2) Develop a React application that demonstrates the use of props to pass data from a parent component to child components. The application should include the parent component named App that serves as the central container for the application. Create two separate child components, Header: Displays the application title or heading. Footer: Displays additional information, such as copyright details or a tagline. Pass data (e.g., title, tagline, or copyright information) from the App component to the Header and Footer components using props. Ensure that the content displayed in the Header and Footer components is dynamically updated based on the data received from the parent component.

Footer.js

```
function  Footer(props)
{
   <div><p>Name :{props.tagline}</p></div>
}
export default Footer
```

Header.js

```
function Header(props)
{
   <div><p>Name :{props.title}</p>  </div>
}
export default Header
```

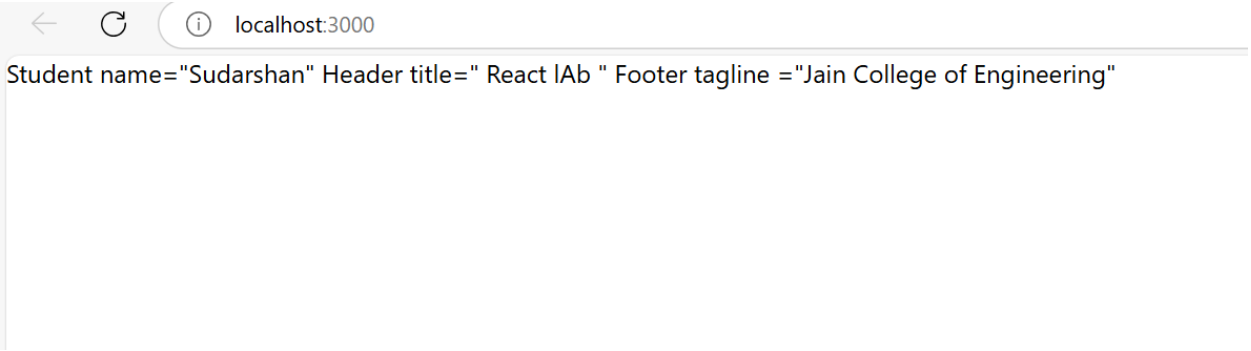
Student.js

```
function Student(props)
{
   <div><p>Name :{props.name}</p></div>
}
export default Student
```

App.js

```
import  Student from './Student.jsx'
import Header from './Header.jsx'
import Footer from './Footer.jsx'
function App() {
  return (
    <>
     Student name="Sudarshan"
     Header  title=" React lAb "
     Footer tagline ="Jain College of Engineering"
    </>
  );
}
export default App;
```

Output:

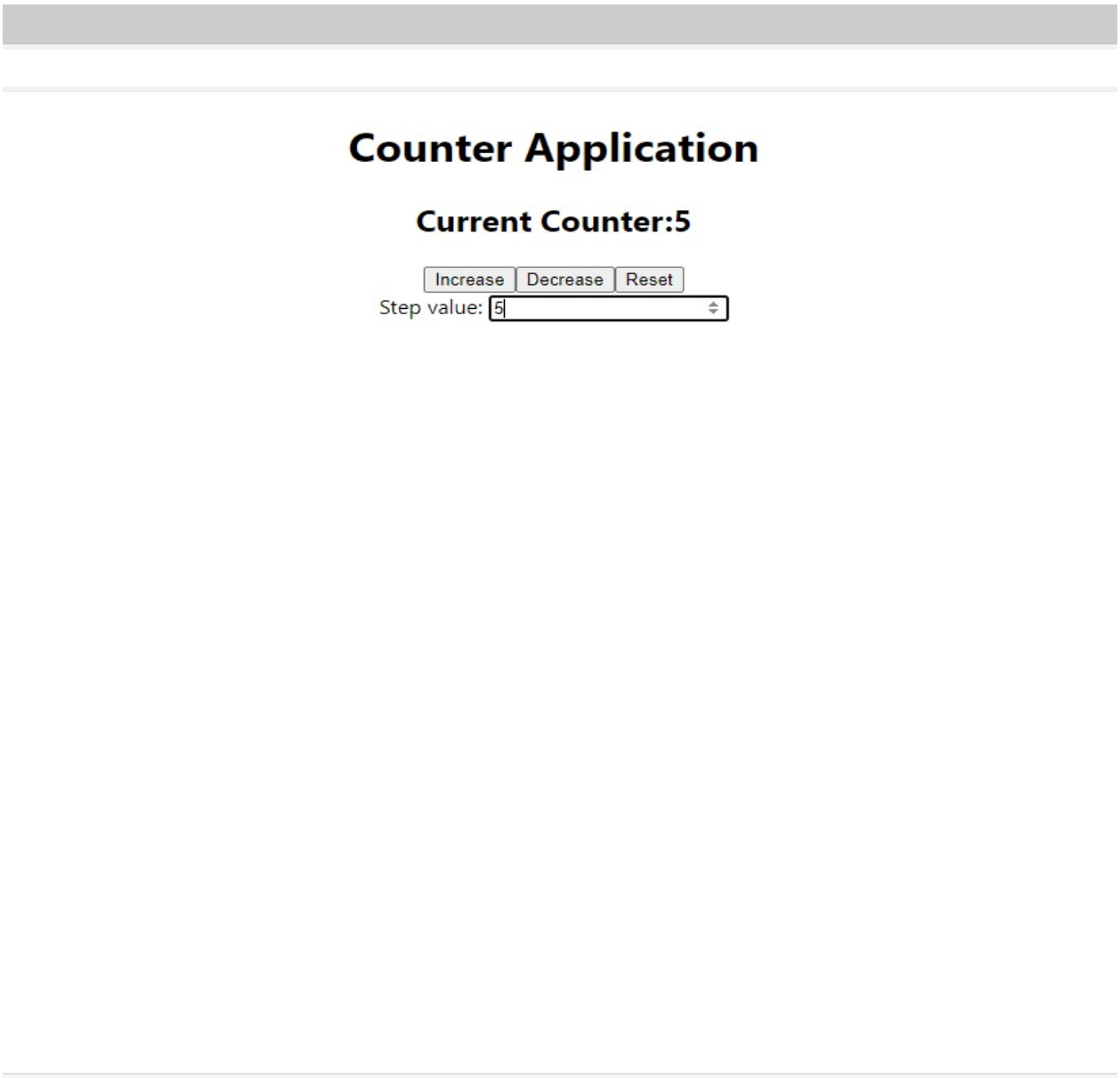

localhost:3000
Student name="Sudarshan" Header title=" React lAb " Footer tagline ="Jain College of Engineering"

3) Create a Counter Application using React that demonstrates state management with the useState hook. Display the current value of the counter prominently on the screen. Add buttons to increase and decrease the counter value. Ensure the counter updates dynamically when the buttons are clicked. Use the useState hook to manage the counter's state within the component. Prevent the counter from going below a specified minimum value (e.g., 0). Add a "Reset" button to set the counter back to its initial value. Include functionality to specify a custom increment or decrement step value.

App.js

```
import React, { useState } from 'react';
const CounterApp = () => {
  const initialCounter = 0;
  const [counter, setCounter] = useState(initialCounter);
  const [step, setStep] = useState(1);
  const increaseCounter = () => {
    setCounter(prevCounter => prevCounter + step);
  };
  const decreaseCounter = () => {
    if (counter - step >= 0) {
      setCounter(prevCounter => prevCounter - step);
    }
  };
  const resetCounter = () => {
    setCounter(initialCounter);
  };
  const handleStepChange = (event) => {
    setStep(Number(event.target.value));
  };
  return (
    <div style={{ textAlign: 'center', marginTop: '50px' }}>
      <h1>Counter Application</h1>
      <h2>Current Counter: {counter}</h2>
      <div>
        <button onClick={increaseCounter}>Increase</button>
        <button onClick={decreaseCounter}>Decrease</button>
        <button onClick={resetCounter}>Reset</button>
      </div>
      <div style={{ marginTop: '10px' }}>
        <label>Step Value: </label>
        <input  type="number" value={step}
          onChange={handleStepChange}
          min="1"/>
      </div>
    </div>
    );
};
export default CounterApp;
```

Output:


Counter Application
Current Counter:5
Increase
Decrease
Reset
Step value: 5

4) Develop a To-Do List Application using React functional components that demonstrates the use of the useState hook for state management. Create a functional component named ToDoFunction to manage and display the todo list. Maintain a list of tasks using state. Provide an input field for users to add new tasks. Dynamically render the list of tasks below the input field. Ensure each task is displayed in a user-friendly manner. Allow users to delete tasks from the list. Mark tasks as completed or pending, and visually differentiate them.

