** Axios + Fetch API + React CRUD**

> **Goal :** By the end of this manual, you will be able to **build, run, test, and modify a real React CRUD app using Axios and Fetch with async/await.**

Core Concepts (Read Before Coding)

1.1 What is Axios? (very clear)

Axios is a promise-based HTTP client for JavaScript used to communicate with backend APIs. It supports GET, POST, PUT, and DELETE methods and automatically converts JSON responses. It also provides features like interceptors, timeouts, and better error handling.

1.2 What is Fetch API?

Fetch is a built-in browser API for making HTTP requests without installing any library. It works with promises and async/await but requires manual JSON conversion and error handling.

1.3 HTTP Methods (you must remember these)

Method	What it does	Lab usage
GET	Read data	Fetch users
POST	Create data	Add user
PUT	Update data	Modify user
DELETE	Remove data	Delete user

API endpoint (we will use this in every experiment):

```
https://jsonplaceholder.typicode.com/users
```

** How Data Travels in APIs**

2.1 Path Parameters

```
/users/3   → means user with id = 3
```

2.2 Query Parameters

```
/users?name=John&city=NY
```

2.3 Request Body (POST / PUT)

Example body:

```
{
  "name": "John",
  "email": "john@test.com"
}
```

2.4 Headers (important in real apps)

```
Content-Type: application/json
Authorization: Bearer token123
```

Axios vs Fetch

Feature	Axios	Fetch
Auto JSON parsing	✅ Yes	❌ No
Error handling	Easy	Manual
Interceptors	✅ Yes	❌ No
Timeout support	✅ Yes	❌ No
Built-in	❌ Install needed	✅ Native

When to use what?

- Use **Axios** → in professional React projects.
- Use **Fetch** → in simple apps or interviews.

Environment Setup (DO THIS CAREFULLY)

Step 1 — Create project

Run in terminal:

```
npx create-react-app axios-fetch-lab
cd axios-fetch-lab
npm install axios
```

Step 2 — Create Axios instance

Create file: **src/api.js**

```
import axios from "axios";

export const api = axios.create({
  baseURL: "https://jsonplaceholder.typicode.com",
});
```

Build CRUD App (Main Experiment)

Replace **src/App.js** with this:

```
import { useEffect, useState } from "react";
import { api } from "./api";

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

  // EXPERIMENT 1 — READ (GET)
  const fetchUsers = async () => {
    const res = await api.get("/users");
    setUsers(res.data);
  };

  // EXPERIMENT 2 — CREATE (POST)
  const addUser = async () => {
    const newUser = { name };
    const res = await api.post("/users", newUser);
    setUsers([...users, res.data]);
  };

  // EXPERIMENT 3 — UPDATE (PUT)
  const updateUser = async (id) => {
    const updated = { name: "Updated Name" };
    await api.put(`/users/${id}`, updated);
    fetchUsers();
  };
  // EXPERIMENT 4 — DELETE
  const deleteUser = async (id) => {
    await api.delete(`/users/${id}`);
    setUsers(users.filter(u => u.id !== id));
  };

  useEffect(() => {
    fetchUsers();
  }, []);

  return (
    <div style={{ padding: 20 }}>
      <h1>Axios CRUD Lab</h1>

      <input value={name} onChange={e => setName(e.target.value)}
placeholder="Enter name" />
      <button onClick={addUser}>Add User</button>

      <ul>
        {users.map(user => (
          <li key={user.id}>
            {user.name}
            <button onClick={() => updateUser(user.id)}>Update</button>
            <button onClick={() => deleteUser(user.id)}>Delete</button>
          </li>
        ))}
      </ul>
    </div>
  );
}

export default App;
```

Compare with Fetch (Side Experiment)

Inside App add:

```
const fetchUsersWithFetch = async () => {
  const res = await fetch("https://jsonplaceholder.typicode.com/users");
  const data = await res.json();
  setUsers(data);
};
```

Then change `useEffect` to:

```
useEffect(() => {
  fetchUsersWithFetch();
}, []);
```

⛎ Observe: Fetch needs **extra code** compared to Axios.

Tasks for You (Practice Sheet)

Try these yourself:

1. Add email field in input.
2. Send `{ name, email }` in POST body.
3. Change baseURL to your own API later.
4. Add a refresh button.
5. Add error message if API fails.