1. Use create-react-app to set up a new project. Edit the App.js file to include a stateful component with useState. Add an input field and a element that displays text based on the input. Dynamically update the content as the user types.

**Step 1: Create a new React app**
First, you need to create a new React app using create-react-app. Open your terminal and run:

**npx create-react-app my-dynamic-app cd my-**

**dynamic-app**

**Step 2: Modify the App.js file**
Open the src/App.js file in your favorite code editor and update the code to include a stateful component using the useState hook. Here's how you can modify it:

```
import React, { useState } from 'react'; import
'./App.css';
function App() {
  const [text, setText] = useState(''); const
  handleChange = (event) => {
  setText(event.target.value);
  };
  return (
    <div className="App">
      <h1>Dynamic Text Display</h1>
      <input type="text"
        value={text}
        onChange={handleChange} placeholder="Type
        something..."
      />
      <p>You typed: {text}</p>
```

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

**Step 3: Run the application**
Back in your terminal, start the development server by running:

**npm start**

OUTPUT:

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.

**Step 1: Create a New React Application**
First, you need to create a new React app using below command. Open your terminal and run:

npx create-react-app react-props-demo navigate to the

project directory:

cd react-props-demo

**Step 2: Define the Components**

**1. App Component (Parent Component)**
In src/App.js, we define the parent component App, which will pass data to the child components using props.

```
import React from 'react'; import Header

from './Header'; import Footer from

'./Footer'; import './App.css';

function App() {

  const title = "Welcome to My React App";

  const tagline = "Building great apps with React";

  const copyright = "© 2025 MyApp, All Rights Reserved";

  return (

    <div className="App">

      <Header title={title} />
```

```
      <Footer tagline={tagline} copyright={copyright} />
    </div>
  );
}
export default App;
```

**2. Header Component (Child Component)**

Create a new file src/Header.js for the Header component, which will receive the title as a prop.

```
import React from 'react';
function Header(props) { return
(
    <header>
      <h1>{props.title}</h1>
    </header>
  );
}
export default Header;
```

**3. Footer Component (Child Component)**

Create a new file src/Footer.js for the Footer component, which will receive the tagline and copyright as props.

```
import React from 'react';
function Footer(props) { return (
    <footer>
      <p>{props.tagline}</p>
      <p>{props.copyright}</p>
    </footer>
  );
```

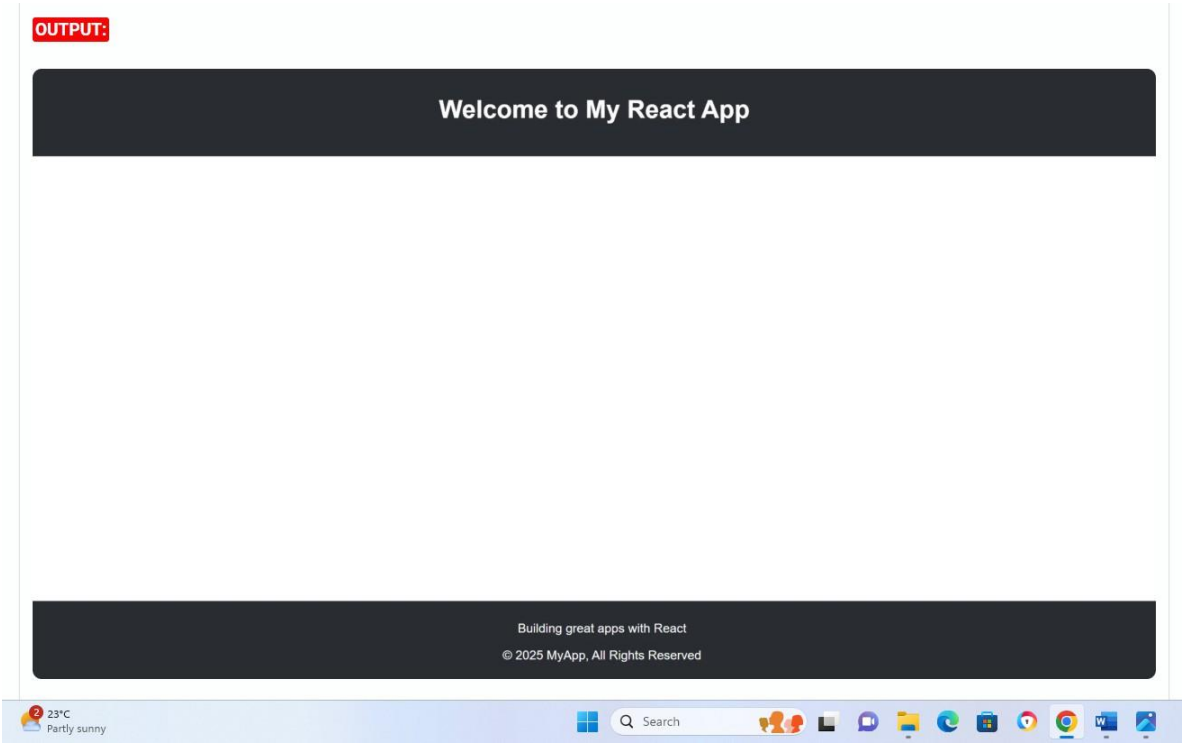
```
}

export default Footer;
```

**Step 3: Add Some Basic Styles (Optional)**
To make the app look better, you can add some basic styles.
Open src/App.css (or create a new file) and add the following styles:

```
.App {
  text-align: center;
  font-family: Arial, sans-serif;
}
header {
  background-color: #282c34; padding: 20px;
  color: white;
}
footer {
  background-color: #282c34; padding: 10px;
  color: white; position:
  absolute; bottom: 0;
  width: 100%; text-
  align: center;
}
```

**Step 4: Run the application**
Back in your terminal, start the development server by running: npm start

OUTPUT:


Welcome to My React App
Building great apps with React
© 2025 MyApp, All Rights Reserved
23°C
Partly sunny
Search

**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.

**Step 1: Create a new React app**
First, you need to create a new React app using below command. Open your terminal and run:

npx create-react-app counter-app

This will set up a new React project in a folder called **counter-app**. After the installation is complete, navigate to the project directory:

cd counter-app

**Step 2: Modify the App.js File**

1. **Navigate to the src folder** in the file explorer on the left-hand side of VSCode.
2. Open the App.js file (which contains the default template code).
3. **Replace the content of App.js with the code provided for the Counter App**. Here's the code to replace inside App.js:

```
import React, { useState } from 'react';
import './App.css';
function App() {
  const [counter, setCounter] = useState(0); const [step,
  setStep] = useState(1);
  const minValue = 0;
  const handleIncrement = () => { setCounter(prevCounter =>
    prevCounter + step);
  };
```

```
const handleDecrement = () => {
  if (counter - step >= minValue) { setCounter(prevCounter =>
    prevCounter - step);
  }
};
const handleReset = () => { setCounter(0);
};
const handleStepChange = (event) => {
  setStep(Number(event.target.value));
};
return (
  <div style={{ textAlign: 'center', marginTop: '50px' }}>
    <h1>Counter Application</h1>
    <div style={{ fontSize: '48px', margin: '20px' }}>
      <span>{counter}</span>
    </div>
    <div>
      <button onClick={handleIncrement}>Increase by {step}</button>
      <button onClick={handleDecrement}>Decrease by {step}</button>
      <button onClick={handleReset}>Reset</button>
    </div>
    <div style={{ marginTop: '20px' }}>
      <label>
        Set Increment/Decrement Step:
        <input type="number"
```

```
            value={step}
            onChange={handleStepChange} min="1"
            style={{ marginLeft: '10px' }}
          />
        </label>
      </div>
    </div>
  );
}
export default App;
```

**Step 3: Modify the App.css (Optional)**
You can adjust the styling if desired. For example, you can modify App.css to ensure the buttons look good:

```
.App {
  text-align: center;
}
button {  margin:
  10px; padding:
  10px;
  font-size: 16px; cursor:
  pointer;
}
input { padding:
  5px;
  font-size: 16px;
}
```

You can also remove any default styling from the App.css file that is not needed for this project.

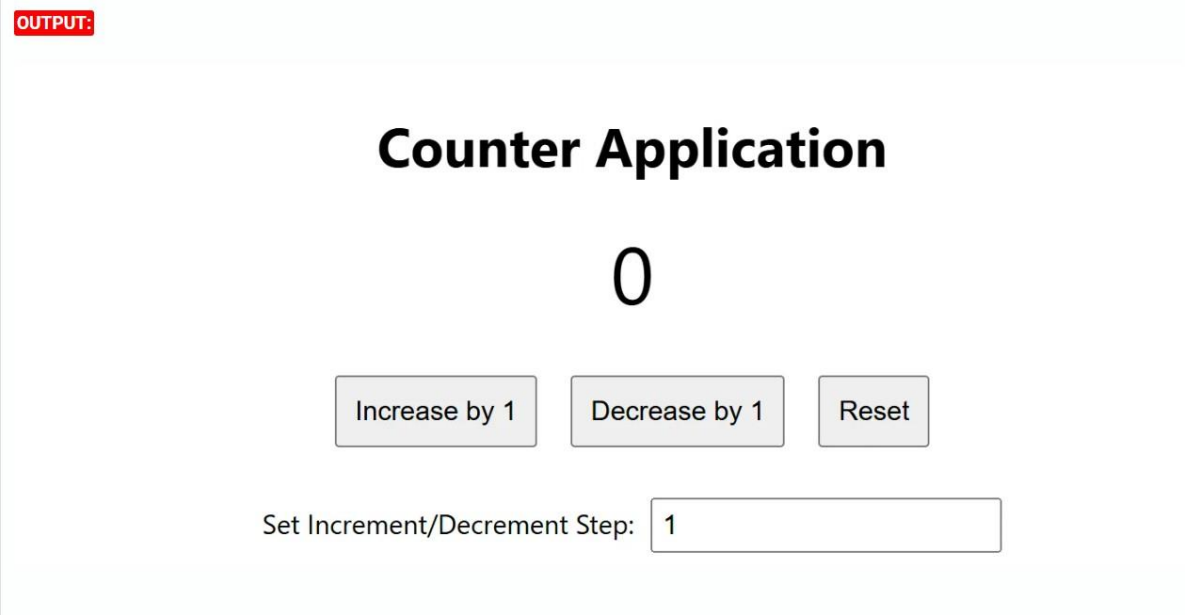
**Step 4: Start the Development Server**

1. In the terminal inside VSCode, run the following command to start the React development

```
npm start
```

1. This will open your browser and navigate to http://localhost:3000/. You should see your Counter Application up and running.

**OUTPUT:**


OUTPUT:
Counter Application
0
Increase by 1
Decrease by 1
Reset
Set Increment/Decrement Step:
1

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 to-do 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.

**Step 1: Create a New React Application**

npx create-react-app todo-app cd

todo-app

**Step 2: Modify the App.js File**

1. **Replace the content of App.js with the code provided for the todo-app**. Here's the code to replace inside App.js:

```
import React, { useState } from 'react';
import './App.css'; // Import CSS file

function App() {
  const [tasks, setTasks] = useState([]);
  const [newTask, setNewTask] = useState('');

  // Add task to the list
  const addTask = () => {
    if (newTask) {
      setTasks([...tasks, newTask]);
      setNewTask('');
    }
  };

  // Delete a task from the list
  const deleteTask = (index) => {
    setTasks(tasks.filter((_, i) => i !== index));
  };

  return (
    <div className="todo-container">
      <h1>To-Do List</h1>
      <input
        type="text"
        value={newTask}
        onChange={(e) => setNewTask(e.target.value)}
        placeholder="Enter a new task"
        className="task-input"
      />
      <button onClick={addTask} className="add-
button">Add</button>
```

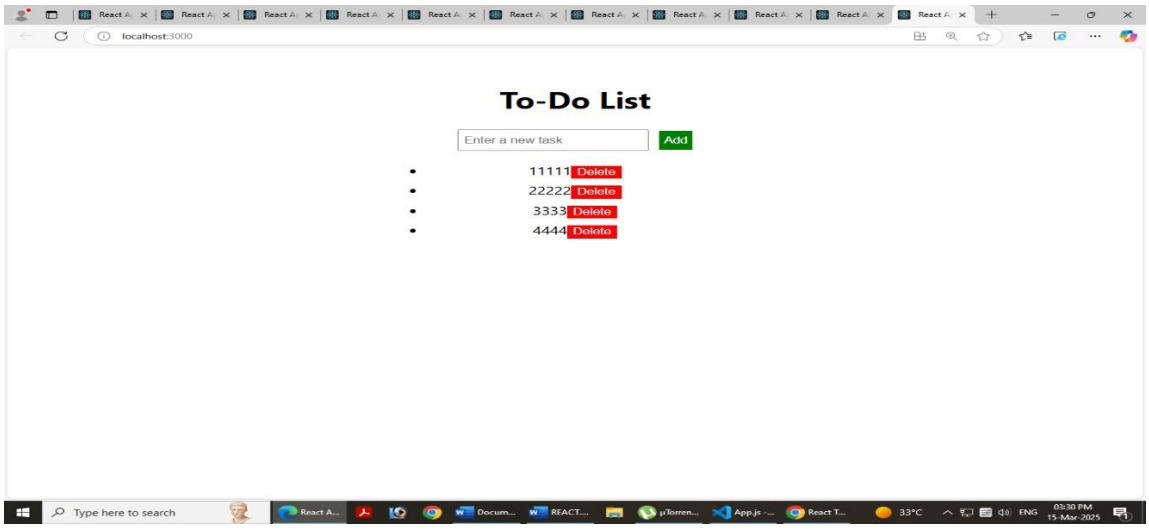
```
33.
34.         <ul className="task-list">
35.           {tasks.map((task, index) => (
36.             <li key={index} className="task-item">
37.               {task}
38.               <button onClick={() => deleteTask(index)}
      className="delete-button">Delete</button>
39.             </li>
40.           ))}
41.         </ul>
42.       </div>
43.     );
44.   }
45.
46.   export default App;
```

**Step 3: Modify the App.css (Optional)**

```
/* Center the container */
.todo-container {
  width: 300px;
  margin: 50px auto;
  text-align: center;
}
/* Style the input field */
.task-input { padding:
  5px; width: 60%;
  margin-right: 10px;
}

/* Style the Add button */
.add-button {
  padding: 5px;
  background-color: green;
  color: white;
  border: none;
```

```
  cursor: pointer;
}

/* Style for the task list */
.task-list { padding:
  0;
}
/* Style for each task item */
.task-item { margin:
  5px 0;
}
/* Style for the delete button */
.delete-button {
  background-color: red;
  color: white;
  border: none;
  cursor: pointer;
}
```

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 an 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.

**Step 1: Create a New React Application**

npx create-react-app figure-gallery

cd figure-gallery

**Step 2: Set Up the Folder Structure**
Create the folder structure. Here's how you can organize the directories:

1. **Inside the src folder**:
    1. Create a components folder.
    1. Inside components, **create BasicFigure.js** and **FigureList.js**.

**BasicFigure.js:**

```
// src/BasicFigure.js

import React from 'react';

const BasicFigure = ({ src, caption }) => {
  return (
    <div>
      <img src={src} alt={caption} />
      <p>{caption}</p>
    </div>
  );
};

export default BasicFigure;
```

**FigureList.js:**

```
import React, { useState } from 'react';

import BasicFigure from './BasicFigure';

const FigureList = () => {
```

```
  // Initial static data
  const [figures, setFigures] = useState([
    { src: 'https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQLeEJ47c_Y9g5VeNDcUWmFMuvpjB4LsrR3ZQ&s', caption: 'Image 1' }, // Example image URL
    { src: 'https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRcKpkc_AQKNOt8OsfV3wsfDGOrr-SkE_MRcg&s', caption: 'Image 2' },
    { src: 'https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSvS8XRlRIzQ_lvu0EZy88MrE-UkMYfDTPjYQ&s', caption: 'Image 3' },
  ]);
  // Function to add a new image
  const addImage = () => {
  const newFigure = {
     src: `https://picsum.photos/200?random=${figures.length + 1}`, // Random image from picsum
     caption: `Image ${figures.length + 1}`,
   };
   setFigures([...figures, newFigure]);
  };
  // Function to remove the last image
  const removeImage = () => {
  setFigures(figures.slice(0, -1));
  };

  return (
    <div>
      <h1>Simple Image Gallery </h1>
      <div className="button-box">
        <button onClick={addImage} className="action-button">Add Image</button>
        <button onClick={removeImage} className="action-button">Remove Image</button>
      </div>
      <div className="figure-list">
        {figures.map((figure, index) => (
```

```
<BasicFigure key={index} src={figure.src || 'https://via.placeholder.com/200'} caption={figure.caption}
/>
          ))}
        </div>
      </div>
    );
  };
  export default FigureList;
```

**App.js**

```
import React, { useState } from "react";
import "./App.css";
function App() {
  const [tasks, setTasks] = useState([]);
  const [taskName, setTaskName] = useState("");
  const [dueDate, setDueDate] = useState("");
  const [filter, setFilter] = useState("all"); // Filter by all, completed, or pending

  const addTask = (e) => {
    e.preventDefault();
    const newTask = {
      id: Date.now(),
      name: taskName,
      dueDate,
      completed: false,
    };
    setTasks([...tasks, newTask]);
    setTaskName("");
    setDueDate("");
  };

  const toggleCompletion = (id) => {
    setTasks(
```

```
    tasks.map((task) =>
      task.id === id ? { ...task, completed: !task.completed } : task
    )
  );
};

const filteredTasks = tasks.filter((task) => {
  if (filter === "completed") return task.completed;
  if (filter === "pending") return !task.completed;
  return true;
});

return (
  <div className="App">
    <h2>Task Reminder</h2>

    {/* Task Form */}
    <form onSubmit={addTask}>
      <input
        type="text"
        placeholder="Task Name"
        value={taskName}
        onChange={(e) => setTaskName(e.target.value)}
        required
      />
      <input
        type="date"
        value={dueDate}
        onChange={(e) => setDueDate(e.target.value)}
        required
      />
      <button type="submit">Add Task</button>
```

```
      </form>

      {/* Filter Buttons */}
      <div>
        <button onClick={() => setFilter("all")}>All</button>
        <button onClick={() => setFilter("completed")}>Completed</button>
        <button onClick={() => setFilter("pending")}>Pending</button>
      </div>
      {/* Task List */}
      <div>
        {filteredTasks.map((task) => (
          <div key={task.id} className={task.completed ? "completed" : ""}>
            <div>
              <strong>{task.name}</strong>
              <br />
              Due: {task.dueDate}
            </div>
            <button onClick={() => toggleCompletion(task.id)}>
              {task.completed ? "Mark Incomplete" : "Mark Completed"}
            </button>
          </div>
        ))}
      </div>
    </div>
  );
}
export default App;
```

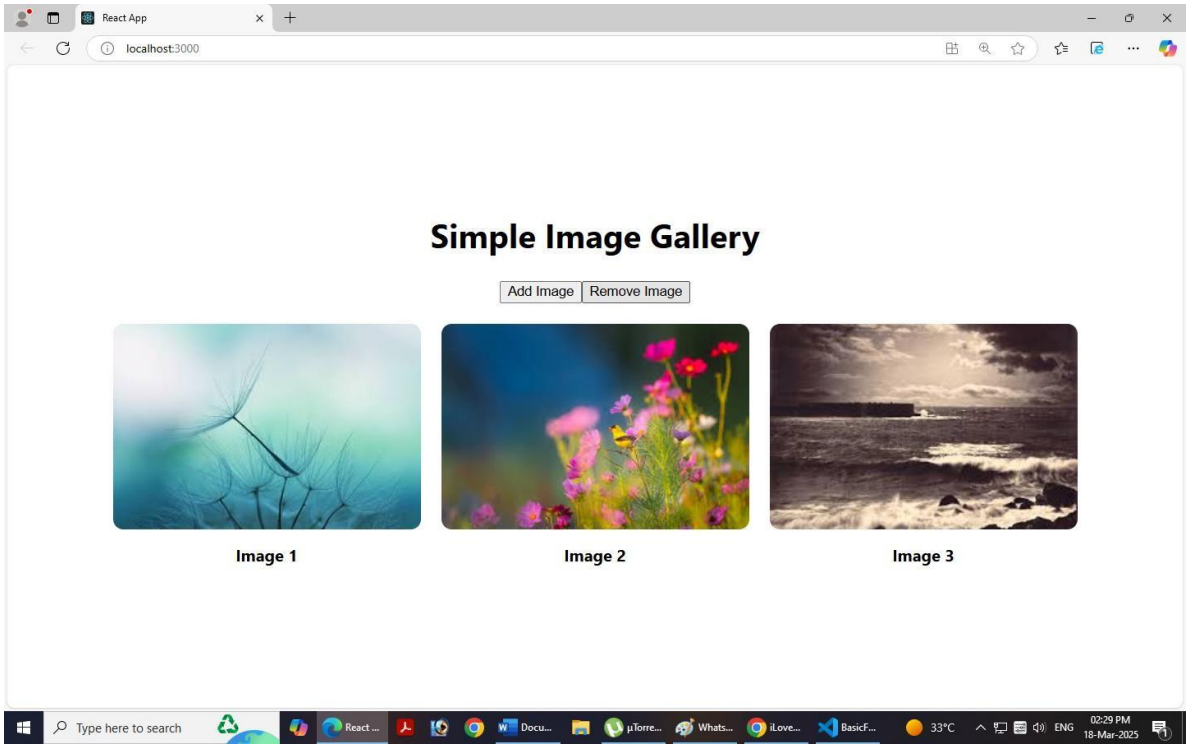
**APP.CSS**

```
.App {
  font-family: Arial, sans-serif;
  display: flex;
  flex-direction: column;
```

```
align-items: center; padding:
20px; background-color:
#f9f9f9;
  }
  form {
    display: flex;
    flex-direction: column;
    margin-bottom: 20px;
  }
  input, button {
    padding: 8px;
    margin: 5px;
    border: 1px solid #ccc;
    border-radius: 4px;
  }
  button {
    background-color: #007bff;
    color: white;
    cursor: pointer;
  }
  button:hover {
    background-color: #0056b3;
  }
  div > button {
    background-color: #f8f9fa;
  }
  button:hover {
    background-color: #e2e6ea;
  }
  .completed {
    background-color: #d4edda;
    text-decoration: line-through;
```

```
  color: gray;
}

div {
  margin-bottom: 10px;
}
```

**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 client-side sanitization to ensure clean input.

**Step 1: Create a New React Application**

npx create-react-app react-form

This will set up a new React project in a folder called **react-form**. After the installation is complete, navigate to the project directory:

cd react-form

**Step 2: Modify the App.js File**

```
import React, { useState } from 'react';

const App = () => {

 // State to hold form data and validation errors

 const [formData, setFormData] = useState({ name: '', email: '', password: '' });

 const [errors, setErrors] = useState({});

 const [showPassword, setShowPassword] = useState(false);

 // Handle input changes  const

 handleChange = (e) => {

  const { name, value } = e.target;

  setFormData({ ...formData, [name]: value });

 };

 // Toggle password visibility

 const togglePasswordVisibility = () => setShowPassword(!showPassword);

 // Simple validation

 const validate = () => {

 let newErrors = {};

  if (!formData.name) newErrors.name = 'Name is required';

  if (!formData.email || !/\S+@\S+\.\S+/.test(formData.email)) newErrors.email = 'Please enter a valid email';
```

```
  if (!formData.password || formData.password.length < 6) newErrors.password = 'Password must be at least 6 characters';
   setErrors(newErrors);
   return Object.keys(newErrors).length === 0;
  };
  // Handle form submission
  const handleSubmit = (e) => {
  e.preventDefault();
   if (validate()) {
    console.log('Form submitted:', formData);
    alert('Form submitted successfully!');
   }
  };
  return (
   <div style={{ maxWidth: '400px', margin: '0 auto', padding: '20px', border: '1px solid #ccc', borderRadius: '8px' }}>
    <h2>Sign Up</h2>
    <form onSubmit={handleSubmit}>
     {/* Name Field */}
     <div>
      <label>Name:</label>
      <input
       type="text"
       name="name"
       value={formData.name}
       onChange={handleChange}
       style={{ width: '100%', padding: '8px', borderColor: errors.name ? 'red' : '#ccc' }}
      />
      {errors.name && <p style={{ color: 'red', fontSize: '12px' }}>{errors.name}</p>}
     </div>
     {/* Email Field */}
```

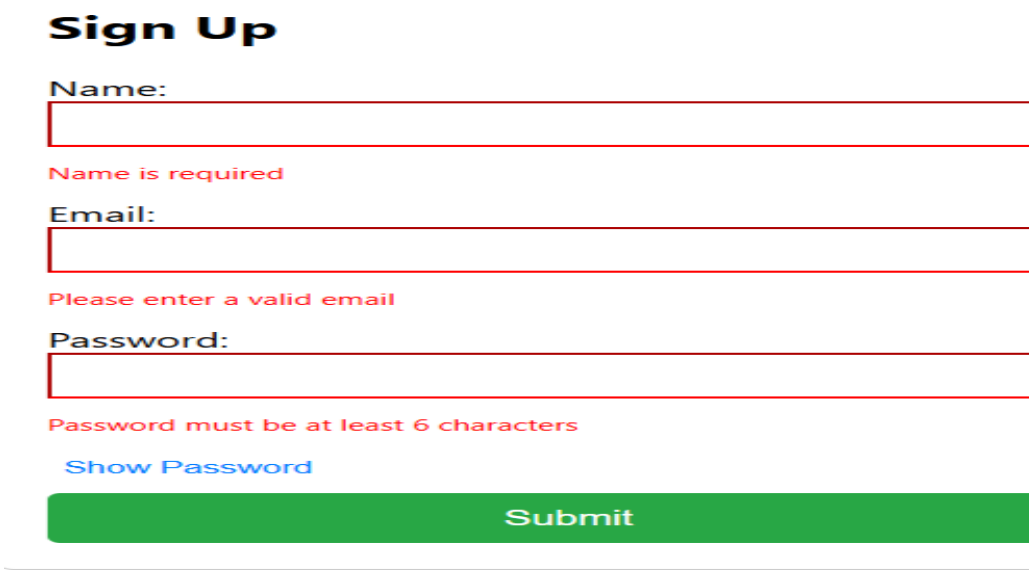
```
      <div>
        <label>Email:</label>
        <input
          type="email"
          name="email"
          value={formData.email}
          onChange={handleChange}
          style={{ width: '100%', padding: '8px', borderColor: errors.email ? 'red' : '#ccc' }}
        />
        {errors.email && <p style={{ color: 'red', fontSize: '12px' }}>{errors.email}</p>}
      </div>
      {/* Password Field */}
      <div>
        <label>Password:</label>
        <input
          type={showPassword ? 'text' : 'password'}
          name="password"
          value={formData.password}
          onChange={handleChange}
          style={{ width: '100%', padding: '8px', borderColor: errors.password ? 'red' : '#ccc' }}
        />
        {errors.password && <p style={{ color: 'red', fontSize: '12px' }}>{errors.password}</p>}
        <button
          type="button"
          onClick={togglePasswordVisibility}
          style={{ marginTop: '5px', background: 'transparent', border: 'none', color: '#007BFF', cursor:
'pointer' }}
        >
          {showPassword ? 'Hide Password' : 'Show Password'}
        </button>
      </div>
```

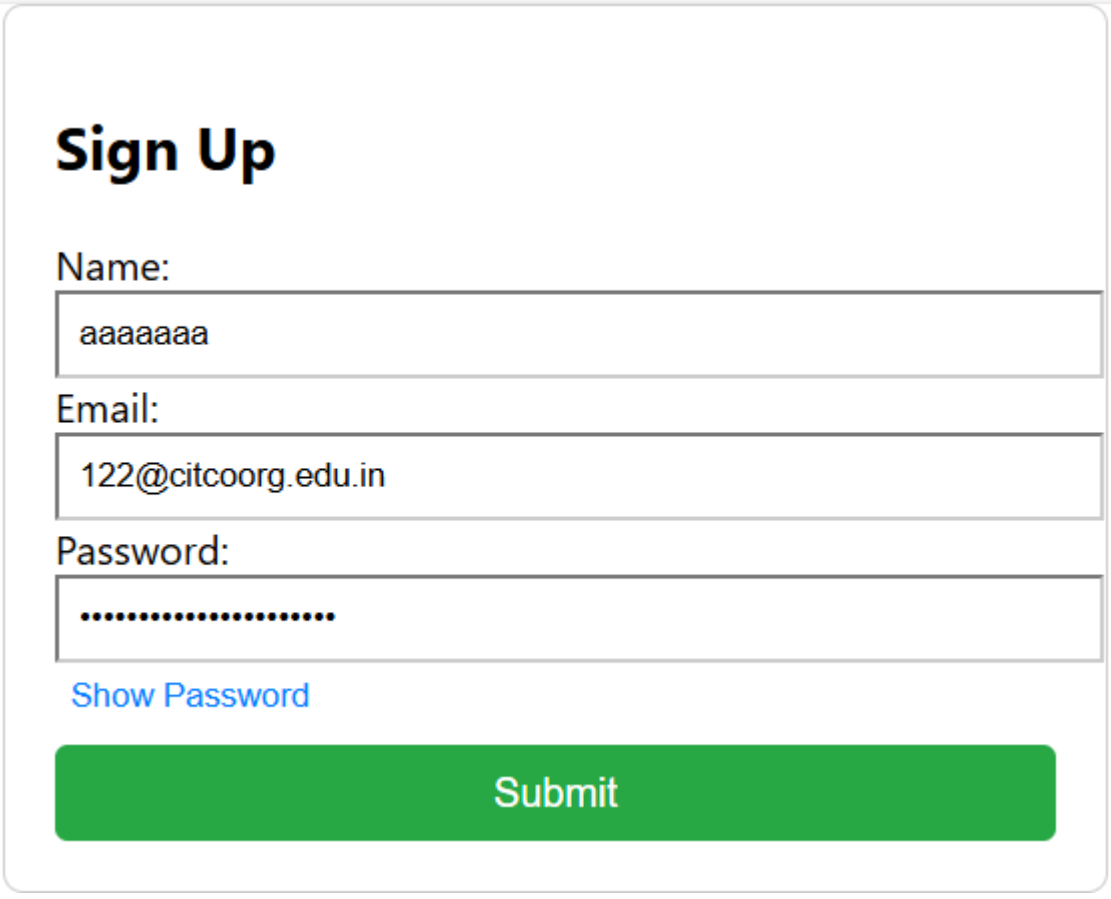
```
      {/* Submit Button */}
      <button
        type="submit"
        style={{
          width: '100%',
          padding: '10px',
          backgroundColor: '#28a745',
          color: 'white',
          border: 'none',
          fontSize: '16px',
          marginTop: '10px',
          borderRadius: '5px',
          cursor: 'pointer',
        }}
      >
        Submit
      </button>
    </form>
  </div>
 );
};
export default App;
```

**Step 3: Run the application**

Back in your terminal, start the development server by running: npm start

**OUTPUT:**


Sign Up
Name:
Name is required
Email:
Please enter a valid email
Password:
Password must be at least 6 characters
Show Password
Submit


Sign Up
Name:
aaaaaaa
Email:
122@citcoorg.edu.in
Password:
Show Password
Submit

**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.

**Step 1: Create a New React Application**

First, you need to create a new React app using below command. Open your terminal and run:

npx create-react-app profile-card-app

This will set up a new React project in a folder called **profile-card-app**. After the installation is complete, navigate to the project directory:

**Step 2: Set Up the Folder Structure**

- Inside the **src folder,** create a new file **ProfileCard.js** to define the ProfileCard component.
- After that copy and paste below code in the **ProfileCard.js** file.

**ProfileCard.js:**

```
import React, { useState } from 'react';const ProfileCard = ({ name, bio, profilePicture }) => {
const [bgColor, setBgColor] = useState('#f0f0f0');const handleMouseEnter = () => {
setBgColor('#d1c4e9');
};const handleMouseLeave = () => {
setBgColor('#f0f0f0');
};return (
<div
className="profile-card"
style={{ backgroundColor: bgColor }}
onMouseEnter={handleMouseEnter}
onMouseLeave={handleMouseLeave}
>
<img
src={profilePicture}
```

```
alt={`${name}'s profile`}
className="profile-picture"
/>
<div className="profile-info">
<h2 className="profile-name">{name}</h2>
<p className="profile-bio">{bio}</p>
</div>
</div>
);
};export default ProfileCard;
```

**Step 3: Modify the App.js File**

- Inside the **src folder** modify the **src/App.js** file.
- Now, use the **ProfileCard** component in **App.js** and pass sample data to display a user's profile.

```
import React from 'react';
import ProfileCard from './ProfileCard';
import './App.css'

const App = () => {
  return (
    <div className="App">
      <ProfileCard
        name="vtucircle"
        bio="vtucircle is the website which provides all the required VTU notes, syllabus, model papers, previous
            year papers of 2021 | 2022 scheme for BE students."
        profilePicture="https://vtucircle.com/wp-content/uploads/2024/11/cropped-vtucircle_icon-1.png"
      />
    </div>
  );
};

export default App;
```

**Step 3: Modify the App.css**

- You can adjust the styling if desired. For example, you can modify **App.css** to ensure the profile

look good. Copy the below code and paste it in the **App.css** file.

Software tool

```
body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  background-color: #f4f7fa;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  margin: 0;
}

.profile-card {
  width: 320px;
  padding: 30px;
  border-radius: 15px;
  text-align: center;
  background-color: #ffffff;
  box-shadow: 0 6px 12px rgba(0, 0, 0, 0.1);
  transition: transform 0.3s ease, box-shadow 0.3s ease, background-color 0.3s ease;
  cursor: pointer;
  overflow: hidden;
  margin: 20px;
}
.profile-card-container {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  width: 100%;
}

.profile-card:hover {
  transform: translateY(-10px);
  box-shadow: 0 12px 24px rgba(0, 0, 0, 0.2);
  background-color: #f3f4f6;
```

```
}

.profile-picture {
  width: 130px;
  height: 130px;
  border-radius: 50%;
  object-fit: cover;
  border: 4px solid #fff;
  transition: transform 0.3s ease, box-shadow 0.3s ease;
}

.profile-card:hover .profile-picture {
  transform: scale(1.1);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}

.profile-info {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

.profile-name {
  font-size: 1.8rem;
  font-weight: 600;
  color: #333;
  margin-bottom: 15px;
  transition: color 0.3s ease;
}

.profile-card:hover .profile-name {
  color: #5e35b1;
}

.profile-bio {
  font-size: 1.1rem;
  color: #555;
  line-height: 1.5;
```

```
  margin-bottom: 0;
  transition: color 0.3s ease;
}

.profile-card:hover .profile-bio {
  color: #444;
}

.profile-card-container {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  width: 100%;
  background-color: #f4f7fa;
}
```

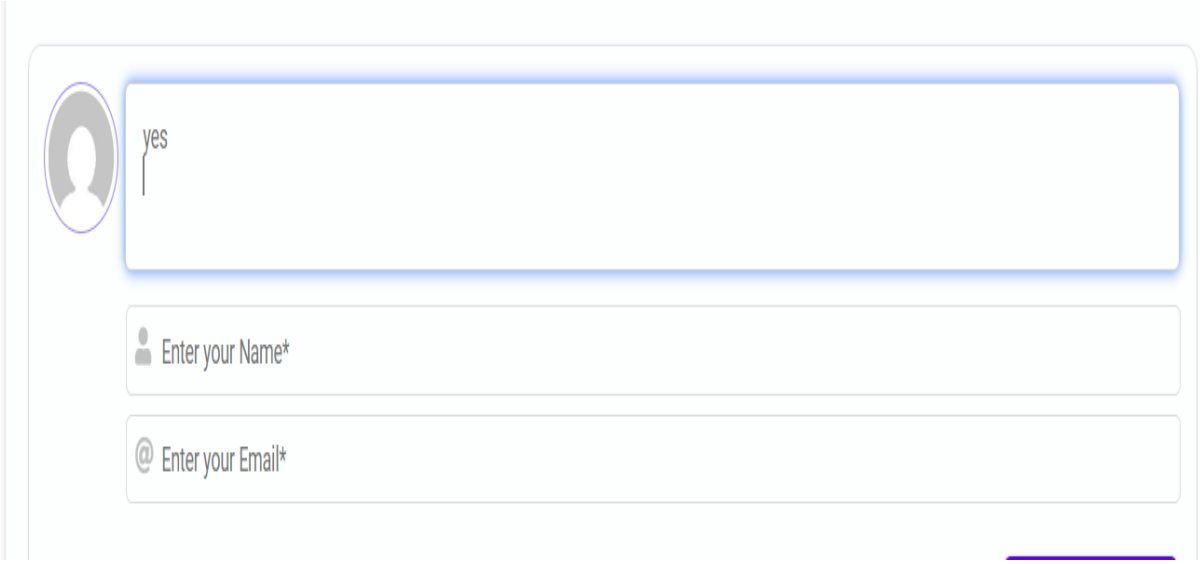
**Step 4: Start the Development Server**

1. In the terminal inside VSCode, run the following command to start the React development.

```
npm start
```

- This will open your browser and navigate to http://localhost:3000/. You should see your **ProfileCard** application up and running.

**OUTPUT:**



**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.

**Step 1: Create a New React Application**

First, you need to create a new React app using below command. Open your terminal and run:

npx create-react-app react-reminder-app

This will set up a new React project in a folder called **react-reminder-app**. After the installation is complete, navigate to the project directory:

React support services

Desktop application generator

cd react-reminder-app

**Step 2: Set Up the Folder Structure**

Create the folder structure. Here's how you can organize the directories:

1. **Inside the src folder**:
    - Create a **components** folder.
    - Inside components, create **Filter.js** , **TaskForm.js** and **TaskList.js** files. Copy below code and paste it into the different files.

**TaskForm.js:**

```
import React, { useState } from 'react';

function TaskForm({ addTask }) {
  const [taskName, setTaskName] = useState('');
  const [dueDate, setDueDate] = useState('');
  const [description, setDescription] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();

    if (taskName && dueDate) {
      const newTask = {
        id: Date.now(),
        name: taskName,
        dueDate: dueDate,
```

```
      description,
      completed: false,
    };
    addTask(newTask);

    setTaskName('');
    setDueDate('');
    setDescription('');
  }
};

return (
  <form onSubmit={handleSubmit}>
    <div>
      <input
        type="text"
        placeholder="Task Name"
        value={taskName}
        onChange={(e) => setTaskName(e.target.value)}
      />
    </div>
    <div>
      <input
        type="date"
        value={dueDate}
        onChange={(e) => setDueDate(e.target.value)}
      />
    </div>
    <div>
      <textarea
        placeholder="Description (optional)"
        value={description}
        onChange={(e) => setDescription(e.target.value)}
      />
    </div>
    <button type="submit">Add Task</button>
```

```
    </form>
  );
}

export default TaskForm;
```

**Filter.js:**

```
import React from 'react';

function Filter({ setFilter }) {
  return (
    <div>
      <button onClick={() => setFilter('all')}>All Tasks</button>
      <button onClick={() => setFilter('completed')}>Completed Tasks</button>
      <button onClick={() => setFilter('not-completed')}>Pending Tasks</button>
    </div>
  );
}

export default Filter;
```

**TaskList.js:**

```
import React from 'react';

function TaskList({ tasks, setTasks }) {
  const toggleTaskCompletion = (taskId) => {
    setTasks(
      tasks.map((task) =>
        task.id === taskId ? { ...task, completed: !task.completed } : task
      )
    );
  };

  const deleteTask = (taskId) => {
    setTasks(tasks.filter((task) => task.id !== taskId));
  };

  return (
```

```
    <div>
      {tasks.length > 0 ? (
        <ul>
          {tasks.map((task) => (
            <li key={task.id}>
              <h3>{task.name}</h3>
              <p>Due Date: {task.dueDate}</p>
              {task.description && <p>Description: {task.description}</p>}
              <p>Status: {task.completed ? 'Completed' : 'Not Completed'}</p>
              <button onClick={() => toggleTaskCompletion(task.id)}>
                {task.completed ? 'Mark as Not Completed' : 'Mark as Completed'}
              </button>
              <button onClick={() => deleteTask(task.id)}>Delete</button>
            </li>
          ))}
        </ul>
      ) : (
        <p>No tasks available!</p>
      )}
    </div>
  );
}

export default TaskList;
```

**Step 3. App Component(src/App.js):**

React support services

JavaScript courses

In your **src/App.js**, import the **Filter.js**, **TaskForm.js** and **TaskList.js** component and use it or copy the below code and paste it into the **App.js** file.

```
import React, { useState } from 'react';
import TaskForm from './components/TaskForm';
import TaskList from './components/TaskList';
import Filter from './components/Filter';
import './App.css';
```

```
function App() {
  const [tasks, setTasks] = useState([]);
  const [filter, setFilter] = useState('all');

  const addTask = (task) => {
    setTasks([...tasks, task]);
  };

  const handleFilterChange = (status) => {
    setFilter(status);
  };

  const filteredTasks = tasks.filter((task) => {
    if (filter === 'completed') return task.completed;
    if (filter === 'not-completed') return !task.completed;
    return true;
  });

  return (
    <div className="App">
      <h1>Task Reminder</h1>
      <TaskForm addTask={addTask} />
      <Filter setFilter={handleFilterChange} />
      <TaskList tasks={filteredTasks} setTasks={setTasks} />
    </div>
  );
}

export default App;
```

React support services

JavaScript courses

**Step 4: Add Styles(src/App.css)**

Add some styles in **src/App.css** to make the layout nicer. Copy the below code and paste it into the **App.css** file.

```
body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
```

```
  margin: 0;
  padding: 0;
  background-color: #f0f4f8;
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
}

.App {
  width: 550px;
  padding: 30px;
  background-color: #ffffff;
  border-radius: 12px;
  box-shadow: 0 4px 16px rgba(0, 0, 0, 0.1);
  transition: transform 0.3s ease, box-shadow 0.3s ease;
}

.App:hover {
  transform: translateY(-5px);
  box-shadow: 0 8px 24px rgba(0, 0, 0, 0.2);
}

h1 {
  font-size: 2.2rem;
  color: #333;
  text-align: center;
  margin-bottom: 10px;
  margin-top: 0;
}

form {
  display: flex;
  flex-direction: column;
  gap: 20px;
}
```

```
input,
textarea {
  padding: 12px;
  font-size: 1rem;
  border: 1px solid #ccc;
  border-radius: 8px;
  transition: border-color 0.3s ease;
}

input:focus,
textarea:focus {
  border-color: #4CAF50;
  outline: none;
}

button {
  background-color: #4CAF50;
  color: white;
  border: none;
  padding: 12px;
  font-size: 1rem;
  border-radius: 8px;
  cursor: pointer;
  transition: background-color 0.3s ease, transform 0.3s ease;
}

button:hover {
  background-color: #45a049;
}

button:active {
  transform: scale(0.98);
}

textarea {
```

```
  resize: vertical;
  min-height: 120px;
}

input[type="date"] {
  padding: 12px;
}

div {
  display: flex;
  flex-direction: column;
  gap: 10px;
}

ul {
  list-style-type: none;
  padding: 0;
}

li {
  background-color: #fafafa;
  margin: 15px 0;
  padding: 20px;
  border-radius: 12px;
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
  transition: transform 0.3s ease, box-shadow 0.3s ease;
}

li:hover {
  transform: translateY(-5px);
  box-shadow: 0 8px 20px rgba(0, 0, 0, 0.2);
}

h3 {
  margin: 0;
  font-size: 1.5rem;
```

```
  color: #333;
  font-weight: 600;
}

p {
  margin: 5px 0;
  color: #666;
}

button {
  background-color: #007BFF;
  color: white;
  border: none;
  padding: 8px 15px;
  font-size: 1rem;
  border-radius: 8px;
  cursor: pointer;
  transition: background-color 0.3s ease, transform 0.3s ease;
  margin-right: 10px;
}

button:hover {
  background-color: #0056b3;
}

button:active {
  background-color: #003f8d;
}

button:last-child {
  background-color: #e74c3c;
}

button:last-child:hover {
  background-color: #c0392b;
}
```

```
button:last-child:active {
  background-color: #7f1c1c;
}

.completed {
  text-decoration: line-through;
  color: #bbb;
}


div {
  display: flex;
  gap: 20px;
  justify-content: center;
}

button {
  background-color: #f1f1f1;
  color: #333;
  padding: 12px 18px;
  font-size: 1rem;
  border: 1px solid #ccc;
  border-radius: 8px;
  cursor: pointer;
  transition: background-color 0.3s ease, transform 0.3s ease;
}

button:hover {
  background-color: #ddd;
}

button:active {
  transform: scale(0.98);
}
```

```
button:focus {
  outline: none;
  border-color: #007BFF;
  }
```

**Step 5: Run the application**

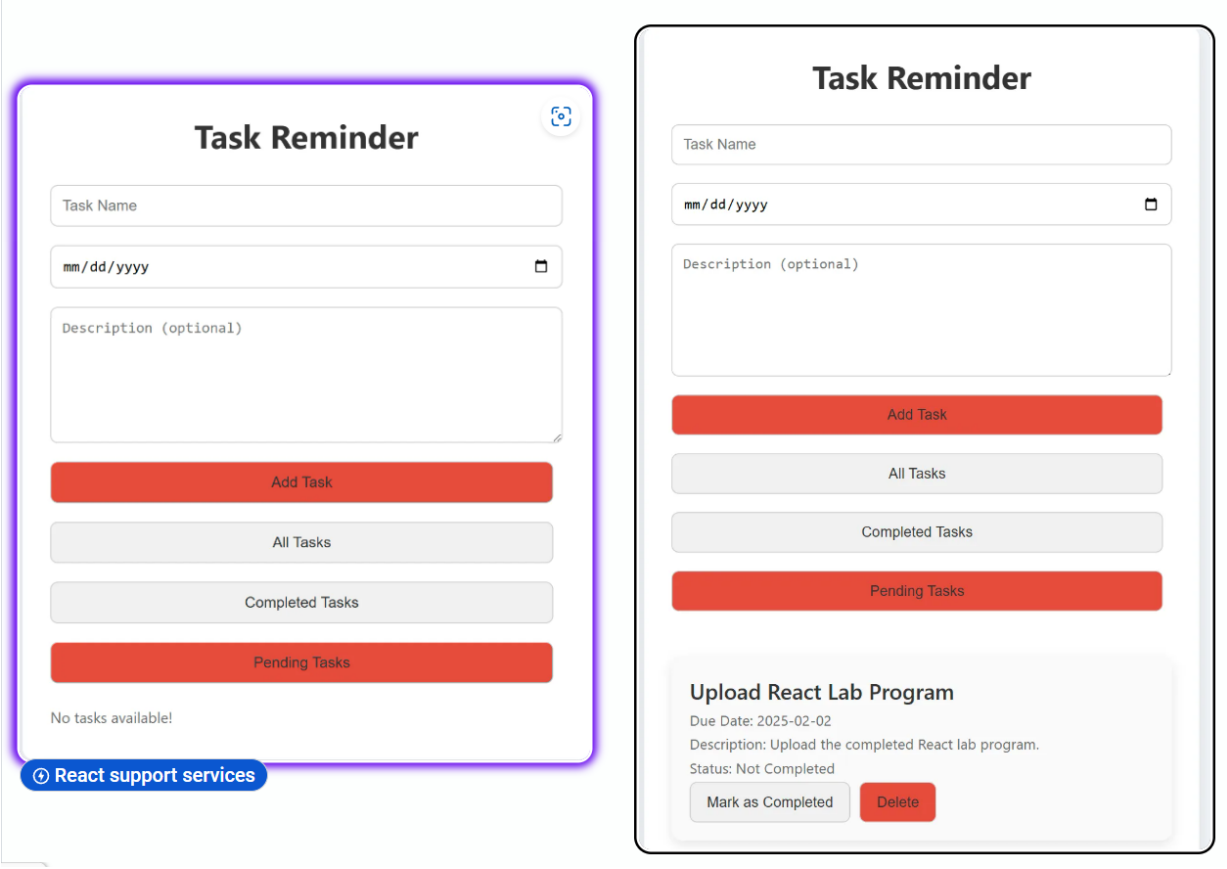
React support services

1. In the terminal inside VSCode, run the following command to start the React development.

```
npm start
```

- This will open your browser and navigate to http://localhost:3000/. You should see your **task reminder** application up and running.

**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.

**Step 1: Create a New React Application**

First, you need to create a new React app using below command. Open your terminal and run:

npx create-react-app my-routing-app

This will set up a new React project in a folder called **my-routing-app**. After the installation is complete, navigate to the project directory:

Programming software

cd my-routing-app

**Step 2: Install react-router-dom**

1. In the terminal inside VSCode, install react-router-dom:

npm install react-router-dom

**Step 3: Set Up the Folder Structure**

Create the folder structure. Here's how you can organize the directories:

1. **Inside the src folder**:
    - Create a **components** folder.
    - Inside components, create **Home.js** , **About.js**, **Contact.js and Navbar.js** files. Copy below code and paste it into the different files.

**Home.js:**

```
import React from 'react';

const Home = () => {
  return (
    <div>
      <h2>Home Page</h2>
      <p>Welcome to the Home Page!</p>
    </div>
  );
};

export default Home;
```

**About.js:**

```
import React from 'react';

const About = () => {
  return (
    <div>
      <h2>About Page</h2>
      <p>Learn more about us on the About Page!</p>
    </div>
  );
};

export default About;
```

**Contact.js:**

```
import React from 'react';

const Contact = () => {
  return (
    <div>
      <h2>Contact Page</h2>
      <p>Get in touch with us through the Contact Page!</p>
    </div>
  );
};

export default Contact;
```

**Navbar.js:**

```
import React from 'react';
import { NavLink } from 'react-router-dom';

const Navbar = () => {
  return (
    <nav>
      <ul>
        <li>
          <NavLink
            to="/"
```

```
          className={({ isActive }) => (isActive ? 'active' : '')}
        >
          Home
        </NavLink>
      </li>
      <li>
        <NavLink
          to="/about"
          className={({ isActive }) => (isActive ? 'active' : '')}
        >
          About
        </NavLink>
      </li>
      <li>
        <NavLink
          to="/contact"
          className={({ isActive }) => (isActive ? 'active' : '')}
        >
          Contact
        </NavLink>
      </li>
    </ul>
  </nav>
 );
};

export default Navbar;
```

**Step 4. App Component(src/App.js):**

In your **src/App.js**, import the **Home.js**, **About.js**, **Contact.js** and **Navbar.js** component and use it or copy the below code and paste it into the **App.js** file.

```
import React from 'react';
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
import Navbar from './components/Navbar';
import Home from './components/Home';
import About from './components/About';
import Contact from './components/Contact';
```

```
import './App.css'

const App = () => {
  return (
    <Router>
      <div>
        <Navbar />
        <div style={{ padding: '20px' }}>
          <Routes>
            <Route path="/" element={<Home />} />
            <Route path="/about" element={<About />} />
            <Route path="/contact" element={<Contact />} />
          </Routes>
        </div>
      </div>
    </Router>
  );
};

export default App;
```

**Step 5: Add Styles(src/App.css)**

Add some styles in **src/App.css** to make the layout nicer. Copy the below code and paste it into the **App.css** file.

```
body {
  font-family: Arial, sans-serif;
  background-color: #f4f4f4;
  margin: 0;
  padding: 0;
}

div {
  margin: 0 auto;
  max-width: 960px;
  padding: 20px;
}
```

```
h2 {
  color: #333;
  padding-bottom: 20px;
}

nav {
  background-color: #333;
  padding: 10px;
  border-radius: 5px;
  margin-bottom: 20px;
}

ul {
  list-style: none;
  display: flex;
  gap: 15px;
  justify-content: center;
  margin: 0;
  padding: 0;
}

li {
  display: inline;
}

a {
  text-decoration: none;
  color: white;
  padding: 8px 16px;
  border-radius: 4px;
}

a:hover {
  background-color: #444;
}
```

```
a.active {
  background-color: #1e90ff;
  color: white;
  font-weight: bold;
}

p {
  color: #555;
  font-size: 1.1rem;
  line-height: 1.6;
}
```