Todoform.js

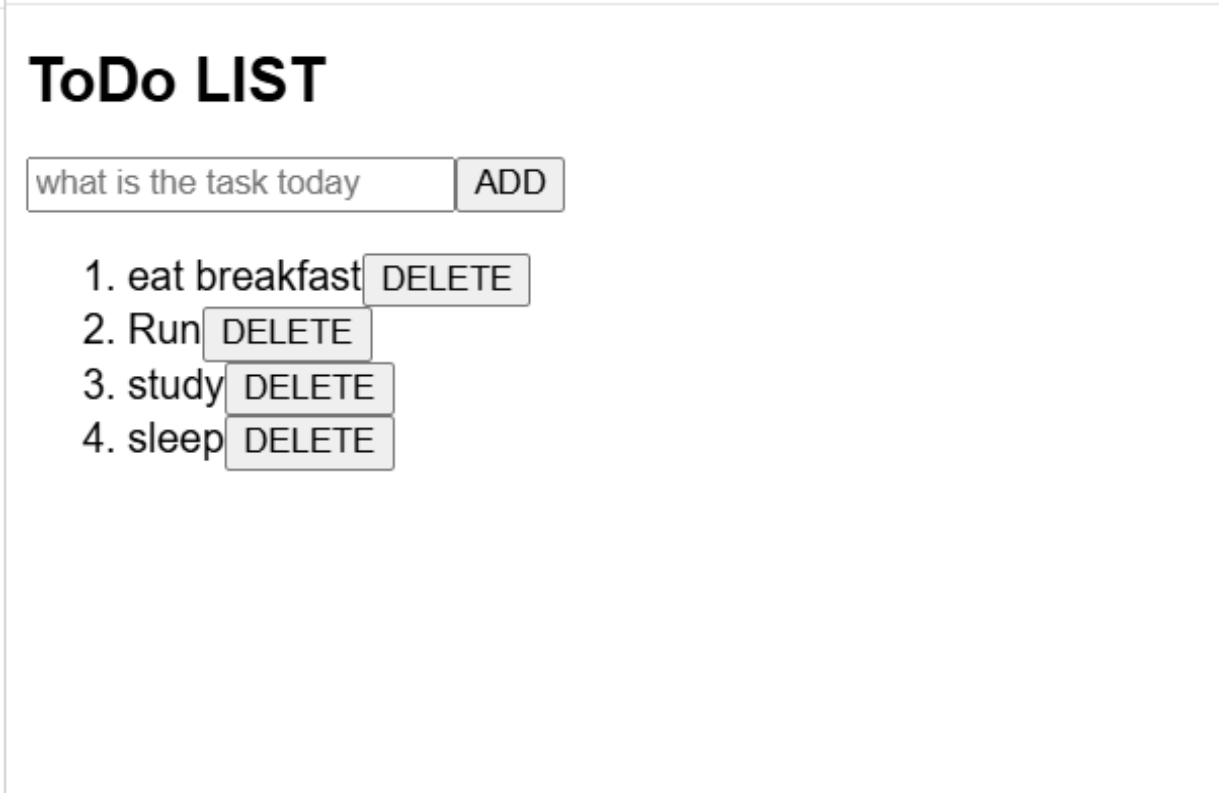
```
import React,{useState} from "react"
function  Todoform()
{
   const[tasks,settasks]=useState(["eat breakfast", "Run", "study"]);
   const[newTask,setNewTask]=useState("");
   function handleinputchange(event)
   {
      setNewTask(event.target.value);
   }
   function todoinput()
   {
    settasks(t=>[...t,newTask]);
    setNewTask("");
   }
   function deleteTask(index)
   {
    const updatedTasks=tasks.filter((element,i)=>i!==index);
    settasks(updatedTasks);
   }
   return(
<div className="To-do-list">
          <h1> ToDo LIST </h1>
          <div>
             <input  type="text"
             placeholder="what is the task today "
             value={newTask}
             onChange={handleinputchange} />
          <button  className="add-button"
           onClick= {()=>todoinput()}>
           ADD
          </button>
          </div>
       <ol>
          {tasks.map((task, index) =>
                <li key={index}>
                   <span className="text"> {task}</span>
                <button
                className="delete-button "
```

```
                onClick={()=> deleteTask(index)}>
                DELETE
              </button>
              </li>
          )}
        </ol>
</div>);
}
export default Todoform;
```

App.js

```
import './App.css';
import Todoform from './Todoform';
function App() {
  return (
    <div className="App">
    <Todoform> </Todoform>
    </div>
  );
}
export default App;
```

Output:



5) Develop a React application that demonstrates component composition and the use of props to pass data. Create two components: FigureList: A parent component responsible for rendering multiple child components. BasicFigure: A child component designed to display an image and its associated caption. Use the FigureList component to dynamically render multiple BasicFigure components. Pass image URLs and captions as props from the FigureList component to each BasicFigure component. Style the BasicFigure components to display the image and caption in ans aesthetically pleasing manner. Arrange the BasicFigure components within the FigureList in a grid or list format. Allow users to add or remove images dynamically. Add hover effects or animations to the images for an interactive experience.

App.js

```
import React, { useState } from "react";
import "./App.css";
        const BasicFigure = ({ imageUrl, caption, onDelete }) => {
          return (
            <div className="figure-container">
              <img src={imageUrl} alt={caption} className="figure-image" />
              <p className="figure-caption">{caption}</p>
              <button className="delete-btn" onClick={onDelete}>Remove</button>
            </div>
          );
        };
        const FigureList = () => {
          const [figures, setFigures] = useState([
            { id: 1, imageUrl: "https://picsum.photos/200/300", caption: "Image 1" },
            { id: 2, imageUrl: "https://picsum.photos/200/300", caption: "Image 2" },
          ]);
          const addFigure = () => {
            const newId = figures.length + 1;
            setFigures([
              ...figures,
              { id: newId, imageUrl: `https://via.placeholder.com/150`, caption: `Image
        ${newId}` },
            ]);
          };
          const removeFigure = (id) => {
            setFigures(figures.filter((figure) => figure.id !== id));
          };

          return (
            <div className="figure-list">
              <button className="add-btn" onClick={addFigure}>
                Add Image
              </button>
              <div className="figure-grid">
```

```
            {figures.map((figure) => (
              <BasicFigure
                key={figure.id}
                imageUrl={figure.imageUrl}
                caption={figure.caption}
                onDelete={() => removeFigure(figure.id)}
              />
            ))}
          </div>
        </div>
      );
    };
    const App = () => {
      return (
        <div className="App">
          <h1>Image Gallery</h1>
          <FigureList />
        </div>
      );
    };

export default App;
```

