Great—this is the perfect next step

Now you'll connect **Express.js + MySQL** and store real data instead of memory.

I'll give you a **simple, beginner-friendly API** for a table with:
`id, name, city, phone`

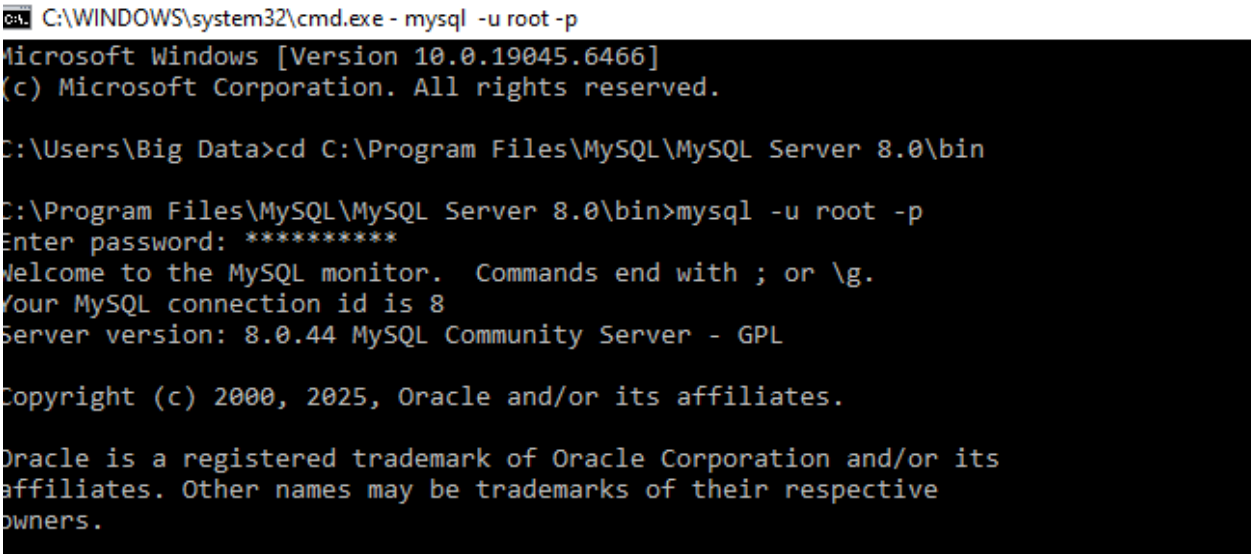
□ Step 1: Install Packages

Run in your project folder:

```
npm install express mysql2
```

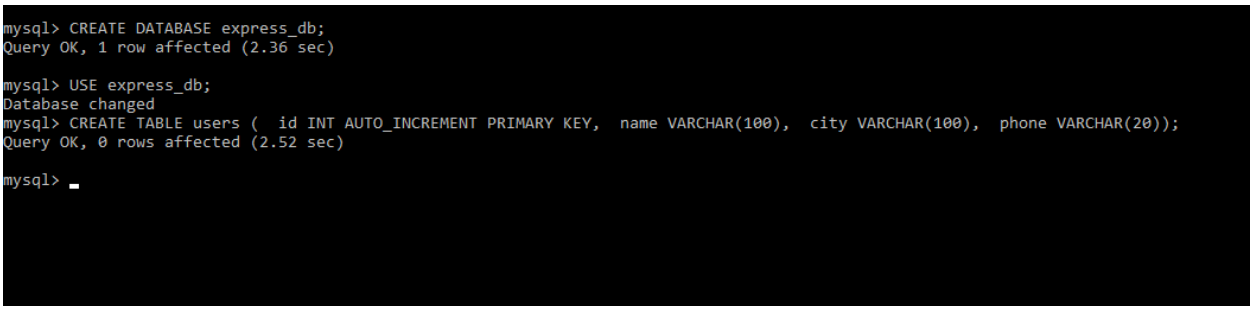
□ Step 2: Create MySQL Database & Table

Open MySQL and Run:-



And perform following task :-

```
CREATE DATABASE express_db;

USE express_db;

CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100),
  city VARCHAR(100),
  phone VARCHAR(20)
);
```



