React Router **PART-1**

React Router - *Simple Routing, Route Parameters, Query Parameters, Links, Nested Routes, React Forms - Controlled Components, Filters, Typed Input, Edit Form, Number Input, Date Input, Text Input, Update API, Delete API.*

React Router

React Router is the **Standard Routing Library for React Applications**. It allows you to build **Single-Page Applications (SPAs)** with **Navigation** without reloading the entire page.

In a normal website, moving between pages triggers a full page reload. With React Router, navigation happens **Client-Side**, which means only the necessary components update — making apps **faster and smoother**.

Why Do We Need React Router?

In normal websites:

- Clicking a link → reloads the entire page.

In React with React Router:

- Clicking a link → changes the URL → only updates the **relevant component**.
- No full page reload.

❖ Better user experience

❖ Faster navigation

❖ Enables **Multi-Page App behavior inside a Single-Page App**

Core Concepts

(a) Router

The **container** that wraps your application.

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

function App() {
  return (
    <BrowserRouter>
      <MyRoutes />
    </BrowserRouter>
  );
}
```

(b) Routes & Route

Defines which component should render at a specific URL.

```
import { Routes, Route } from "react-router-dom";
function MyRoutes() {
  return (
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/about" element={<About />} />
      <Route path="/products" element={<Products />} />
    </Routes>
  );
}
```

(c) Link

Used instead of <a> for navigation **without reload**.

```
import { Link } from "react-router-dom";
function Navbar() {
  return (
    <nav>
      <Link to="/">Home</Link> | <Link to="/about">About</Link> | <Link
to="/products">Products</Link>
    </nav>
  );
}
```

(d) useNavigate (programmatic navigation)

Lets you navigate **via code**.

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

function Login() {
  const navigate = useNavigate();

  const handleLogin = () => {
    // After login, redirect to dashboard
    navigate("/dashboard");
  };

  return <button onClick={handleLogin}>Login</button>;
}
```

(e) useParams (Dynamic Routes)

Pass variables in URLs.

```
<Route path="/product/:id" element={<ProductDetail />} />

function ProductDetail() {
  const { id } = useParams();
  return <h2>Product ID: {id}</h2>;
}
```

❖ /product/5 → shows "Product ID: 5".

Types of Routers

- **BrowserRouter** → Uses HTML5 history API (most common).
- **HashRouter** → Uses # in URLs (e.g., example.com/#/about).
- **MemoryRouter** → Stores history in memory (used in tests).

Advanced Features

1. Nested Routes

```
<Routes>
  <Route path="dashboard" element={<Dashboard />}>
    <Route path="profile" element={<Profile />} />
    <Route path="settings" element={<Settings />} />
  </Route>
</Routes>
```

❖ URL /dashboard/profile → loads Profile inside Dashboard.

2. Route Guards (Protected Routes)

```
function PrivateRoute({ children }) {
  const isAuth = localStorage.getItem("auth");
  return isAuth ? children : <Navigate to="/login" />;
}
<Route path="/dashboard" element={<PrivateRoute><Dashboard /></PrivateRoute>} />
```

3. 404 Page (Fallback)

```
<Route path="*" element={<NotFound />} />
```

Summary

- **React Router** enables navigation **without page reloads**.
- Core elements:
  - BrowserRouter, Routes, Route
  - Link, useNavigate, useParams
- Supports **dynamic routes, nested routes, protected routes, and 404 pages**.
- Makes SPAs behave like **multi-page apps**.

Simple Routing in React Router

This is the most basic setup for navigation between pages in a React app.

1. Create a React Project

If you don't already have one, create it using Create **React App**.

npx create-react-app my-router-app

cd my-router-app

2. Install React Router

Inside your project folder:

npm install react-router-dom

3. Basic Setup

App.js

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

function App() {
  return (
    <BrowserRouter>
      <nav>
        <Link to="/">Home</Link> |{" "}
        <Link to="/about">About</Link> |{" "}
        <Link to="/contact">Contact</Link>
      </nav>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
      </Routes>
    </BrowserRouter>
  );
}
```

