1. Create a Vite + React Project

Step 1: Install Node.js

Make sure Node.js is installed:

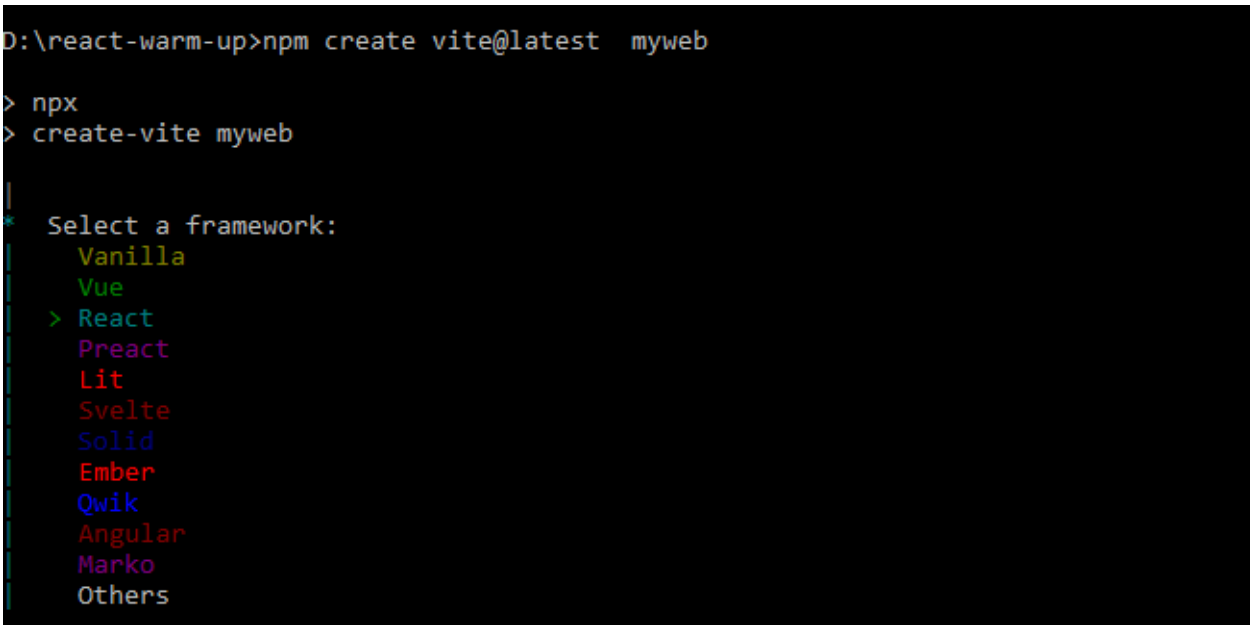
```
node -v
npm -v
```

Step 2: Create Vite project

Run:

npm create vite@latest myweb

select framework React (using down arrow button) as shown below



And select Variant as Javascript (using Down arrow button) as shown below.

```
D:\react-warm-up>npm create vite@latest  myweb

> npx
> create-vite myweb

|
o  Select a framework:
|  React
|
*  Select a variant:
|    TypeScript
|    TypeScript + React Compiler
|  > JavaScript
|    JavaScript + React Compiler
|    RSC
|    React Router v7 ⧉ https://reactrouter.com
|    TanStack Router ⧉ https://tanstack.com/router
|    RedwoodSDK ⧉ https://rwsdk.com
|    Vike ⧉ https://vike.dev
—
```

And after it press yes as shown below for install with npm and start now :-

```
D:\react-warm-up>npm create vite@latest  myweb

> npx
> create-vite myweb

|
o  Select a framework:
|  React
|
o  Select a variant:
|  JavaScript
|
*  Install with npm and start now?
|  > Yes /   No
—
```

