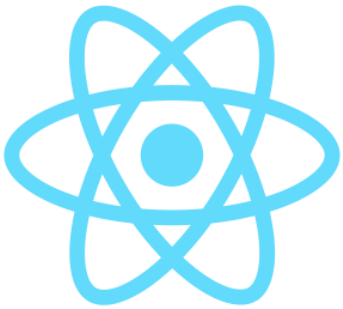
KG COLLEGE OF ARTS AND SCIENCE

co-Kreate your Genius

LABORATORY MANUAL



INTERACTIVE JAVASCRIPT PROGRAMMING B.Sc. CS / B.Sc. IT / B.Sc. CT

2024 BATCH

IV Semester

KG COLLEGE OF ARTS AND SCIENCE

co-Kreate your Genius

KG COLLEGE OF ARTS AND SCIENCE
Autonomous Institution | Affiliated to Bharathiar University
Accredited with A++ Grade by NAAC
ISO 9001:2015 Certified Institution
KGiSL Campus, Saravanampatti, Coimbatore – 641 035

SYLLABUS

S. No.	List of Programs	No. of Hours
Level 1: React Basics (Getting Started)		
1.	Create a counter with increment, decrement, and reset buttons.	2
2.	Build a to-do list that allows users to add, delete, and mark tasks as complete.	4
Level 2: Forms & User Interaction		
3.	Create a form with name, email, and password fields. Display entered data below the form after submission.	4
4.	Create a color picker with multiple buttons (Red, Blue, Green). When clicked, change the background color of a box.	2
Level 3: Conditional Rendering & Logic		
5.	Display a dynamic greeting message based on the current system time. Use conditional rendering in React to update the message according to the hour of the day.	2
6.	Develop a React application that accepts two numeric inputs and dynamically performs addition, subtraction, multiplication, and division. Display the computed result in real time as the user interacts with the inputs.	4
Level 4: Data & Component Navigation		
7.	Fetch and display a list of users from JSON Placeholder API. Show name and email.	2
8.	Create a component with 3 tabs (Home, About, Contact). Clicking each tab shows different content.	4
Level 5: Theming & Routing		

9.	Add a Dark Mode / Light Mode toggle. The theme should change the background and text colors.	2
10.	Create a mini blog with 2 pages: Home page shows a list of posts, Post page shows full content when clicked.	4
Capstone Project		
Develop a React-Based Task Management System with User Authentication, Database Integration, API Connectivity, Theme Toggle, and Dynamic Routing.		
TOTAL HOURS		**30**

EXPERIMENT - 1

Create a Counter with increment ,decrement and reset buttons.

AIM:

To develop a simple **Counter Application** in **React (Vite)** that displays a number and provides **Increment**, **Decrement**, and **Reset** buttons to modify the value dynamically.

ALGORITHM:

Step 1: Start

Step 2: Install and create a React project using **Vite**.

Step 3: Open the App.jsx file.

Step 4: Create a state variable named **count** using the useState() hook.

Step 5: Create three functions:

- **increment** → increases count by 1
- **decrement** → decreases count by 1
- **reset** → sets count back to 0

Step 6: Design the UI with:

- A heading
- A box showing the count
- Three buttons: Increment, Decrement, Reset

Step 7: Assign **onClick** event handlers to each button.

Step 8: Display updated value automatically using React state.

Step 10: End the program.

SOURCE CODE:

File: Main.jsx

```
import React from "react";
import { createRoot } from "react-dom/client";
import App from "./App";
import "./index.css";

createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

File: Src/App.jsx

```
import { useState } from "react";

function App() {
  const [count, setCount] = useState(0);
  const increment = () => {
    setCount(count + 1);
  };
  const decrement = () => {
    setCount(count - 1);
  };
 const reset = () => {
    setCount(0);
  };
  return (
    <div style={{ textAlign: "center", marginTop: "50px" }}>
      <h1>React Counter App</h1>
      <h2>{count}</h2>
      <button onClick={increment} style={{ margin: "10px" }}>
        Increment
      </button>
```

```
      <button onClick={decrement} style={{ margin: "10px" }}>
        Decrement
      </button>
       <button onClick={reset} style={{ margin: "10px" }}>
        Reset
      </button>
    </div>
  );
}
export default App;
```

EXPLANATION:

import { useState } from "react";

Imports the **useState Hook** for creating state variables.

function App() {

Defines the main App component.

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

Creates a state variable count with initial value **0**.
setCount is used to update it.

Increment Function : Adds 1 to the current count.

const increment = () => {

```
  setCount(count + 1);
};
```

Decrement Function : Subtracts 1 from the current count.

```
const decrement = () => {
  setCount(count - 1);
};
```

Reset Function : Sets count back to 0.

```
const reset = () => {
  setCount(0);
};
```

UI Section: Displays the heading and the counter output dynamically.

Buttons

```
<button onClick={increment}>Increment</button>
<button onClick={decrement}>Decrement</button>
<button onClick={reset}>Reset</button>
```

Each button is linked to its respective function.

export default App;

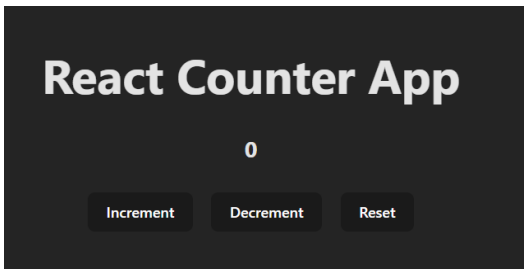
Exports the App component for rendering in the browser.

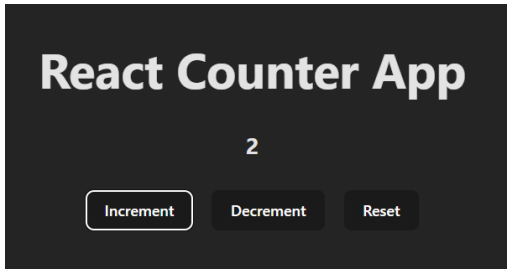
OUTPUT :

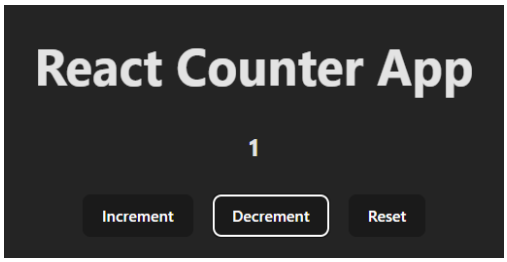
Live_Demo_Video :

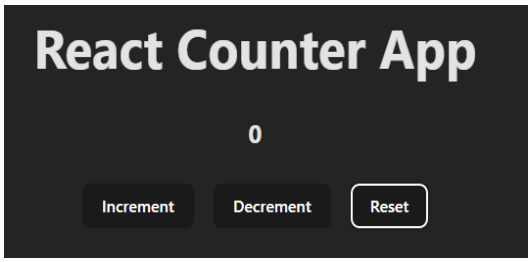
Counter

Screenshot :









RESULT :

The ReactJS program successfully shows a working counter using useState and event handling, allowing users to increment, decrement, and reset the count dynamically.

EXPERIMENT - 2

Build a to-do list that allows users to add, delete, and mark tasks as complete.

AIM:

Create a React (Vite) To-Do List app demonstrating component state management, form handling, and event-driven updates. Users can add new tasks, delete tasks, and toggle a task as complete/incomplete.

ALGORITHM:

Step 1: Start the app.

Step 2: Create a tasks state array to hold task objects (id, text, complete**d).**

Step 3: Create a controlled input to type a new task.

Step 4: On form submit:

- Prevent default.
- Create a new task object and add it to tasks.
- Clear input.

Step 5: Render tasks as a list. For each task:

- Show text (strike-through if completed).
- Provide a **Delete** button that removes the task by id.
- Provide a **checkbox** or click action to toggle completed.

Step 6: Update state immutably for add/delete/toggle operations.

Step 7: Stop.

SOURCE CODE:

File: Main.jsx