□ Step 3: Full Express + MySQL Code

Create file: `app.js:-`

```
const express = require('express');
const mysql = require('mysql2');

const app = express();
app.use(express.json());

// ------------------
// MySQL Connection
const db = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: '', // put your MySQL password
  database: 'express_db'
});

// Connect to DB
db.connect(err => {
  if (err) {
    console.log('Database connection failed:', err);
  } else {
    console.log('Connected to MySQL');
  }
});

// ------------------
// GET all users
app.get('/users', (req, res) => {
  db.query('SELECT * FROM users', (err, results) => {
    if (err) {
      res.status(500).json({ error: err.message });
    } else {
      res.json(results);
    }
  });
});

// ------------------
// GET single user
app.get('/users/:id', (req, res) => {
  const id = req.params.id;

  db.query('SELECT * FROM users WHERE id = ?', [id], (err, results) => {
    if (err) {
      res.status(500).json({ error: err.message });
    } else if (results.length === 0) {
      res.status(404).json({ error: "User not found" });
    } else {
      res.json(results[0]);
    }
  });
});
```

```
// ------------------
// CREATE user
app.post('/users', (req, res) => {
  const { name, city, phone } = req.body;

  const sql = 'INSERT INTO users (name, city, phone) VALUES (?, ?, ?)';
  db.query(sql, [name, city, phone], (err, result) => {
    if (err) {
      res.status(500).json({ error: err.message });
    } else {
      res.json({
        message: "User created",
        id: result.insertId
      });
    }
  });
});

// ------------------
// UPDATE user
app.put('/users/:id', (req, res) => {
  const id = req.params.id;
  const { name, city, phone } = req.body;

  const sql = 'UPDATE users SET name=?, city=?, phone=? WHERE id=?';
  db.query(sql, [name, city, phone, id], (err, result) => {
    if (err) {
      res.status(500).json({ error: err.message });
    } else {
      res.json({ message: "User updated" });
    }
  });
});

// ------------------
// DELETE user
app.delete('/users/:id', (req, res) => {
  const id = req.params.id;

  db.query('DELETE FROM users WHERE id=?', [id], (err, result) => {
    if (err) {
      res.status(500).json({ error: err.message });
    } else {
      res.json({ message: "User deleted" });
    }
  });
});

// ------------------
app.listen(3000, () => {
  console.log('Server running on http://localhost:3000');
});
```

□ Test with Windows CMD (IMPORTANT)

□ Create user

```
curl -X POST http://localhost:3000/users -H "Content-Type: application/json" -d "{\"name\":\"Alice\",\"city\":\"Mumbai\",\"phone\":\"1234567890\"}"
```

□ Get all users

```
curl http://localhost:3000/users
```

□ Get single user

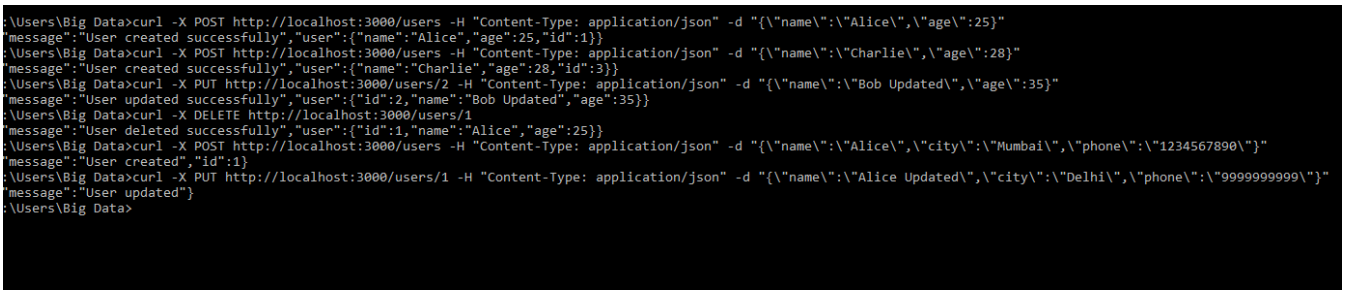
```
curl http://localhost:3000/users/1
```

□ Update user

```
curl -X PUT http://localhost:3000/users/1 -H "Content-Type: application/json" -d "{\"name\":\"Alice Updated\",\"city\":\"Delhi\",\"phone\":\"9999999999\"}"
```

□ Delete user

```
curl -X DELETE http://localhost:3000/users/1
```



Great! You're doing the right thing by understanding **line by line**. I'll explain this in **very simple beginner language**.

