Overview

Year / Semester	III Year		**Academic Year**	2024-25
Laboratory Title	REACT		**Laboratory Code**	BCSL657B
Total Contact Hours	20 Hours		**Duration of SEE**	0:0:1:0
IA Marks	50 Marks		**SEE Marks**	100
Lab Manual Author		**Sign -**	**Date**	
Checked By		**Sign -**	**Date**	

Objectives

React is a powerful JavaScript library designed for building fast, interactive, and scalable web applications. Here are the main objectives of React

1. Build Dynamic and Interactive UIs.
2. Component-Based Architecture
3. Efficient Rendering with Virtual DOM
4. Enable State Management and Data Flow
5. Simplify Development with Hooks
6. Provide Routing for Single-Page Applications (SPAs)

Description

1.0 Learning Objectives

By learning React.js, you will gain essential skills to build modern, interactive, and scalable web applications. The following learning objectives will guide you through the key concepts and functionalities of React.

1. Understanding the Basics of React
2. Setting Up a React Development Environment
3. Mastering JSX (JavaScript XML)
4. Creating and Managing Components
5. Managing State and Handling Events
6. Understanding React Hooks
7. Implementing React Router for Navigation
8. Styling React Applications
9. Debugging and Testing React Applications

2.0 Learning Outcomes

The students should be able to:
After completing the React program, students should be able to:

1. Understand React Fundamentals.
2. Set Up and Configure a React Development Environment
3. Develop and Manage React Components

4. Implement State Management and Event Handling
5. Work with React Router for Navigation
6. Fetch and Display Data from APIs
7. Apply Styling and UI Design in React
8. Debug and Optimize React Applications
9. Deploy a React Application
10. Develop Real-World React Projects

Prerequisites

1. Basic Knowledge of HTML, CSS, and JavaScript
2. Understanding of HTML structure and elements.
3. Familiarity with CSS for styling web pages.
4. Basic knowledge of JavaScript.

Base Course

This Base Course covers fundamental to intermediate concepts of React, ensuring that students can build dynamic and interactive web applications.

Resources Required

1. Hardware Requirements

Processor: Intel i3 or higher (i5/i7 recommended
RAM: Minimum 4GB (Recommended 8GB+ for smooth development)
Storage: At least 10GB free space (for dependencies and projects)
Operating System: Windows, macOS, or Linux

2. Software Requirements

Node.js (LTS Version) – Required to run React applications
npm (Node Package Manager) – Comes with Node.js for managing dependencies
Visual Studio Code (VS Code) – Code editor for writing React applications

General Instructions

1. Student should be punctual to the Lab
1. Required to prepare the Lab report every week
2. Required to maintain the Lab record properly
3. Should use the resources properly Student should be punctual to the Lab
4. Required to prepare the Lab report every week
5. Required to maintain the Lab record properly
6. Should use the resources properly

CONTENTS

Expt. 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 element that displays text based on the input. Dynamically update the 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 7	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. 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	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		

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.		

Evaluation Scheme

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.
● 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

Reference

1. React Tutorials & Courses

Experiments

1.0 Experiment (Sand Preparation)

Experiment No	Date Planed	Date Conducted	Marks
01.			

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

1.1 Learning Objectives

Creating a React project with create-react-app, use useState in App.js to manage input state, and dynamically update displayed text as the user types

1.2 Aim

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

1.3 Material / Equipment Required

PC/Laptop (Windows, macOS, or Linux)
Processor: Intel Core i3 or higher (Recommended)
RAM: Minimum **4GB** (Recommended: **8GB or more**)
Storage: At least **10GB free space**
Software : VS Code, Node.js & npm (Runtime & Package Manager)

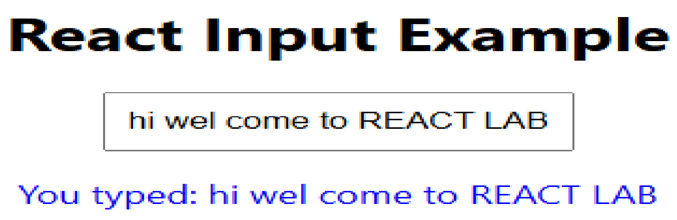
1.4 Theory / Hypothesis

1. Download and install **Node.js** from https://nodejs.org/ (LTS version recommended).
2. node –v
3. npm –v
4. npm install web-vitals
5. npx create-react-app my-dynamic-app
6. cd my-dynamic-app
7. npm start

1.5 Procedure / Program / Activity

```
import React, { useState } from "react";
function App() {
const [text, setText] = useState("");
 return (
<div style={{ textAlign: "center", marginTop: "50px" }}>
<h1>React Input Example</h1>
<input
type="text"
placeholder="Type something..."
value={text}
onChange={(e) => setText(e.target.value)}
style={{ padding: "10px", fontSize: "16px" }}
/>
<p style={{ fontSize: "20px", marginTop: "20px" }}>
You typed: <strong>{text}</strong>
</p>
</div>
);
}
export default App;
```

1.6 Results & Analysis


React Input Example

hi wel come to REACT LAB

You typed: hi wel come to REACT LAB

Experiments

2.0 Experiment (Sand Preparation)

Experiment No	Date Planed	Date Conducted	Marks
02			

TITLE: 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.

2.1 Learning Objectives

Create a React app with an App component as the parent. Use props to pass a title to a Header and Footer component.

2.2 Aim

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.

2.3 Material / Equipment Required

PC/Laptop (Windows, macOS, or Linux)
Processor: Intel Core i3 or higher (Recommended)
RAM: Minimum **4GB** (Recommended: **8GB or more**)
Storage: At least **10GB free space**
Software : VS Code, Node.js & npm (Runtime & Package Manager)

2.4 Theory / Hypothesis

1. Download and install **Node.js** from https://nodejs.org/ (LTS version recommended).
2. node –v
3. npm –v
4. npm install web-vitals
5. npx create-react-app my-app
6. cd my-app
7. npm start
8. Create Header.js
9. Create Footer.js
10. Create App.js

2.5 Procedure / Program / Activity

Create Header.js

```
import React from "react";
function Header({ title }) {
 return (
 <header style={{ backgroundColor: "#282c34", color: "white", padding: "10px", textAlign:
"center" }}>
<h1>{title}</h1>
</header>
 );
}
export default Header;
```