Output:

6) Design and implement a React Form that collects user input for name, email, and password. Form Fields are Name, Email, Password. Ensure all fields are filled before allowing form submission.Validate the email field to ensure it follows the correct email format (e.g., example@domain.com). Optionally enforce a minimum password length or complexity. Display error messages for invalid or missing inputs. Provide visual cues (e.g., red borders) to highlight invalid fields. Prevent form submission until all fields pass validation. Log or display the entered data upon successful submission (optional). Add a "Show Password" toggle for the password field. Implement clientside sanitization to ensure clean input.

App.js

```
import React, { useState } from "react";
import "./App.css";
const App = () => {
        const [formData, setFormData] = useState({
          name: "",
          email: "",
          password: "",
        });
        const [errors, setErrors] = useState({
          name: "",
          email: "",
          password: "",
        });
        const [showPassword, setShowPassword] = useState(false);
        const handleInputChange = (e) => {
          const { name, value } = e.target;
          setFormData({
            ...formData,
            [name]: value.trim(), // sanitize by trimming spaces
          });
        };
        const validateForm = () => {
          const newErrors = { name: "", email: "", password: "" };
          if (!formData.name) {
            newErrors.name = "Name is required";
          }
          const emailPattern = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}$/;
          if (!formData.email) {
            newErrors.email = "Email is required";
          } else if (!emailPattern.test(formData.email)) {
            newErrors.email = "Please enter a valid email address";
          }
          if (!formData.password) {
            newErrors.password = "Password is required";
          } else if (formData.password.length < 8) {
```

```
    newErrors.password = "Password must be at least 8 characters long";
   }

   setErrors(newErrors);
   return Object.values(newErrors).every((error) => !error);
  };

  const handleSubmit = (e) => {
   e.preventDefault();
   if (validateForm()) {
    console.log("Form Data Submitted:", formData);
    alert("Form submitted successfully!");
    // Optionally reset the form
    setFormData({
     name: "",
     email: "",
     password: "",
    });
    setErrors({
     name: "",
     email: "",
     password: "",
    });
   }
  };
  const togglePasswordVisibility = () => {
   setShowPassword((prevState) => !prevState);
  };

  return (
   <div className="form-container">
    <h1>React Registration Form</h1>
    <form onSubmit={handleSubmit} className="form">
     {/* Name Input */}
     <div className={`form-group ${errors.name ? "error" : ""}`}>
      <label htmlFor="name">Name</label>
      <input
       type="text"
       id="name"
       name="name"
       value={formData.name}
       onChange={handleInputChange}
       className={errors.name ? "input-error" : ""}
      />
      {errors.name && <span className="error-
message">{errors.name}</span>}
```

```
        </div>

        {/* Email Input */}
        <div className={`form-group ${errors.email ? "error" : ""}`}>
          <label htmlFor="email">Email</label>
          <input
            type="email"
            id="email"
            name="email"
            value={formData.email}
            onChange={handleInputChange}
            className={errors.email ? "input-error" : ""}
          />
          {errors.email && <span className="error-
message">{errors.email}</span>}
        </div>

        {/* Password Input */}
        <div className={`form-group ${errors.password ? "error" : ""}`}>
          <label htmlFor="password">Password</label>
          <input
            type={showPassword ? "text" : "password"}
            id="password"
            name="password"
            value={formData.password}
            onChange={handleInputChange}
            className={errors.password ? "input-error" : ""}
          />
          <button
            type="button"
            onClick={togglePasswordVisibility}
            className="show-password-btn"
          >
            {showPassword ? "Hide Password" : "Show Password"}
          </button>
          {errors.password && <span className="error-
message">{errors.password}</span>}
        </div>

        {/* Submit Button */}
        <button type="submit" className="submit-btn">
          Submit
        </button>
      </form>
    </div>
  );
```

