**PART A**

**Experiment No. 06**

**A.1 Aim:** Create react application to implement components with state and hook.

**Objective:** To learn and understand how to pass prop and create reusable components that structure your app.

**A.2 Prerequisite:**

HTML, CSS, Javascript

**A.3 Outcome:**

After successful completion of these applications students will be able to

Understand and implement react library as a front end.

**A.4 Theory**

**React**.js is a popular JavaScript library for building user interfaces, particularly single-page applications where you need a fast, interactive user experience. It's developed by Facebook and allows developers to create large web applications that can change data, without reloading the page. The main goal is to be fast, scalable, and simple. It works only on user interfaces in the application; this corresponds to the view in the MVC template.

**Components**: React is all about components. You build your UIs out of components. Components are reusable and can be nested within each other. A React application is typically a tree of components. They are of 2 types: functional and class

**JSX**: JSX is a syntax extension for JavaScript. It looks like HTML but works inside JavaScript. React uses JSX to describe UIs. JSX is optional and not required to use React, but it's recommended as it visually resembles the output it generates.

**Props**: Short for properties, props are how you pass data from a parent component to a child component. Props are read-only and should not be modified by the child component.Tasks to be completed

**State**: State offers a way to store and manage data that changes over time in a component. Unlike props, the state is local to the component and can be changed from within the component.

**Lifecycle Methods**: React components have several lifecycle methods that you can override to run code at particular times in the process. Examples include mounting, updating, and unmounting phases.

**Hooks**: Introduced in React 16.8, hooks are functions that let you “hook into” React state and lifecycle features from function components. They make it possible to use state and other React features without writing a class.

Example

import React, { useState } from 'react';

function ExampleComponent() {

// useState hook to add state to the function component

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

return (

<div>

<p>You clicked {count} times</p>

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

Click me

</button>

</div>

);

}

export default ExampleComponent;

Tasks to be completed:

**Implementing a React Registration Form with State Management and Hooks**

You are a front-end developer at a company tasked with creating a user-friendly registration form for an upcoming web application. The form should allow users to input their name, email, and password. To ensure the form is dynamic and interactive, you will leverage React.js for component creation, state management using hooks, and basic CSS styling.

Create a react js registration form to demonstrate component creation and rendering along with mainating of state using hooks. (Use CSS wherever required)

**Problem Statement:**

Your goal is to design a simple yet functional registration form using React.js. The form must:

1. Allow users to input their **name**, **email**, and **password**.
2. Utilize **React hooks** (e.g., useState) to manage and maintain the state of the form inputs dynamically.
3. Display the submitted data in a pop-up alert on form submission.
4. Clear the form fields once the user submits the form successfully.
5. Incorporate **CSS styling** to make the form aesthetically pleasing and user-friendly.
6. Ensure proper validation of input fields, such as making fields required for submission.

**Task:**

As part of the case study, develop a complete React application that fulfills the following requirements:

* Create a **functional React component** to render the registration form.
* Use the useState hook to manage the state of form fields (name, email, password).
* Handle form submission by preventing default browser behavior and alerting the user with the input data.
* Reset the form to its initial state upon successful submission.
* Add **CSS styling** for form layout and design, ensuring it is responsive and easy to interact with.

**Key Points:**

1. **Component creation**: Design a single functional component for the registration form.
2. **State management**: Use hooks for maintaining the state of user input.
3. **Event handling**: Implement form submission and state updates via event handlers.
4. **User interaction**: Ensure proper validation and feedback mechanisms are in place.
5. **Styling**: Use basic CSS to style the form components for better user experience.

**Deliverables:**

1. A working React application with a functional registration form.
2. CSS file for form styling.
3. A demonstration of how state is updated and maintained using hooks.
4. Form validation with an alert box showing user inputs on form submission.