---

1. Import Libraries

```
const express = require('express');
```

- Loads **Express framework**.
- Used to create server and APIs.

```
const mysql = require('mysql2');
```

- Loads **MySQL library**.
- Used to connect Node.js with MySQL database.

---

2. Create App

```
const app = express();
```

- Creates your **server app**.
- You will define routes using `app`.

```
app.use(express.json());
```

- Middleware to **read JSON data from request body**.
- Required for POST/PUT.
- Without this → `req.body` will be `undefined`.

□ 3. MySQL Connection

```
const db = mysql.createConnection({
```

- Creates a **database connection object**.

```
  host: 'localhost',
```

- Database is running on your local machine.

```
  user: 'root',
```

- MySQL username (default is `root`).

```
  password: '',
```

- Your MySQL password (empty in your case).

```
  database: 'express_db'
```

- Name of your database.

```
});
```

- Ends connection configuration.

□ 4. Connect to Database

```
db.connect(err => {
```

- Tries to connect to MySQL.
- `err` will contain error if connection fails.

```
  if (err) {
    console.log('Database connection failed:', err);
  }
```

- If error → print error in console.

```
  else {
    console.log('Connected to MySQL');
  }
```

- If success → show success message.

```
});
```

- End of connection function.

□ 5. GET All Users

```
app.get('/users', (req, res) => {
```

- Handles GET request at `/users`.

```
  db.query('SELECT * FROM users', (err, results) => {
```

- Runs SQL query to fetch all users.

```
    if (err) {
      res.status(500).json({ error: err.message });
    }
```

- If error → send HTTP 500 (server error).

```
    else {
      res.json(results);
    }
```

- If success → send all users as JSON.

```
  });
});
```

- End of query and route.

6. GET Single User

```
app.get('/users/:id', (req, res) => {
```

- `:id` is dynamic URL parameter.

```
  const id = req.params.id;
```

- Extract `id` from URL.

```
  db.query('SELECT * FROM users WHERE id = ?', [id], (err, results) => {
```

- `?` is placeholder → prevents SQL injection.
- `[id]` replaces `?`.

```
    else if (results.length === 0) {
```

- If no user found.

```
      res.status(404).json({ error: "User not found" });
```

- Send 404 error.

```
    else {
      res.json(results[0]);
    }
```

- Send first user (only one result).

---

7. CREATE User (POST)

```
app.post('/users', (req, res) => {
```

- Handles POST request (create data).

```
const { name, city, phone } = req.body;
```

- Extract data from JSON body.

```
const sql = 'INSERT INTO users (name, city, phone) VALUES (?, ?, ?)';
```

- SQL query with placeholders.

```
db.query(sql, [name, city, phone], (err, result) => {
```

- Inserts values into DB.

```
    id: result.insertId
```

- Returns ID of newly inserted record.

□ 8. UPDATE User

```
app.put('/users/:id', (req, res) => {
```

- Handles update request.

```
const id = req.params.id;
```

- Get ID from URL.

```
const { name, city, phone } = req.body;
```

- Get updated data.

```
const sql = 'UPDATE users SET name=?, city=?, phone=? WHERE id=?';
```

- SQL update query.

```
db.query(sql, [name, city, phone, id], (err, result) => {
```

- Executes update.

```
  res.json({ message: "User updated" });
```

- Sends success response.

□ 9. DELETE User

```
app.delete('/users/:id', (req, res) => {
```

- Handles delete request.

```
  const id = req.params.id;
```

- Get ID.

```
  db.query('DELETE FROM users WHERE id=?', [id], (err, result) => {
```

- Deletes user from database.

```
  res.json({ message: "User deleted" });
```

- Sends confirmation.

□ 10. Start Server

```
app.listen(3000, () => {
```

- Starts server on port **3000**.

```
  console.log('Server running on http://localhost:3000');
```

- Prints message in console.

```
});
```

- End server start.

Simple Summary (Very Important)

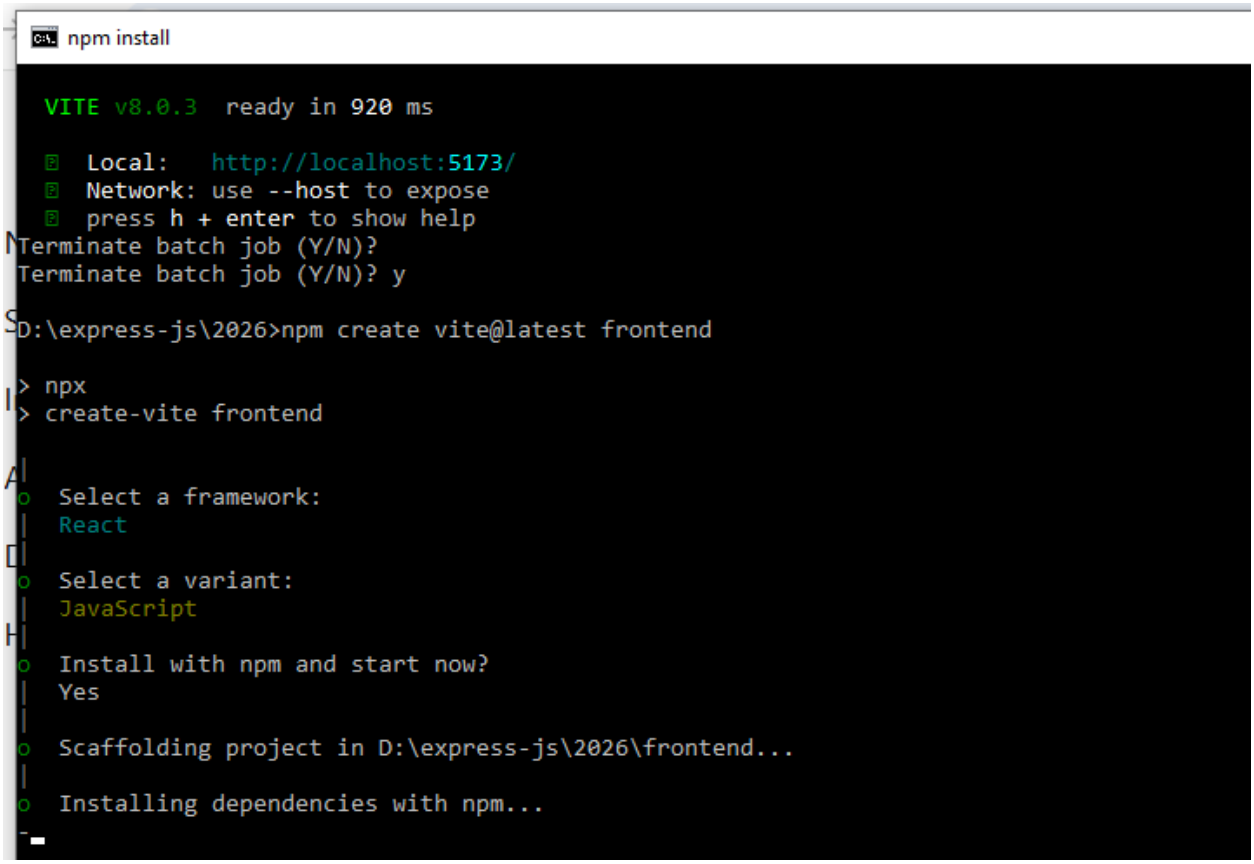
- `express()` → create server
- `app.get()` → read data
- `app.post()` → insert data
- `app.put()` → update data
- `app.delete()` → delete data
- `req.params` → URL data
- `req.body` → JSON data
- `db.query()` → run SQL
- `?` → safe values (prevents SQL injection)

Here’s a clean **React + Vite frontend** that connects to your Express + MySQL backend (CRUD for users).