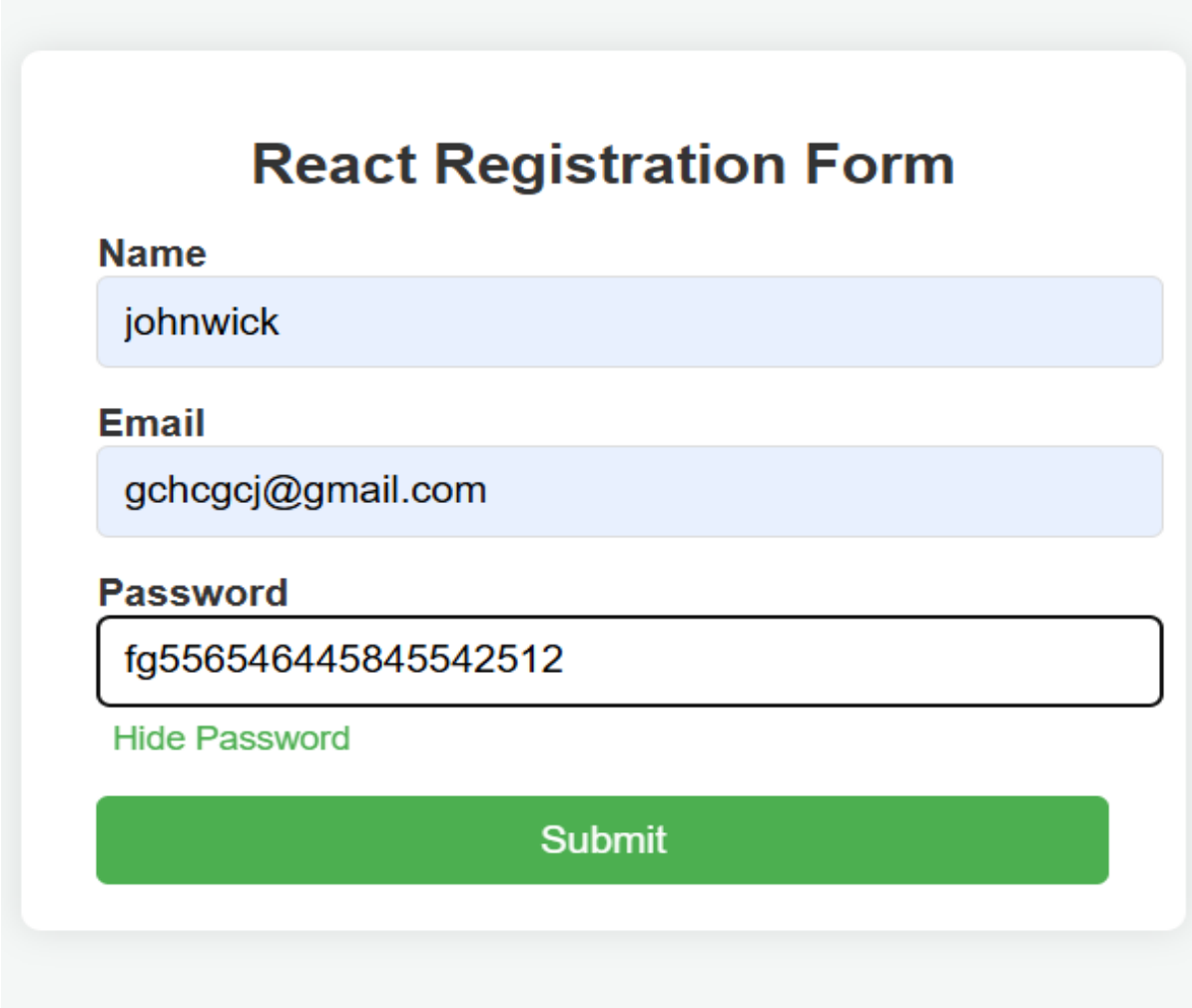
```
        };

        export default App;
```

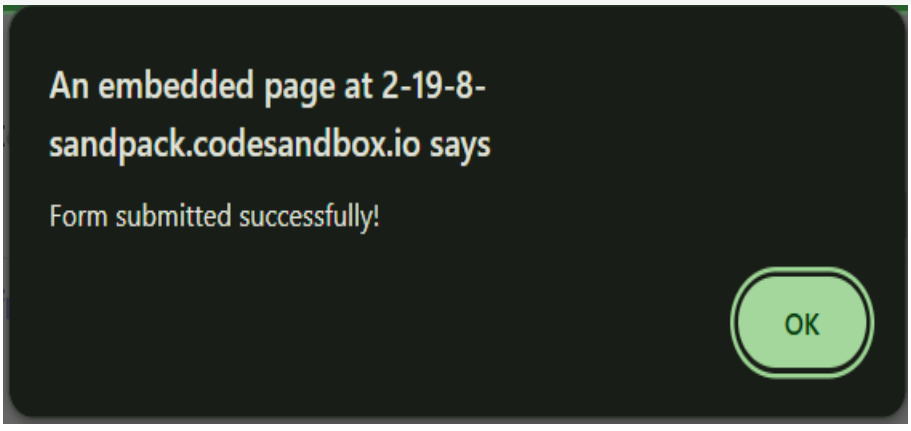
App.css

```
body {
  font-family: Arial, sans-serif;
  background-color: #f4f7f6;
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
  margin: 0;
}

.form-container {
  background: white;
  padding: 20px 30px;
  border-radius: 8px;
  box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);
  width: 100%;
  max-width: 400px;
}

h1 {
  text-align: center;
  color: #333;
}

/* Form styling */
.form {
  display: flex;
  flex-direction: column;
}

.form-group {
  margin-bottom: 15px;
}

label {
  font-weight: bold;
  margin-bottom: 5px;
  color: #333;
}
input {
  width: 100%;
```

```
  padding: 10px;
  font-size: 16px;
  border: 1px solid #ddd;
  border-radius: 5px;
}

input:focus {
  border-color: #4CAF50;
}

.error {
  border: 1px solid red;
}

.input-error {
  border: 1px solid red;
}

.error-message {
  color: red;
  font-size: 12px;
  margin-top: 5px;
}

.show-password-btn {
  background: none;
  border: none;
  color: #4CAF50;
  cursor: pointer;
  font-size: 14px;
  text-align: right;
  margin-top: 5px;
}

.submit-btn {
  padding: 10px 20px;
  background-color: #4CAF50;
  color: white;
  border: none;
  border-radius: 5px;
  font-size: 16px;
  cursor: pointer;
}
.submit-btn:hover {
  background-color: #45a049;
}
```