**PART B**

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)

|  |  |
| --- | --- |
| Roll No. : S020 | Name: Husain Chhil |
| Class : MBA.Tech DS | Batch : J1 |
| Date of Experiment : 10.10.24 | Date/Time of Submission : 10.10.24 |
| Grade : |  |

**B.1 Code:**

App.js

import React from "react";

import RegistrationForm from "./RegistrationForm";

import "./App.css"; // We'll create a separate CSS file for styling

function App() {

  return (

    <div *className*="app-container">

      <h1>Register</h1>

      <*RegistrationForm* />

    </div>

  );

}

export default App;

RegistrationForm.js

import React, { useState } from "react";

function RegistrationForm() {

  // Step 1: Set up the form fields state using useState hook

  const [formData, setFormData] = useState({

    name: "",

    email: "",

    password: "",

  });

  // Step 2: Handle input changes

  const handleChange = (*e*) => {

    const { name, value } = *e*.target;

    setFormData((*prevData*) => ({

      ...*prevData*,

      [name]: value,

    }));

  };

  // Step 3: Handle form submission

  const handleSubmit = (*e*) => {

*e*.preventDefault(); // Prevent default form submission behavior

    alert(`Name: ${formData.name}\nEmail: ${formData.email}\nPassword: ${formData.password}`);

    // Reset form state after successful submission

    setFormData({

      name: "",

      email: "",

      password: "",

    });

  };

  return (

    <form *className*="registration-form" *onSubmit*={handleSubmit}>

      <div>

        <label *htmlFor*="name">Name:</label>

        <input

*type*="text"

*id*="name"

*name*="name"

*value*={formData.name}

*onChange*={handleChange}

*required*

        />

      </div>

      <div>

        <label *htmlFor*="email">Email:</label>

        <input

*type*="email"

*id*="email"

*name*="email"

*value*={formData.email}

*onChange*={handleChange}

*required*

        />

      </div>

      <div>

        <label *htmlFor*="password">Password:</label>

        <input

*type*="password"

*id*="password"

*name*="password"

*value*={formData.password}

*onChange*={handleChange}

*required*

        />

      </div>

      <button *type*="submit">Register</button>

    </form>

  );

}

export default RegistrationForm;

index.js

import React from 'react';

import ReactDOM from 'react-dom/client';

import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

    <*App* />

);

App.css

*.app-container* {

  display: flex;

  flex-direction: column;

  align-items: center;

  justify-content: center;

  height: 100vh;

  font-family: Arial, sans-serif;

}

h1 {

  color: #333;

}

*.registration-form* {

  background-color: #f9f9f9;

  padding: 30px;

  border-radius: 10px;

  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

  max-width: 400px;

  width: 100%;

}

*.registration-form* div {

  margin-bottom: 15px;

  margin-right: 15px;

}

*.registration-form* label {

  display: block;

  margin-bottom: 5px;

  color: #555;

}

*.registration-form* input {

  width: 100%;

  padding: 10px;

  border-radius: 5px;

  border: 1px solid #ccc;

}

button {

  width: 100%;

  padding: 10px;

  background-color: #007bff;

  color: white;

  border: none;

  border-radius: 5px;

  cursor: pointer;

  font-size: 16px;

}

button*:hover* {

  background-color: #0056b3;

}

@media (max-width: 600px) {

*.registration-form* {

    padding: 15px;