Your app will run at:

http://localhost:5173

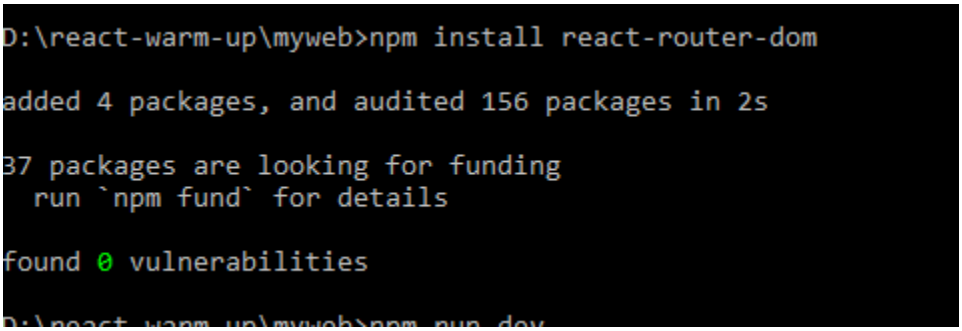
Note to run your app:-

```
cd myweb

npm run dev
```

2. Install React Router (for 2 pages)

We need routing to create multiple pages:

```
npm install react-router-dom
```

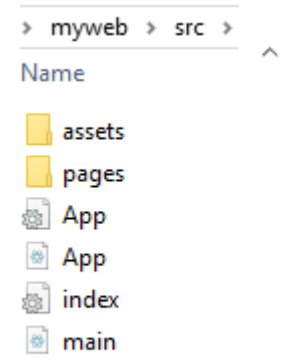


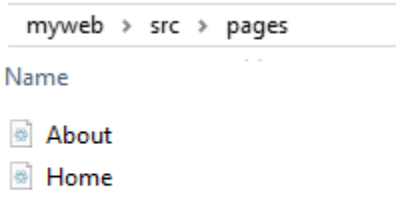
3. Project Structure (important)

Inside `src/`:

```
src/
├── pages/
│   ├── Home.jsx
│   ├── About.jsx
│
├── App.jsx
├── main.jsx
```

as shown below project structure :-





4. Setup Router in `main.jsx`

Edit `src/main.jsx`:

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.jsx'
import { BrowserRouter } from 'react-router-dom'

ReactDOM.createRoot(document.getElementById('root')).render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
)
```

5. Create Pages

`src/pages/Home.jsx`

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


function Home() {

  return (

    <div>

      <h1>Home Page</h1>

      <p>Welcome to the Home page</p>

        <Link to="/about">Go to About Page</Link>

    </div>

  );

}


export default Home;
```

---

src/pages/About.jsx

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


function About() {
  return (
    <div>
      <h1>About Page</h1>
      <p>This is the About page</p>
          <Link to="/">Go to Home</Link>
    </div>
  );
}