Output:


React Registration Form
Name
johnwick
Email
gchcgcj@gmail.com
Password
fg5565464458455425l2
Hide Password
Submit


An embedded page at 2-19-8-sandpack.codesandbox.io says
Form submitted successfully!
OK

7) Develop a React Application featuring a ProfileCard component to display a user's profile information, including their name, profile picture, and bio. The component should demonstrate flexibility by utilizing both external CSS and inline styling for its design. Display the following information: Profile picture, User's name, A short bio or description Use an external CSS file for overall structure and primary styles, such as layout, colors, and typography. Apply inline styles for dynamic or specific styling elements, such as background colors or alignment. Design the ProfileCard to be visually appealing and responsive. Ensure the profile picture is displayed as a circle, and the name and bio are appropriately styled. Add hover effects or animations to enhance interactivity. Allow the background color of the card to change dynamically based on a prop or state.

Profilecard.jsx

```
import React from "react";
import "./profilecard.css";
const Profilecard = ({ name, bio, profilePic, bgColor }) => {
const cardStyle = {
   backgroundColor: bgColor || "#f0f0f0",
   transition: "background-color 0.3s ease-in-out",
 };

 return (
   <div className="profile-card" style={cardStyle}>
     <img className="profile-pic" src={profilePic} alt={`${name}'s profile`} />
     <h2 className="profile-name">{name}</h2>
     <p className="profile-bio">{bio}</p>
   </div>
 );
};
export default ProfileCard;
```

Proffilecard.css

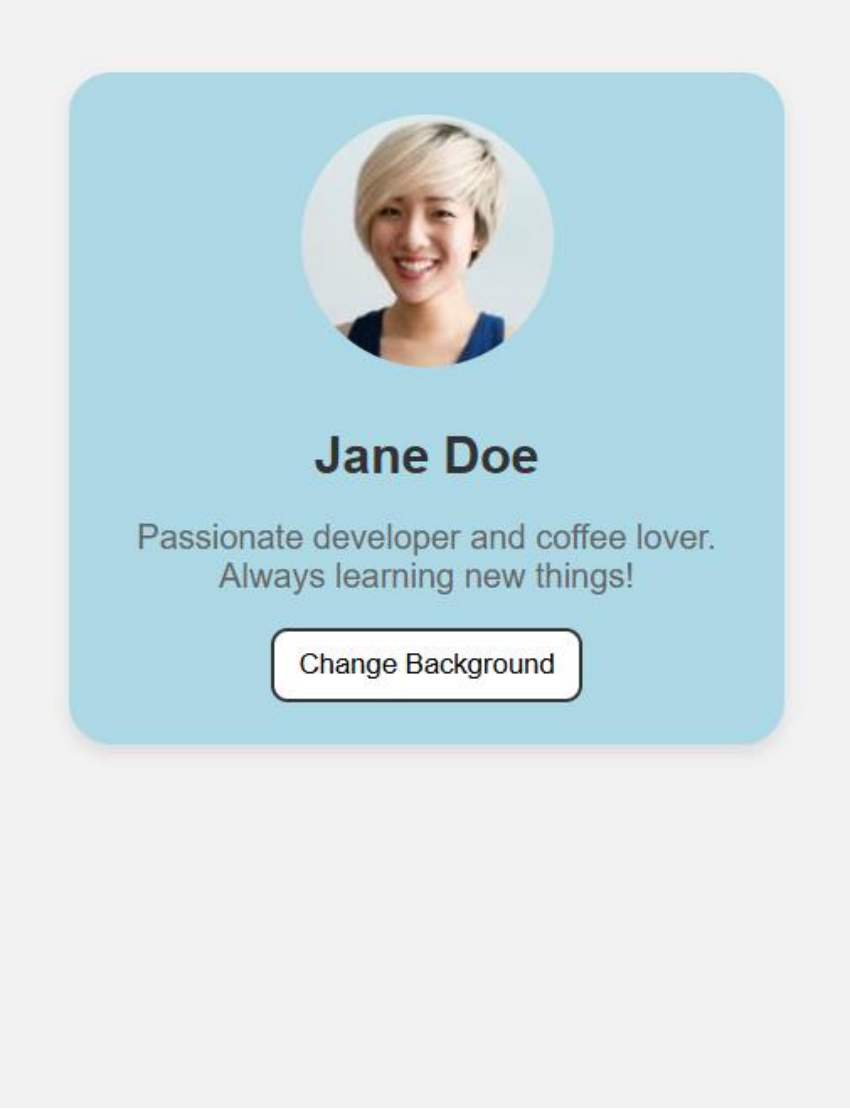
```
.profile-card {
   width: 300px;
   padding: 20px;
   border-radius: 10px;
   text-align: center;
   box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
   font-family: Arial, sans-serif;
   transition: transform 0.3s ease-in-out;
 }

 .profile-card:hover {
   transform: scale(1.05);
 }

 .profile-pic {
```

```
    width: 100px;
    height: 100px;
    border-radius: 50%;
    object-fit: cover;
    border: 3px solid #fff;
  }

  .profile-name {
    font-size: 1.5em;
    margin: 10px 0;
    color: #333;
  }

  .profile-bio {
    font-size: 1em;
    color: #666;
  }
```