```
function Home() {
  return <h2>🏠 Home Page</h2>;
}

function About() {
  return <h2>ℹ️ About Page</h2>;
}

function Contact() {
  return <h2>📞 Contact Page</h2>;
}

export default App;
```

4. Run the Project

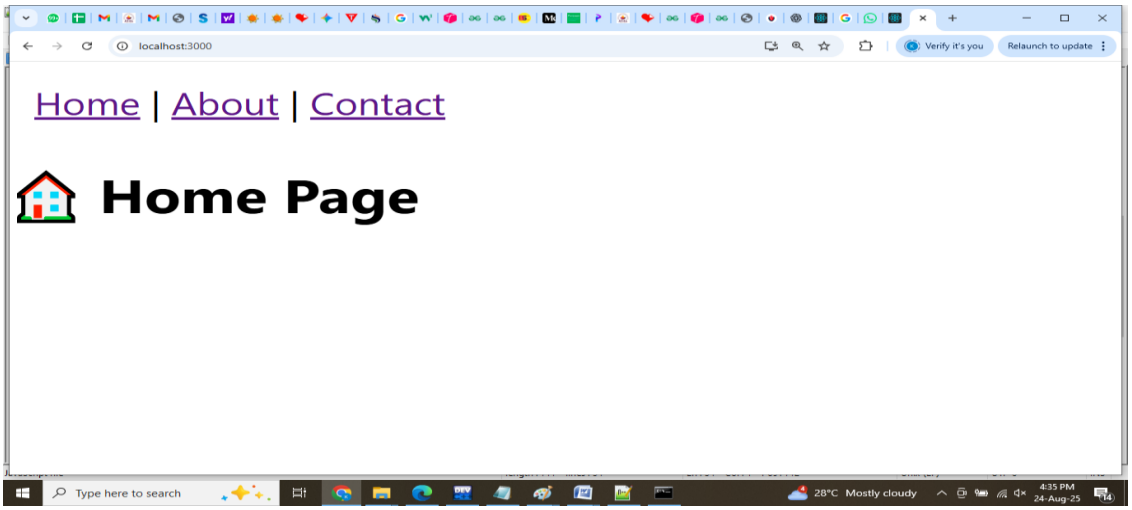
npm start

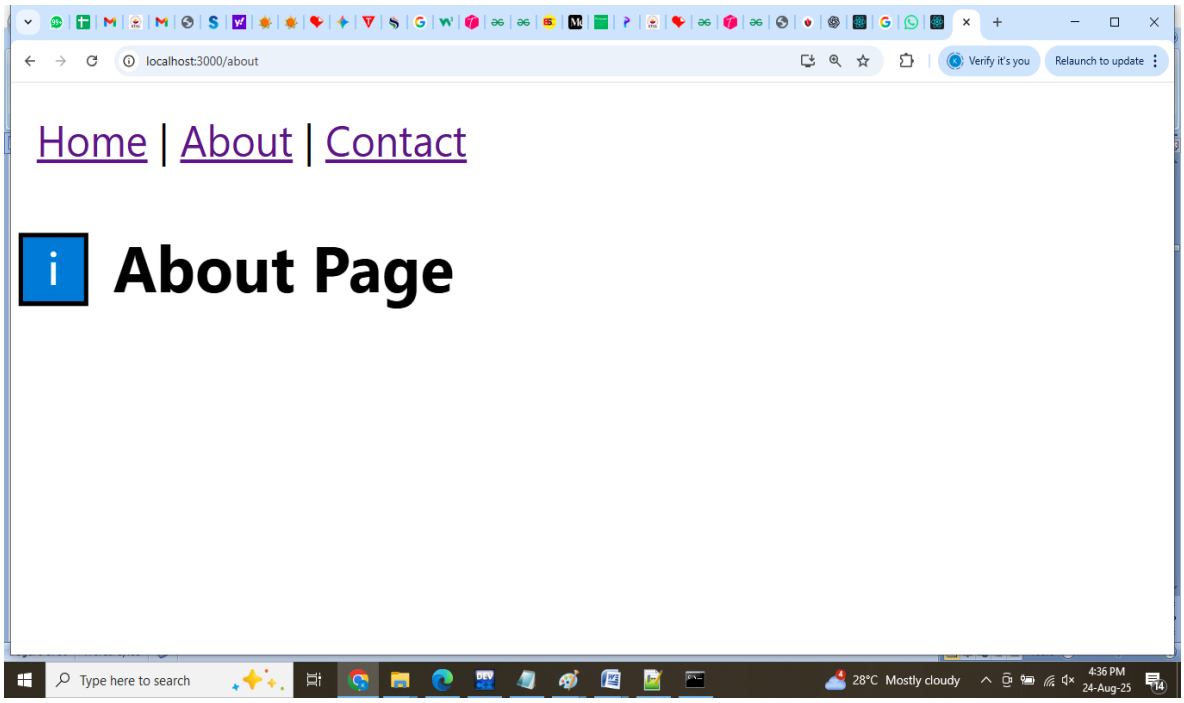
How it Works

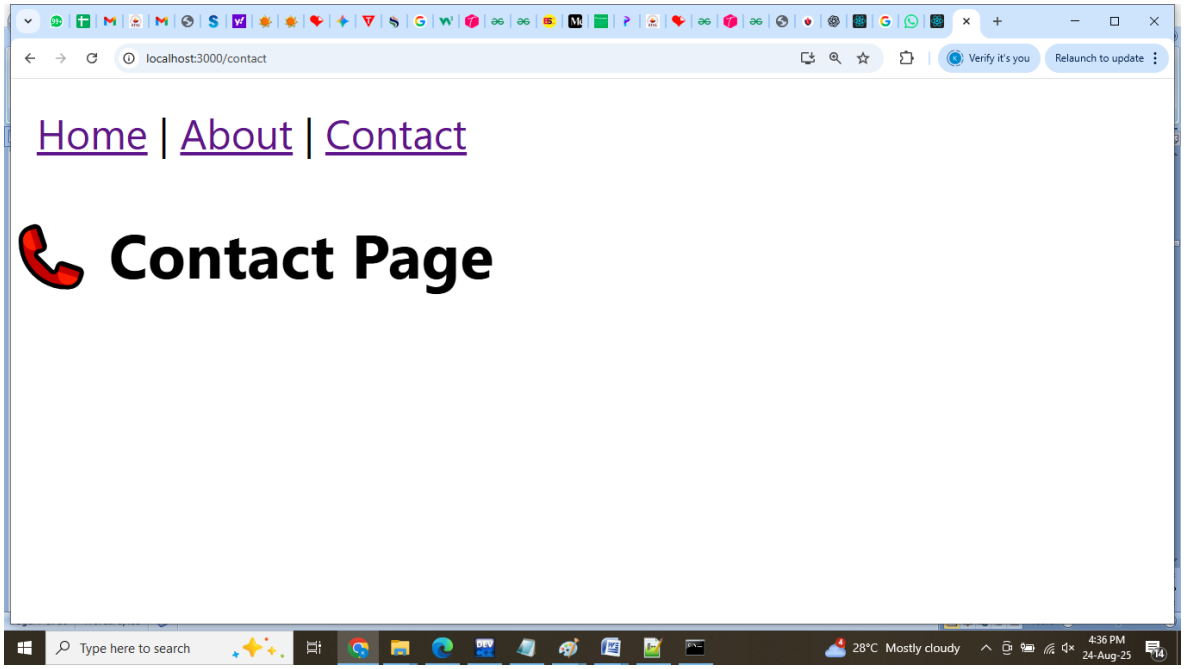
- **BrowserRouter** → Wraps the app, enabling routing.
- **Routes & Route** → Define which component should render for each path.
- **Link** → Used instead of <a> tags to navigate **without page reload**.

Example behavior:

- Visiting / → Shows **Home Page**.
- Visiting /about → Shows **About Page**.
- Visiting /contact → Shows **Contact Page**.







Key Points

- Simple Routing = Just mapping **URLs → Components**.
- Ideal for **small apps** (e.g., static websites).
- Later it can be extended with **nested routes, dynamic routes, and protected routes**.

Route Parameters

Route Parameters allows you to pass **dynamic values** inside the URL.

- **Example**:
  - /products/1 → Shows product with ID 1.
  - /products/2 → Shows product with ID 2.
- This avoids writing separate routes for each item.

Defining a Route with Parameters

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

function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/products/:id" element={<ProductDetail />} />
      </Routes>
    </BrowserRouter>
  );
}

function ProductDetail() {
  const { id } = useParams(); // Access the parameter
  return <h2>Product Details for ID: {id}</h2>;
}

export default App;
```