**Step 6: Set Up the Entry Point (index.js)**

1. Open **src/index.js** and ensure the entry point is correct:

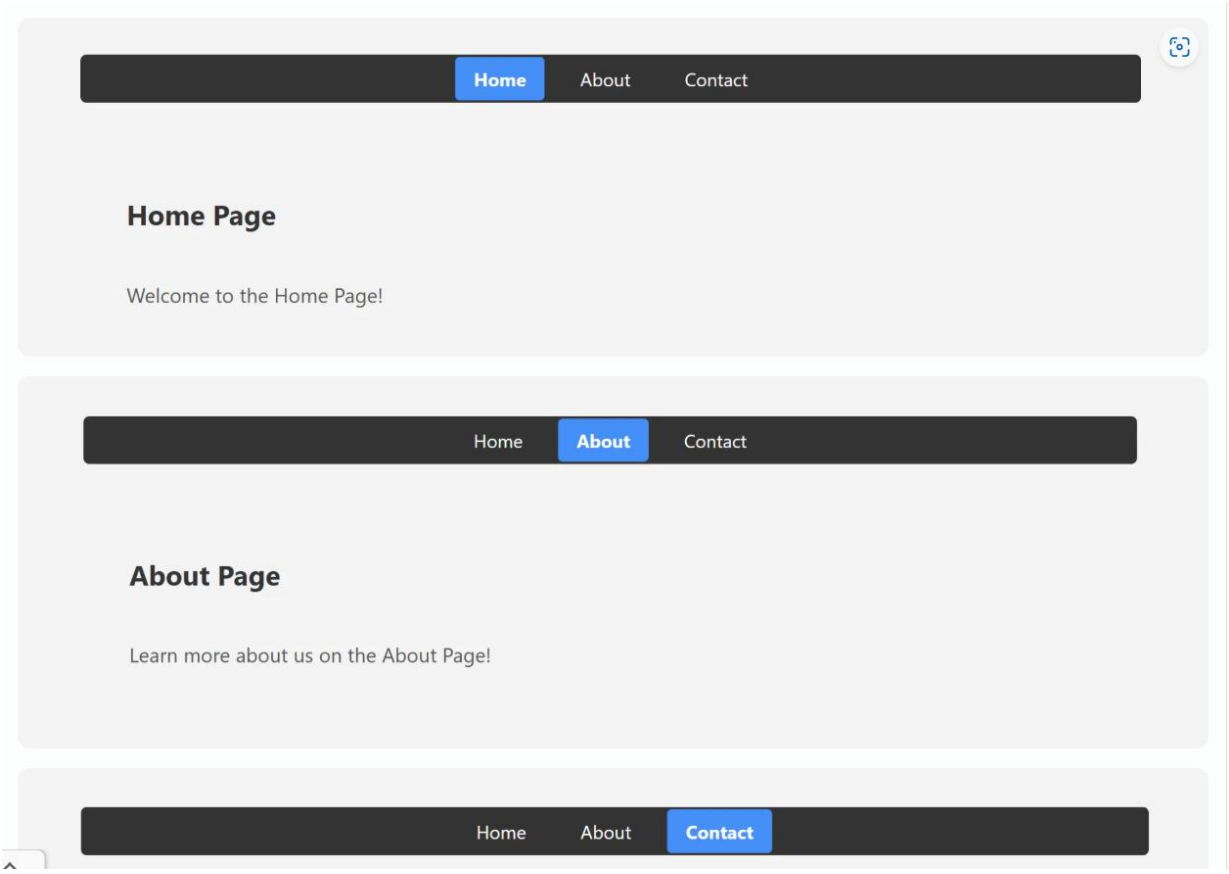
```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';

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

root.render(<App />);
```

**Step 7: Run the App**

1. Now that you've set up everything, go back to your terminal and run:

```
npm start
```

**OUTPUT:**


Home
About
Contact
Home Page
Welcome to the Home Page!
Home
About
Contact
About Page
Learn more about us on the About Page!
Home
About
Contact

**10.** Design a React application featuring a class-based component that demonstrates the use of lifecycle methods to interact with an external API. The component should fetch and update data dynamically based on user interactions or state changes. Use the componentDidMount lifecycle method to fetch data from an API when the component is initially rendered. Display the fetched data in a structured format, such as a table or list. Use the componentDidUpdate lifecycle method to detect changes in the component's state or props. Trigger additional API calls to update the displayed data based on user input or actions (e.g., filtering, searching, or pagination). Implement error handling to manage issues such as failed API requests or empty data responses. Display appropriate error messages to the user when necessary. Allow users to perform actions like filtering, searching, or refreshing the data. Reflect changes in the displayed data based on these interactions.

**Step 1: Create a New React Application**

- First, you need to create a new React app using below command. Open your terminal and run:

Front-end development tool

Software as a Service

npx create-react-app data-fetcher

This will set up a new React project in a folder called **data-fetcher**. After the installation is complete, navigate to the project directory:

cd data-fetcher

**Step 2: Update src/App.js:**

- **Navigate to the src folder** in the file explorer on the left-hand side of VSCode.
- Open the **App.js** file (which contains the default template code).
- **Replace the content of App.js with the code provided for the data-fetcher**. Here's the code to replace inside App.js:

```
import React, { Component } from 'react';

const API_URL = 'https://jsonplaceholder.typicode.com/users';

class DataFetcher extends Component {
  constructor(props) {
    super(props);
    this.state = {
      data: [],
      filteredData: [],
      searchQuery: '',
      error: null,
      loading: false,
```

```
  };
}

componentDidMount() {
  this.fetchData();
}

fetchData = async () => {
  this.setState({ loading: true, error: null });
  try {
    const response = await fetch(API_URL);
    if (!response.ok) {
      throw new Error('Failed to fetch data');
    }
    const data = await response.json();
    this.setState({ data, filteredData: data, loading: false });
  } catch (error) {
    this.setState({ error: error.message, loading: false });
  }
};

componentDidUpdate(prevProps, prevState) {
  if (prevState.searchQuery !== this.state.searchQuery) {
    this.filterData();
  }
}

handleSearchChange = (event) => {
  this.setState({ searchQuery: event.target.value });
};

filterData = () => {
  const { data, searchQuery } = this.state;
  if (searchQuery.trim() === '') {
    this.setState({ filteredData: data });
  } else {
```

```
    const filteredData = data.filter((item) =>
      item.name.toLowerCase().includes(searchQuery.toLowerCase())
    );
    this.setState({ filteredData });
  }
};

renderError = () => {
  const { error } = this.state;
  return error ? <div className="error">{`Error: ${error}`}</div> : null;
};

render() {
  const { filteredData, searchQuery, loading } = this.state;

  return (
    <div className="data-fetcher">
      <h1>User Data</h1>

      {this.renderError()}

      <div className="search-bar">
        <input
          type="text"
          value={searchQuery}
          onChange={this.handleSearchChange}
          placeholder="Search by name"
        />
      </div>

      {loading ? (
        <div>Loading...</div>
      ) : (
        <table>
          <thead>
            <tr>
```

```
            <th>Name</th>
            <th>Email</th>
            <th>City</th>
          </tr>
        </thead>
        <tbody>
          {filteredData.length > 0 ? (
            filteredData.map((item) => (
              <tr key={item.id}>
                <td>{item.name}</td>
                <td>{item.email}</td>
                <td>{item.address.city}</td>
              </tr>
            ))
          ) : (
            <tr>
              <td colSpan="3">No results found.</td>
            </tr>
          )}
        </tbody>
      </table>
    )}

    <button onClick={this.fetchData}>Refresh Data</button>
  </div>
 );
 }
}

export default DataFetcher;
```

Front-end development tool

**Step 3: Update src/index.js:**

- Replace the default content of **src/index.js** with this code to ensure the component is rendered in your application:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
```

```
import './App.css';
import DataFetcher from './App';

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

root.render(
  <React.StrictMode>
    <DataFetcher />
  </React.StrictMode>
);
```

**Step 4: Modify the App.css**

- You can adjust the styling if desired. For example, you can modify **App.css** to ensure the UI look good:

```
* {
  padding: 0;
  margin: 0;
  box-sizing: border-box;
}

body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 0;
  background-color: #f4f4f4;
}


button {
  border-radius: 5px;
  border: none;
  cursor: pointer;
  color: #fff;
  font-weight: bold;
  background: red;
  margin-top: 20px;
  padding: 10px;
```

```
}


.data-fetcher {
  width: 80%;
  margin: 0 auto;
  padding: 20px;
  background-color: #fff;
  border-radius: 8px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}

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

.search-bar {
  margin: 20px 0;
  text-align: center;
}

.search-bar input {
  padding: 8px;
  width: 60%;
  font-size: 16px;
  border: 1px solid #000;
  border-radius: 4px;
}

table {
  width: 100%;
  margin-top: 20px;
  border-collapse: collapse;
}
```

```
table th,
table td {
  padding: 10px;
  text-align: left;
  border-bottom: 1px solid #ddd;
}

.error {
  color: red;
  text-align: center;
}
```

**Step 5: Start the Development Server**

1. In the terminal inside VSCode, run the following command to start the React development.

Front-end development tool

```
npm start
```

**OUTPUT:**