App.js

```
import React from "react";
import Profilecard from  "./ProfileCard";
function App() {
  return (
    <div style={{display: "flex", justifyContent: "center", marginTop: "50px" }}>
      <Profilecard
        name="John Doe"
        bio="Web Developer & Tech Enthusiast"
        profilePic="https://via.placeholder.com/100"
        bgColor="#e3f2fd"
      />
    </div>
  );
}
export default App;
```

Output:


Jane Doe
Passionate developer and coffee lover.
Always learning new things!
Change Background

8. Develop a Reminder Application that allows users to efficiently manage their tasks. The application should include the following functionalities: Provide a form where users can add tasks along with due dates. The form includes task name,Due date,An optional description. Display a list of tasks dynamically as they are added. Show relevant details like task name, due date, and completion status. Include a filter option to allow users to view all Tasks and Display all tasks regardless of status. Show only tasks marked as completed. Show only tasks that are not yet completed.

App.js

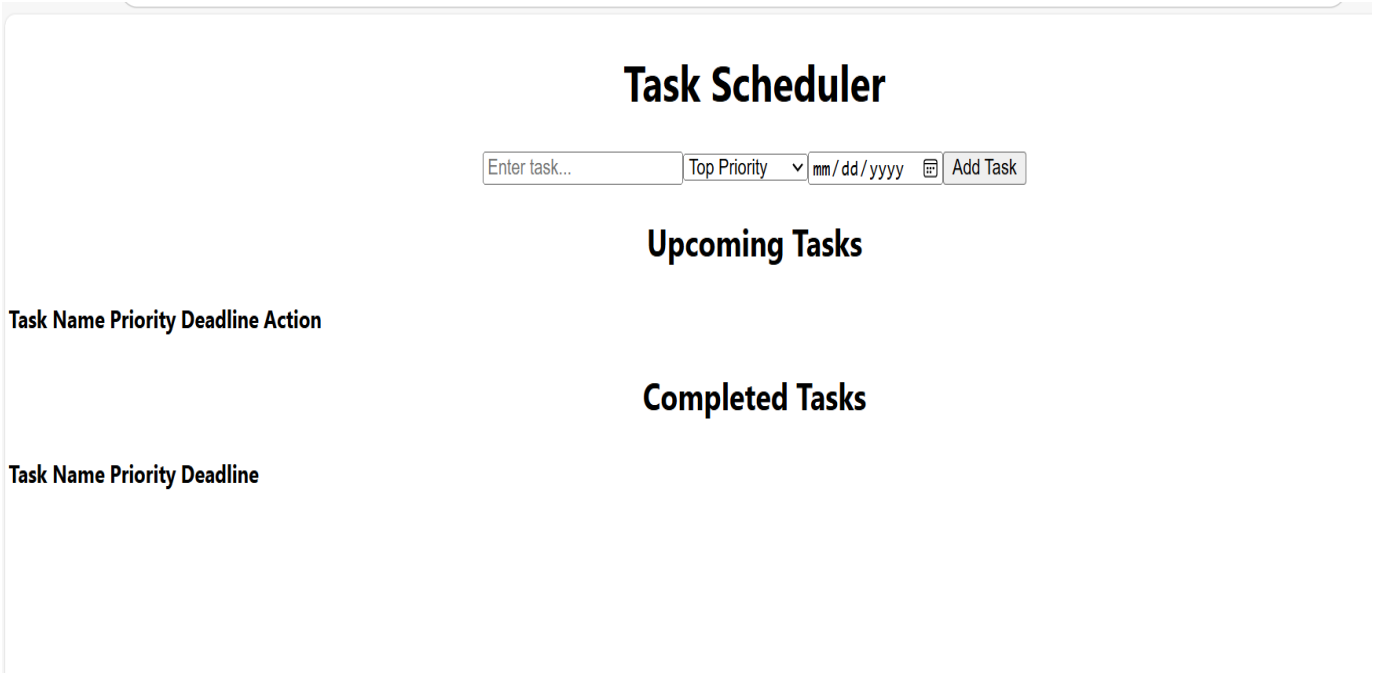
```
import React, { useState } from "react";
import "./App.css"; // Import your CSS file here
function App() {
    const [tasks, setTasks] = useState([]);
    const [completedTasks, setCompletedTasks] = useState([]);
    const [task, setTask] = useState("");
    const [priority, setPriority] = useState("top");
    const [deadline, setDeadline] = useState("");
 const handleTaskChange = (e) => {
        setTask(e.target.value);
    };const handlePriorityChange = (e) => {
        setPriority(e.target.value);
    };const handleDeadlineChange = (e) => {
        setDeadline(e.target.value);
    } const addTask = () => {
        if (task.trim() === "" || deadline === "") {
            alert("Please enter a task and select a valid deadline.");
            return;
        } const selectedDate = new Date(deadline);
        const currentDate = new Date();

        if (selectedDate <= currentDate) {
            alert("Please select a future date for the deadline.");
            return;
        }

        const newTask = {
            id: tasks.length + 1,
            task,
            priority,
            deadline,
            done: false,
        };setTasks([...tasks, newTask]);
          setTask("");
        setPriority("top");
        setDeadline("");
    }const markDone = (id) => {
```

```
        const updatedTasks = tasks.map((t) =>
            t.id === id ? { ...t, done: true } : t);
        setTasks(updatedTasks);
       const completedTask = tasks.find((t) => t.id === id);
        if (completedTask) {
            setCompletedTasks([...completedTasks, completedTask]);
        }
    };
 const upcomingTasks = tasks.filter((t) => !t.done);
return (
        <div className="App">
            <header>
                <h1>Task Scheduler</h1>
            </header>
            <main>
                <div className="task-form">
                    <input
                        type="text"
                        id="task"
                        placeholder="Enter task..."
                        value={task}
                        onChange={handleTaskChange/>
                    <select
                        id="priority"
                        value={priority}
                        onChange={handlePriorityChange>
                        <option value="top">Top Priority</option>
                        <option value="middle">Middle Priority</option>
                        <option value="low">Less Priority</option>
                    </select>
                    <input
                        type="date"
                        id="deadline"
                        value={deadline}
                        onChange={handleDeadlineChange />
                    <button id="add-task" onClick={addTask}>
                        Add Task
                    </button>
                </div>
                <h2 className="heading">Upcoming Tasks</h2>
                <div className="task-list" id="task-list">
                    <table>
                        <thead>
                            <tr>
                                <th>Task Name</th>
                                <th>Priority</th>
```

```
                <th>Deadline</th>
                <th>Action</th>
              </tr>
            </thead>
            <tbody>
              {upcomingTasks.map((t) => (
                <tr key={t.id}>
                  <td>{t.task}</td>
                  <td>{t.priority}</td>
                  <td>{t.deadline}</td>
                  <td>
                    {!t.done && (
                      <button
                        className="mark-done"
                        onClick={() => markDone(t.id)}
                      >Mark Done
                      </button>
                    )}
                  </td>
                </tr>
              ))}
            </tbody>
          </table>
        </div>
        <div className="completed-task-list">
          <h2 className="cheading">Completed Tasks</h2>
          <table>
            <thead>
              <tr>
                <th>Task Name</th>
                <th>Priority</th>
                <th>Deadline</th>
              </tr>
            </thead>
            <tbody>
              {completedTasks.map((ct) => (
                <tr key={ct.id}>
                  <td>{ct.task}</td>
                  <td>{ct.priority}</td>
                  <td>{ct.deadline}</td>
                </tr>
              ))}
            </tbody>
          </table>
        </div>
      </main>
```

```
        </div>
    );
}
export default App;
```