- Visiting /products/5 will render "**Product Details for ID: 5**".

Multiple Parameters

You can define **more than one** parameter:

```
<Route path="/products/:category/:id" element={<ProductDetail />} />
function ProductDetail() {
  const { category, id } = useParams();
  return <h2>Category: {category}, Product ID: {id}</h2>;
}
```

- /products/electronics/101 → "Category: electronics, Product ID: 101".

Optional Parameters

- React Router v6 does not directly support :id? like v5 did.
- Instead, you define **two routes**:

```
<Routes>
  <Route path="/products" element={<ProductList />} />
  <Route path="/products/:id" element={<ProductDetail />} />
</Routes>
```

Example with Links

```
import { Link } from "react-router-dom";
function ProductList() {
  const products = [
    { id: 1, name: "Laptop" },
    { id: 2, name: "Phone" },
    { id: 3, name: "Tablet" },
  ];
  return (
    <div>
      <h2>Products</h2>
      {products.map((p) => (
        <div key={p.id}>
          <Link to={`/products/${p.id}`}>{p.name}</Link>
        </div>
      ))}
    </div>
  );
}
```

❖ Clicking a product → navigates to /products/:id → renders ProductDetail.

Summary

- **Route Parameters** = Dynamic segments in the URL.
- Access them with useParams().
- Useful for **detail pages, user profiles, filtering, categories, etc.**
- Multiple parameters and optional routes are possible.

Query Parameters

Query parameters are extra info in the **URL after a ?**.

- Example:
  - /products?category=electronics
  - /search?q=laptop&page=2
- Unlike **Route Params** (/products/:id), query params are **optional key-value pairs**.

Accessing Query Parameters

React Router v6 provides the **useSearchParams** hook:

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

function SearchPage() {
  const [searchParams] = useSearchParams();

  const query = searchParams.get("q");   // ?q=laptop
  const page = searchParams.get("page"); // ?page=2

  return (
    <div>
      <h2>Search Results</h2>
      <p>Query: {query}</p>
      <p>Page: {page}</p>
    </div>
  );
}

export default SearchPage;
```

- Visiting /search?q=laptop&page=2 →

Query: laptop
Page: 2

Setting Query Parameters

You can also update them with setSearchParams:

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

function FilterProducts() {
  const [searchParams, setSearchParams] = useSearchParams();

  const setCategory = (category) => {
    setSearchParams({ category }); // updates ?category=value
  };

  return (
    <div>
      <button onClick={() => setCategory("electronics")}>Electronics</button>
      <button onClick={() => setCategory("fashion")}>Fashion</button>
      <p>Current Category: {searchParams.get("category")}</p>
    </div>
  );
}

export default FilterProducts;
```

❖ Clicking a button updates the URL:

- /products?category=electronics
- /products?category=fashion

Example with Shopping Cart

Let's say we want to **filter products by category**:

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

function ProductList({ products }) {
  const [searchParams] = useSearchParams();
  const category = searchParams.get("category");
```

```
  const filteredProducts = category
    ? products.filter((p) => p.category === category)
    : products;

  return (
    <div>
      <h2>Products {category && `(Category: ${category})`}</h2>
      {filteredProducts.map((p) => (
        <div key={p.id}>
          {p.name} - ${p.price}
        </div>
      ))}
    </div>
  );
}
```

❖ Visiting /products?category=electronics → shows only **electronics products**.

Key Differences (Route Params vs Query Params)

Feature	Route Params (/products/:id)	Query Params (/products?id=1)
Position in URL	Part of the path	After ? in URL
Usage	Identifies a **specific resource**	Provides **extra info / filters**
Accessed with	useParams()	useSearchParams()
Example	/products/10	/products?id=10&category=tech

Summary:

- **Use Route Params** for unique IDs (e.g., product details).
- **Use Query Params** for filtering, sorting, search, pagination.

Links in React Router

In React Router, **<Link>** is used instead of **<a>** for **navigation**.

❖ **Why?**

- <a href="/page"> reloads the entire page (traditional web).
- <Link to="/page"> updates the **URL and UI** without reloading (SPA behavior).

Basic Example

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

function Navbar() {
  return (
    <nav>
      <Link to="/">Home</Link> |{" "}
      <Link to="/products">Products</Link> |{" "}
      <Link to="/cart">Cart</Link>
    </nav>
  );
}

export default Navbar;
```

❖ Clicking any link changes the **URL** and loads the correct component **without refresh**.

Links with Route Params

```
<Link to={`/products/${product.id}`}>
  {product.name}
</Link>
```

❖ If product.id = 2, the link becomes /products/2.

Links with Query Params

```
<Link to="/products?category=electronics">
  Electronics
</Link>
```

```
<Link to="/products?category=fashion">
  Fashion
</Link>
```

❖ Clicking updates the URL with **query string**.

Styling Links

React Router provides <NavLink> which works like <Link> but adds an **active class** when the link matches the current route.

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

function Navbar() {
  return (
    <nav>
      <NavLink
        to="/"
        style={({ isActive }) => ({ color: isActive ? "red" : "black" })}
      >
        Home
      </NavLink> |{" "}
      <NavLink to="/cart">Cart</NavLink>
    </nav>
  );
}
```

❖ The active link will appear **red**.

Programmatic Navigation (without clicking a link)

Sometimes you want to navigate **in code** (e.g., after login).
React Router gives useNavigate() hook:

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

function LoginButton() {
  const navigate = useNavigate();
```

```
  const handleLogin = () => {
    // logic for login
    navigate("/cart"); // go to cart after login
  };

  return <button onClick={handleLogin}>Login</button>;
}
```

Summary

- <Link> → normal navigation
- <NavLink> → navigation with active styling
- useNavigate() → navigate programmatically
- Can pass **route params** and **query params** with links

Nested Routes

Nested Routes let you render components **inside other components** based on the URL structure.

❖ **Example:**

- /products → Show list of products.
- /products/:id → Show details of a specific product **inside** the Products page.

❖ Instead of defining every route separately, we can **nest them under /products**.

Basic Setup

In App.js:

```
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Navbar from "./Navbar";
import ProductLayout from "./ProductLayout";
import ProductList from "./ProductList";
import ProductDetail from "./ProductDetail";
import Cart from "./Cart";
import { useState } from "react";
```

```
function App() {
  const [products] = useState([
    { id: 1, name: "Laptop", price: 1000, description: "High-performance laptop",
category: "electronics" },
    { id: 2, name: "Phone", price: 500, description: "Latest smartphone", category:
"electronics" },
    { id: 3, name: "T-Shirt", price: 30, description: "Cotton T-shirt", category: "fashion" },
    { id: 4, name: "Shoes", price: 80, description: "Running shoes", category: "fashion" },
  ]);

  const [cart, setCart] = useState([]);
  const addToCart = (product) => setCart([...cart, product]);

  return (
    <BrowserRouter>
      <Navbar cartCount={cart.length} />
      <Routes>
        <Route path="/" element={<h2>Welcome to Shop</h2>} />

        {/* Nested routes for products */}
        <Route path="products" element={<ProductLayout />}>
          <Route index element={<ProductList products={products} />} />
          <Route path=":id" element={<ProductDetail products={products}
addToCart={addToCart} />} />
        </Route>

        <Route path="/cart" element={<Cart cart={cart} />} />
      </Routes>
    </BrowserRouter>
  );
}

export default App;
```

ProductLayout.js

This component will act as a **wrapper** for nested routes.

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

function ProductLayout() {
  return (
    <div>
      <h2>Products</h2>
      <nav>
        <Link to="/products">All</Link> |{" "}
        <Link to="/products?category=electronics">Electronics</Link> |{" "}
        <Link to="/products?category=fashion">Fashion</Link>
      </nav>
      <hr />
      {/* Nested route components appear here */}
      <Outlet />
    </div>
  );
}

export default ProductLayout;
```

Behavior

- /products → Shows **ProductList**.
- /products/1 → Still inside **ProductLayout**, but now shows **ProductDetail for Laptop**.
- Navbar at the top stays intact, ProductLayout stays intact, only the **Outlet changes**.

Why Use Nested Routes?

✓ Cleaner route definitions.
✓ Keeps shared layout (headers, tabs, filters) without re-writing code.
✓ Perfect for sections like Products, User Profiles, Settings pages, etc.

Controlled Components

In React, forms are usually built using **Controlled Components** (where React controls the state of inputs).

- In **Controlled Components**, the form inputs are linked to React's useState.
- React state is the **Single Source Of Truth** for form values.

Example:

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

function ControlledForm() {
  const [name, setName] = useState("");

  const handleSubmit = (e) => {
    e.preventDefault();
    alert(`Hello, ${name}`);
  };

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
   <input type="text" value={name} onChange={(e) => setName(e.target.value)} />
      </label>
      <button type="submit">Submit</button>
    </form>
  );
}
```

❖ value is bound to state, and onChange updates it.

Filters with Forms

Filters are just **controlled inputs** that update state, which can be used to filter data.

```
function ProductFilter({ onFilter }) {
  const [category, setCategory] = useState("");
```

```
  return (
    <div>
      <label>
        Filter by Category:
        <select value={category} onChange={(e) => {
          setCategory(e.target.value);
          onFilter(e.target.value);
        }}>
          <option value="">All</option>
          <option value="electronics">Electronics</option>
          <option value="fashion">Fashion</option>
        </select>
      </label>
    </div>
  );
}
```

Typed Input

Each input type (text, number, date) is controlled the same way, but React ensures values match the type.

Text Input

```
const [text, setText] = useState("");

<input
  type="text"
  value={text}
  onChange={(e) => setText(e.target.value)}
  placeholder="Enter your name"
/>
```

Number Input

```
const [age, setAge] = useState("");

<input
  type="number"
  value={age}
  onChange={(e) => setAge(e.target.value)}
  placeholder="Enter your age"
/>
```

Date Input

```
const [dob, setDob] = useState("");

<input
  type="date"
  value={dob}
  onChange={(e) => setDob(e.target.value)}
/>
```

Edit Form

When editing, initialize form fields with existing data.

```
function EditProductForm({ product, onSave }) {
  const [name, setName] = useState(product.name);
  const [price, setPrice] = useState(product.price);

  const handleSubmit = (e) => {
    e.preventDefault();
    onSave({ ...product, name, price });
  };

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input type="text" value={name} onChange={(e) => setName(e.target.value)} />
      </label>
      <br />
      <label>
        Price:
      <input type="number" value={price} onChange={(e) => setPrice(e.target.value)} />
      </label>
      <br />
      <button type="submit">Save</button>
    </form>
  );
}
```

- This allows editing of product details with initial values prefilled.

Putting It Together

We can now combine all of this in a **Shopping Cart Example**:

- Filter products (<select>).
- Add/Edit product forms (text, number, date).
- Controlled components keep everything in sync.

Summary:

- **Controlled components** → inputs controlled by React state.
- **Filters** → just controlled inputs that change displayed data.
- **Typed inputs** (text, number, date) → handled the same way.
- **Edit form** → initialize state with existing data.

Update API (PUT Request)

React Router **itself doesn't send requests**. Instead, it helps us **Navigate** between pages like:

❖ React Router doesn't actually do the fetch itself — it simply decides:

- /products → Show list of products
- /products/new → Show form for **creating** a product (POST)
- /products/:id/edit → Show form for **editing** an existing product (PUT)

❖ So the **navigation** is handled by React Router, and the **API calls** are handled by fetch inside your form.

Example:

1. Create a React Project

If you don't already have one, create it using Create **React App**.

npx create-react-app my-update-api-demo

cd my-update-api-demo

2. Install React Router

Inside your project folder:

npm install react-router-dom

3. Basic Setup

App.js

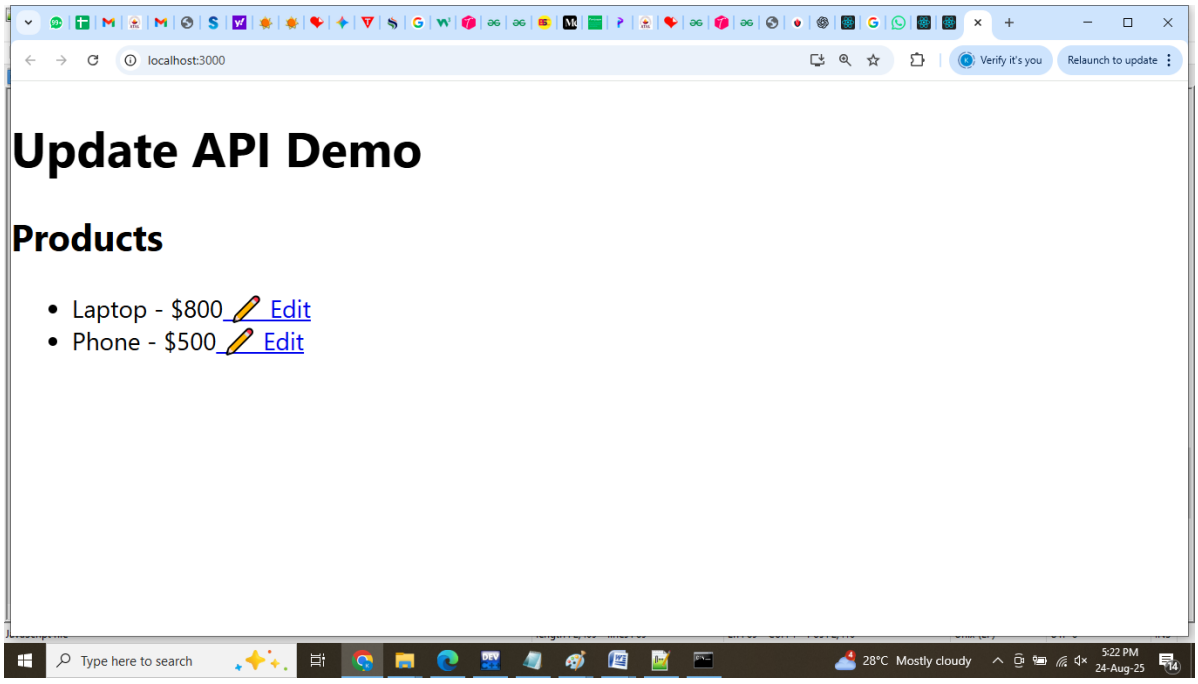
```
import { useState } from "react";
import { BrowserRouter, Routes, Route, Link, useNavigate, useParams } from "react-router-dom";

// Home Page: Show product list
function Home({ products }) {
  return (
    <div>
      <h2>Products</h2>
      <ul>
        {products.map((p) => (
          <li key={p.id}>
            {p.name} - ${p.price}
            <Link to={`/edit/${p.id}`}> ☒ Edit</Link>
          </li>
        ))}
      </ul>
    </div>
  );
}

// Edit Form
function EditForm({ products, setProducts }) {
  const { id } = useParams();
  const navigate = useNavigate();

  const product = products.find((p) => p.id.toString() === id);
  const [name, setName] = useState(product?.name || "");
  const [price, setPrice] = useState(product?.price || "");
  const [loading, setLoading] = useState(false);

  const handleUpdate = async (e) => {
    e.preventDefault();
```