  }

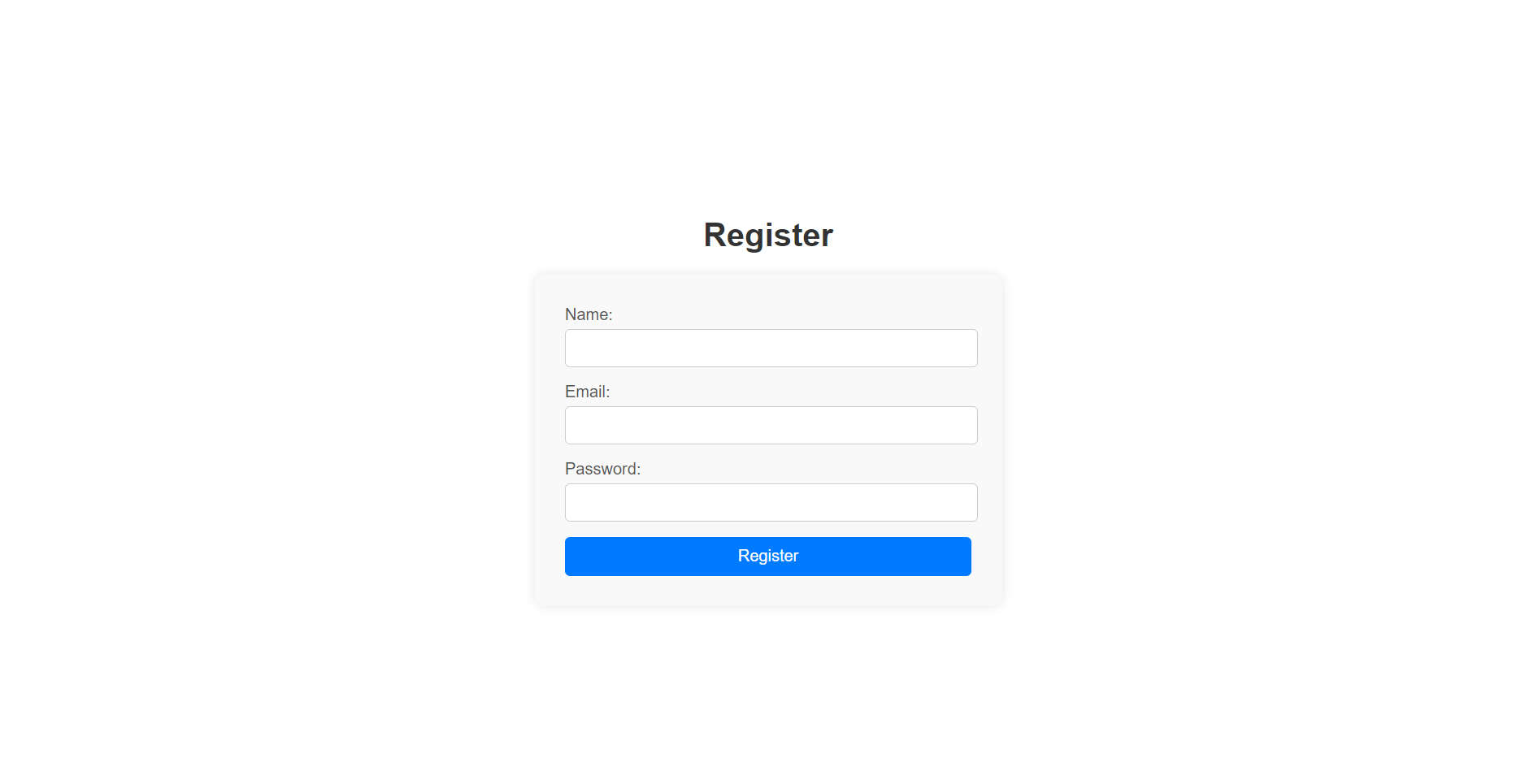
  button {

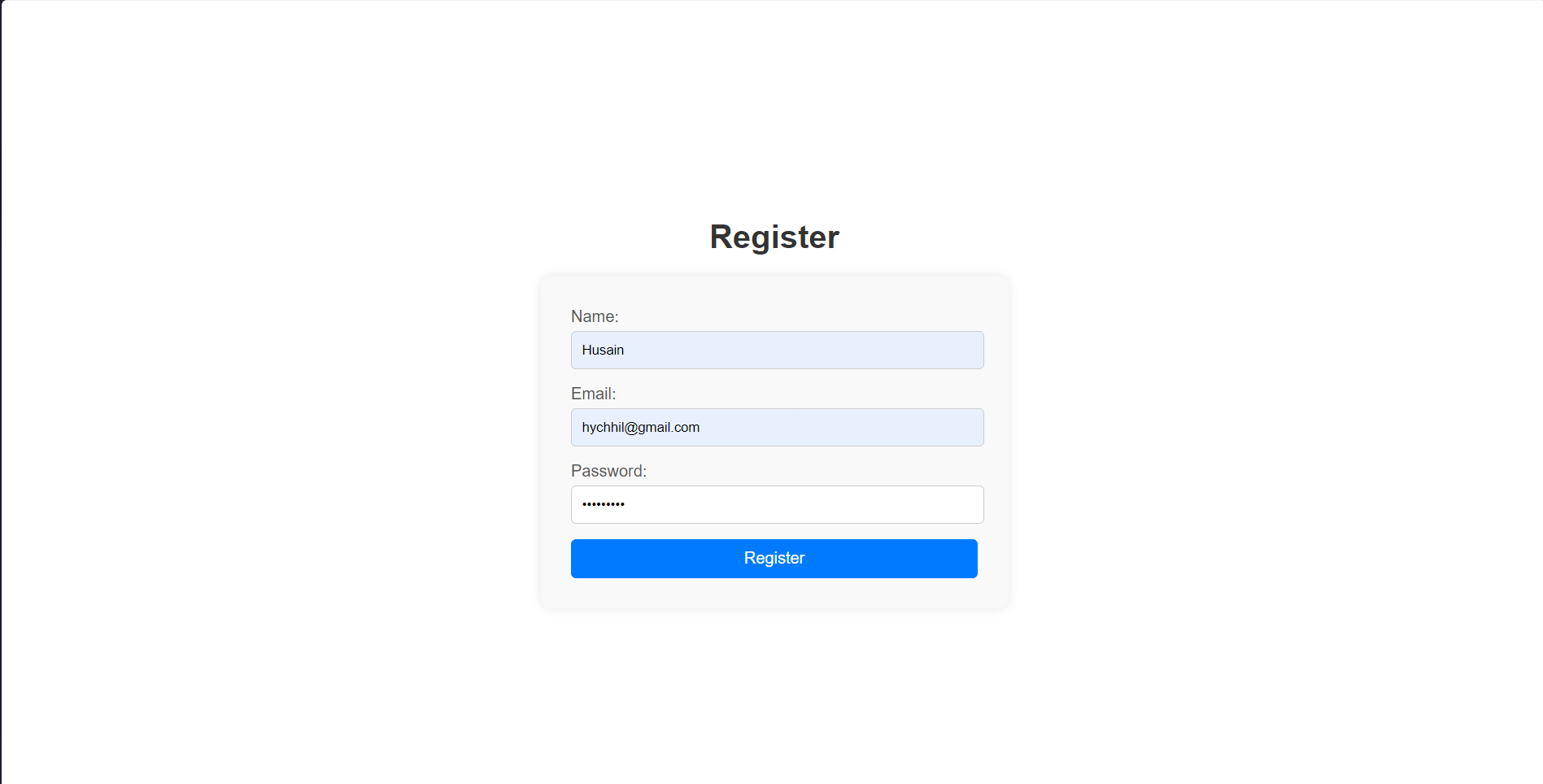
    font-size: 14px;