Output:



9)Design a React application that demonstrates the implementation of routing using the react-router-dom library. The application should include the Navigation Menu: Create a navigation bar with links to three distinct pages, Home, About, Contact. Develop separate components for each page (Home, About, and Contact) with appropriate content to differentiate them. Configure routes using react-router-dom to render the corresponding page component based on the selected link. Use BrowserRouter and Route components for routing. Highlight the active link in the navigation menu to indicate the current page

1. **Create a folder Components**
2. **Create a file Header.js**

Header.js

```
export default function Header(){
        return(
         <h1> Header: React  Router Tutorial </h1>
        )
}
```

About.js

```
import Header  from '../Components/Header'
export default function(){
        return(
                <>
                <Header />
                <h2> About  </h2>
                </>

        )
}
```

Contact.js

```
import Header  from '../Components/Header'
export default function(){
            return(
                <>
                <Header />
                <h2> Contact </h2></>

            )
}
```

Home.js

```
export default function(){
            return(
               <h2> Home  </h2>
            )
        }
```

App.js

```
import { BrowserRouter,Routes,Route } from 'react-router-dom';
import Home from './Pages/Home';
import Contact from './Pages/Contact';
import About from './Pages/About';
import './App.css';
export default function App() {
        return (
            <div>
             <BrowserRouter>
             <Routes>
             <Route index  element = {<Home/>} />
             <Route path ='/home' element ={<Home />}/>
             </Routes>
             <Routes>
             <Route index  element = {<Contact/>} />
             <Route path ='/contact' element ={<Contact />}/>
             </Routes>
             <Routes>
             <Route index  element = {<About/>}/>
             <Route path ='/about' element ={<About />}/>
             </Routes>
             </BrowserRouter>
            </div>
      );
}
```

Output: