ACS COLLEGE OF ENGINEETRING

(Accredited by NBA, NAAC Approved by A.I.C.T.E. New Delhi, Recognized by Govt. of Karnataka & Affiliated to V.T U.,

Kambipura, Mysore Road, Bangalore -560 074

DEPARTMENT OF COMPUTERSCIENCE AND ENGINEERING

LABORATORYMANUAL

React Laboratory (BCSL657B)

(Effectivefromtheacademicyear2024-2025)

Preparedby,

Mrs. Ashwini S

Department of Computer Science & Engineering

ACS College of Engineering

Kambipura,MysoreRoad,Bengaluru-56007

Template for Practical Course and if AEC is a practical Course Annexure-V

REACT		Semester	6
Course Code	**BCSL657B**	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	0:0:1:0	SEE Marks	50
Credits	01	Exam Hours	100
Examination type (SEE)	Practical		

Course objectives:

- Enable students to develop React applications utilizing functional and class-based components, effectively managing state with hooks and lifecycle methods .
- Introduce, how to pass data dynamically between parent and child components using props, ensuring modular and reusable component design.
- Create dynamic and responsive applications, integrating forms, validation, task management systems, and styled components.
- Use React Router for navigation, external API integration for dynamic data handling, and CSS styling techniques for modern UI/UX design.

Sl.NO	**Experiments**
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 <h1> element that displays text based on the input. Dynamically update the <h1> content as the user types.
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.
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.
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.

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.
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.
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.
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.
9.	9Design 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

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.

Course outcomes (Course Skill Set):

At the end of the course the student will be able to:

● Illustrate React basics and state components.
● Develop React applications that utilize component composition, passing data through props. ● Use dynamic state updates, event handling, and custom logic to increment, decrement, and reset state values. ● Implement forms in React that collect and validate user input.
● Demonstrate interaction with external APIs, dynamic content generation and manage state in real-time applications.

Template for Practical Course and if AEC is a practical Course Annexure-V

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together

Continuous Internal Evaluation (CIE):

CIE marks for the practical course are **50 Marks**.

The split-up of CIE marks for record/ journal and test are in the ratio **60:40**. ● Each experiment is to be evaluated for conduction with an observation sheet and record write-up. Rubrics for the evaluation of the journal/write-up for hardware/software experiments are designed by the faculty who is handling the laboratory session and are made known to students at the beginning of the practical session.

● Record should contain all the specified experiments in the syllabus and each experiment write-up will be evaluated for 10 marks.

● Total marks scored by the students are scaled down to **30 marks** (60% of maximum marks).

● Weightage to be given for neatness and submission of record/write-up on time. ● Department shall conduct a test of 100 marks after the completion of all the experiments listed in the syllabus.

● In a test, test write-up, conduction of experiment, acceptable result, and procedural knowledge will carry a weightage of 60% and the rest 40% for viva-voce.

● The suitable rubrics can be designed to evaluate each student's performance and learning ability.

● The marks scored shall be scaled down to **20 marks** (40% of the maximum marks). The Sum of scaled-down marks scored in the report write-up/journal and marks of a test is the total CIE marks scored by the student.

Semester End Evaluation (SEE):

● SEE marks for the practical course are 50 Marks.

● SEE shall be conducted jointly by the two examiners of the same institute, examiners are appointed by the Head of the Institute.

● The examination schedule and names of examiners are informed to the university before the conduction of the examination. These practical examinations are to be conducted between the schedule mentioned in the academic calendar of the University.

● All laboratory experiments are to be included for practical examination. ● (Rubrics) Breakup of marks and the instructions printed on the cover page of the answer script to be strictly adhered to by the examiners. **OR** based on the course requirement evaluation rubrics shall be decided jointly by examiners.

Template for Practical Course and if AEC is a practical Course Annexure-V

- Students can pick one question (experiment) from the questions lot prepared by the examiners jointly.
- Evaluation of test write-up/ conduction procedure and result/viva will be conducted jointly by examiners.
- General rubrics suggested for SEE are mentioned here, writeup-20%, Conduction procedure and result in -60%, Viva-voce 20% of maximum marks. SEE for practical shall be evaluated for 100 marks and scored marks shall be scaled down to 50 marks (however, based on course type, rubrics shall be decided by the examiners)

Change of experiment is allowed only once and 15% of Marks allotted to the procedure part are to be made zero.

The minimum duration of SEE is 02 hours

Suggested Learning Resources:

Books:

1. Beginning React JS Foundations Building User Interfaces with ReactJS: An Approachable Guide, Chris Minnick, Wiley publications , 2022.
2. Learning React Functional Web Development with React and Redux , Alex Banks, Eve Porcello · 2017

Web links and Video Lectures (e-Resources):

- https://www.youtube.com/watch?v=V9i3cGD-mts
- https://youtu.be/PHaECbrKgs0
- https://youtu.be/uvEAvxWvwOs
- https://www.geeksforgeeks.org/state-management-with-usestate-hook-in-react/
- https://youtu.be/KU-I2M9Jm68
- https://youtu.be/H63Pd_lXkeQ
- https://youtu.be/oTIJunBa6MA
- https://youtu.be/3EbYJrAOpUs

REACT LAB MANUAL - INDEX

INDEX

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 an element that displays text based on the input. Dynamically update the content as the user types.**

Solution:

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
```

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

```
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;
```

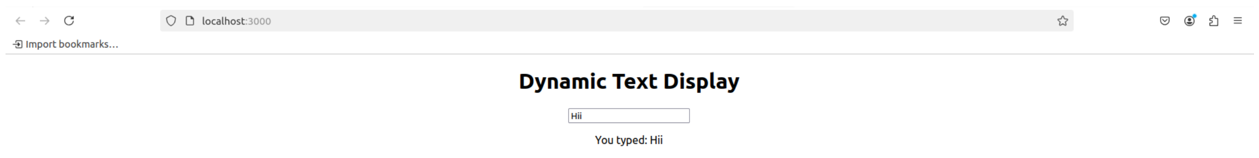
Step 3: Run the application

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

<u>npm start</u>

This will open the app in your default web browser, typically at **http://localhost:3000**, and you should see an input field where you can type, and the content will update dynamically as you type.

<u>OUTPUT:</u>

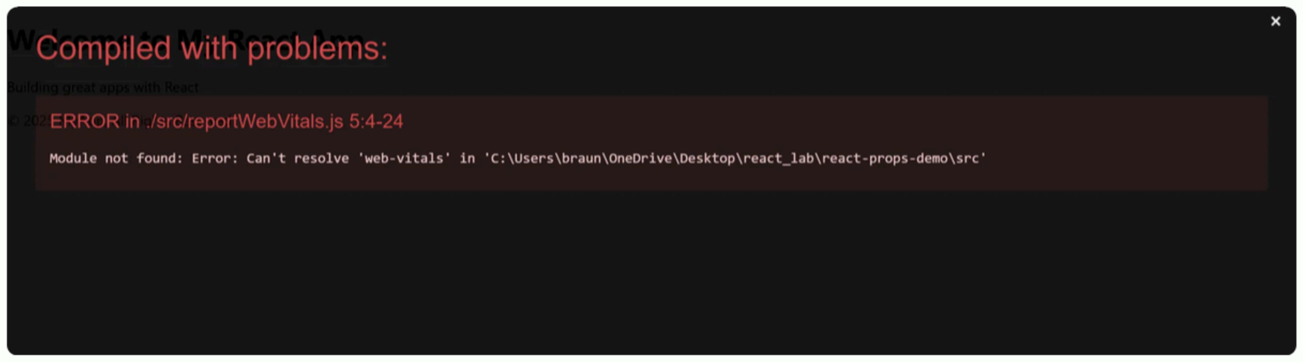


Dynamic Text Display

Hii

You typed: Hii

"Fixing the Module not found: Error: Can't resolve 'web-vitals' Error in React"