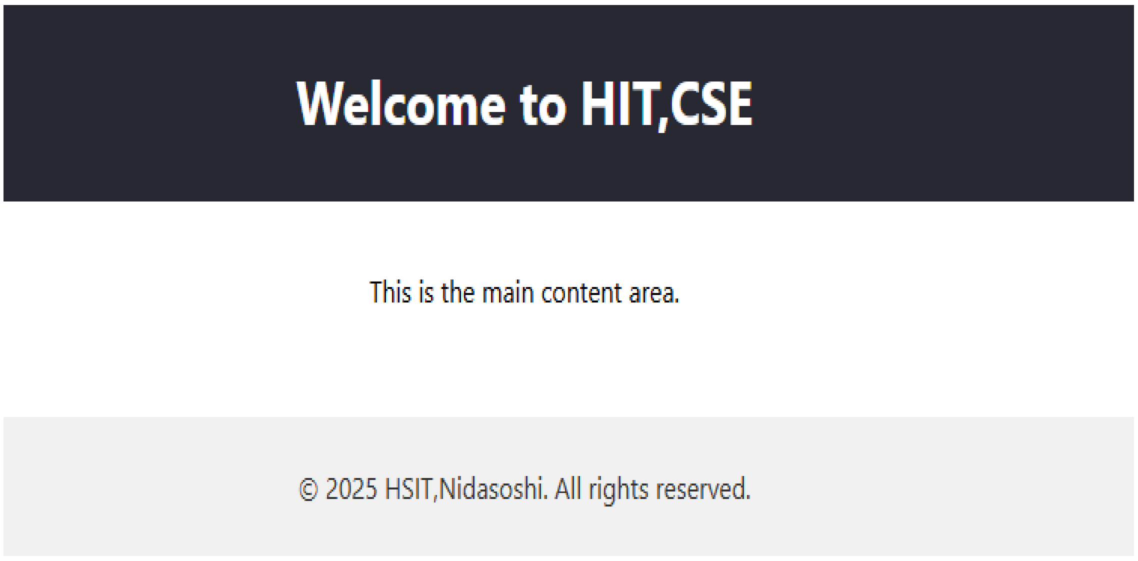
<u>Create Footer.js</u>

```
 import React from "react";
 function Footer({ copyright }) {
 return (
 <footer style={{ backgroundColor: "#f1f1f1", color: "#333", padding: "10px", textAlign:
 "center", marginTop: "20px" }}>
 <p>{copyright}</p>
 </footer>
 );
 }
 export default Footer;
```

<u>Create APP.js</u>

```
 import React from "react";
 import Header from "./Header";
 import Footer from "./Footer";
 function App() {
 const appTitle = "Welcome to HIT,CSE";
 const copyrightText = "© 2025 HSIT,Nidasoshi. All rights reserved.";
 return (
 <div>
 <Header title={appTitle} />
 <main style={{ textAlign: "center", padding: "20px" }}>
 <p>This is the main content area.</p>
 </main>
 <Footer copyright={copyrightText} />
 </div>
 );
 }
 export default App
```

2.6 Results & Analysis


Welcome to HIT,CSE

This is the main content area.

© 2025 HSIT,Nidasoshi. All rights reserved.

Experiments

3.0 Experiment (Sand Preparation)

Experiment No	Date Planed	Date Conducted	Marks
03			

TITLE: Create a Counter Application using React that demonstrates state management with the use State 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 use State 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.

3.1 Learning Objectives

Creating a React counter app using use State. Display the counter, with buttons to increase, decrease (min 0), and reset.

3.2 Aim

Create a Counter Application using React that demonstrates state management with the use State 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 use State 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.

3.3 Material / Equipment Required

PC/Laptop (Windows, macOS, or Linux)
Processor: Intel Core i3 or higher (Recommended)
RAM: Minimum **4GB** (Recommended: **8GB or more**)
Storage: At least **10GB free space**
Software : VS Code, Node.js & npm (Runtime & Package Manager)

3.4 Theory / Hypothesis

1. Install **Node.js** from https://nodejs.org/.

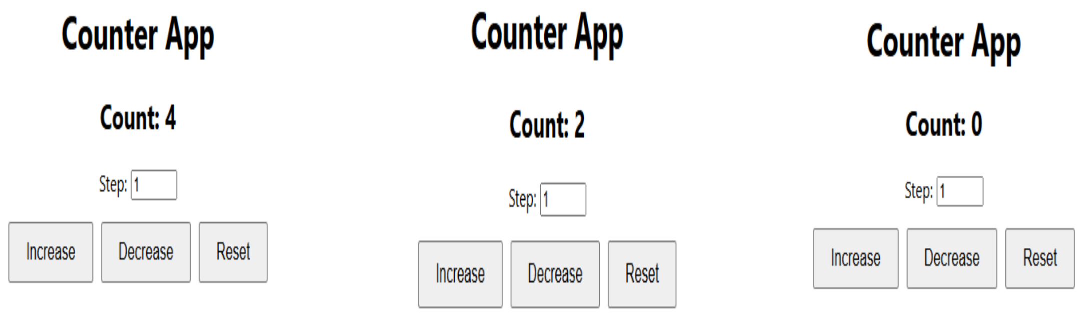
2. node –v
3. npm –v
4. npm install web-vitals
5. npx create-react-app counter-app
6. cd counter-app
7. npm start

3.5 Procedure / Program / Activity

App.js

```
import React, { useState } from "react";
function App() {
const [count, setCount] = useState(0);
const [step, setStep] = useState(1); // Default step value
const increase = () => setCount(prevCount => prevCount + step);
const decrease = () => setCount(prevCount => Math.max(0, prevCount - step));
const reset = () => setCount(0);
return (
<div style={{ textAlign: "center", marginTop: "50px" }}>
<h1>Counter App</h1>
<h2>Count: {count}</h2>
<label>Step: </label>
<input
type="number"
value={step}
onChange={(e) => setStep(Math.max(1, Number(e.target.value)))}
style={{ width: "50px", marginBottom: "10px" }}
/>
<br />
<button onClick={increase} style={buttonStyle}>Increase</button>
<button onClick={decrease} style={buttonStyle}>Decrease</button>
<button onClick={reset} style={buttonStyle}>Reset</button>
</div>
);
}
const buttonStyle = {
margin: "5px",
padding: "10px 20px",
fontSize: "16px",
cursor: "pointer"
};
export default App;
```

3.6 Results & Analysis



Experiments

4.0 Experiment (Sand Preparation)

Experiment No	Date Planed	Date Conducted	Marks
04			

TITLE: Develop a To-Do List Application using React functional components that demonstrates the use of the use State hook for state management. Create a functional component named ToDoFunction to manage and display the ToDoList. 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.

4.1 Learning Objectives

Build a React To-Do List app using useState in a ToDoFunction component. Allow users to add, delete, and toggle tasks between completed/pending states.

4.2 Aim

Develop a To-Do List Application using React functional components that demonstrates the use of the use State 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.

4.3 Material / Equipment Required

PC/Laptop (Windows, macOS, or Linux)
Processor: Intel Core i3 or higher (Recommended)
RAM: Minimum **4GB** (Recommended: **8GB or more**)
Storage: At least **10GB free space**
Software : VS Code, Node.js & npm (Runtime & Package Manager)

4.4 Theory / Hypothesis

1. Install **Node.js** from https://nodejs.org/.
2. node –v
3. npm –v
4. npm install web-vitals
5. npx create-react-app todo-app
6. cd todo-app
7. npm start

4.5 Procedure / Program / Activity

ToDoFunction.js

```
import React, { useState } from 'react';
function ToDoFunction() {
const [tasks, setTasks] = useState([]);
const [newTask, setNewTask] = useState('');

const handleAddTask = () => {
if (newTask.trim() !== '') {
setTasks([...tasks, { text: newTask, isCompleted: false }]);
setNewTask('');
}
};
const handleDeleteTask = (index) => {
const updatedTasks = tasks.filter((_, i) => i !== index);
setTasks(updatedTasks);
};
const handleToggleCompletion = (index) => {
const updatedTasks = tasks.map((task, i) =>
i === index ? { ...task, isCompleted: !task.isCompleted } : task
);
setTasks(updatedTasks);
};
return (
<div style={styles.container}>
```

```
<h1>To-Do List</h1>
<div style={styles.inputContainer}>
<input
 type="text"
 value={newTask}
 onChange={(e) => setNewTask(e.target.value)}
placeholder="Add a new task"
style={styles.input}
/>
<button onClick={handleAddTask} style={styles.addButton}>
Add
</button>
</div>
<ul style={styles.list}>
{tasks.map((task, index) => (
<li key={index} style={styles.listItem}>
<span
 onClick={() => handleToggleCompletion(index)}
 style={{
...styles.taskText,
 textDecoration: task.isCompleted ? 'line-through' : 'none',
 color: task.isCompleted ? 'gray' : 'black',
 }}
 >
{task.text}
 </span>
<button
onClick={() => handleDeleteTask(index)}
style={styles.deleteButton}
>
Delete
</button>
</li>
))}
</ul>
</div>
);
}
const styles = {
container: {
textAlign: 'center',
marginTop: '50px',
},
inputContainer: {
marginBottom: '20px',
},
input: {
```

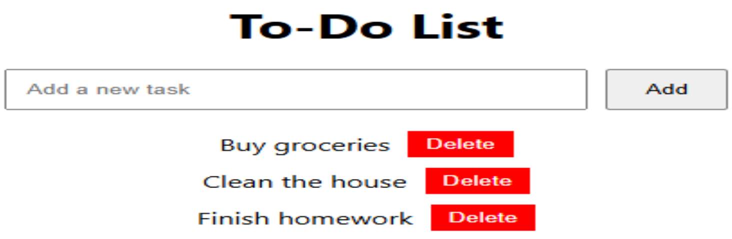
```
padding: '10px',
width: '300px',
marginRight: '10px',
},
addButton: {
padding: '10px 20px',
},
list: {
listStyleType: 'none',
padding: 0,
},
listItem: {
display: 'flex',
justifyContent: 'center',
alignItems: 'center',
marginBottom: '10px',
},
taskText: {
cursor: 'pointer',
marginRight: '10px',
},
deleteButton: {
padding: '5px 10px',
backgroundColor: 'red',
color: 'white',
border: 'none',
cursor: 'pointer',
},
};
export default ToDoFunction;
```

App.Js

```
import React from 'react';
import ToDoFunction from './ToDoFunction';
function App() {
return (
<div>
<ToDoFunction />
</div>
);
}
export default App;
```

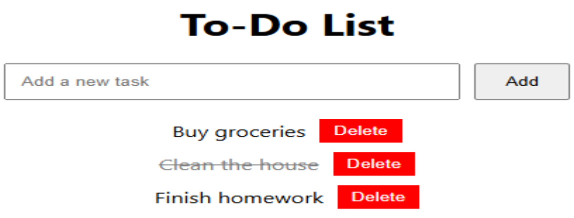
4.6 Results & Analysis

Insert task:


To-Do List

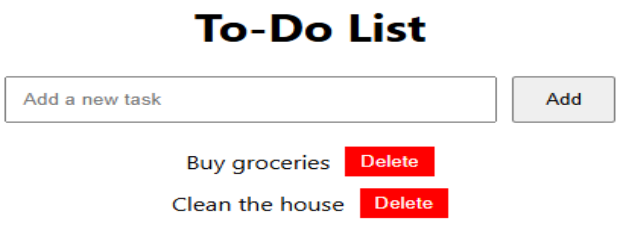
Buy groceries Delete
Clean the house Delete
Finish homework Delete

Toggle: Says task completed


To-Do List

Buy groceries Delete
~~Clean the house~~ Delete
Finish homework Delete

Delete:


To-Do List

Buy groceries Delete
Clean the house Delete

Experiments

5.0 Experiment (Sand Preparation)

Experiment No	Date Planed	Date Conducted	Marks
05			

TITLE: Develop a React application that demonstrates component composition and the use of props to pass data. Create two components: Figure List: A parent component responsible for rendering multiple child components. Basic Figure: A child component designed to display an image and its associated caption. Use the Figure List component to dynamically render multiple Basic Figure components. Pass image URLs and captions as props from the Figure List component to each Basic Figure component. Style the Basic Figure components to display the image and caption in an aesthetically pleasing manner. Arrange the Basic Figure components within the Figure List 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.

5.1 Learning Objectives

Create a React app with a Figure List parent component that renders multiple Basic Figure child components. Use props to pass image URLs and captions. Display images in a styled grid/list, allowing users to add/remove images dynamically. Enhance with hover effects or animations.

5.2 Aim

Develop a React application that demonstrates component composition and the use of props to pass data. Create two components: Figure List: A parent component responsible for rendering multiple child components. Basic Figure: A child component designed to display an image and its associated caption. Use the Figure List component to dynamically render multiple Basic Figure components. Pass image URLs and captions as props from the Figure List component to each Basic Figure component. Style the Basic Figure components to display the image and caption in an aesthetically pleasing manner. Arrange the Basic Figure components within the Figure List 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.

5.3 Material / Equipment Required

PC/Laptop (Windows, macOS, or Linux)
Processor: Intel Core i3 or higher (Recommended)
RAM: Minimum **4GB** (Recommended: **8GB or more**)
Storage: At least **10GB free space**
Software : VS Code, Node.js & npm (Runtime & Package Manager)

5.4 Theory / Hypothesis

1. npx create-react-app figure-gallery
2. cd figure-gallery
3. Create a components folder.
4. Inside components, create BasicFigure.js and FigureList.js.
5. npm start

5.5 Procedure / Program / Activity

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

```
// 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;
```

App.js

```
// 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;
```

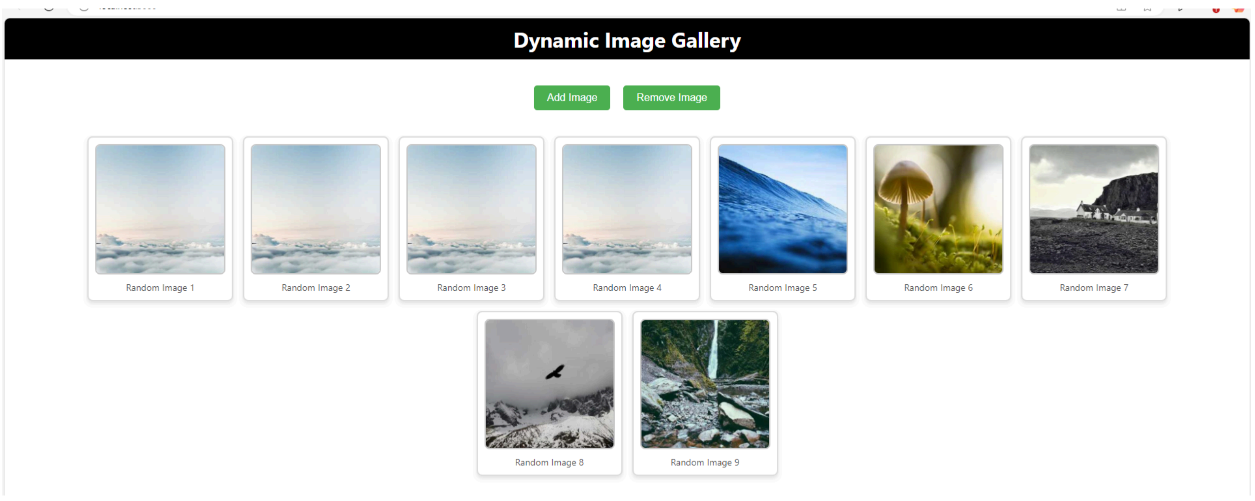
App.css

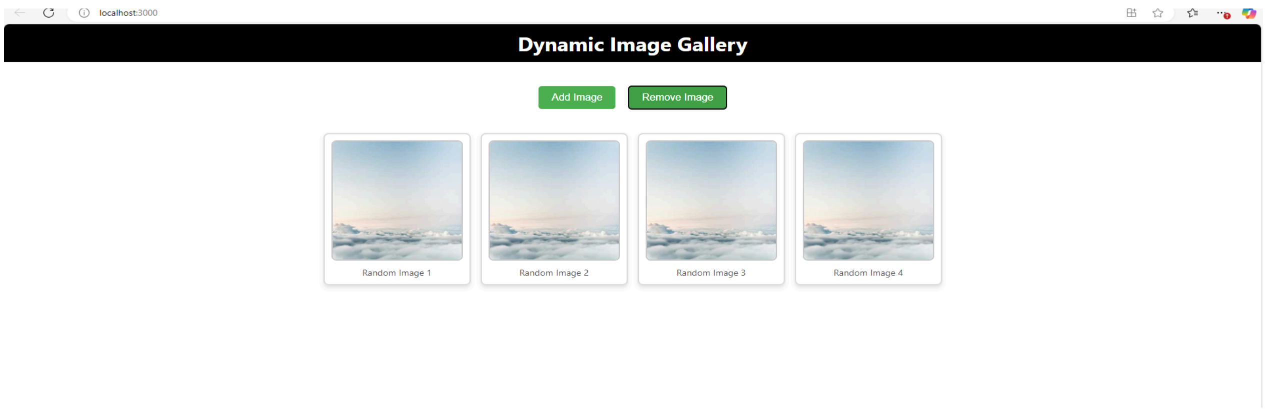
```
*{
  padding: 0;
  margin: 0;
  box-sizing: border-box;
}
h1 {
  background: #000;
  color: #fff;
  padding: 10px;
  text-align: center;
}
.figure-list-container {
  display: flex;
  flex-direction: column;
  align-items: center;
  margin: 20px;
}
.button-box {
  display: block;
  text-align: center;
  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;
}
```

5.6 Results & Analysis





Experiments

6.0 Experiment (Sand Preparation)

Experiment No	Date Planed	Date Conducted	Marks
06			

TITLE: 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.

6.1 Learning Objectives

Build a React form with fields for Name, Email, and Password. Validate inputs, ensuring all fields are filled, the email is correctly formatted, and the password meets complexity requirements. Show error messages and visual cues for invalid inputs. Add a "Show Password" toggle and sanitize inputs. Prevent submission until validation passes and log/display entered data on success

6.2 Aim

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.

6.3 Material / Equipment Required

PC/Laptop (Windows, macOS, or Linux)
Processor: Intel Core i3 or higher (Recommended)
RAM: Minimum **4GB** (Recommended: **8GB or more**)
Storage: At least **10GB free space**
Software : VS Code, Node.js & npm (Runtime & Package Manager)

6.4 Theory / Hypothesis

1. Install **Node.js** from https://nodejs.org/.
2. node –v
3. npm –v
4. npm install web-vitals
5. npx create-react-app my-form-app
6. cd my-form-app
7. npm start
8. Create UserForm.js

6.5 Procedure / Program / Activity

UserForm.js

```
import React, { useState } from 'react';
function UserForm() {
// State variables for input fields and error messages.
const [name, setName] = useState('');
const [email, setEmail] = useState('');
const [password, setPassword] = useState('');
const [errors, setErrors] = useState({});
const [showPassword, setShowPassword] = useState(false);
const [submittedData, setSubmittedData] = useState(null);
// Simple sanitization function: trim whitespace.
const sanitizeInput = (input) => input.trim();
// Validate fields and return an errors object.
const validate = () => {
const newErrors = {};
const cleanName = sanitizeInput(name);
const cleanEmail = sanitizeInput(email);
const cleanPassword = sanitizeInput(password);
if (!cleanName) {
newErrors.name = 'Name is required.';
}
```

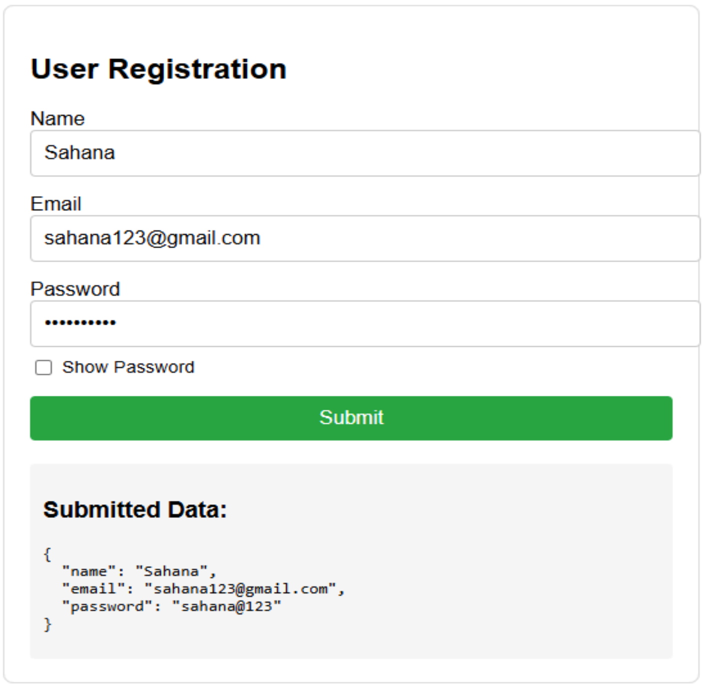
```
if (!cleanEmail) {
newErrors.email = 'Email is required.';
} else if (!/\S+@\S+\.\S+/.test(cleanEmail)) {
newErrors.email = 'Email format is invalid.';
 }
if (!cleanPassword) {
newErrors.password = 'Password is required.';
} else if (cleanPassword.length < 6) {
newErrors.password = 'Password must be at least 6 characters.';
}
 return newErrors;
};
// Handle form submission.
const handleSubmit = (e) => {
e.preventDefault();
const formErrors = validate();
setErrors(formErrors);
if (Object.keys(formErrors).length === 0) {
// Clean inputs
const formData = {
name: sanitizeInput(name),
email: sanitizeInput(email),
password: sanitizeInput(password),
};
// Log the data and update state.
console.log('Submitted Data:', formData);
setSubmittedData(formData);

// Optionally clear the form:
// setName('');
// setEmail('');
// setPassword('');
}
};
  return (
<div style={styles.container}>
<h2>User Registration</h2>
<form onSubmit={handleSubmit} noValidate>
{/* Name Field */}
<div style={styles.formGroup}>
<label htmlFor="name">Name</label>
<input
id="name"
type="text"
value={name}
onChange={(e) => setName(e.target.value)}
style={{
 ...styles.input,
borderColor: errors.name ? 'red' : '#ccc',
}}
/>
{errors.name && <p style={styles.errorText}>{errors.name}</p>}
</div>
{/* Email Field */}
<div style={styles.formGroup}>
<label htmlFor="email">Email</label>
<input
id="email"
type="email"
value={email}
onChange={(e) => setEmail(e.target.value)}
 style={{
...styles.input,
borderColor: errors.email ? 'red' : '#ccc',
}}
/>
{errors.email && <p style={styles.errorText}>{errors.email}</p>}
</div>
{/* Password Field */}
<div style={styles.formGroup}>
<label htmlFor="password">Password</label>
<input
  id="password"
 type={showPassword ? 'text' : 'password'}
 value={password}
 onChange={(e) => setPassword(e.target.value)}
  style={{
...styles.input,
borderColor: errors.password ? 'red' : '#ccc',
}}
/>
{errors.password && <p style={styles.errorText}>{errors.password}</p>}
<div style={styles.showPasswordContainer}>
<input
type="checkbox"
 id="showPassword"
 checked={showPassword}
onChange={(e) => setShowPassword(e.target.checked)}
/>
<label htmlFor="showPassword" style={styles.showPasswordLabel}>
Show Password
  </label>
  </div>
 </div>
```

```
{/* Submit Button */}
<button type="submit" style={styles.submitButton}>
 Submit
</button>
</form>
{/* Optionally display the submitted data */}
{submittedData && (
<div style={styles.result}>
<h3>Submitted Data:</h3>
<pre>{JSON.stringify(submittedData, null, 2)}</pre>
 </div>
)}
</div>
);
}
// Inline styles for the component.
const styles = {
container: {
maxWidth: '500px',
margin: '30px auto',
padding: '20px',
border: '1px solid #ddd',
borderRadius: '8px',
fontFamily: 'Arial, sans-serif',
},
formGroup: {
 marginBottom: '15px',
},
input: {
 width: '100%',
padding: '10px',
fontSize: '16px',
border: '1px solid #ccc',
borderRadius: '4px',
outline: 'none',
},
 errorText: {
color: 'red',
marginTop: '5px',
 fontSize: '0.9em',
},submitButton: {
width: '100%',
padding: '10px',
fontSize: '16px',
backgroundColor: '#28a745',
color: '#fff',
border: 'none',
borderRadius: '4px',
cursor: 'pointer',
},
result: {
marginTop: '20px',
backgroundColor: '#f7f7f7',
padding: '10px',
borderRadius: '4px',
},
showPasswordContainer: {
marginTop: '8px',
display: 'flex',
alignItems: 'center',
},
showPasswordLabel: {
marginLeft: '5px',
fontSize: '0.9em',
},
};
export default UserForm;
```

App.Js

```
import React from 'react';
import UserForm from './UserForm';
function App() {
return (
<div>
<UserForm />
</div>
);
}
export default App;
```

6.6 Results & Analysis



User Registration

Name
Sahana

Email
sahana123@gmail.com

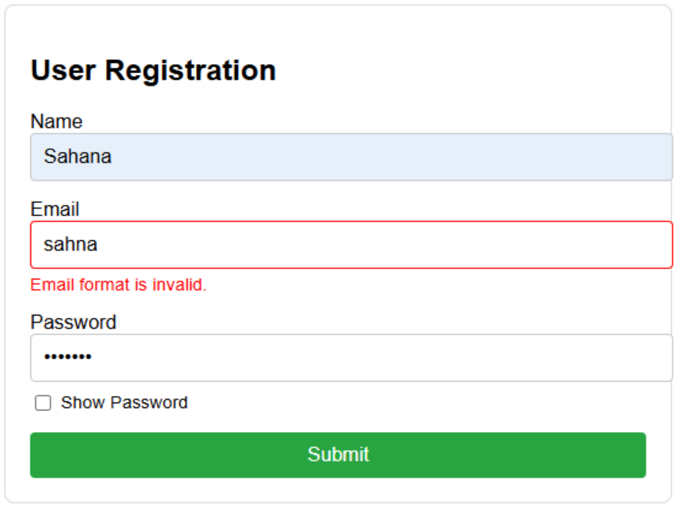
Password
••••••••••

Show Password

Submit

Submitted Data:

```
{
  "name": "Sahana",
  "email": "sahana123@gmail.com",
  "password": "sahana@123"
}
```



User Registration

Name
Sahana

Email
sahna

Email format is invalid.

Password
•••••••

Show Password

Submit

7.0 Experiment (Sand Preparation)

Experiment No	Date Planed	Date Conducted	Marks
07			

TITLE: Develop a React Application featuring a Profile Card 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 Profile Card 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.

7.1 Learning Objectives

Create a React Profile Card component displaying a user's name, profile picture, and bio. Use external CSS for layout and typography, and inline styles for dynamic elements like background color. Ensure a circular profile picture, responsive design, and interactive hover effects. Allow the background color to change via props or state

7.2 Aim

Develop a React Application featuring a Profile Card 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 Profile Card 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.

7.3 Material / Equipment Required

PC/Laptop (Windows, macOS, or Linux)
Processor: Intel Core i3 or higher (Recommended)
RAM: Minimum **4GB** (Recommended: **8GB or more**)
Storage: At least **10GB free space**
Software : VS Code, Node.js & npm (Runtime & Package Manager)

7.4 Theory / Hypothesis

1. Install **Node.js** from https://nodejs.org/

2. node –v
3. npm –v
4. npm install web-vitals
5. npx create-react-app profilecrd-app
6. cd profilecrd-app
7. npm start

7.5 Procedure / Program / Activity

/*ProfileCard.js*/

```
import React, { useState } from "react";
import "./ProfileCard.css";

const ProfileCard = ({ name, bio, profilePicture, initialBgColor }) => {
  const [bgColor, setBgColor] = useState(initialBgColor);

  const handleMouseEnter = () => {
    setBgColor("#f4a261");
  };
  const handleMouseLeave = () => {
    setBgColor(initialBgColor);
  };
  return (
    <div
      className="profile-card"
      style={{ backgroundColor: bgColor }}
      onMouseEnter={handleMouseEnter}
      onMouseLeave={handleMouseLeave}
    >
      <img src={profilePicture} alt={name} className="profile-picture" />
      <h2 className="profile-name">{name}</h2>
      <p className="profile-bio">{bio}</p>
    </div>
  );
};

const App = () => {
  return (
    <div className="app-container">
      <ProfileCard
        name="John Doe"
        bio="Web Developer | Tech Enthusiast | Lifelong Learner"
        profilePicture="https://via.placeholder.com/150"
        initialBgColor="#2a9d8f"
      />
    </div>
  );
};
export default App;
```

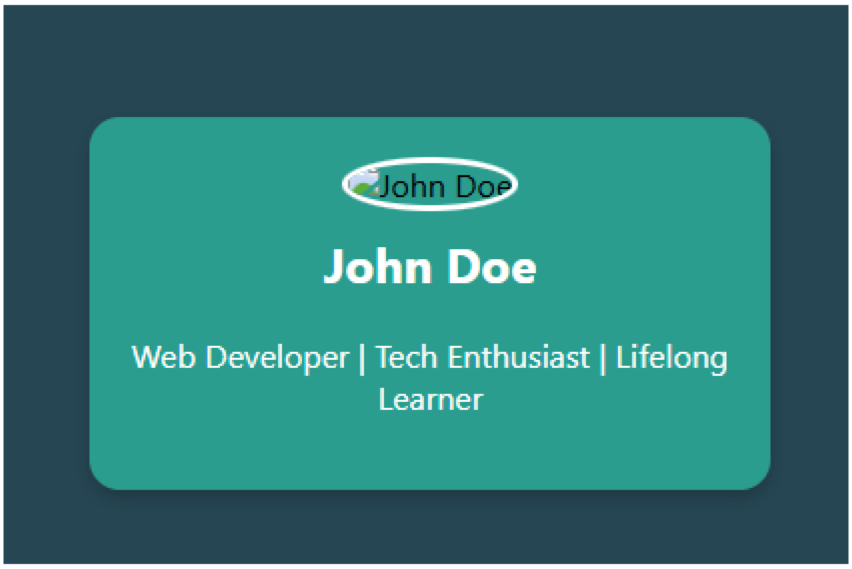
/* ProfileCard.css */

```
.profile-card {
   width: 300px;
   padding: 20px;
   border-radius: 15px;
   text-align: center;
   transition: background-color 0.3s ease-in-out, transform 0.2s;
   box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
  }
   .profile-card:hover {
   transform: scale(1.05);
  }
   .profile-picture {
   width: 100px;
   height: 100px;
   border-radius: 50%;
   border: 3px solid white;
   object-fit: cover;
  }
   .profile-name {
   font-size: 1.5em;
   margin-top: 10px;
   color: white;
  }
   .profile-bio {
   font-size: 1em;
   color: white;
  }
   .app-container {
   display: flex;
   justify-content: center;
   align-items: center;
   height: 100vh;
   background-color: #264653;
  }
```

/*App.JS*/

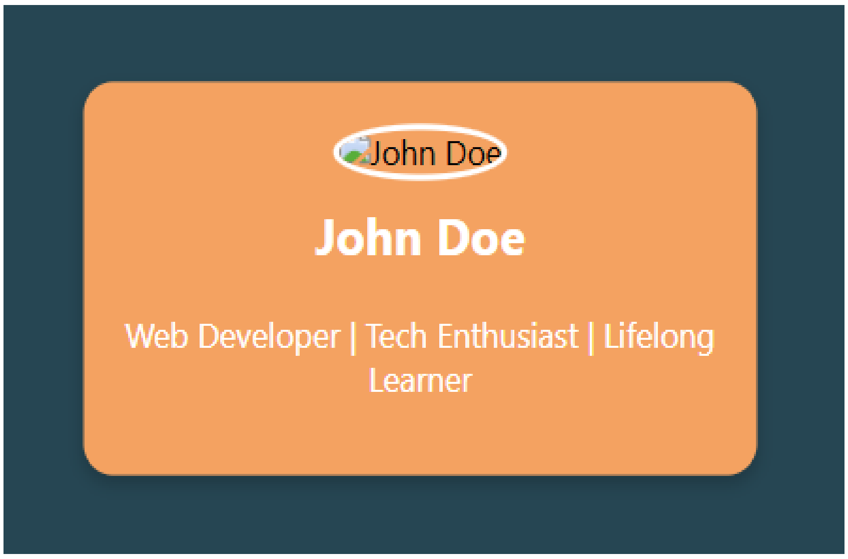
```
import React from "react";
import ProfileCard from "./ProfileCard";
import "./ProfileCard.css";
const App = () => {
  return (
    <div className="app-container">
      <ProfileCard
        name="John Doe"
        bio="Web Developer | Tech Enthusiast | Lifelong Learner"
        profilePicture="https://via.placeholder.com/150"
        initialBgColor="#2a9d8f"
      />
    </div>
  );
};
export default App;
```

7.6 Result & Analysis


John Doe

John Doe

Web Developer | Tech Enthusiast | Lifelong Learner



8.0 Experiment (Sand Preparation)

Experiment No	Date Planed	Date Conducted	Marks
08			

TITLE: 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.

8.1 Learning Objectives

Build a React Reminder App with a form to add tasks (name, due date, optional description). Display tasks dynamically with details and completion status. Add filters to view all, completed, or pending tasks.

8.2 Aim

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.

8.3 Material / Equipment Required

PC/Laptop (Windows, macOS, or Linux)
Processor: Intel Core i3 or higher (Recommended)
RAM: Minimum **4GB** (Recommended: **8GB or more**)
Storage: At least **10GB free space**
Software : VS Code, Node.js & npm (Runtime & Package Manager)

8.4 Theory / Hypothesis

1. node –v
2. npm –v
3. npx create-react-app react-reminder-app
4. cd react-reminder-app
5. 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.

8.5 Procedure / Program / Activity

<u>TaskForm.js</u>

```
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;
```

