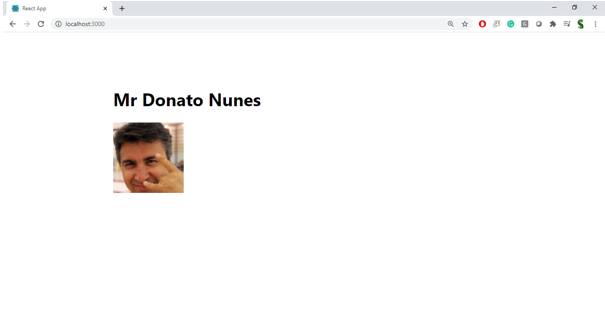
**Create a React Application “fetchuserapp” which will retrieve the user details from** [**https://api.randomuser.me/**](https://api.randomuser.me/) **and display the title, firstname and image of a user.**

**Create a component named “Getuser” and in the asynchronous method “ComponentDidMount ()” invoke the URL using fetch method and the response can be displayed in the render method of the component.**

**Code Snippet in Getuser Component:**

****

**Expected Output:**

****

**OUTPUT:**