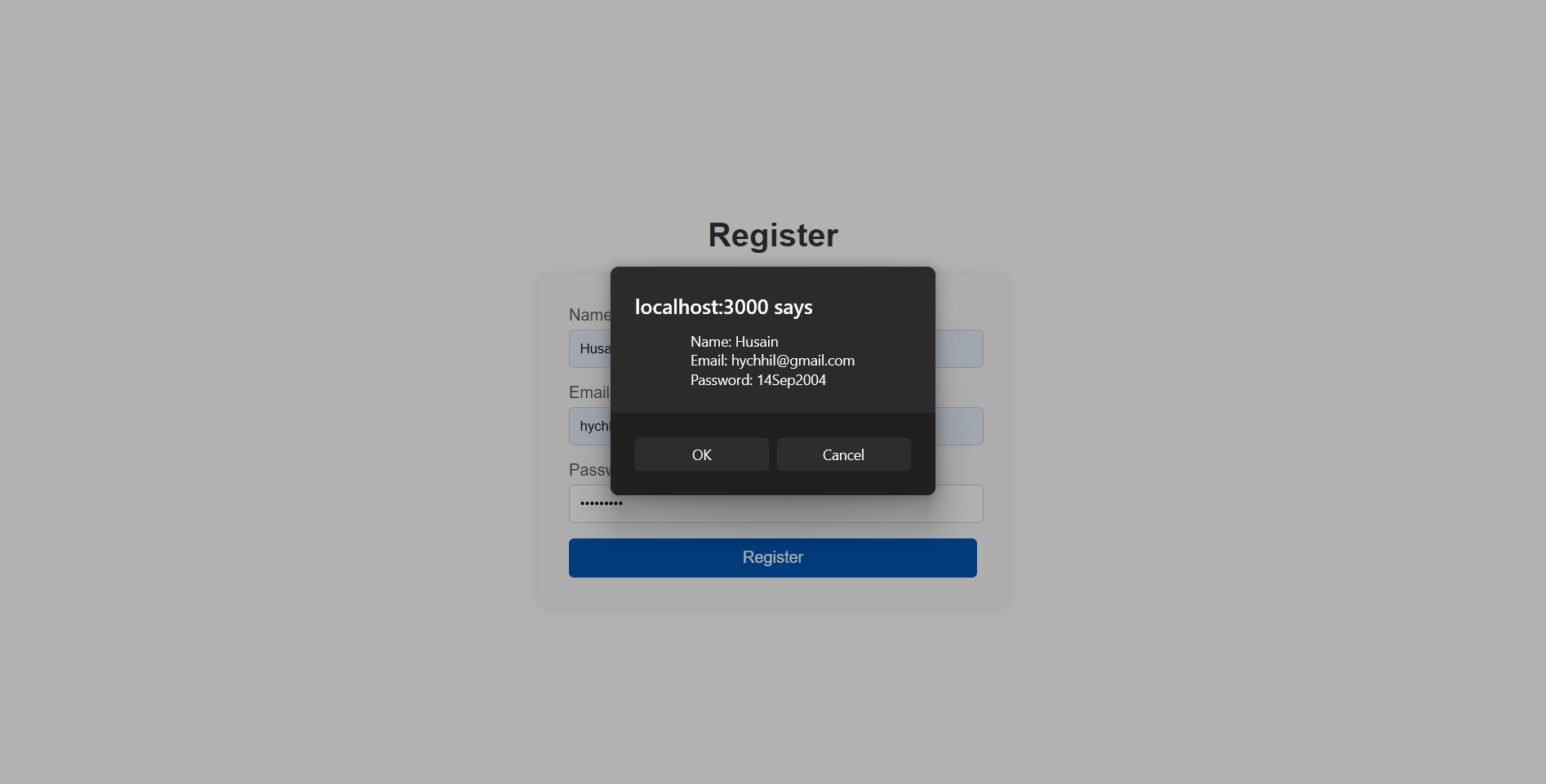
  }

}

**B.2 Output**



****

****

B.3 Conclusion:

In this practical, we developed a functional React component for a registration form that manages form data using useState and handles form submission with validation. The outcomes of this practical include gaining hands-on experience with state management, event handling, and integrating CSS for user-friendly interface design. The project successfully demonstrates how to implement dynamic forms and user interaction in React.

B.3 Observations and Learning:

Component Creation: React components are reusable pieces of the UI. I observed how functional components enable modular, maintainable code.

State Management: Using the useState hook is essential to track changes in form data. I learned how state updates are re-rendered to reflect the user's input dynamically.

Form Submission Handling: Proper form handling in React prevents page refreshes and enables controlled form submission. This reinforces how React manages user interaction smoothly.

Styling Impact: CSS plays a crucial role in enhancing the form’s visual appeal and usability. Styling contributes to a responsive and accessible design.

B.4 Question of Curiosity:

What is a component in React?

A component in React is a reusable, self-contained piece of the user interface. It can manage its own state and render HTML based on data passed to it, known as props.

What are props and how are they used in this app?

Props (short for "properties") are inputs to components, allowing data to be passed from one component to another. In this app, props could be used to pass initial form values or functions that handle events like form submission.

How do we manage form data using React hooks?

We manage form data using the useState hook. Each form field (like name, email, password) has its own state, which updates as the user inputs data. This state is then used to display and process the form information.

Why is state important in this application?

State is crucial because it stores the form data and re-renders the component every time the data changes. Without state, the form wouldn't be able to dynamically reflect user input or track the form's current values.

How would you handle form submission and validation in this app?

On form submission, I would prevent the default browser behavior using event.preventDefault(), validate the input values (like checking if fields are empty or if email follows the correct format), and alert the user with the form data. If valid, the form can be reset to its initial state.

How can CSS be used to enhance user experience in this React form?

CSS can be used to create a visually appealing layout, ensure responsive design on various devices, style form elements like buttons and input fields, and provide feedback (e.g., changing border color on invalid input). These enhancements improve user interaction and accessibility.

What improvements could be made to this form?

Improvements could include adding real-time validation (e.g., showing error messages as the user types), handling errors more gracefully, and providing more advanced styling and transitions to improve the user experience. Additionally, integrating a backend for storing the form data could make it more functional.