App.js

```
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>
);
}
```

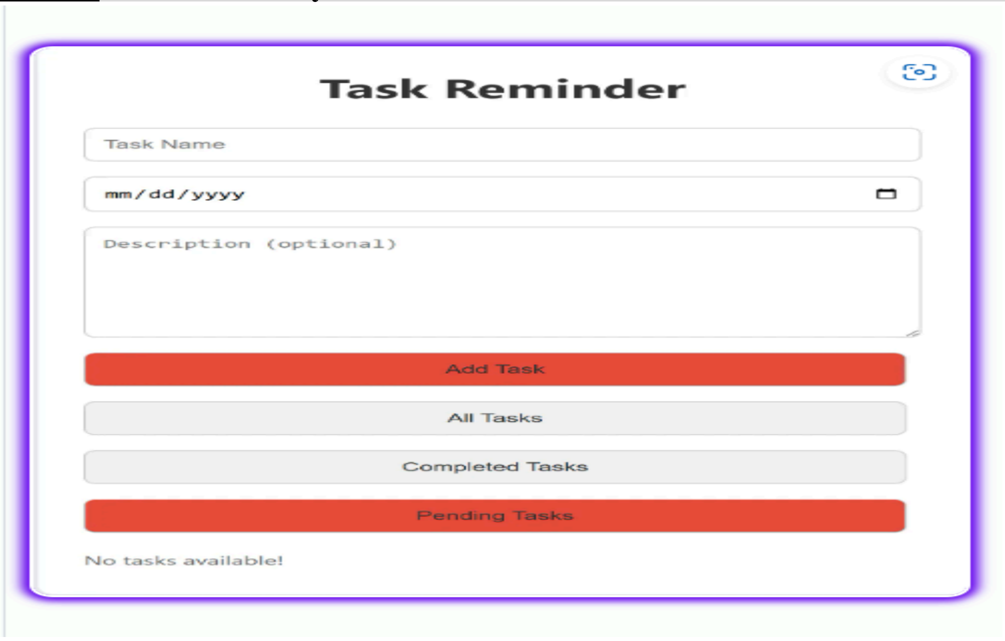
```
export default App;
```

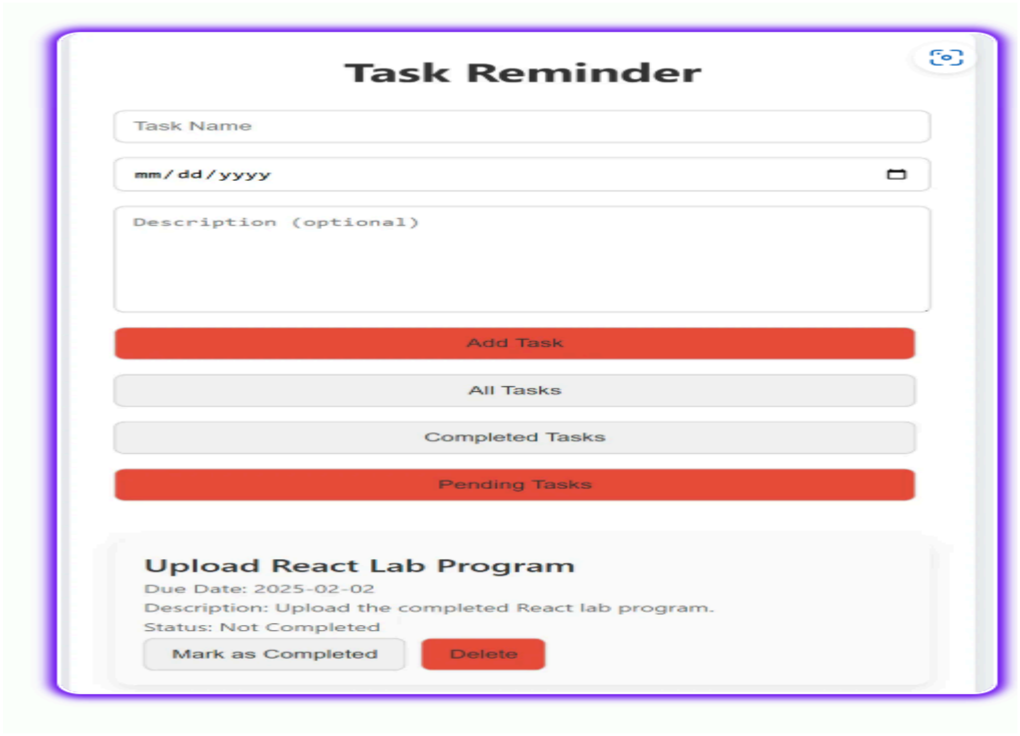
App.css

```
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;
  }
```

8.6 Result & Analysis





Upload React Lab Program
Due Date: 2025-02-02
Description: Upload the completed React lab program.
Status: Not Completed

9.0 Experiment (Sand Preparation)

Experiment No	Date Planed	Date Conducted	Marks
09			

TITLE: 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 Browser Router and Route components for routing. Highlight the active link in the navigation menu to indicate the current page

9.1 Learning Objectives

Create a React app with react-router-dom for routing. Add a navigation bar with links to Home, About, and Contact pages. Use BrowserRouter and Route to render components dynamically. Highlight the active link for better UX.

9.2 Aim

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 Browser Router and Route components for routing. Highlight the active link in the navigation menu to indicate the current page

9.3 Material / Equipment Required

PC/Laptop (Windows, macOS, or Linux)
Processor: Intel Core i3 or higher (Recommended)
RAM: Minimum **4GB** (Recommended: **8GB or more**)
Storage: At least **10GB free space**
Software : VS Code, Node.js & npm (Runtime & Package Manager)

9.4 Theory / Hypothesis

1. node –v
2. npm –v
3. npm install web-vitals
4. npx create-react-app my-routing-app
5. cd my-routing-app
6. npm install react-router-dom
7. Inside the src folder:
8. 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.

9. 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.

10. Open `src/index.js` and ensure the entry point is correct:
11. npm start

9.5 Procedure / Program / Activity

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;
```

App.js

```
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;
```

App.css

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

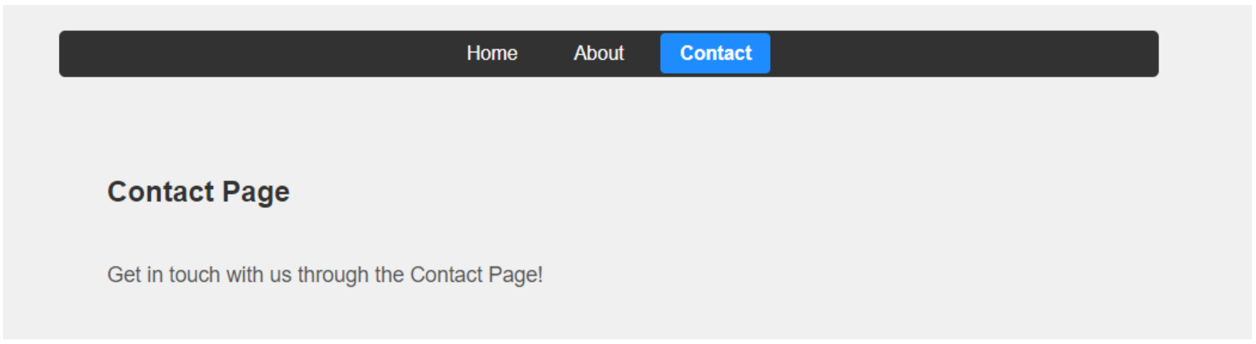
```
}
div {
  margin: 0 auto;
  max-width: 960px;
  padding: 20px;
}
h2 {
  color: #333;
  padding-bottom: 20px;
}
nav {
  background-color: #333;
  padding: 10px;
  border-radius: 5px;
  margin-bottom: 20px;
}