```
import React from "react";
import { createRoot } from "react-dom/client";
import App from "./App";
import "./index.css";

createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

File: Src/App.jsx

```
import { useState } from "react";
export default function App() {
  const [tasks, setTasks] = useState([]);       // list of task objects
  const [text, setText] = useState("");         // controlled input
  const addTask = (e) => {
    e.preventDefault();
    const trimmed = text.trim();
    if (!trimmed) return;                       // ignore empty
    const newTask = {
      id: Date.now(),
      text: trimmed,
      completed: false,
    };
    setTasks([newTask, ...tasks]);              // add to front
    setText("");
  };
  const deleteTask = (id) => {
    setTasks(tasks.filter((t) => t.id !== id));
  };
  const toggleTask = (id) => {
    setTasks(
      tasks.map((t) =>
        t.id === id ? { ...t, completed: !t.completed } : t
```

```
    )
  );
};
return (
  <div className="app">
    <h1>To-Do List</h1>
    <form onSubmit={addTask} className="task-form">
      <input
        value={text}
        onChange={(e) => setText(e.target.value)}
        placeholder="Enter new task"
        aria-label="Task"
      />
      <button type="submit">Add</button>
    </form>
    <ul className="task-list">
      {tasks.length === 0 && <li className="empty">No tasks yet</li>}
      {tasks.map((task) => (
        <li key={task.id} className="task-item">
          <input
            type="checkbox"
            checked={task.completed}
            onChange={() => toggleTask(task.id)}
```

```
          />
          <span
            onClick={() => toggleTask(task.id)}
            className={task.completed ? "done" : ""}
          >
            {task.text}
          </span>
          <button className="delete" onClick={() => deleteTask(task.id)}>
            Delete
          </button>
        </li>
      ))}
    </ul>
  </div>
 );
}
```

File: src/index.css

```
/* --- Global Styling --- */
* {
 margin: 0;
 padding: 0;
 box-sizing: border-box;
 font-family: "Poppins", sans-serif;
```

```
}

body {
  background: linear-gradient(135deg, #d9e4ff, #f7d9ff);
  min-height: 100vh;
  display: flex;
  justify-content: center;
  align-items: center;
}

/* --- Container --- */
.app {
  background: #ffffff;
  width: 450px;
  padding: 25px 30px;
  border-radius: 18px;
  box-shadow: 0px 8px 25px rgba(0, 0, 0, 0.15);
  animation: fadeIn 0.5s ease-in-out;
}

@keyframes fadeIn {
  from { opacity: 0; transform: translateY(10px); }
  to { opacity: 1; transform: translateY(0); }
```

```
}

h1 {
  text-align: center;
  margin-bottom: 20px;
  font-size: 28px;
  color: #333;
}

/* --- Form Input --- */
.task-form {
  display: flex;
  gap: 10px;
  margin-bottom: 20px;
}

.task-form input {
  flex: 1;
  padding: 12px;
  border-radius: 8px;
  border: 2px solid #ddd;
  font-size: 16px;
  transition: 0.3s;
```

```
}

.task-form input:focus {
  border-color: #7d5fff;
  outline: none;
}

.task-form button {
  padding: 12px 20px;
  background: #7d5fff;
  color: white;
  border: none;
  border-radius: 8px;
  cursor: pointer;
  transition: 0.3s;
  font-size: 16px;
}

.task-form button:hover {
  background: #5d3fd3;
}

/* --- Task List --- */
```

```
.task-list {
  list-style: none;
}

.empty {
  text-align: center;
  padding: 15px;
  color: #777;
  font-style: italic;
}

.task-item {
  background: #f4f4ff;
  padding: 12px 15px;
  border-radius: 10px;
  margin-bottom: 12px;
  display: flex;
  align-items: center;
  gap: 12px;
  justify-content: space-between;
  transition: 0.3s;
}
```

```
.task-item:hover {
  background: #ebe6ff;
}

/* --- Task Text --- */
.task-item span {
  flex: 1;
  cursor: pointer;
  font-size: 16px;
}

.task-item span.done {
  text-decoration: line-through;
  color: #888;
}

/* --- Delete Button --- */
.delete {
  background: #ff7675;
  border: none;
  padding: 8px 12px;
  border-radius: 8px;
  cursor: pointer;
```

```
  color: white;
  font-size: 14px;
  transition: 0.2s;
}

.delete:hover {
  background: #e65555;
}
```

EXPLANATION:

Step 1: Import necessary modules

In src/main.jsx, React, ReactDOM, App component, and CSS are imported.

Step 2: Mount the App component

<App /> is rendered inside the HTML element with id root.

Step 3: Import useState

import { useState } from "react";
This brings the useState hook to manage component state.

Step 4: Create State Variables

tasks → stores an array of task objects { id, text, completed }.
text → stores the input value (controlled input).

Step 5: Add new task (addTask function)

- Step 5.1: e.preventDefault() prevents page refresh.

- Step 5.2: trim() removes extra spaces and avoids empty tasks.
- Step 5.3: Create newTask with a unique id using Date.now().
- Step 5.4: setTasks([newTask, ...tasks]) inserts the task at the top.
- Step 5.5: Clear input field with setText("").

Step 6: Delete a Task (deleteTask)

- Step 6.1: Filter out the task using id.
- Step 6.2: Update the tasks array using setTasks().

Step 7: Toggle Task Completion (toggleTask)

- Step 7.1: Loop through tasks using .map().
- Step 7.2: Find the task with the matching id.
- Step 7.3: Flip the completed value (true → false, or false → true).
- Step 7.4: Keep updates immutable.

Step 8: Render Input Form

- Step 8.1: Input is controlled using value={text}.
- Step 8.2: onChange updates text.
- Step 8.3: Submit button calls addTask.

Step 9: Render Task List

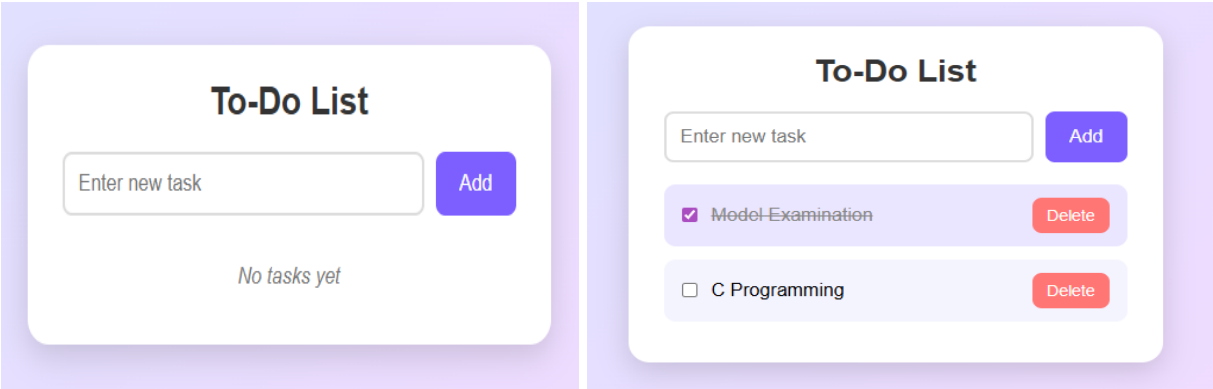
- Step 9.1: tasks.map() displays each task item.
- Step 9.2: Checkbox toggles task completion.

- Step 9.3: Clicking text also toggles completion.
- Step 9.4: Delete button removes specific task.

Step 10: Display Message for Empty List

If there are no tasks, show: "No tasks yet".

OUTPUT :



RESULT :

The ReactJS To-Do List app works as intended: users can add, delete, and mark tasks complete. State updates render immediately, demonstrating controlled inputs, immutable updates, and event handling.

EXPERIMENT - 3

Create a Form with Name, Email, and Password Fields & Display Entered Data

AIM:

To create a React application using Vite that contains a form with **Name**, **Email**, and **Password** fields, and displays the submitted data below the form using state management and event handling.

ALGORITHM:

Step 1: Start the React-Vite project and open App.jsx.

Step 2: Import useState to manage form field values and submitted data.

Step 3: Create state variables:

- name → store entered name
- email → store user email
- password → store user password
- submittedData → store submitted information

Step 4: Create a function handleSubmit that:

- Prevents page refresh using e.preventDefault()
- Saves the form data to submittedData
- Clears the input fields

Step 5: Create the form UI with three input fields:

- Name
- Email
- Password
  (All controlled using value and onChange)

Step 6: Add a submit button that triggers handleSubmit.

Step 7: Below the form, display the submitted name, email, and password only after form submission.

Step 8: Stop.

SOURCE CODE:

File : Src/App.jsx

```
import { useState } from "react";

function App() {
  const [name, setName] = useState("");
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [submittedData, setSubmittedData] = useState(null);

  const handleSubmit = (e) => {
    e.preventDefault();
    const data = { name, email, password };
    setSubmittedData(data);

    setName("");
    setEmail("");
    setPassword("");
  };

  return (
```

```
<div className="card">
  <h1>Registration Form</h1>

  <form onSubmit={handleSubmit}>
    <label>Name</label>
    <input
      type="text"
      className="input-box"
      value={name}
      onChange={(e) => setName(e.target.value)}
      required
    />

    <label>Email</label>
    <input
      type="email"
      className="input-box"
      value={email}
      onChange={(e) => setEmail(e.target.value)}
      required
    />

    <label>Password</label>
```

```
      <input
        type="password"
        className="input-box"
        value={password}
        onChange={(e) => setPassword(e.target.value)}
        required
      />

      <button type="submit" className="btn">
        Submit
      </button>
    </form>

    {submittedData && (
      <div className="output-card">
        <h2>Submitted Data</h2>
        <p><strong>Name:</strong> {submittedData.name}</p>
        <p><strong>Email:</strong> {submittedData.email}</p>
        <p><strong>Password:</strong> {submittedData.password}</p>
      </div>
    )}
  </div>
);
```

```
}

export default App;
```

File : src/index.css

```
:root {
  --bg: #f0f2f5;
  --text: #222;
  --primary: #4A90E2;
  --card-bg: #ffffff;
  --radius: 14px;
  --shadow: 0 6px 18px rgba(0, 0, 0, 0.1);
  --input-border: #d0d0d0;
  --input-focus: #4A90E2;
  --success: #4CAF50;
}

* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
  font-family: system-ui, sans-serif;
}
```

```
body {
  background: var(--bg);
  color: var(--text);
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
  padding: 20px;
}

h1 {
  font-size: 2rem;
  text-align: center;
  margin-bottom: 20px;
}

button {
  cursor: pointer;
  transition: 0.3s;
}

button:hover {
  opacity: 0.9;
```

```
}

/* -------------------------- */
/* ADDED STYLES FOR COMPONENTS */
/* -------------------------- */

/* Card */
.card {
  max-width: 420px;
  margin: 40px auto;
  padding: 28px;
  border-radius: 14px;
  background: var(--card-bg);
  box-shadow: var(--shadow);
}
/* Input */
.input-box {
  width: 100%;
  padding: 12px;
  margin: 8px 0 18px 0;
  border-radius: 10px;
  border: 1px solid var(--input-border);
  font-size: 16px;
```

```
  transition: 0.3s;
}
.input-box:focus {
  border-color: var(--input-focus);
  outline: none;
}
/* Button */
.btn {
  width: 100%;
  padding: 14px;
  margin-top: 10px;
  border: none;
  border-radius: 10px;
  background: var(--primary);
  color: white;
  font-size: 17px;
  font-weight: 600;
  letter-spacing: 0.5px;
  cursor: pointer;
  transition: 0.3s;
}

/* Output Card */
```

```
.output-card {
  margin-top: 25px;
  padding: 20px;
  border-radius: 12px;
  background: #e9f7ef;
  border-left: 6px solid var(--success);
  box-shadow: 0 4px 12px rgba(0,0,0,0.08);
}
```

EXPLANATION:

Step 1: import { useState } from "react";

Imports React's useState hook to store and update form values.

Step 2: Create state variables

- name, email, password → input fields
- submittedData → holds final submitted values

Step 3: handleSubmit function

- e.preventDefault() stops page reload
- Creates a data object with the entered values
- Saves it in submittedData
- Clears form fields using setName(""), etc.

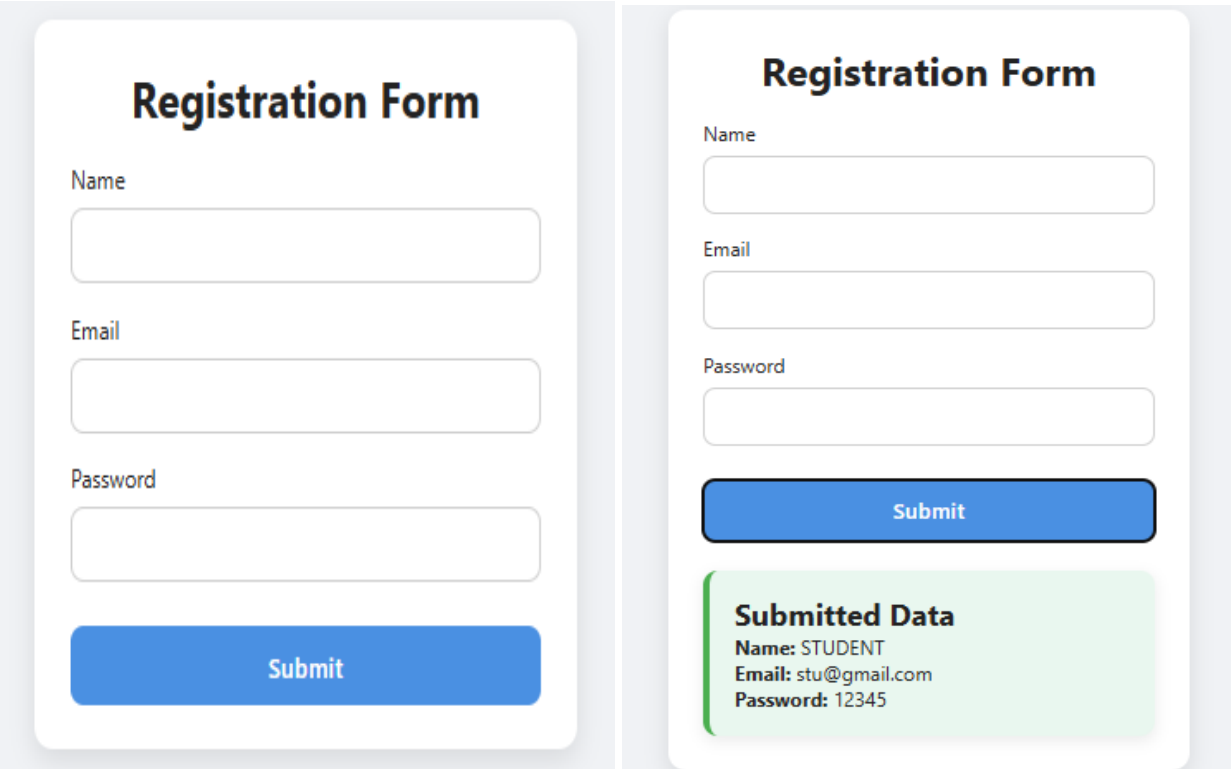
Step 4: Create the form UI

- Each input uses value and onChange (controlled inputs).
- required ensures fields cannot be empty.

Step 5: Display submitted data

- Condition submittedData && ensures display only after submit
- Shows Name, Email, Password neatly below the form.

OUTPUT :



RESULT :

The ReactJS program successfully collects Name, Email, and Password through a form, and displays the submitted data below the form using state management and event handling.

EXPERIMENT - 4

Create a color picker with multiple buttons (Red, Blue, Green). When clicked, change the background color of a box.

AIM:

To create a React application using Vite that contains multiple color buttons (Red, Blue, Green). When the user clicks a button, the background color of a display box changes using state management and event handling.

ALGORITHM:

Step 1: Start the React-Vite project and open App.jsx.

Step 2: Import useState to store the selected color.

Step 3: Create a state variable:

- color → stores the current background color.

Step 4: Create functions for each button:

- A function that updates the color state to Red
- A function that updates the color state to Blue
- A function that updates the color state to Green

Step 5: Create UI buttons labeled Red, Blue, and Green.
Each button uses onClick to change the color state.

Step 6: Create a box (a div) whose background color is controlled by the state variable.

Step 7: When clicking a button, update the color state → box color should change immediately.

Step 8: Stop.

SOURCE CODE:

File: src/App.jsx

```
import { useState } from "react";

function App() {
  const [color, setColor] = useState("white");

  return (
    <div className="container">
      <h1 className="title">Color Picker</h1>

      <div className="btn-group">
        <button className="btn red" onClick={() => setColor("red")}>
          Red
        </button>
        <button className="btn blue" onClick={() => setColor("blue")}>
          Blue
        </button>
        <button className="btn green" onClick={() => setColor("green")}>
          Green
        </button>
      </div>
      <div className="color-box" style={{ backgroundColor: color }}></div>
      {/* CSS inside the same file */}
```

```
<style>{`
  :root {
    --red: #e74c3c;
    --blue: #3498db;
    --green: #2ecc71;
    --shadow: 0 5px 15px rgba(0, 0, 0, 0.15);
    --box-border: #333;
  }
  body {
    font-family: system-ui, sans-serif;
    background: #f5f5f5;
    padding: 40px;
    text-align: center;
  }

  .container {
    max-width: 500px;
    margin: auto;
  }

  .title {
    font-size: 2rem;
    margin-bottom: 25px;
```

```
}

.btn-group {
  display: flex;
  justify-content: center;
  gap: 15px;
  margin-bottom: 30px;
}
.btn {
  padding: 12px 22px;
  border: 1px solid black;
  font-size: 16px;
  font-weight: 600;
  border-radius: 8px;
  color: white;
  cursor: pointer;
  transition: 0.3s;
  box-shadow: var(--shadow);
}
.btn:hover {
  transform: scale(1.05);
  border:1px solid ;
  border-radius: 10px 10px ;
```

```
        }
         .red { background: var(--red); }
        .blue { background: var(--blue); }
        .green { background: var(--green); }
        .color-box {
          width: 280px;
          height: 160px;
          margin: 0 auto;
          border: 2px solid var(--box-border);
          border-radius: 10px;
          transition: background-color 0.4s ease;
          box-shadow: var(--shadow);
        }
      `}</style>
    </div>
  );
}
export default App;
```

EXPLANATION:

Step 1: import { useState } from "react"

Imports the useState hook to store and update the selected color.

Step 2: Create state variable

color is used to hold the current background color of the box.

Step 3: Button click handlers

Each button uses onClick to update the color state:

- setColor("red")
- setColor("blue")
- setColor("green")

Step 4: UI Buttons

Three buttons are created and displayed at the top of the app.

Step 5: Color Box

A rectangular box (div) is created.
Its background color changes according to the color state.

Step 6: Automatic Update

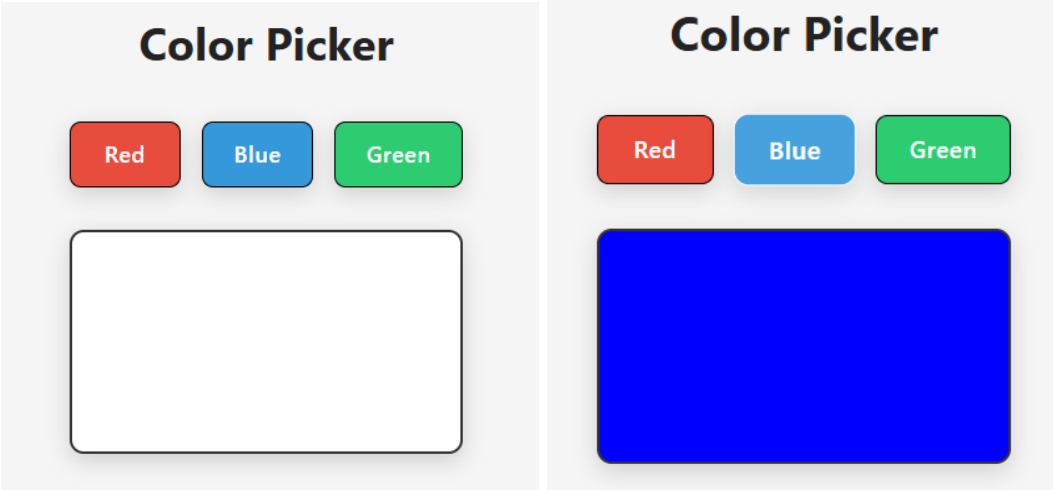
React re-renders the display box immediately when the color state changes.

OUTPUT :

Live_Demo_Video :

Colorpicker

Screenshot :



RESULT :

The ReactJS program successfully changes the background color of a display box when users click the Red, Blue, or Green buttons. The update happens dynamically using React state management and event handling.

EXPERIMENT - 5

Display a dynamic greeting message based on the current system time. Use conditional rendering in React to update the message according to the hour of the day.

AIM:

To create a React application using Vite that displays a greeting message (Good Morning, Good Afternoon, Good Evening, Good Night) based on the current system time using conditional rendering.

ALGORITHM:

Step 1: Start the React-Vite project and open App.jsx.

Step 2: Import useState and useEffect to handle time-based updates.

Step 3: Create a state variable:

- greeting → stores the current greeting message.

Step 4: Use useEffect() to:

- Get the current hour using new Date().getHours()
- Check conditions:
    - If hour < 12 → Good Morning
    - If hour between 12–17 → Good Afternoon
    - If hour between 17–20 → Good Evening
    - Else → Good Night
- Update greeting state.

Step 5: Display the greeting message with inline styles for:

- Font size
- Font weight
- Text color
- Background box styling

Step 6: Stop.

SOURCE CODE:

File: src/App.jsx

```
import { useState, useEffect } from "react";
```

```
function App() {
  const [greeting, setGreeting] = useState("");

  useEffect(() => {
    const hour = new Date().getHours().toString().padStart(2,'0');
    const min = new Date().getMinutes().toString().padStart(2,'0');
    const secs = new Date().getSeconds().toString().padStart(2,'0');

    if (hour < 12) {
      setGreeting(`Good Morning<br/>${hour}:${min}:${secs}`);
    } else if (hour >= 12 && hour < 17) {
      setGreeting(`Good Afternoon<br/>${hour}:${min}:${secs}`);
    } else if (hour >= 17 && hour < 20) {
      setGreeting(`Good Evening<br/>${hour}:${min}:${secs}`);
    } else {
      setGreeting(`Good Night<br/>${hour}:${min}:${secs}`);
    }
  }, []);

  return (
    <div
      style={{
        textAlign: "center",
```

```
    marginTop: "80px",
    fontFamily: "Arial, sans-serif",
  }}
 >
<div
    style={{
      display: "inline-block",
      padding: "20px 40px",
      borderRadius: "12px",
      backgroundColor: "#66b5eeff",
      boxShadow: "0 0 10px rgba(0,0,0,0.2)",
    }}
  >
    <h1
      style={{
        margin: 0,
        fontSize: "30px",
        color: "#0a0202ff",
      }}
      dangerouslySetInnerHTML={{ __html: greeting }}
    ></h1>

    <p style={{ marginTop: "10px", color: "#0e0a0aff" }}>
```

```
        This greeting updates based on your system time.
      </p>
    </div>
  </div>
 );
}
export default App;
```

EXPLANATION:

Step 1: import { useState, useEffect } from "react"

useState stores the greeting message.
useEffect runs once when the component loads and checks the time.

Step 2: Create greeting state

greeting holds the final greeting message to display.

Step 3: useEffect()

- Gets the current hour using new Date().getHours().
- Uses conditional statements to set the correct greeting:
    - Morning
    - Afternoon
    - Evening
    - Night

Step 4: Display Greeting

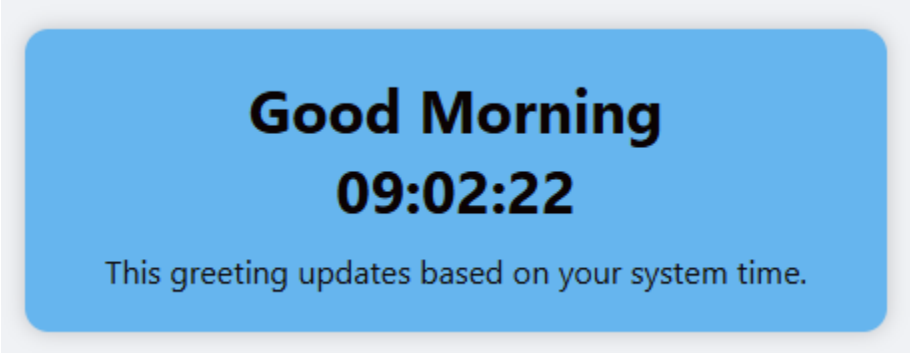
The greeting is displayed inside a styled box.
Styling includes:

- Rounded corners
- Shadow
- Custom font
- Large heading

Step 5: Conditional Rendering

React updates the UI automatically when greeting is assigned inside useEffect.

OUTPUT :



RESULT :

The React-Vite application successfully displays a dynamic greeting message such as **“Good Morning!”**, **“Good Afternoon!”**, **“Good Evening!”**, or **“Good Night!”** based on the user’s system time. The message appears inside a styled UI box with clean formatting.

EXPERIMENT - 6

Real-Time Arithmetic Operations Application

AIM:

To develop a React application using Vite that accepts two numeric inputs and performs addition, subtraction, multiplication, and division in real time. The computed results must update dynamically as the user types.

ALGORITHM:

Step 1: Start the React-Vite project and open App.jsx.

Step 2: Import useState to manage two input values.

Step 3: Create two state variables:

- num1 → stores first number
- num2 → stores second number

Step 4: Store both values as numbers using Number() or parseFloat().

Step 5: Compute all four operations:

- addition → num1 + num2
- subtraction → num1 - num2
- multiplication → num1 * num2
- division → num1 / num2 (check for divide-by-zero case)

Step 6: Display results below the input fields and update them dynamically whenever the user types.

Step 7: Stop.

SOURCE CODE:

File: src/App.jsx

```
import { useState } from "react";


function App() {
  const [num1, setNum1] = useState("");
  const [num2, setNum2] = useState("");
  // Convert input to numbers safely
  const a = Number(num1);
  const b = Number(num2);
  return (
    <div
      style={{
        textAlign: "center",
        marginTop: "50px",
        fontFamily: "Arial, sans-serif",


      }}
    >
      <h1>Real-Time Calculator</h1>
      {/* Input Fields */}
      <input
```

```
  type="number"
  placeholder="Enter first number"
  value={num1}
  onChange={(e) => setNum1(e.target.value)}
  style={{
    padding: "10px",
    marginRight: "10px",
    borderRadius: "6px",
    border: "1px solid #444",
    boxShadow:"5px 5px solid"
  }}
/>
<input
  type="number"
  placeholder="Enter second number"
  value={num2}
  onChange={(e) => setNum2(e.target.value)}
  style={{
    padding: "10px",
    borderRadius: "6px",
    border: "1px solid #444",
  }}
/>
```

```
    {/* Result Box */}
    <div
      style={{
        width: "340px",
        margin: "30px auto",
        padding: "20px",
        borderRadius: "12px",
        backgroundColor: "#c6a0a0ff",
        boxShadow: "0 0 10px rgba(0,0,0,0.2)",
        textAlign: "left",
      }}
    >
      <h2>Results</h2>
      <p><strong>Addition:</strong> {a + b}</p>
      <p><strong>Subtraction:</strong> {a - b}</p>
      <p><strong>Multiplication:</strong> {a * b}</p>
      <p>
        <strong>Division:</strong>{" "}
        {b === 0 ? "Cannot divide by zero" : a / b}
      </p>
    </div>
  </div>
);
```

}

export default App;

EXPLANATION:

Step 1: Import useState

Used to manage the values typed in both input fields.

Step 2: Create state variables

num1 and num2 store the current values entered by the user.

Step 3: Convert values to numbers

Number(num1) and Number(num2) ensure calculations work correctly.

Step 4: Real-time computation

No submit button is required.
React automatically re-renders results when inputs change.

Step 5: Display results

All four operations are displayed inside a styled result box.

Step 6: Division check

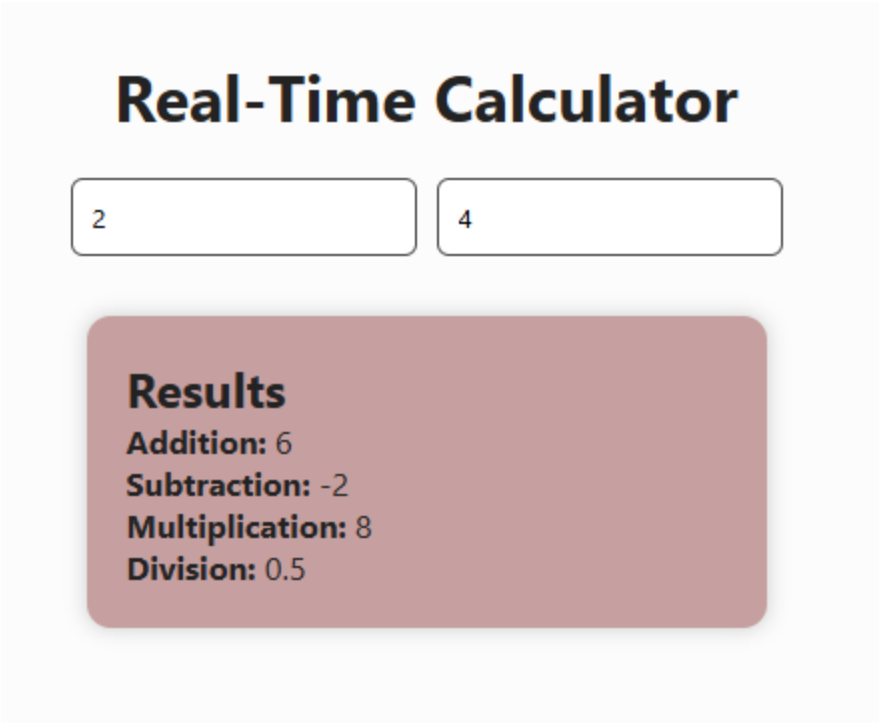
If the second value is zero, it safely displays:
"Cannot divide by zero"

OUTPUT :

Live_Demo_Video :

Calculatore

Screenshot :



RESULT :

The React application successfully accepts two numeric values and dynamically performs addition, subtraction, multiplication, and division. The results are updated instantly in real time as the user types, demonstrating state management and automatic re-rendering in React.

EXPERIMENT - 7

Fetching and Displaying API Data

AIM:

To create a React application using Vite that fetches a list of users from the JSON and displays each user's **name** and **email** on the screen.

ALGORITHM:

Step 1: Start the React-Vite project and open App.jsx.

Step 2: Import useState and useEffect for storing and loading API data.

Step 3: Create a state variable:

- users → array to store the fetched user data.

Step 4: Inside useEffect():

- Use fetch() to call the JSON Placeholder API.
- Convert the response to JSON.
- Save the returned data into the users state.

Step 5: Use .map() to display each user's **name** and **email**.

Step 6: Add basic styling for a neat list display.

Step 7: Stop.

SOURCE CODE:

File: src/App.jsx

```
import { useState, useEffect } from "react";


function App() {
  const [users, setUsers] = useState([]);


  useEffect(() => {
```

```
  fetch("https://jsonplaceholder.typicode.com/users")
    .then((res) => res.json())
    .then((data) => setUsers(data));
}, []);

return (
  <>
    <style>{`
      body {
        background: #e3ecf7;
        font-family: Arial, sans-serif;
      }

      .container {
        text-align: center;
        margin-top: 30px;
      }

      .title {
        font-size: 26px;
        margin-bottom: 20px;
        color: #1f2d3d;
      }
```

```
.user-list {
  width: 350px;
  margin: auto;
}

.user {
  padding: 12px 16px;
  margin-bottom: 12px;
  background: #d6e4f5;
  border-radius: 10px;
  text-align: left;
  border-left: 4px solid #4a90e2;
  transition: 0.3s;
}

.user:hover {
  background: #c7daf1;
}

.user p {
  font-size: 14px;
  margin: 4px 0;
```

```
      }

      strong {
        color: #1f2d3d;
      }
    `}</style>

    <div className="container">
      <h1 className="title">User List</h1>

      <div className="user-list">
        {users.map((user) => (
          <div key={user.id} className="user">
            <p><strong>Name:</strong> {user.name}</p>
            <p><strong>Email:</strong> {user.email}</p>
          </div>
        ))}
      </div>
    </div>
  </>
 );
}
```

export default App;

EXPLANATION:

Step 1: Import Hooks

useState stores user data.
useEffect loads data when the component mounts.

Step 2: users State

users is initialized as an empty array and later filled with fetched data.

Step 3: API Call with fetch()

- Fetches user data from JSON Placeholder.
- Converts response to JSON.
- Updates the users state.

Step 4: Display Data Using map()

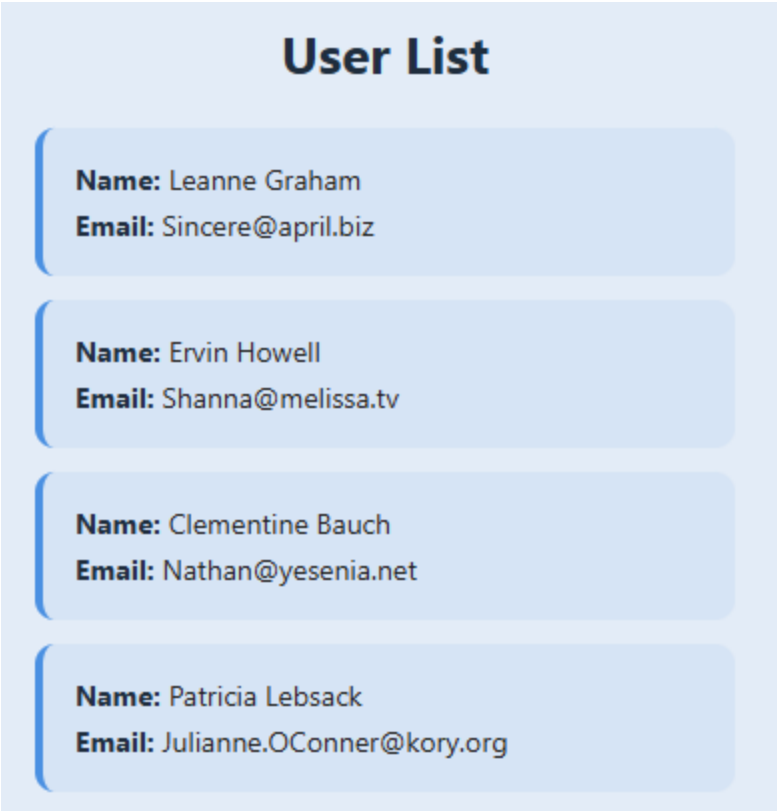
Each user is shown inside a styled card with:

- Name
- Email

Step 5: Styling

A central box is styled with shadows, rounded corners, and padding for clarity.

OUTPUT :



RESULT :

The React application successfully fetches a list of users from the JSON Placeholder API and displays each user's **name** and **email** in a clean, styled layout using useEffect, useState, and array mapping.

EXPERIMENT - 8

Tab Component (Home, About, Contact)

AIM:

To develop a React application with **three tabs** — *Home, About, Contact* — where clicking each tab **dynamically updates and displays** the corresponding content using **conditional rendering**.

ALGORITHM:

Step 1: Start a new React Vite project or use an existing one.

Step 2: Create a functional component named **Tabs**.

Step 3: Create a state variable activeTab using useState() to track the current tab.

Step 4: Render three buttons: **Home**, **About**, **Contact**.

Step 5: When a button is clicked, update the activeTab state.

Step 6: Use conditional rendering to display content based on the value of activeTab.

Step 7: Add simple styling to highlight the active tab and format the content box.

Step 8: Export and render the component inside App.jsx.

SOURCE CODE:

File : src/App.jsx

```
import { useState } from "react";

function Tabs() {
  const [activeTab, setActiveTab] = useState("home");

  return (
    <div style={styles.container}>
      <h2>Tab Navigation Example</h2>

      {/* Tab Buttons */}
      <div style={styles.tabContainer}>
        <button
```

```
          style={activeTab === "home" ? styles.activeBtn : styles.btn}
          onClick={() => setActiveTab("home")}
        >
          Home
        </button>

        <button
          style={activeTab === "about" ? styles.activeBtn : styles.btn}
          onClick={() => setActiveTab("about")}
        >
          About
        </button>

        <button
          style={activeTab === "contact" ? styles.activeBtn : styles.btn}
          onClick={() => setActiveTab("contact")}
        >
          Contact
        </button>
      </div>

      {/* Conditional Content */}
      <div style={styles.contentBox}>
        {activeTab === "home" && (
          <p>Welcome to the Home tab! This is the homepage content.</p>
        )}

        {activeTab === "about" && (
          <p>About Us: We are learning React conditional rendering and tabs.</p>
        )}

        {activeTab === "contact" && (
          <p>Contact Us at: contact@example.com</p>
        )}
      </div>
    </div>
  );
}

const styles = {
```

```
  container: {
    textAlign: "center",
    padding: "20px",
    fontFamily: "Arial",
  },
  tabContainer: {
    marginBottom: "20px",
  },
  btn: {
    padding: "10px 20px",
    margin: "5px",
    cursor: "pointer",
    background: "#ddd",
    border: "1px solid #aaa",
    borderRadius: "5px",
  },
  activeBtn: {
    padding: "10px 20px",
    margin: "5px",
    cursor: "pointer",
    background: "#4caf50",
    color: "white",
    border: "1px solid #3e8e41",
    borderRadius: "5px",
  },
  contentBox: {
    padding: "20px",
    border: "1px solid #ccc",
    width: "300px",
    margin: "0 auto",
    borderRadius: "5px",
    background: "#f9f9f9",
  },
};

export default Tabs;
```

EXPLANATION:

Step 1: useState for Active Tab

const [activeTab, setActiveTab] = useState("home");

This stores which tab is currently selected.

Step 2: Tab Buttons - Each button updates the state on click:

onClick={() => setActiveTab("home")}

The active tab button gets a different color using inline styles.

Step 3 : Conditional Rendering

React checks which tab is active:

{activeTab === "home" && (Home content)}

Only the matching content appear**s.**

Step 4: Styling

styles.activeBtn → for selected tab

styles.btn → for normal tabs

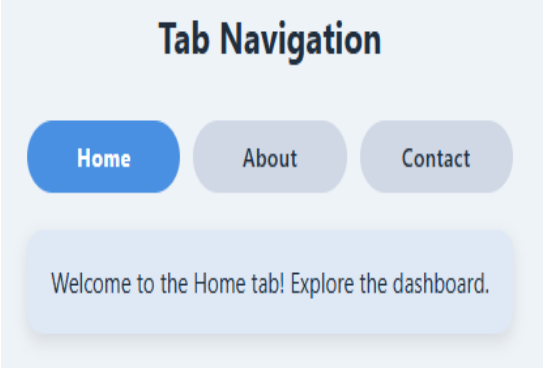
styles.contentBox → to display text nicely

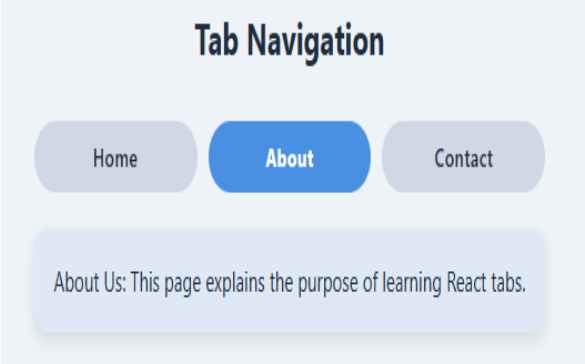
OUTPUT :

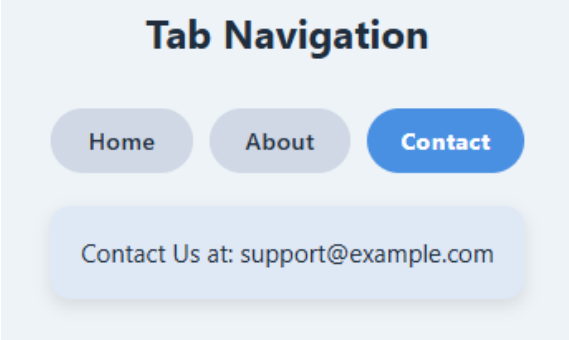
Live Demo Output :

Tab_Navigation

Screen-Shot :







RESULT :

The React application successfully displays three tabs—Home, About, and Contact—and shows the correct content when each tab is clicked using state management and conditional rendering.

EXPERIMENT - 9

Dark / Light Mode Toggle

AIM :

Add a Dark Mode / Light Mode toggle to a React + Vite app so the page background and text colors switch when the user toggles the theme.

ALGORITHM :

Step 1: Start a new React-Vite project and open App.jsx.

Step 2: Import the useState hook to manage the theme toggle state.

Step 3: Create a state variable isDark to store the current theme mode.

Step 4: Define two style objects: lightTheme and darkTheme with background and text colors.

Step 5: Create a variable currentTheme to choose the theme based on isDark.

Step 6: Design a button that toggles the theme by updating isDark using setIsDark(!isDark).

Step 7: Apply the selected theme style (currentTheme) to the main container using inline CSS.

Step 8: Add some sample text or content to visually confirm color changes.

Step 9: Stop.

SOURCE CODE :

File: src/App.jsx

```
import { useState } from "react";
```

```
export default function App() {
  const [isDark, setIsDark] = useState(false);
  const lightTheme = {
    backgroundColor: "#ffffff",
    color: "#111827",
  };
  const darkTheme = {
    backgroundColor: "#0f172a",
    color: "#e6eef8",
  };
  const currentTheme = isDark ? darkTheme : lightTheme;
  return (
    <div
      style={{
        minHeight: "100vh",
        padding: "40px 20px",
        transition: "background-color 250ms ease, color 250ms ease",
        ...currentTheme,
        fontFamily: "Arial, sans-serif",
      }}
    >
      <div style={{ maxWidth: 800, margin: "0 auto", textAlign: "center" }}>
        <h1 style={{ marginBottom: 6 }}>Theme Toggle Example</h1>
```

```
<p style={{ marginTop: 0, marginBottom: 18 }}>
  Click the button to switch between Light and Dark mode.
</p>
<button
  onClick={() => setIsDark((s) => !s)}
  aria-pressed={isDark}
  style={{
    padding: "10px 16px",
    borderRadius: 8,
    border: "none",
    cursor: "pointer",
    boxShadow: "0 2px 6px rgba(0,0,0,0.12)",
    background: isDark ? "#1f2937" : "#f3f4f6",
    color: isDark ? "#e6eef8" : "#111827",
    fontWeight: 600,
    marginBottom: 24,
  }}
>
  {isDark ? "Switch to Light Mode" : "Switch to Dark Mode"}
</button>
<div
  style={{
    padding: 20,
```

```
          borderRadius: 10,
          border: `1px solid ${isDark ? "rgba(255,255,255,0.06)" :
"rgba(0,0,0,0.08)"}`,
          background: isDark ? "rgba(255,255,255,0.02)" : "rgba(0,0,0,0.02)",
          textAlign: "left",
        }}
      >
        <h2 style={{ margin: "0 0 8px 0" }}>Preview Area</h2>
        <p style={{ margin: 0 }}>
          This text and the page background change according to the selected theme.
        </p>
      </div>
    </div>
  </div>
 );
}
```

EXPLANATION :

Step 1: Import useState
Needed to store whether the theme is dark or light.

Step 2: Create isDark state
Holds the current theme mode (true = dark, false = light).

Step 3: Create theme objects
lightTheme and darkTheme contain background and text colors.

Step 4: Select active theme
currentTheme chooses which theme to apply based on isDark.

Step 5: Apply theme
Spread currentTheme into the main container style so the page updates instantly.

Step 6: Add toggle button
Clicking the button reverses the value of isDark and switches themes.

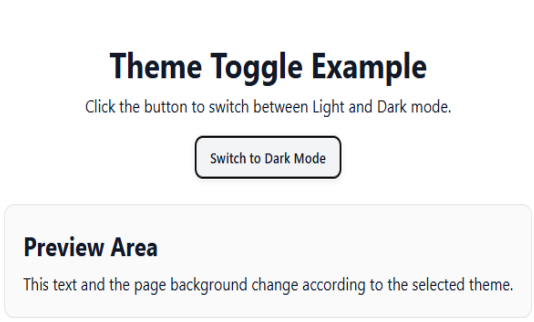
Step 7: Display preview content
Allows the user to visually verify the theme change.

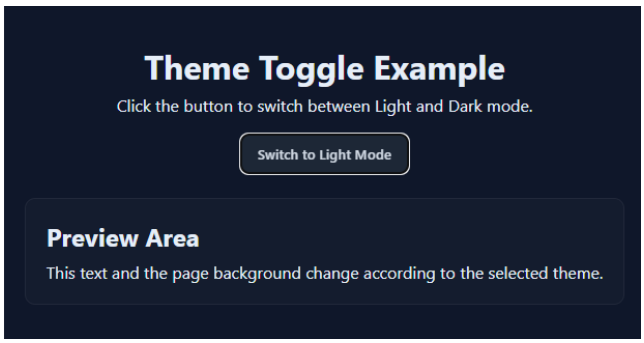
OUTPUT :

Live_Demo_Video :

[Light_Theme_Dark_Theme]

ScreenShot :





RESULT:

The app successfully toggles between Dark and Light modes. Clicking the button switches the page background and text colors instantly and smoothly, reflecting the selected theme.

EXPERIMENT - 10

Create a mini blog with 2 pages: Home page shows a list of posts, Post page shows full content when clicked.

AIM :

To develop a mini React blog application with two pages:

1. **Home Page** – displays a list of blog posts
2. **Post Page** – displays full content of a selected post using React Router navigation

ALGORITHM

Step 1: Create a new React-Vite project and install React Router.

Step 2: Create two components: Home.jsx and Post.jsx.

Step 3: Define an array of posts with id, title, and content.

Step 4: In Home.jsx, display all post titles and add a button/link for “Read More”.

Step 5: Enable navigation to the Post page using useNavigate() or <Link>.

Step 6: In Post.jsx, extract the post ID from the URL using useParams().

Step 7: Find the selected post based on its ID.

Step 8: Display the full content of the selected post.

Step 9: Add a “Back to Home” button.

Step 10: Configure routes in main.jsx using <BrowserRouter> and <Routes>.

Step 11: Run and test the blog.