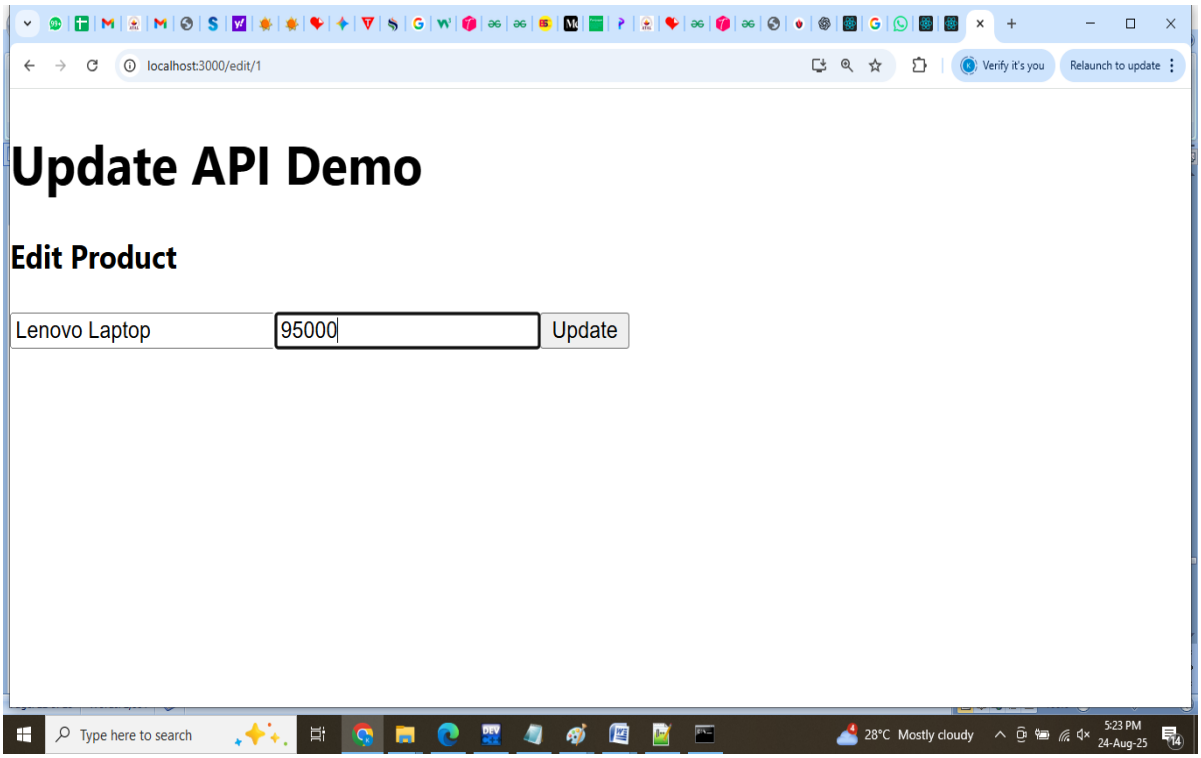
```
    setLoading(true);

    const updatedProduct = { ...product, name, price };

    try {
      // ✅PUT request
      const response = await fetch(`https://jsonplaceholder.typicode.com/posts/${id}`, {
        method: "PUT",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(updatedProduct),
      });

      if (!response.ok) throw new Error("Failed to update");

      const data = await response.json();

      // Update local state
      setProducts(products.map((p) => (p.id.toString() === id ? data : p)));

      navigate("/"); // go back to Home
    } catch (err) {
      alert(err.message);
    } finally {
      setLoading(false);
    }
  };
  return (
    <form onSubmit={handleUpdate}>
      <h3>Edit Product</h3>
      <input value={name} onChange={(e) => setName(e.target.value)} required />
      <input value={price} onChange={(e) => setPrice(e.target.value)} required />
      <button type="submit" disabled={loading}>
        {loading ? "Updating..." : "Update"}
      </button>
    </form>
  );
}
```

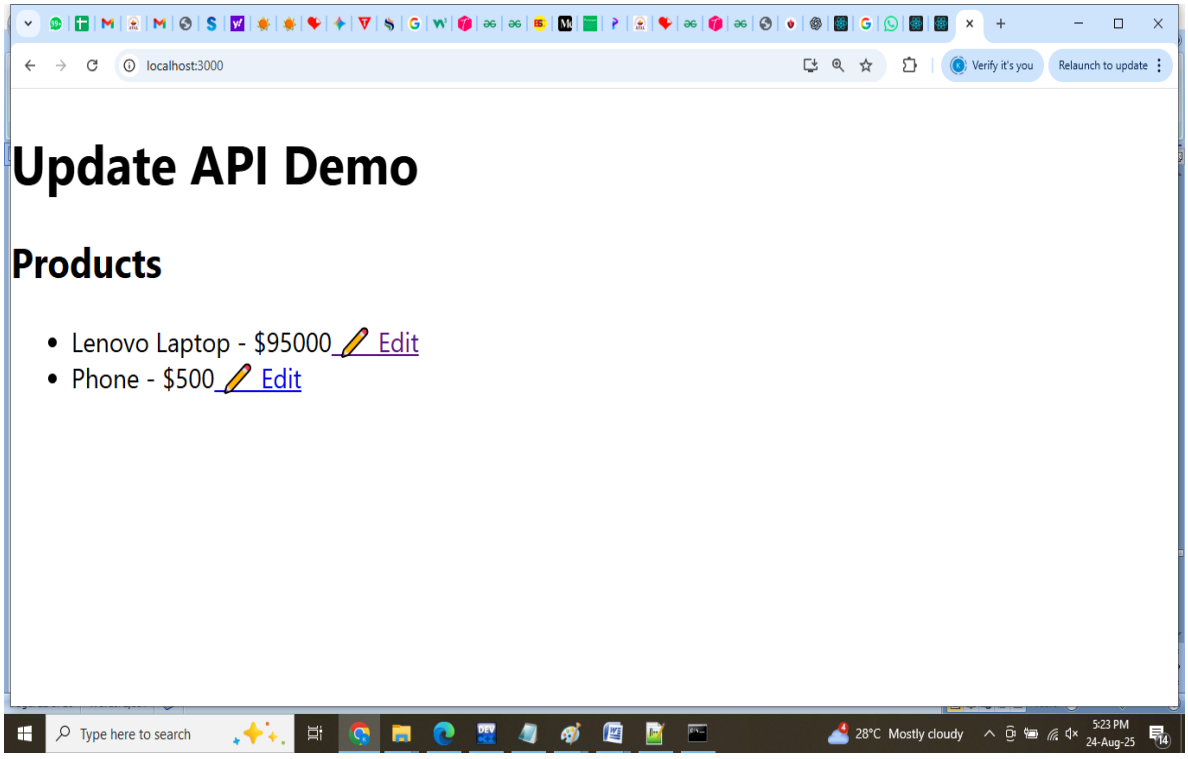
```
// Main App
export default function App() {
  const [products, setProducts] = useState([
    { id: 1, name: "Laptop", price: "800" },
    { id: 2, name: "Phone", price: "500" },
  ]);

  return (
    <BrowserRouter>
      <h1>Update API Demo</h1>
      <Routes>
        <Route path="/" element={<Home products={products} />} />
        <Route path="/edit/:id" element={<EditForm products={products}
setProducts={setProducts} />} />
      </Routes>
    </BrowserRouter>
  );
}
```

4. Run

npm start




localhost:3000/edit/1
Verify it's you
Relaunch to update
Update API Demo
Edit Product
Lenovo Laptop
95000
Update
Type here to search
28°C Mostly cloudy
5:23 PM
24-Aug-25


localhost:3000
Verify it's you
Relaunch to update
Update API Demo
Products
Lenovo Laptop - $95000 Edit
Phone - $500 Edit
Type here to search
28°C Mostly cloudy
5:23 PM
24-Aug-25

Delete API

Delete API integration so that products can be removed from both the **backend** and the **UI list**.

API Endpoint Assumed

DELETE /products/:id

- This will remove a product by its ID.

Example:

1. Create a React Project

If you don't already have one, create it using Create **React App**.

npx create-react-app my-delete-api-demo

cd my-delete-api-demo

2. Install React Router

Inside your project folder:

npm install react-router-dom

3. Basic Setup

Update App.js – Add Delete Support

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

export default function App() {
  const [products, setProducts] = useState([
    { id: 1, name: "Laptop", price: "800" },
    { id: 2, name: "Phone", price: "500" },
  ]);

  const handleDelete = async (id) => {
    try
    {
      // DELETE API request (dummy API used here)
```

```
      const response = await fetch(`https://jsonplaceholder.typicode.com/posts/${id}`,
{
        method: "DELETE",
      });

      if (!response.ok) throw new Error("Failed to delete");

      // Update local state
      setProducts(products.filter((p) => p.id !== id));
    } catch (err) {
      alert(err.message);
    }
  };

  return (
    <div>
      <h1>Delete API Demo</h1>
      <ul>
        {products.map((p) => (
          <li key={p.id}>
            {p.name} - ${p.price}{" "}
            <button onClick={() => handleDelete(p.id)}> Delete</button>
          </li>
        ))}
      </ul>
    </div>
  );
}
```

4.Run

npm start