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;

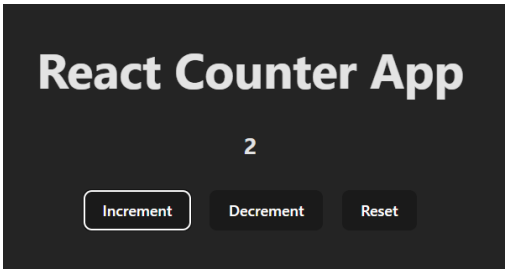
Exports the App component for rendering in the browser.

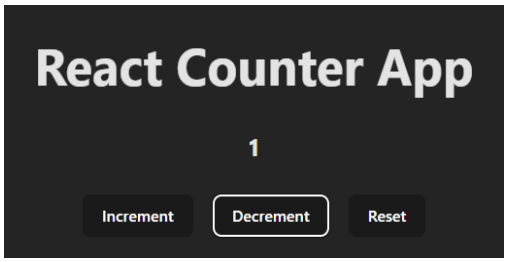
OUTPUT :

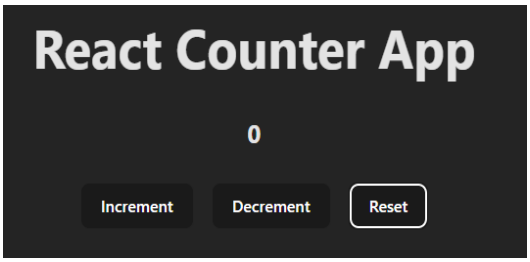
Live_Demo_Video :

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

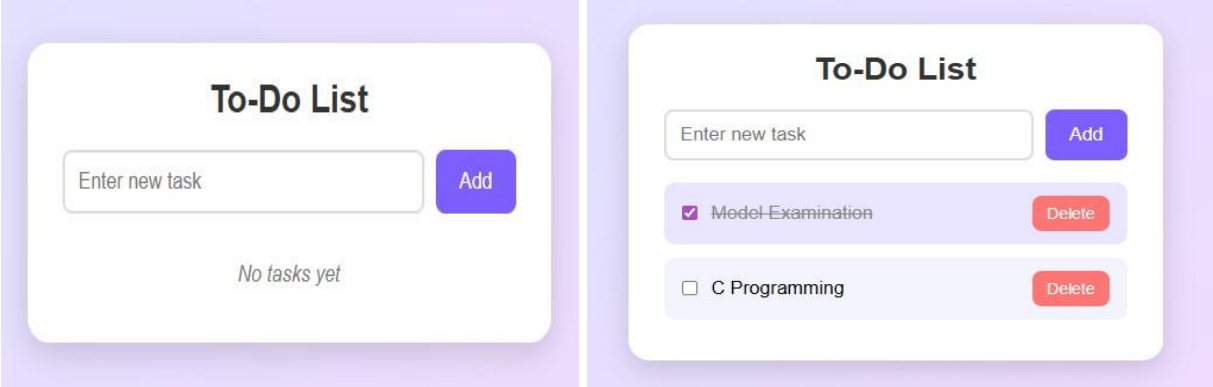
- Step 9.1: tasks.map() displays each task item.
- Step 9.2: Checkbox toggles task completion.

- 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.

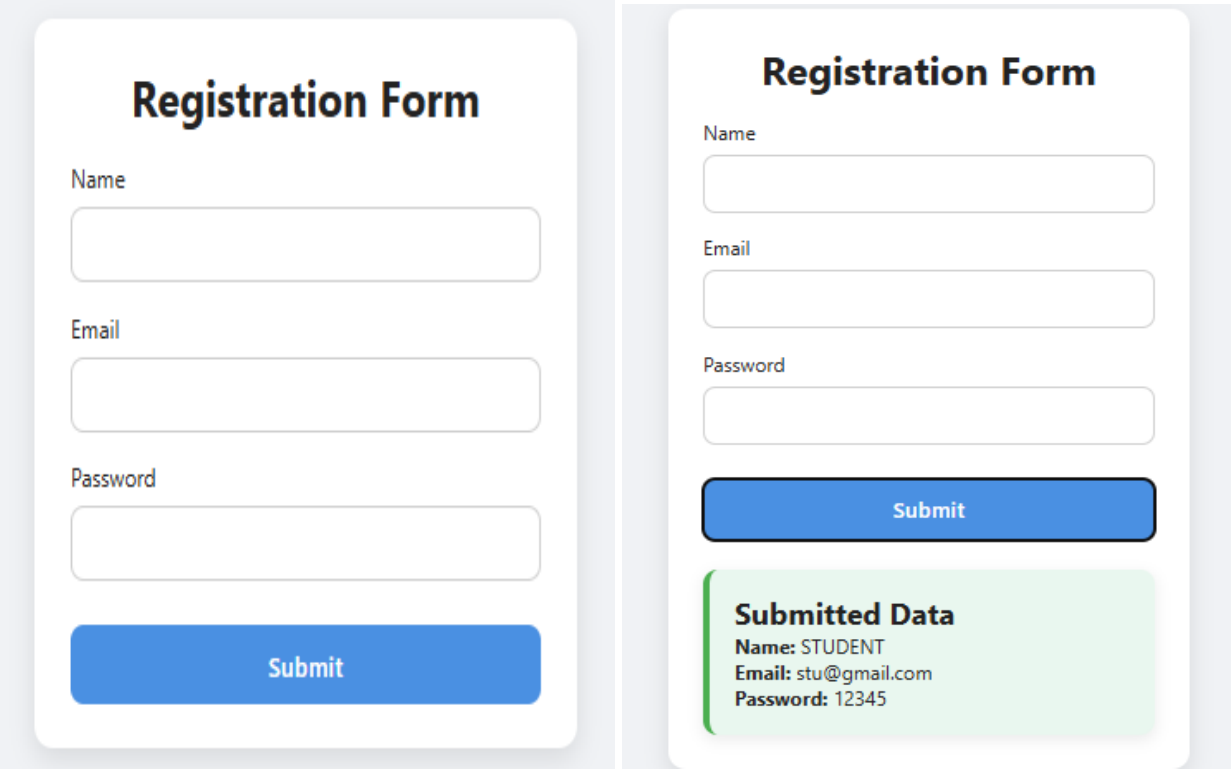
Step 4: Create the form UI

- 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.