export default About;
```

---

6. Setup Routes in `App.jsx`

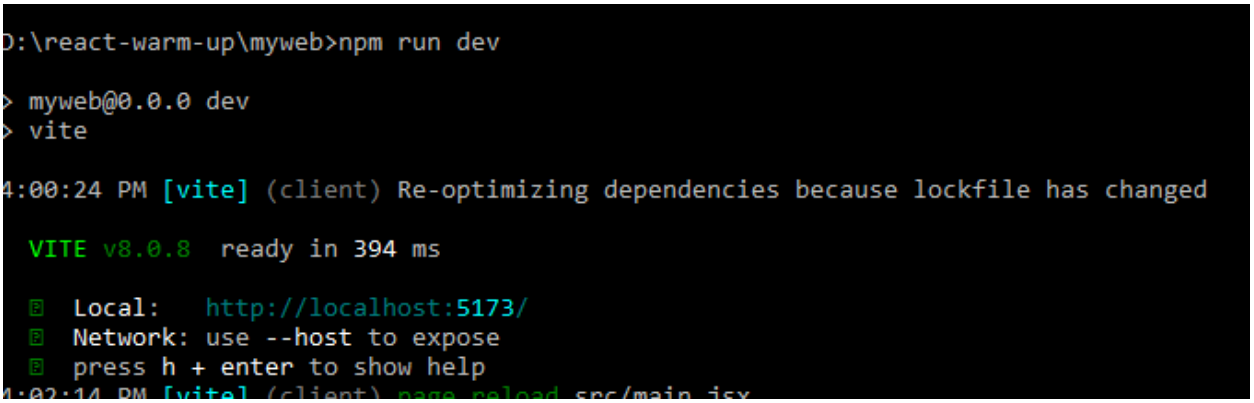
Edit `src/App.jsx`:

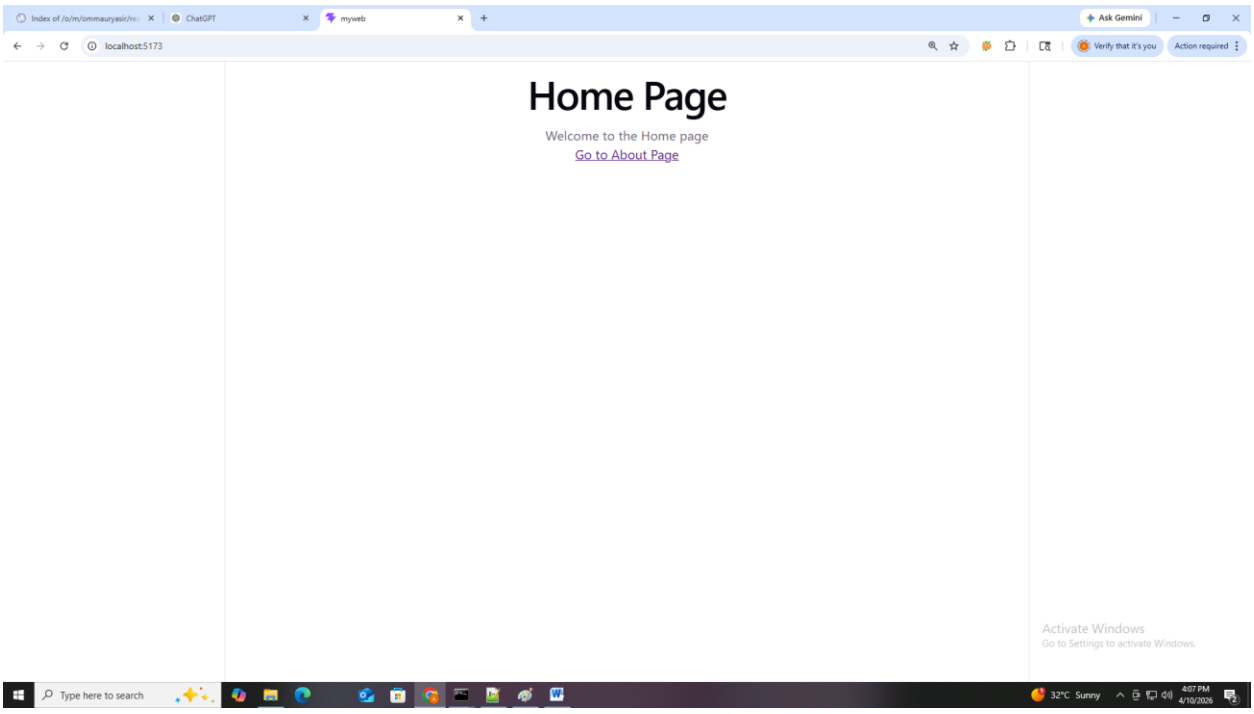
```
import { Routes, Route } from "react-router-dom";
import Home from "./pages/Home";
import About from "./pages/About";

function App() {
  return (
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/about" element={<About />} />
    </Routes>
  );
}

export default App;
```

Run Command:-

npm run dev

and see Output just visit http://localhost:5173/ :-





□ What are React Props?

Props (short for "properties") are used in React to **pass data from one component to another (parent → child)**.

Think of props like:

□ "arguments you send to a function"

Here's the **modern React props way (clean + commonly used today)**

We use:

- **destructuring in function parameters**
- **arrow function components (optional but modern)**
- clean JSX

□ 1. Child Component (Modern Way)

User.jsx

```
const User = ({ name, handleClick }) => {
  return (
    <div>
      <h2>Name: {name}</h2>

      <button onClick={handleClick}>
        Click Me
      </button>
    </div>
  );
};

export default User;
```

---

2. Parent Component

App.jsx

```
import User from "./User";

const App = () => {
  const handleClick = () => {
    alert("Button clicked!");
  };

  return (
    <div>
      <h1>Modern Props Example</h1>

      <User
        name="John Doe"
        handleClick={handleClick}
      />
    </div>
  );
};

export default App;
```

What is “modern” here?

✔ Destructuring props directly:

```
({ name, handleClick })
```

✔ Arrow function components:

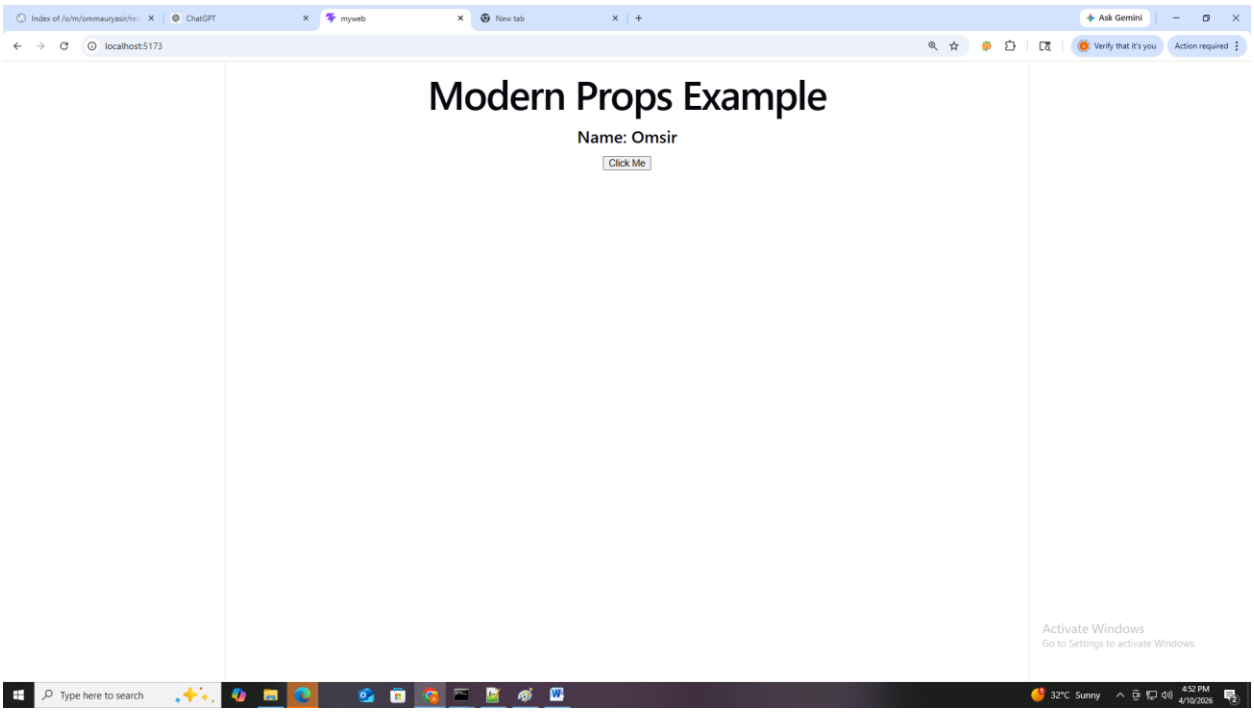
```
const User = () => {}
```

✔ Cleaner and widely used in real projects (React + Vite + Next.js)

Output:-

cd myweb

npm run dev



What is `useState`?

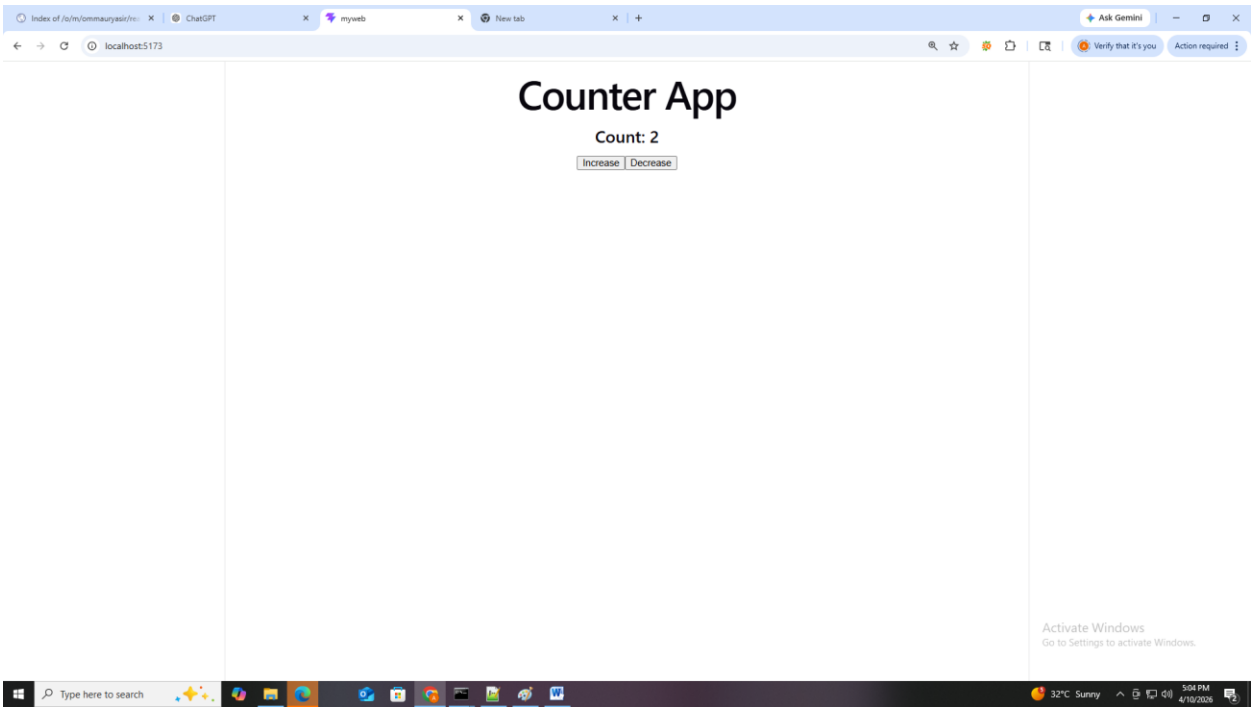
`useState` is a React Hook used to **store and update data (state)** inside a component.

Simple Example (Counter)

App.jsx

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

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

  return (
    <div>
      <h1>Counter App</h1>

      <h2>Count: {count}</h2>

      <button onClick={() => setCount(count + 1)}>
        Increase
      </button>

      <button onClick={() => setCount(count - 1)}>
        Decrease
      </button>
    </div>
  );
};

export default App;
```

Output:-



□ Line-by-line explanation

1 □ Import useState

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

□ We bring `useState` from React

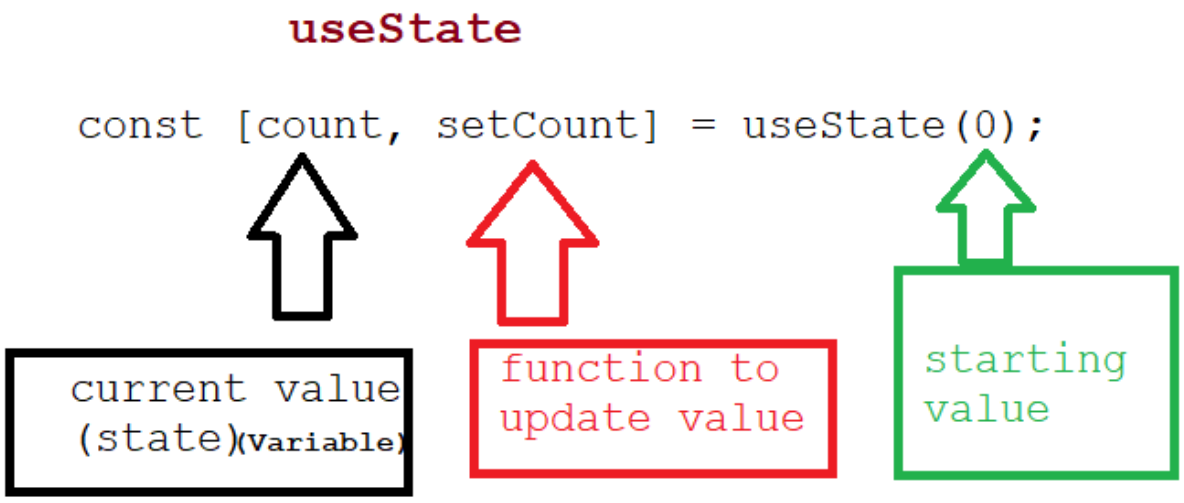
---

2 □ Create state

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

Meaning:

- `count` → current value (state)
- `setCount` → function to update value
- `0` → initial value (starting value)



3 Show state in UI

```
<h2>Count: {count}</h2>
```

Displays current value

---

4 Update state

```
setCount(count + 1)
```

Increases value and re-renders UI automatically

What happens when you click button?

1. Click button
2. `setCount()` runs

3. React updates value
4. UI refreshes automatically

Simple meaning

 `useState` = “memory inside component that updates UI when changed”