```
Compiled with problems:

ERROR in ./src/reportWebVitals.js 5:4-24

Module not found: Error: Can't resolve 'web-vitals' in 'C:\Users\braun\OneDrive\Desktop\react_lab\react-props-demo\src'
```

The error you're seeing occurs because the web-vitals package, which is used for performance monitoring in a React app, is not installed by default in the project or has been removed. Since web-vitals is an optional package, you can safely resolve this issue by either installing the package or removing the code that imports it.

Option 1: Install the web-vitals package
if you want to keep the performance monitoring functionality and resolve the error, simply install the web-vitals package.

1. In the terminal, navigate to your project folder (if not already there):

   cd my-dynamic-app

2. Install web-vitals by running the following command:

   npm install web-vitals

3. After installation is complete, restart the development server:

   npm start

This should resolve the error, and your application should compile correctly.

Option 2: Remove the Web Vitals Code (If Not Needed)
If you don't need performance monitoring and want to get rid of the error, you can safely remove the import and usage of web-vitals from your code.

1. Open src/reportWebVitals.js and remove its contents or just comment out the code:
2. Save the file, and the application should compile without the error. You can now continue developing your app.

2. **Develop a React application that demonstrates the use of props to pass data from 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.**

Solution:

Step 1: Create a New React Application

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

```
npx create-react-app react-props-demo
```

This will set up a new React project in a folder called **react-props-demo**. After the installation is complete, 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} />

    );

}

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:

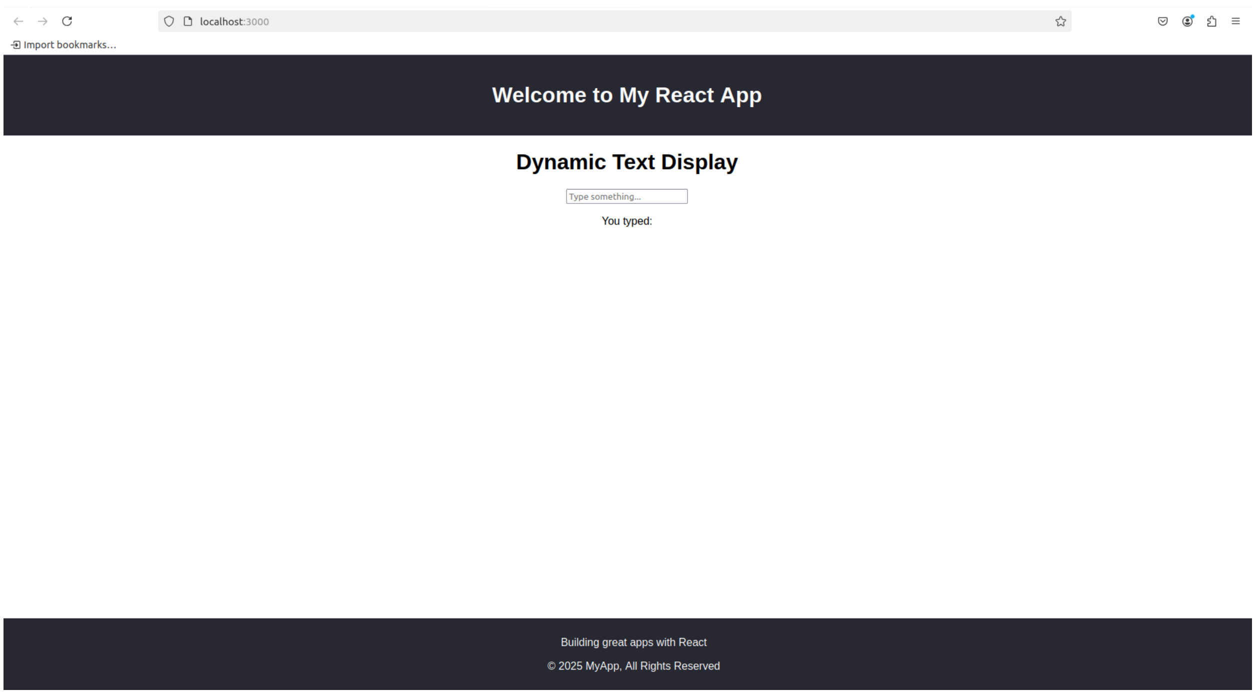
```
.App-header {

  background-color: #282c34;

  min-height: 100vh;

  display: flex;

  flex-direction: column;

  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}


.App-link {
  color: #61dafb;
}


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:



localhost:3000

Import bookmarks...

Welcome to My React App

Dynamic Text Display

Type something...

You typed:

Building great apps with React

© 2025 MyApp, All Rights Reserved

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.**

Solution:

Step 1: Create a new React app

First, you need to create a new React app using the 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:

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.

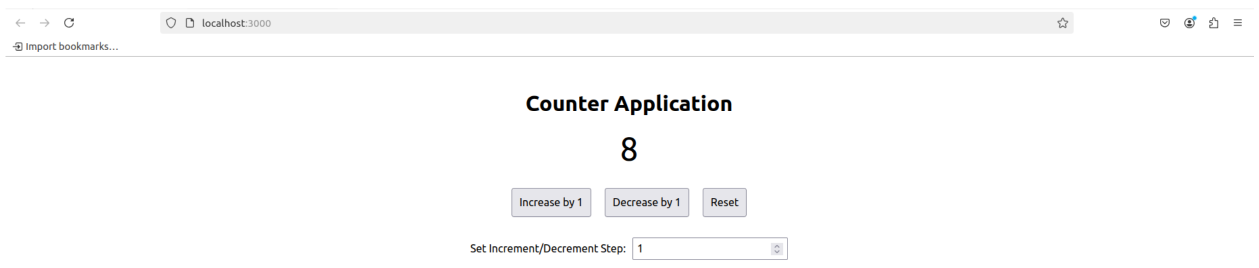
Step 4: Start the Development Server

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

```
npm start
```

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

OUTPUT:



4. After installation is complete, restart the development server:

npm start

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.**

Solution:

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 todo-app
```

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

```
cd todo-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 todo-app**. Here's the code to replace inside App.js:

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

import './App.css';



const ToDoFunction = () => {

  const [tasks, setTasks] = useState([]);

  const [newTask, setNewTask] = useState('');



  const addTask = () => {
        if (newTask.trim()) {

        setTasks([

        ...tasks,

        { id: Date.now(), text: newTask, completed: false },

        ]);

        setNewTask('');

        }

};


const deleteTask = (taskId) => {

        setTasks(tasks.filter(task => task.id !== taskId));

};


const toggleTaskCompletion = (taskId) => {

        setTasks(tasks.map(task =>

        task.id === taskId

        ? { ...task, completed: !task.completed }

        : task

        ));

};


return (

        <div className="todo-container">

        <h2 className="todo-header">To-Do List</h2>
```

```
<div className="todo-input-wrapper">

<input

type="text"

value={newTask}

onChange={(e) => setNewTask(e.target.value)}

placeholder="Add a new task..."

className="todo-input"

/>

<button className="add-task-button" onClick={addTask}>Add Task</button>

</div>


<ul className="todo-list">

{tasks.map((task) => (

<li

key={task.id}

className={`todo-item ${task.completed ? 'completed' : ''}`}

>

<span

className="task-text"

onClick={() => toggleTaskCompletion(task.id)}

>

{task.text}

</span>
```

```
            <button
            className="delete-button"
            onClick={() => deleteTask(task.id)}
            >
            ❌
            </button>
            </li>
            ))}
            </ul>
            </div>
  );
};

export default ToDoFunction;
```

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:

```
.todo-container {
  font-family: 'Arial', sans-serif;
  max-width: 500px;
  margin: 50px auto;
  padding: 20px;
  border: 1px solid #e0e0e0;
  border-radius: 8px;
  background-color: #f9f9f9;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
  text-align: center;
}

.todo-header {
  color: #4A90E2;
  font-size: 2rem;
  margin-bottom: 20px;
}

.todo-input-wrapper {
  display: flex;
  justify-content: center;
  margin-bottom: 20px;
}

.todo-input {
  width: 70%;
  padding: 10px;
  border-radius: 4px;
  border: 1px solid #ccc;
  font-size: 1rem;
  outline: none;
}
```

```
.add-task-button {
  padding: 10px 15px;
  margin-left: 10px;
  background-color: #4CAF50;
  color: white;
  border: none;
  border-radius: 4px;
  font-size: 1rem;
  cursor: pointer;
  transition: background-color 0.3s;
}

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

.todo-list {
  list-style-type: none;
  padding-left: 0;
  margin: 0;
}

.todo-item {
  display: flex;

  align-items: center;

  justify-content: space-between;

  background-color: #fff;

  padding: 12px;

  margin: 10px 0;

  border-radius: 5px;

  border: 1px solid #ddd;

  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

  transition: transform 0.2s ease-in-out;

}


.todo-item:hover {

  transform: scale(1.03);

}


.todo-item.completed {

  background-color: #f1f1f1;

  text-decoration: line-through;

  color: #aaa;

}


.task-text {

  cursor: pointer;
  font-size: 1.1rem;
  color: #333;
  transition: color 0.3s;
}

.task-text:hover {
  color: #4CAF50;
}

.delete-button {
  background: none;
  border: none;
  font-size: 1.1rem;
  color: #ff6347;
  cursor: pointer;
  transition: color 0.3s;
}

.delete-button:hover {
  color: #ff4500;
}
```

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

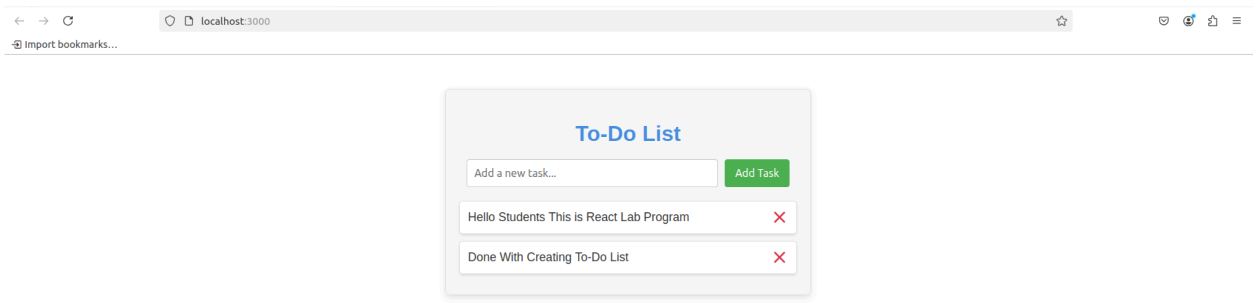
Step 4: Start the Development Server

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

npm start

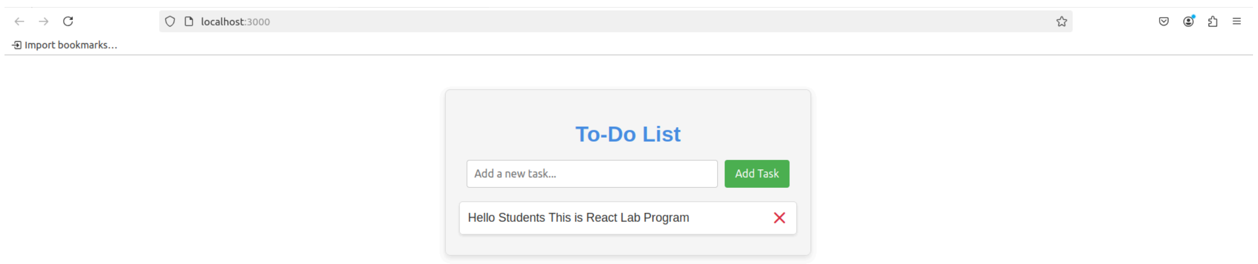
This will open your browser and navigate to http://localhost:3000/. You should see your Counter Application up and running.

OUTPUT:



Hello Students This is React Lab Program

Done With Creating To-Do List



Hello Students This is React Lab Program

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.**

Solution:

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 figure-gallery
```

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

```
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:
   - Create a components folder.
   - Inside components, create BasicFigure.js and FigureList.js.

BasicFigure.js:

```
// BasicFigure.js

import React from 'react';
```

```
const BasicFigure = ({ imageUrl, caption }) => {

  return (

    <div className="figure">

      <img src={imageUrl} alt={caption} className="figure-image" />

      <p className="figure-caption">{caption}</p>

    </div>

  );

};

export default BasicFigure;
```

FigureList.js:

If you want to use your own local images, follow these steps: Create a folder called images inside the public folder. Place your image (for example, placeholder-image.jpg) inside the public/images folder. In your FigureList.js, instead of using an online URL for placeholder images, reference your local image from the public/images folder. When referencing files from the public folder, you can use a relative path starting with /images/.

```
// FigureList.js

import React, { useState } from 'react';

import BasicFigure from './BasicFigure';


const FigureList = () => {

    const [figures, setFigures] = useState([

        { imageUrl: 'https://picsum.photos/400', caption: 'Random Image 1' },

        { imageUrl: 'https://picsum.photos/400', caption: 'Random Image 2' },

        { imageUrl: 'https://picsum.photos/400', caption: 'Random Image 3' },
    { imageUrl: 'https://picsum.photos/400', caption: 'Random Image 4' },

  ]);

  const addFigure = () => {
    const newFigure = {
      imageUrl: `https://picsum.photos/400?random=${figures.length + 1}`,
      caption: `Random Image ${figures.length + 1}`,
    };
    setFigures([...figures, newFigure]);
  };

  const removeFigure = () => {
    const updatedFigures = figures.slice(0, -1);
    setFigures(updatedFigures);
  };

  return (
    <div className="figure-list-container">
      <div className='button-box'>
        <button onClick={addFigure} className="action-button">Add
Image</button>
        <button onClick={removeFigure} className="action-button">Remove
Image</button>
      </div>
      <div className="figure-list">
        {figures.map((figure, index) => (
          <BasicFigure key={index} imageUrl={figure.imageUrl}
caption={figure.caption} />
        ))}
      </div>
    </div>
  );
};
export default FigureList;
```

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

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

```
// App.js
import React from 'react';
import FigureList from './components/FigureList';
import './App.css';


const App = () => {
  return (
    <div className="app">
      <h1>Dynamic Image Gallery</h1>
      <FigureList />
    </div>
  );
};


export default App;
```

Step 4: Add Some Basic 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.

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

h1 {
  background: #000;
  color: #fff;
  padding: 10px;
  text-align: center;
}

.figure-list-container {
  display: flex;
  flex-direction: column;
  align-items: center;
  margin: 20px;
}
.button-box {
  display: block;
  text-align: center;
  padding: 10px;
  margin-bottom: 20px;
}

.action-button {
  padding: 10px 20px;
  margin: 10px;
  background-color: #4CAF50;
  color: white;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  font-size: 16px;
  transition: background-color 0.3s ease;
}
```

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

.figure-list {
  display: flex;
  flex-wrap: wrap;
  justify-content: center;
  gap: 15px;
}

.figure-list img {
  max-width: 200px;
  max-height: 200px;
  border: 2px solid #ccc;
  border-radius: 8px;
}

figure {
  display: flex;
  flex-direction: column;
  align-items: center;
```

```
}

figcaption {
  margin-top: 8px;
  font-size: 14px;
  color: #555;
}

.figure {
  display: flex;
  flex-direction: column;
  align-items: center;
  border: 2px solid #ddd;
  border-radius: 8px;
  padding: 10px;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
  transition: transform 0.2s ease, box-shadow 0.2s ease;
}

.figure:hover {
  transform: translateY(-5px);
  box-shadow: 0 6px 12px rgba(0, 0, 0, 0.2);
```

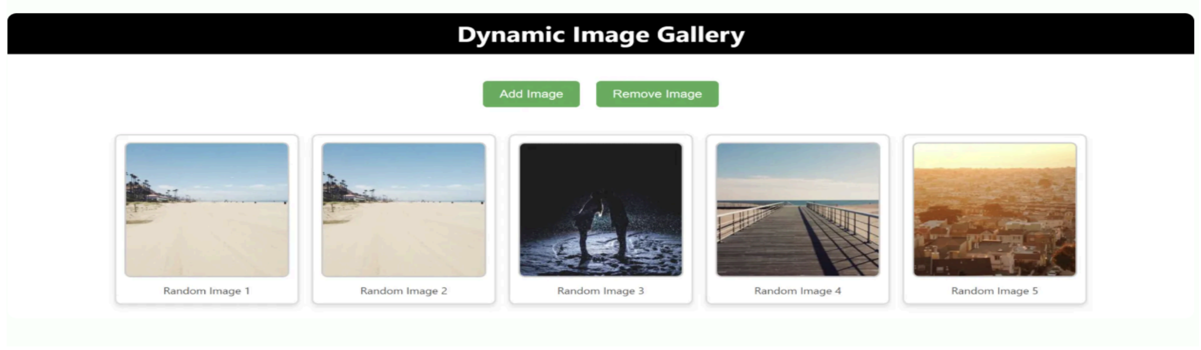
```
}

.figure-image {
  max-width: 200px;
  max-height: 200px;
  border-radius: 8px;
  object-fit: cover;
}

.figure-caption {
  margin-top: 10px;
  font-size: 14px;
  color: #555;
  text-align: center;
}
```

Step 5: Run the application

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

npm start

OUTPUT:



6. **Design and implement a React Form that collects user input for name, email, and password. Form Fields are Name, Email, and 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.**

Solution:

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-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: 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 Form.js file

Form.js

```
import React, { useState, useEffect, useCallback } from 'react';
import './Form.css';

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

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

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

  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData((prevState) => ({
      ...prevState,
      [name]: value.trim(),
    }));
  };
  const validateForm = useCallback(() => {
    let isValid = true;
    const newErrors = { name: '', email: '', password: '' };

    if (!formData.name) {
      newErrors.name = 'Name is required.';
      isValid = false;
    }

    const emailPattern = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/;
    if (!formData.email || !emailPattern.test(formData.email)) {
      newErrors.email = 'Please enter a valid email address.';
      isValid = false;
    }

    if (!formData.password) {
      newErrors.password = 'Password is required.';
      isValid = false;
    } else if (formData.password.length < 6) {
      newErrors.password = 'Password must be at least 6 characters long.';
      isValid = false;
    }

    setErrors(newErrors);
    setIsFormValid(isValid);
  }, [formData]);

  useEffect(() => {
    validateForm();
  }, [formData, validateForm]);

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

    if (isFormValid) {
      console.log('Form Data:', formData);
      setFormData({
        name: '',
        email: '',
        password: '',
    });
   }
  };

  return (
   <div className="form-container">
    <h2 className="form-title">Registration Form</h2>
    <form onSubmit={handleSubmit} className="form">
     <div className="form-group">
      <label htmlFor="name" className="form-label">Name</label>
      <input
       type="text"
       id="name"
       name="name"
       value={formData.name}
       onChange={handleChange}
       className={`form-input ${errors.name ? 'error' : ''}`}
       placeholder="Enter your name"
      />
      {errors.name && <div
className="error-message">{errors.name}</div>}
     </div>
```

```
        <div className="form-group">
          <label htmlFor="email" className="form-label">Email</label>
          <input
            type="email"
            id="email"
            name="email"
            value={formData.email}
            onChange={handleChange}
            className={`form-input ${errors.email ? 'error' : ''}`}
            placeholder="Enter your email"
          />
          {errors.email && <div
className="error-message">{errors.email}</div>}
        </div>


        <div className="form-group">
          <label htmlFor="password"
className="form-label">Password</label>
          <input
            type={showPassword ? 'text' : 'password'}
            id="password"
            name="password"
            value={formData.password}
            onChange={handleChange}
            className={`form-input ${errors.password ? 'error' : ''}`}
            placeholder="Enter your password"
          />
          {errors.password && <div
className="error-message">{errors.password}</div>}
        </div>

        <div className="form-group password-toggle">
          <label>
            <input
              type="checkbox"
              checked={showPassword}
              onChange={() => setShowPassword(!showPassword)}
            />
            Show Password
          </label>
        </div>

        <div className="form-group">
          <button type="submit" className="form-submit"
disabled={!isFormValid}>
            Submit
          </button>
        </div>
      </form>
    </div>
  );
};
export default Form;
```

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

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

```
import React from 'react';
import './App.css';
import Form from './components/Form';

function App() {
  return (
    <div className="App">
      <Form />
    </div>
  );
}
```

```
export default App;
```

Step 4: Add Some Basic 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.

```
.form-container {
  width: 100%;
  max-width: 500px;
  margin: 0 auto;
  padding: 20px;
  background-color: #f7f7f7;
  border-radius: 8px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}

.form-title {
  text-align: center;
  font-size: 24px;
  margin-bottom: 20px;
  color: #333;
}

.form {
  display: flex;
  flex-direction: column;
}

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

.form-label {
  font-size: 14px;
  font-weight: 600;
  color: #555;
}

.form-input {
  width: 100%;
  padding: 12px;
  margin-top: 5px;
  border: 1px solid #ddd;
  border-radius: 4px;
  font-size: 16px;
  box-sizing: border-box;
```

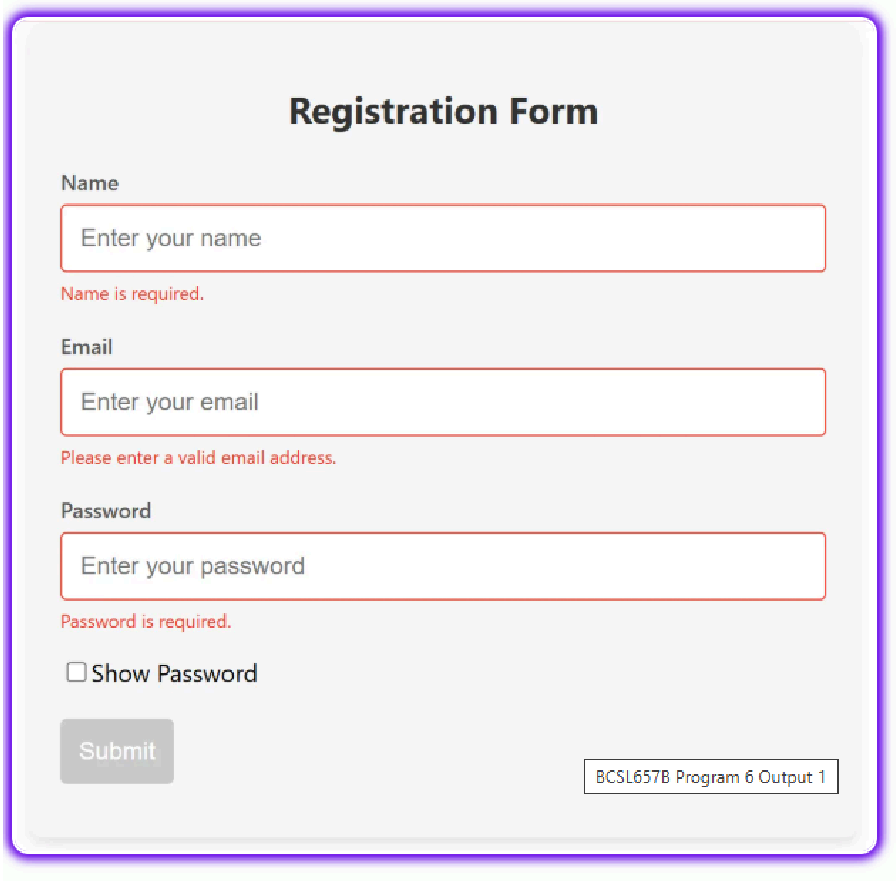
```
}

.form-input.error {
  border-color: red;
}

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

.password-toggle {
  margin-bottom: 20px;
}

.form-submit {
  padding: 12px;
  background-color: #4CAF50;
  color: white;
  border: none;
  border-radius: 4px;
  font-size: 16px;

  cursor: pointer;

  transition: background-color 0.3s;

}


.form-submit:disabled {

  background-color: #ccc;

  cursor: not-allowed;

}

.form-submit:hover:not(:disabled) {

  background-color: #45a049;

}
```

Step 5: Run the application

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

npm start

OUTPUT:



Registration Form

Name

Enter your name

Name is required.

Email

Enter your email

Please enter a valid email address.

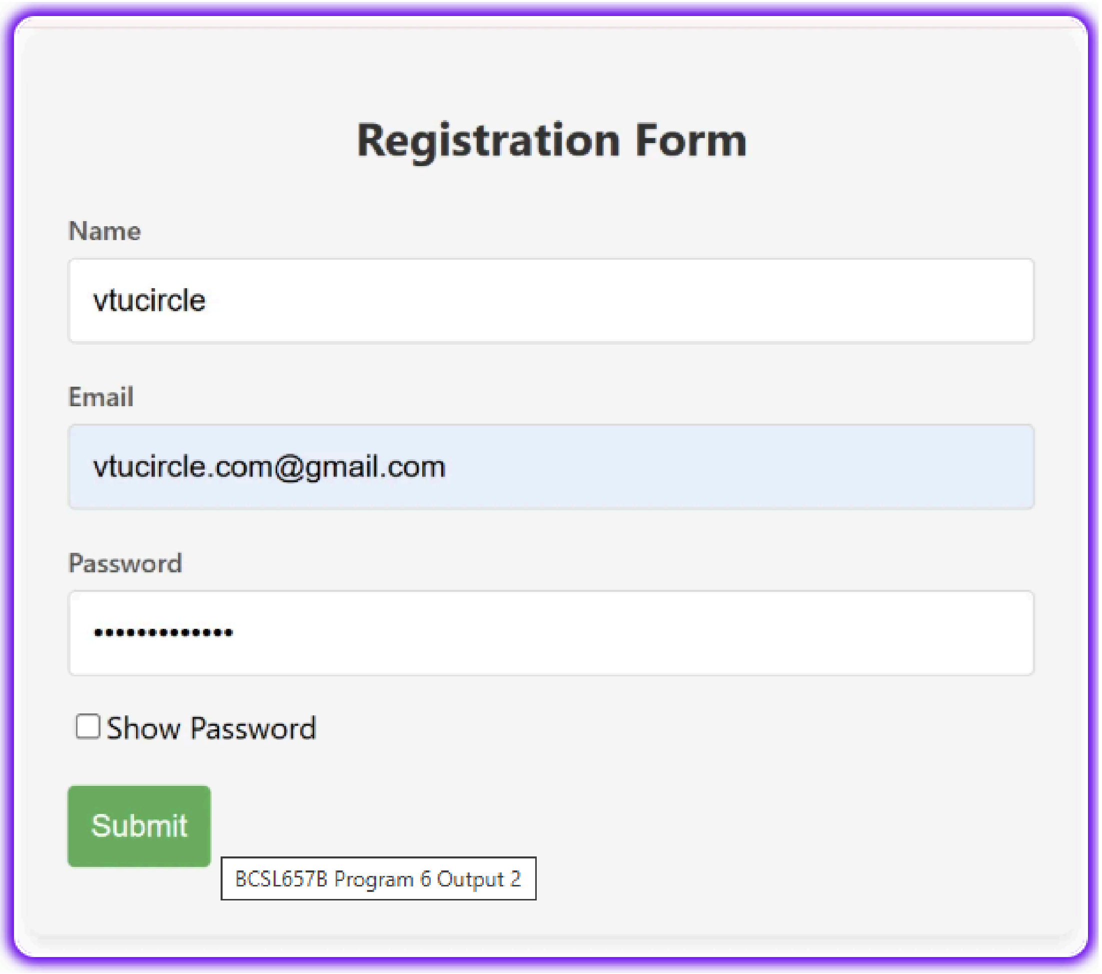
Password

Enter your password

Password is required.

☐ Show Password

BCSL657B Program 6 Output 1


Registration Form

Name

vtucircle

Email

vtucircle.com@gmail.com

Password

••••••••••••••

☐ Show Password

BCSL657B Program 6 Output 2

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.

Solution:

Step 1: Create a New React Application

First, you need to create a new React app using the 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:

```
cd profile-card-app
```

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.

```
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="vtu circle 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 looks good. Copy the below code and paste it in the App.css file.

```
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;

}
```

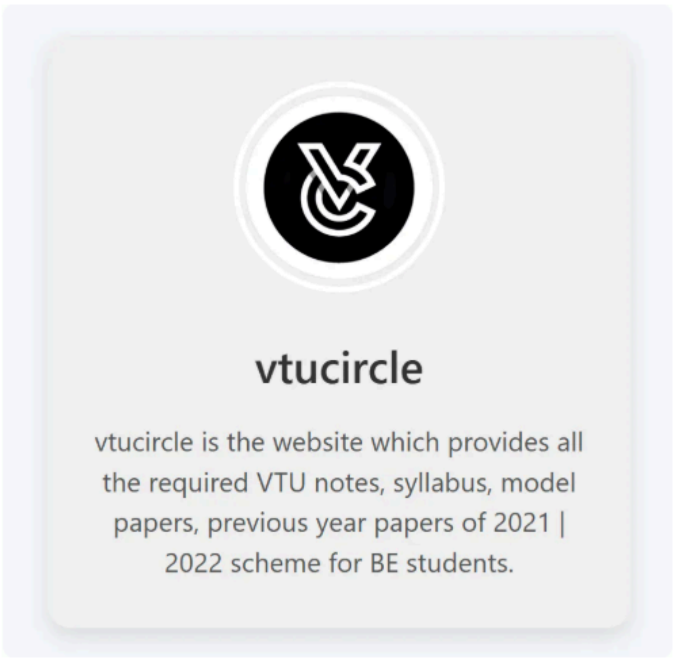
```
.profile-card-container {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  width: 100%;
}

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

OUTPUT:



vtucircle

vtucircle is the website which provides all the required VTU notes, syllabus, model papers, previous year papers of 2021 | 2022 scheme for BE students.

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.

Solution:

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:

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):

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;
```

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;
  }
```

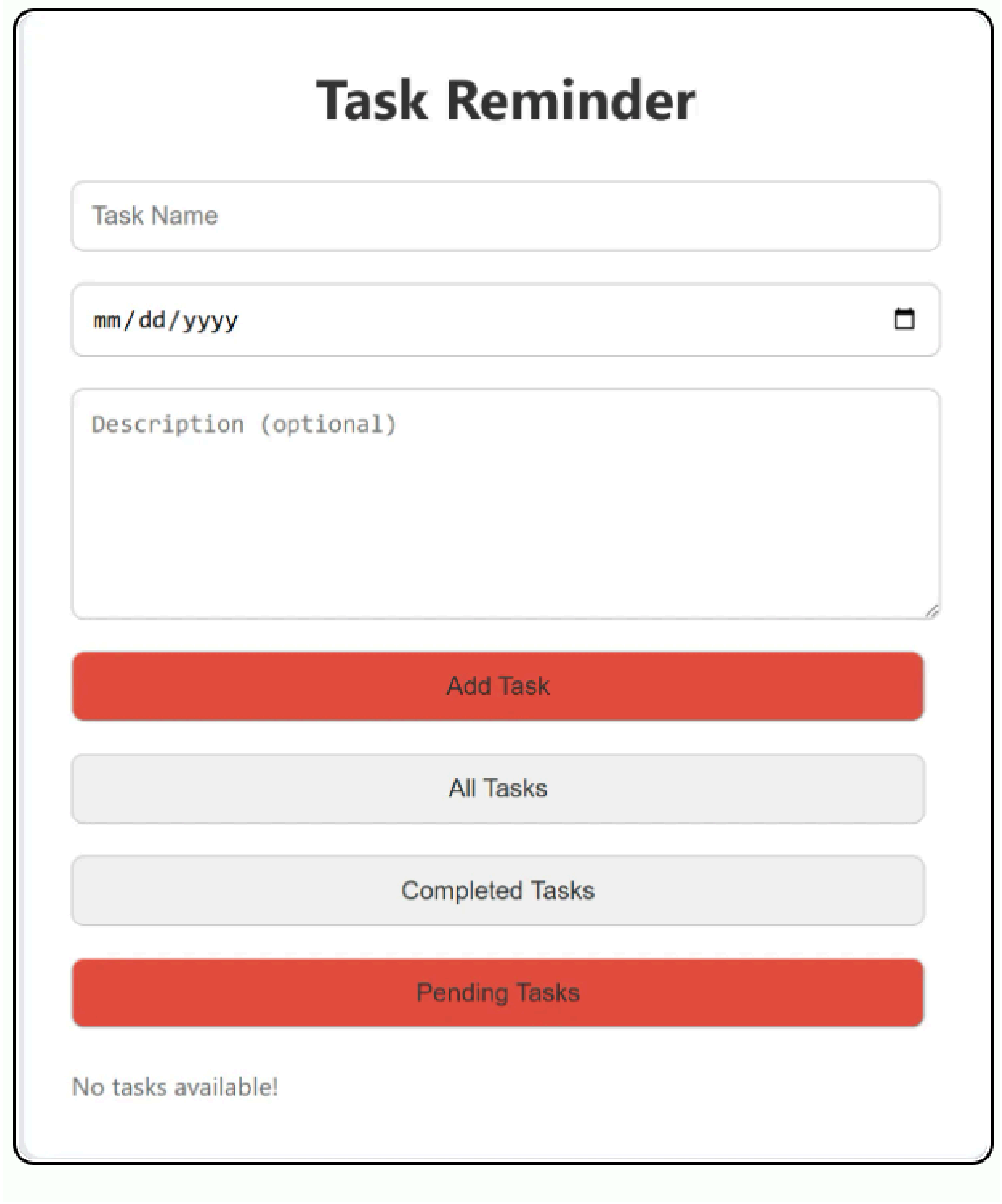
Step 5: Run the application

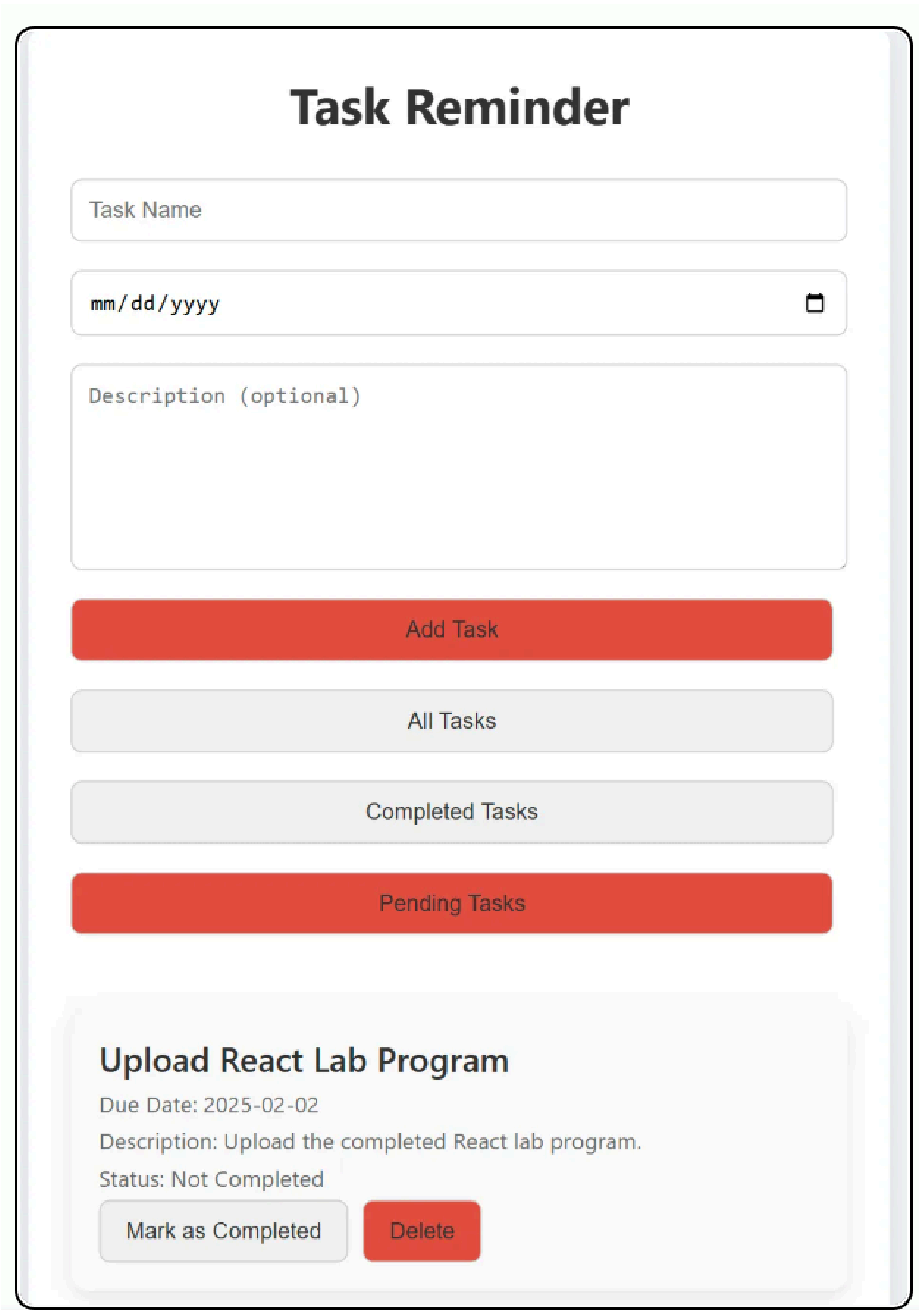
1. In the terminal inside VSCode, run the following command to start the React development.

```
npm start
```

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

OUTPUT:


Task Reminder
Task Name
mm/dd/yyyy
Description (optional)
Add Task
All Tasks
Completed Tasks
Pending Tasks
No tasks available!



Upload React Lab Program

Due Date: 2025-02-02

Description: Upload the completed React lab program.

Status: Not Completed

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.**

Solution:

Step 1: Create a New React Application

First, you need to create a new React app using the 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:

```
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 />);
```

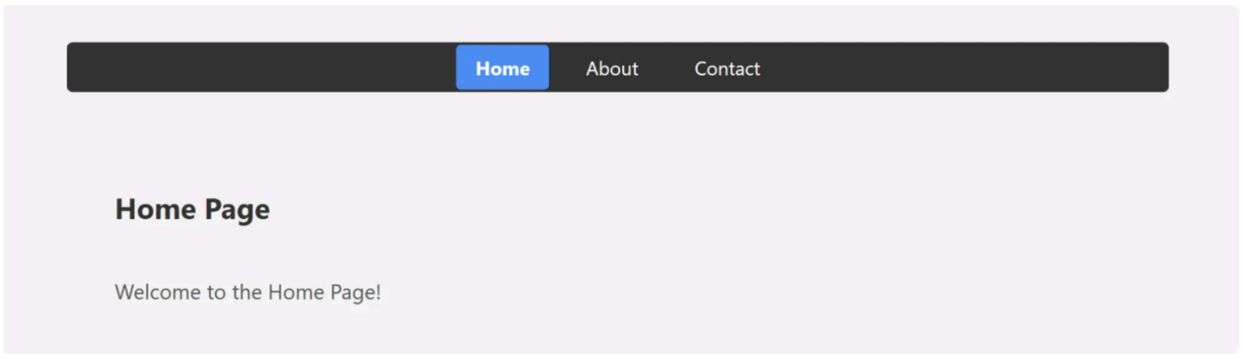
Step 7: Run the App

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

```
npm start
```

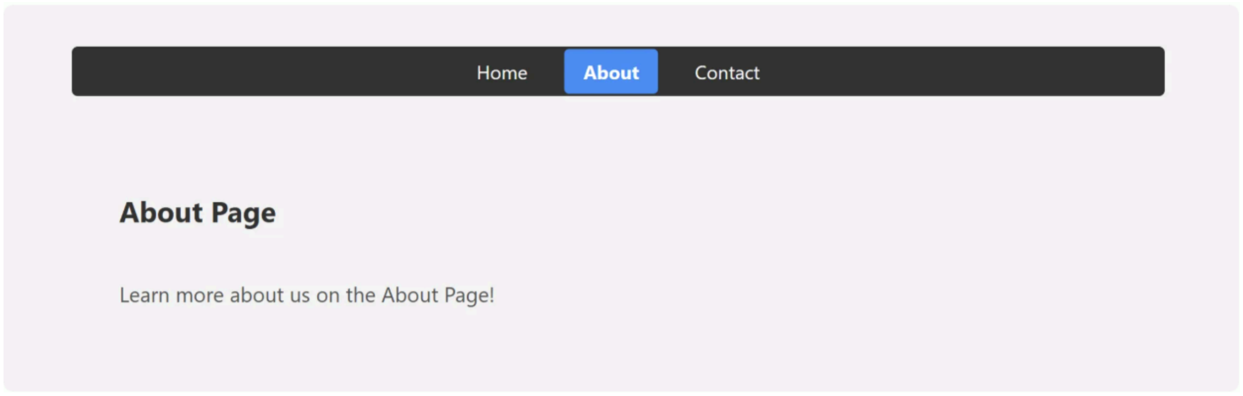
This will start your React app, and it should automatically open in your default browser at http://localhost:3000.

OUTPUT:



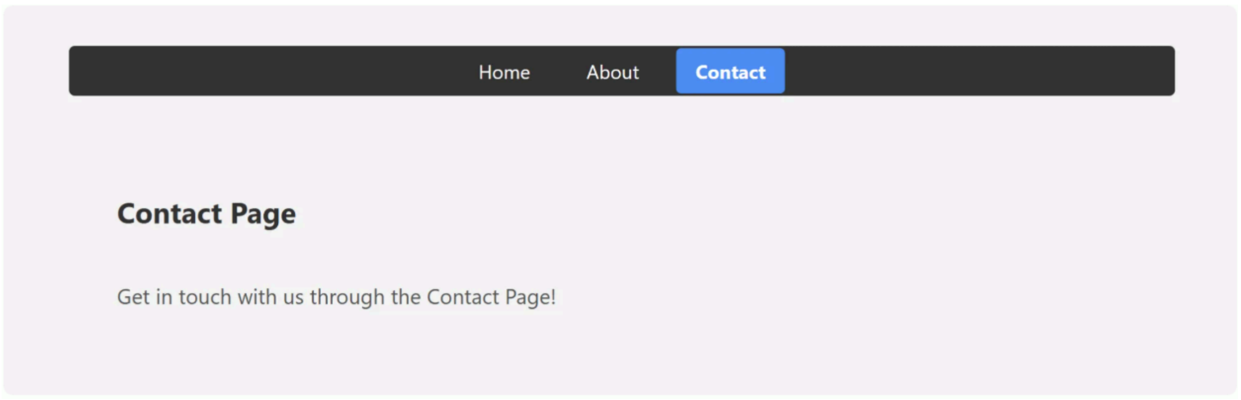
Home Page

Welcome to the Home Page!



About Page

Learn more about us on the About Page!



Contact Page

Get in touch with us through the Contact Page!

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.**

Solution:

Step 1: Create a New React Application

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

```
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;
```

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;
}
```

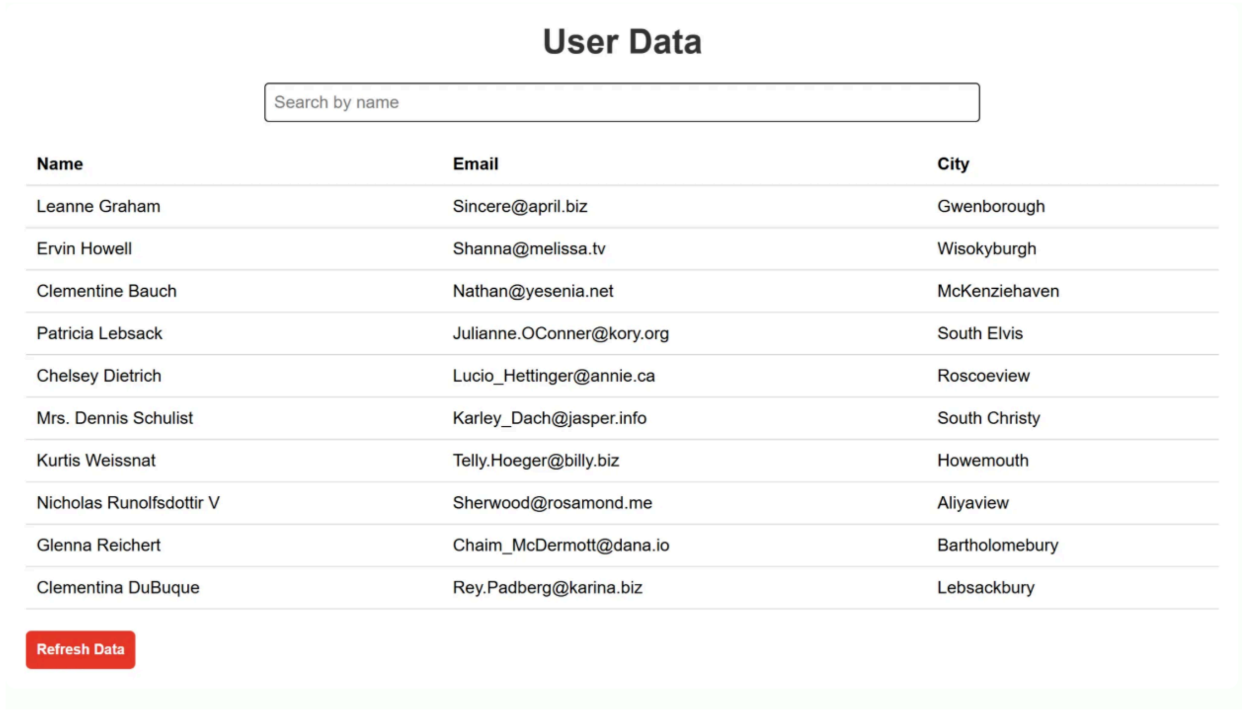
Step 5: Start the Development Server

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

npm start

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

<u>OUTPUT:</u>



User Data

Search by name

Name	Email	City
Leanne Graham	Sincere@april.biz	Gwenborough
Ervin Howell	Shanna@melissa.tv	Wisokyburgh
Clementine Bauch	Nathan@yesenia.net	McKenziehaven
Patricia Lebsack	Julianne.OConner@kory.org	South Elvis
Chelsey Dietrich	Lucio_Hettinger@annie.ca	Roscoeview
Mrs. Dennis Schulist	Karley_Dach@jasper.info	South Christy
Kurtis Weissnat	Telly.Hoeger@billy.biz	Howemouth
Nicholas Runolfsdottir V	Sherwood@rosamond.me	Aliyaview
Glenna Reichert	Chaim_McDermott@dana.io	Bartholomebury
Clementina DuBuque	Rey.Padberg@karina.biz	Lebsackbury

Refresh Data