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;
}
```

index.js

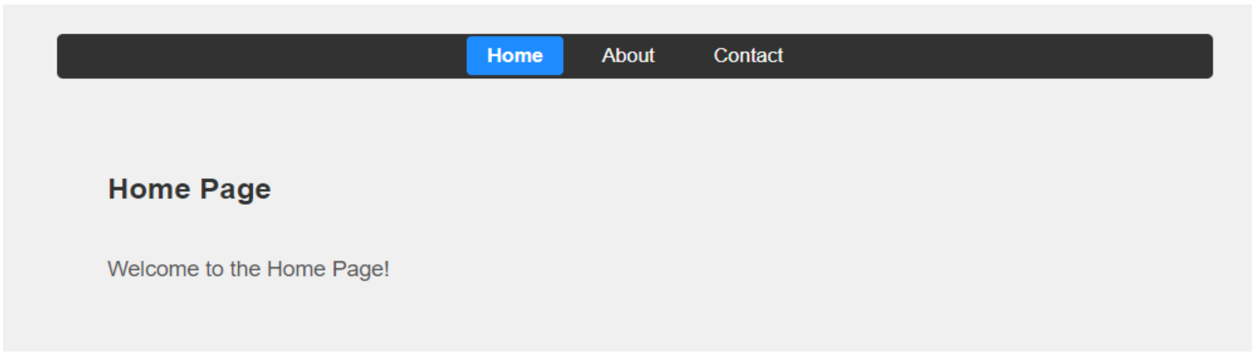
```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
const rootElement = document.getElementById('root');
const root = ReactDOM.createRoot(rootElement);
root.render(<App />);
```

9.6 Result & Analysis



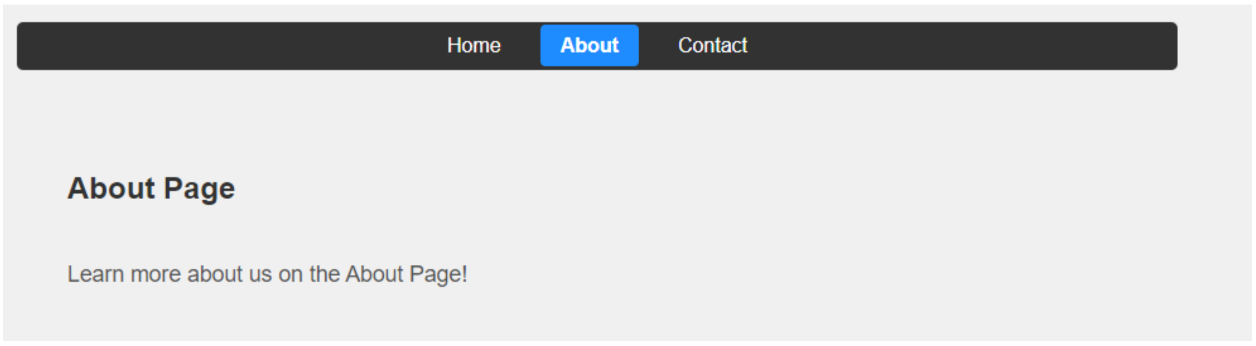
Contact Page

Get in touch with us through the Contact Page!



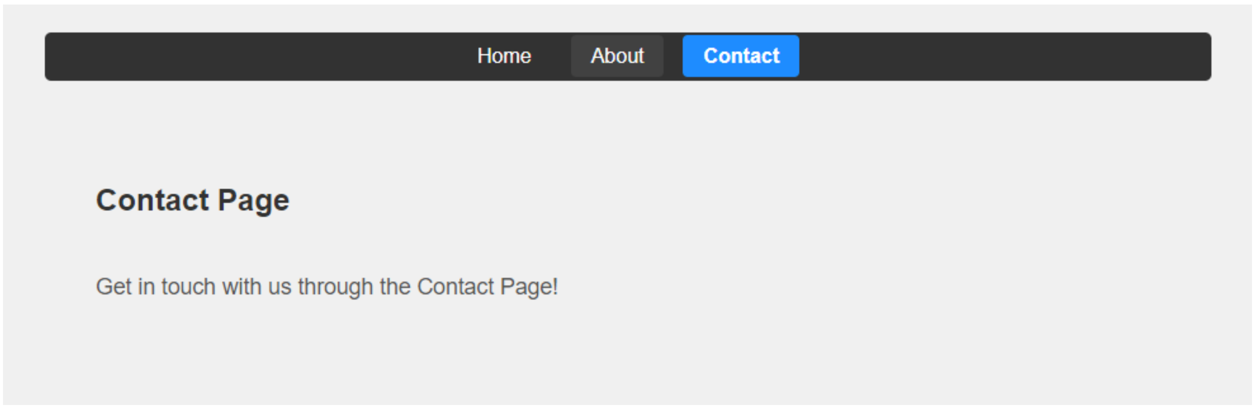
Home Page

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!

10.0 Experiment (Sand Preparation)

Experiment No	Date Planed	Date Conducted	Marks
10			

TITLE: 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 component Did Mount 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 component Did Update 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.

10.1 Learning Objectives

Create a React class-based component that uses lifecycle methods to fetch and update data from an API. Use component Did Mount for initial data fetching and component Did Update for updates based on user actions (filtering, searching, pagination). Display data in a table or list with error handling and interactive controls

10.2 Aim

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 component Did Mount 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 component Did Update 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.

10.3 Material / Equipment Required

PC/Laptop (Windows, macOS, or Linux)
Processor: Intel Core i3 or higher (Recommended)
RAM: Minimum **4GB** (Recommended: **8GB or more**)
Storage: At least **10GB free space**
Software : VS Code, Node.js & npm (Runtime & Package Manager)

10.4 Theory / Hypothesis

1. npx create-react-app data-fetcher
2. cd data-fetcher
3. npm start

10.5 Procedure / Program / Activity

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;
```

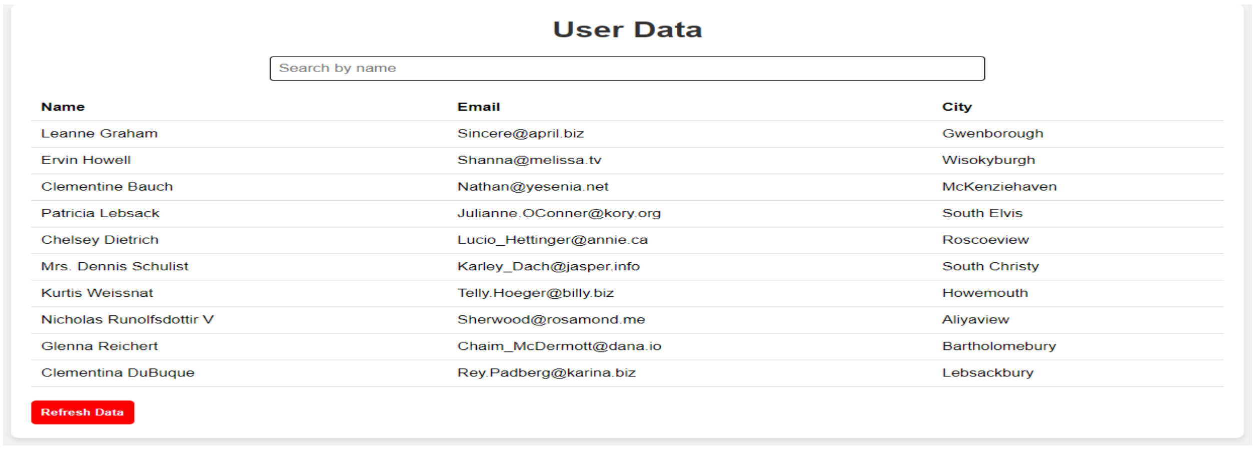
index.js

```
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>
);
```

App.css

```
* {
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;
}
```

10.6 Result & Analysis

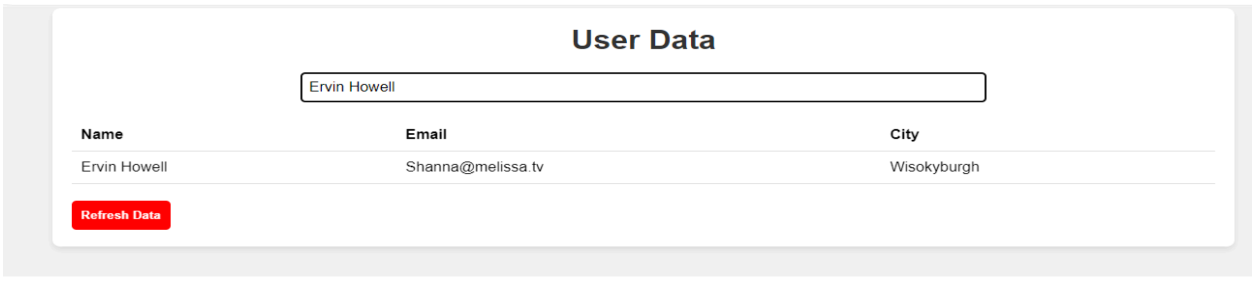


User Data

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



User Data

Ervin Howell

Name	Email	City
Ervin Howell	Shanna@melissa.tv	Wisokyburgh

Refresh Data