SOURCE CODE :

```
import React from "react";

import ReactDOM from "react-dom/client";
```

```
import { BrowserRouter, Routes, Route } from "react-router-dom";

import Home from "./Home";
import Post from "./posts";

ReactDOM.createRoot(document.getElementById("root")).render(
  <BrowserRouter>
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/post/:id" element={<Post />} />
    </Routes>
  </BrowserRouter>
);
```

File : src/Home.jsx

```
import { Link } from "react-router-dom";

export default function Home() {
  const posts = [
    {
      id: 1,
      title: "React Introduction",
      content:
```

```
      "React is a JavaScript library used to build fast and interactive user interfaces..."
    },
    {
      id: 2,
      title: "Understanding Components",
      content:
        "Components are the reusable building blocks of a React application..."
    }
  ];

  return (
    <div
      style={{
        padding: "30px",
        fontFamily: "Arial, sans-serif",
        maxWidth: "900px",
        margin: "0 auto",
      }}
    >
      <h1
        style={{
          marginBottom: "20px",
```

```
    textAlign: "center",
    color: "#222",
  }}
>
  📘 Mini Blog – Home
</h1>

<p
  style={{
    textAlign: "center",
    marginBottom: "30px",
    color: "#555",
    fontSize: "15px",
  }}
>
  Select a blog post to view the full article.
</p>

 {posts.map((post) => (
  <div
    key={post.id}
    style={{
      padding: "20px",
```

```
    marginBottom: "20px",
    borderRadius: "12px",

    // Card Background (Soft Blue)
    background: "linear-gradient(135deg, #e8f0ff, #f2f7ff)",

    border: "1px solid #cdd9f0",
    boxShadow: "0 2px 8px rgba(0,0,0,0.1)",
  }}
>
  <h2 style={{ margin: "0 0 10px 0", color: "#1a3d7c" }}>
    {post.title}
  </h2>

  <p style={{ marginBottom: "12px", color: "#444" }}>
    {post.content.substring(0, 90)}...
  </p>

  <Link
    to={`/post/${post.id}`}
    style={{
      textDecoration: "none",
      color: "#0b5ed7",
```

```
            fontWeight: "bold",
          }}
        >
          Read Full Post →
        </Link>
      </div>
    ))}
  </div>
 );
}
```

File : src/Post.jsx

```
import { useParams, Link } from "react-router-dom";

export default function Post() {
 const { id } = useParams();

 const posts = [
  {
   id: 1,
   title: "React Introduction",
   content:
```

```
      "React is a JavaScript library developed by Facebook.\n\nIt helps build fast, responsive user interfaces by using components and a Virtual DOM.\n\nReact allows developers to break large UIs into smaller reusable parts, making development efficient."
    },
    {
      id: 2,
      title: "Understanding Components",
      content:
      "Components are independent building blocks in React.\n\nEach component manages its own structure and UI using JSX.\n\nThey improve reusability, readability, and organization in React applications."
    }
  ];

  const post = posts.find((p) => p.id === parseInt(id));

  if (!post) {
    return (
      <h2 style={{ padding: 20, fontFamily: "Arial" }}>Post Not Found</h2>
    );
  }

  return (
    <div
```

```
  style={{
    padding: "30px",
    fontFamily: "Arial, sans-serif",
    maxWidth: "800px",
    margin: "0 auto",
  }}
>
  <Link
    to="/"
    style={{
      display: "inline-block",
      marginBottom: "20px",
      textDecoration: "none",
      color: "#0b5ed7",
      fontWeight: 600,
    }}
  >
    ← Back to Home
  </Link>

  <div
    style={{
      padding: "25px",
```

```
        borderRadius: "12px",
        background: "linear-gradient(135deg, #fff8e6, #fff2cc)",
        border: "1px solid #f0d9a8",
        boxShadow: "0 2px 8px rgba(0,0,0,0.1)",
      }}
    >
      <h1 style={{ marginTop: 0, color: "#8a5400" }}>{post.title}</h1>

      <p
        style={{
          lineHeight: "1.7",
          whiteSpace: "pre-line",
          color: "#444",
          fontSize: "16px",
        }}
      >
        {post.content}
      </p>
    </div>
  </div>
 );
}
```

EXPLANATION :

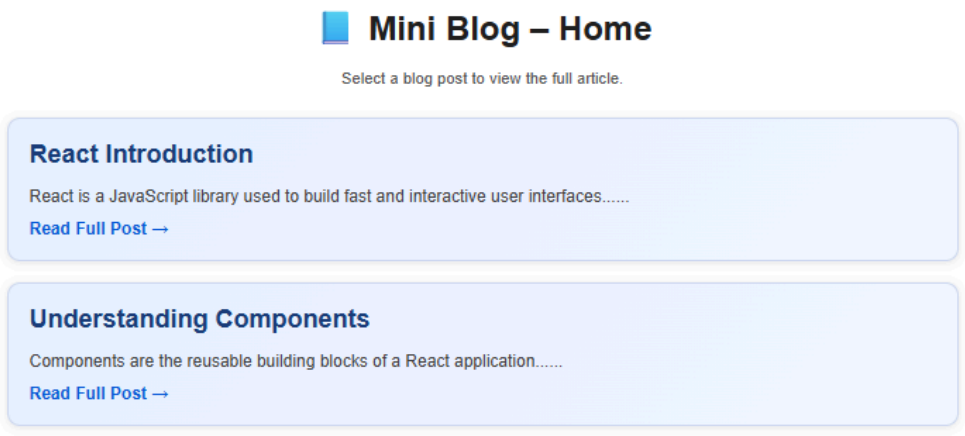
Step 1: Import and install React Router for page navigation.
Step 2: Create a list of posts in a separate file.
Step 3: Home page displays the list of posts using .map().
Step 4: Each post has a **Read More** link that passes the post ID to the Post page.
Step 5: Post page uses useParams() to extract the post ID from the URL.
Step 6: The correct post is found using .find().
Step 7: Full post content is displayed.
Step 8: A "Back to Home" link navigates back to the Home page.

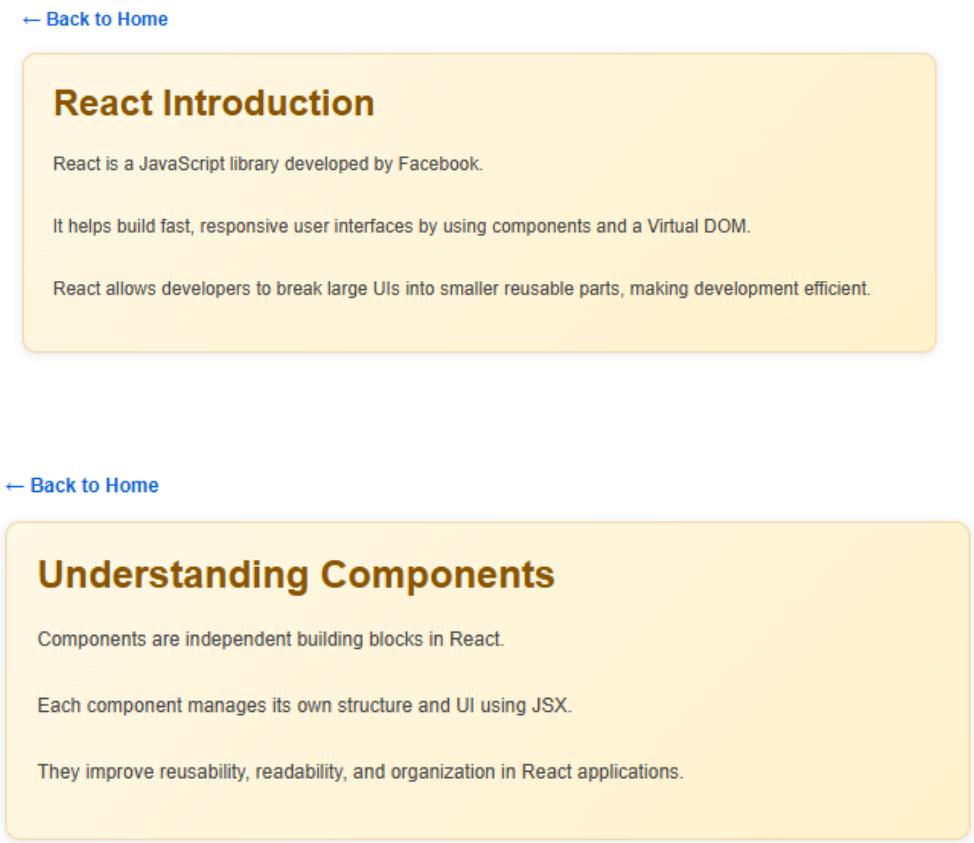
OUTPUT :

Live_Demo_Video :

[Mini_Vlog]

ScreenShot :





RESULT :

The React mini blog application successfully displays a list of posts on the Home page and shows the full content on a separate Post page when a post is clicked using React Router.