Here's a **simple React form with BOTH live preview + submit button**

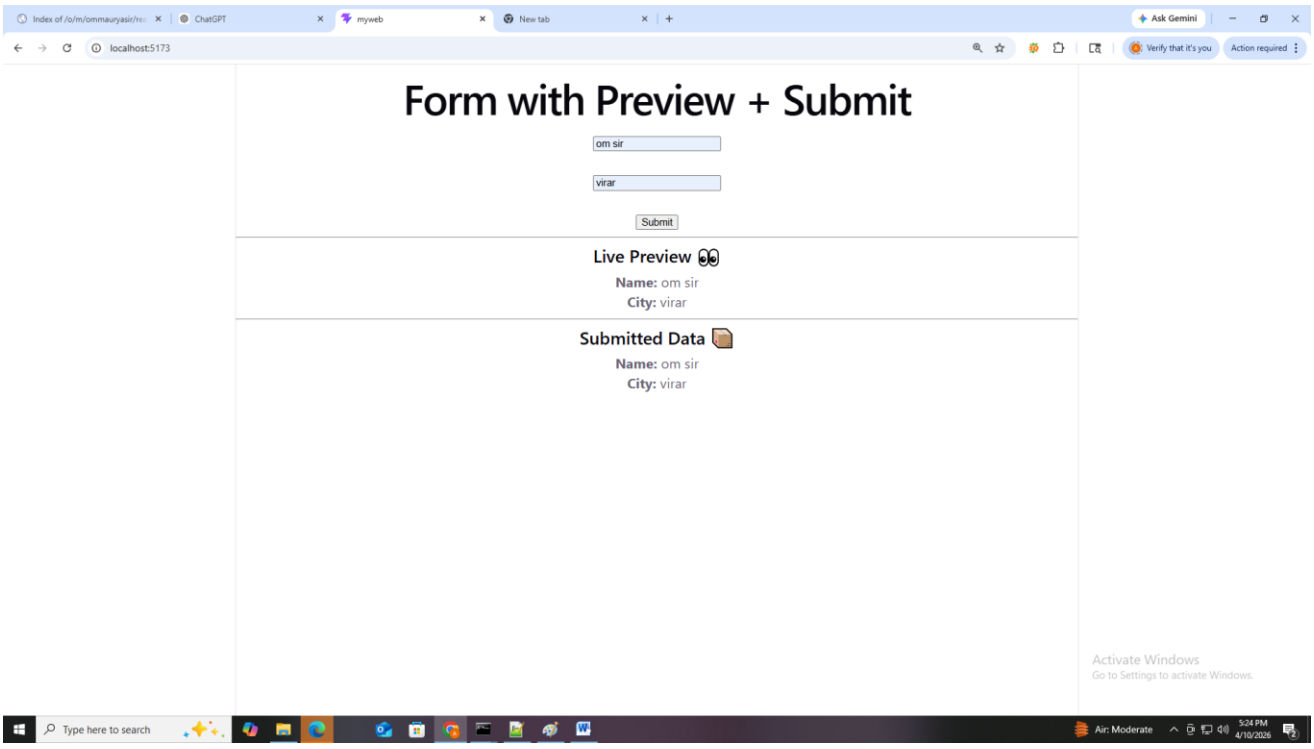
Form with Live Preview + Submit

App.jsx

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

const App = () => {
  const [formData, setFormData] = useState({
    name: "",
    city: ""
  });

  const [submittedData, setSubmittedData] = useState(null);

  const handleChange = (e) => {
    const { name, value } = e.target;

    setFormData((prev) => ({
      ...prev,
      [name]: value
    }));
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    setSubmittedData(formData);
  };

  return (
    <div>
      <h1>Form with Preview + Submit</h1>

      {/* Form */}
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          name="name"
          placeholder="Enter name"
          value={formData.name}
          onChange={handleChange}
        />

        <br /><br />

        <input
          type="text"
          name="city"
          placeholder="Enter city"
          value={formData.city}
          onChange={handleChange}
```

```
        />

        <br /><br />

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

      <hr />

      {/* Live Preview */}
      <h2>Live Preview □</h2>
      <p><b>Name:</b> {formData.name}</p>
      <p><b>City:</b> {formData.city}</p>

      <hr />

      {/* Submitted Data */}
      <h2>Submitted Data □</h2>

      {submittedData ? (
        <>
          <p><b>Name:</b> {submittedData.name}</p>
          <p><b>City:</b> {submittedData.city}</p>
        </>
      ) : (
        <p>No data submitted yet</p>
      )}
    </div>
  );
};

export default App;
```

OUTPUT:-


Index of /o/m/ommauryasir/re
ChatGPT
myweb
New tab
Ask Gemini
localhost:5173
Verify that it's you
Action required
Form with Preview + Submit
om sir
virar
Submit
Live Preview
Name: om sir
City: virar
Submitted Data
Name: om sir
City: virar
Activate Windows
Go to Settings to activate Windows.
Type here to search
Air: Moderate
5:24 PM
4/10/2026

□ How it works

□ 1. Typing in input

- Updates `formData`
- Shows **live preview immediately**

---

2. Live Preview section

```
formData.name
formData.city
```

□ Always shows current typing

---

3. Submit button clicked

```
setSubmittedData(formData);
```

□ Saves current form data into another state

---

4. Submitted data section

```
submittedData ? show data : "No data"
```

□ Only updates when user clicks Submit

□ Final Behavior

While typing:

```
Live Preview updates instantly □
```

After clicking Submit:

```
Submitted Data section updates □
```

□ Key Concept

✔ `formData` → live typing state
✔ `submittedData` → saved final data

Here is a **simple line-by-line explanation** of your React code □□

□ 1. Import useState

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

□ Brings `useState` hook from React
□ Used to store and update data inside the component

□ 2. Create Component

```
const App = () => {
```

□ This is a **React functional component** named `App`
□ Arrow function style

□ 3. First state (form data)

```
const [formData, setFormData] = useState({
  name: "",
  city: ""
});
```

□ Creates state object for form inputs

Meaning:

- `formData` → current values
- `setFormData` → function to update values
- Initial state:

```
{ name: "", city: "" }
```

□ 4. Second state (submitted data)

```
const [submittedData, setSubmittedData] = useState(null);
```

□ Stores data AFTER submit
□ Starts as `null` (nothing submitted yet)

✍□ 5. handleChange function

```
const handleChange = (e) => {
```

□ Runs when user types in input

```
const { name, value } = e.target;
```

□ Extracts:

- `name` → input field name ("name" or "city")
- `value` → typed text

```
setFormData((prev) => ({
  ...prev,
  [name]: value
}));
```

□ Updates state safely

Meaning:

- `prev` → previous state (old data)
- `...prev` → copy old data from previous state
- `[name]` → dynamically update correct field
- `value`→ new typed value

✔ If typing in "city", only city updates

□ 6. handleSubmit function

```
const handleSubmit = (e) => {
```

□ Runs when form is submitted

```
e.preventDefault();
```

□ Stops page reload (default HTML behavior)

```
setSubmittedData(formData);
```

□ Saves current form data into `submittedData`

□ 7. return (UI section)

```
return (
```

□ Everything inside is what appears on screen (JSX)

□ 8. Main container

```
<div>
```

 Wrapper for all elements

---

9. Heading

```
<h1>Form with Preview + Submit</h1>
```

 Title of the page

---

10. Form start

```
<form onSubmit={handleSubmit}>
```

 Form element
 When submitted → calls `handleSubmit`

✍ 11. Name input

```
<input
  type="text"
  name="name"
  placeholder="Enter name"
  value={formData.name}
  onChange={handleChange}
/>
```

Meaning:

- `type="text"` → text input
- `name="name"` → key in state object
- `value={formData.name}` → controlled input
- `onChange` → updates state when typing

✍ 12. City input

```
<input
  type="text"
  name="city"
  placeholder="Enter city"
  value={formData.city}
  onChange={handleChange}
/>
```

 Same logic as name input
 Updates `city` in state

---

13. Submit button

```
<button type="submit">Submit</button>
```

 Submits the form
 Triggers `handleSubmit`

---

14. Horizontal line

```
<hr />
```

 Just a visual separator

---

15. Live Preview

```
<p><b>Name:</b> {formData.name}</p>
<p><b>City:</b> {formData.city}</p>
```

 Shows real-time typing from `formData`

---

16. Submitted section title

```
<h2>Submitted Data □</h2>
```

□ Heading for submitted result

□ 17. Conditional rendering

```
{submittedData ? (
```

□ If data exists → show it
□ If not → show fallback text

□ 18. Show submitted data

```
<>
  <p><b>Name:</b> {submittedData.name}</p>
  <p><b>City:</b> {submittedData.city}</p>
</>
```

□ Displays saved data after submit

□ 19. Else case

```
) : (
  <p>No data submitted yet</p>
)}
```

□ If nothing submitted → show message

□ 20. Close component

```
);
};

export default App;
```

□ Ends component
□ `export default` allows usage in other files

□ FINAL SUMMARY

This code does 3 things:

✔ Takes input (name + city)
✔ Shows live preview while typing
✔ Stores data after submit