---

1. Create Vite React App

```
npm create vite@latest frontend
```




```
cd frontend
```

and install axios :-

```
npm install axios
```

```
npm run dev
```

---

2. Folder Structure

```
frontend/
├── src/
│   ├── App.jsx
│   ├── api.js
│   ├── components/
│   │   ├── UserForm.jsx
│   │   └── UserList.jsx
│   └── main.jsx
```

---

3. API File (src/api.js)

```
import axios from "axios";

const API = axios.create({
  baseURL: "http://localhost:3000"
});

export const getUsers = () => API.get("/users");
export const createUser = (data) => API.post("/users", data);
export const updateUser = (id, data) => API.put(`/users/${id}`, data);
export const deleteUser = (id) => API.delete(`/users/${id}`);
```

---

4. User Form Component

src/components/UserForm.jsx

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

const UserForm = ({ onSubmit, selectedUser }) => {
  const [form, setForm] = useState({
    name: "",
    city: "",
    phone: ""
  });

  useEffect(() => {
    if (selectedUser) {
      setForm(selectedUser);
    }
  }, [selectedUser]);

  const handleChange = (e) => {
    setForm({ ...form, [e.target.name]: e.target.value });
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    onSubmit(form);
    setForm({ name: "", city: "", phone: "" });
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        name="name"
        placeholder="Name"
        value={form.name}
        onChange={handleChange}
        required
      />
      <input
        name="city"
        placeholder="City"
        value={form.city}
        onChange={handleChange}
        required
      />
      <input
        name="phone"
        placeholder="Phone"
        value={form.phone}
        onChange={handleChange}
        required
      />
      <button type="submit">
        {selectedUser ? "Update" : "Add"} User
      </button>
```

```
    </form>
  );
};

export default UserForm;
```

---

5. User List Component

src/components/UserList.jsx

```
const UserList = ({ users, onEdit, onDelete }) => {
  return (
    <div>
      <h2>User List</h2>
      {users.map((user) => (
        <div key={user.id} style={{ marginBottom: "10px" }}>
          <strong>{user.name}</strong> - {user.city} - {user.phone}
          <br />
          <button onClick={() => onEdit(user)}>Edit</button>
          <button onClick={() => onDelete(user.id)}>Delete</button>
        </div>
      ))}
    </div>
  );
};

export default UserList;
```

---

6. Main App Component

src/App.jsx

```
import { useEffect, useState } from "react";
import {
  getUsers,
  createUser,
  updateUser,
  deleteUser
} from "./api";

import UserForm from "./components/UserForm";
import UserList from "./components/UserList";

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

  const fetchUsers = async () => {
    const res = await getUsers();
    setUsers(res.data);
  };

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

  const handleSubmit = async (data) => {
    if (selectedUser) {
      await updateUser(selectedUser.id, data);
      setSelectedUser(null);
    } else {
      await createUser(data);
    }
    fetchUsers();
  };

  const handleDelete = async (id) => {
    await deleteUser(id);
    fetchUsers();
  };

  const handleEdit = (user) => {
    setSelectedUser(user);
  };

  return (
    <div style={{ padding: "20px" }}>
      <h1>CRUD App</h1>

      <UserForm onSubmit={handleSubmit} selectedUser={selectedUser} />

      <UserList
        users={users}
```

```
        onEdit={handleEdit}
        onDelete={handleDelete}
      />
    </div>
  );
}

export default App;
```

---

7. main.jsx (default Vite)

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";

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

---

8. Important Fix (CORS)

Your backend needs CORS enabled:

```
npm install cors
```

Add this in Express app.js file :-

```
const cors = require('cors');

app.use(cors({
  origin: 'http://localhost:5173'
}));
```

Done!

You now have:

- Fetch users
- Add user
- Update user
- Delete user

Your Output:-

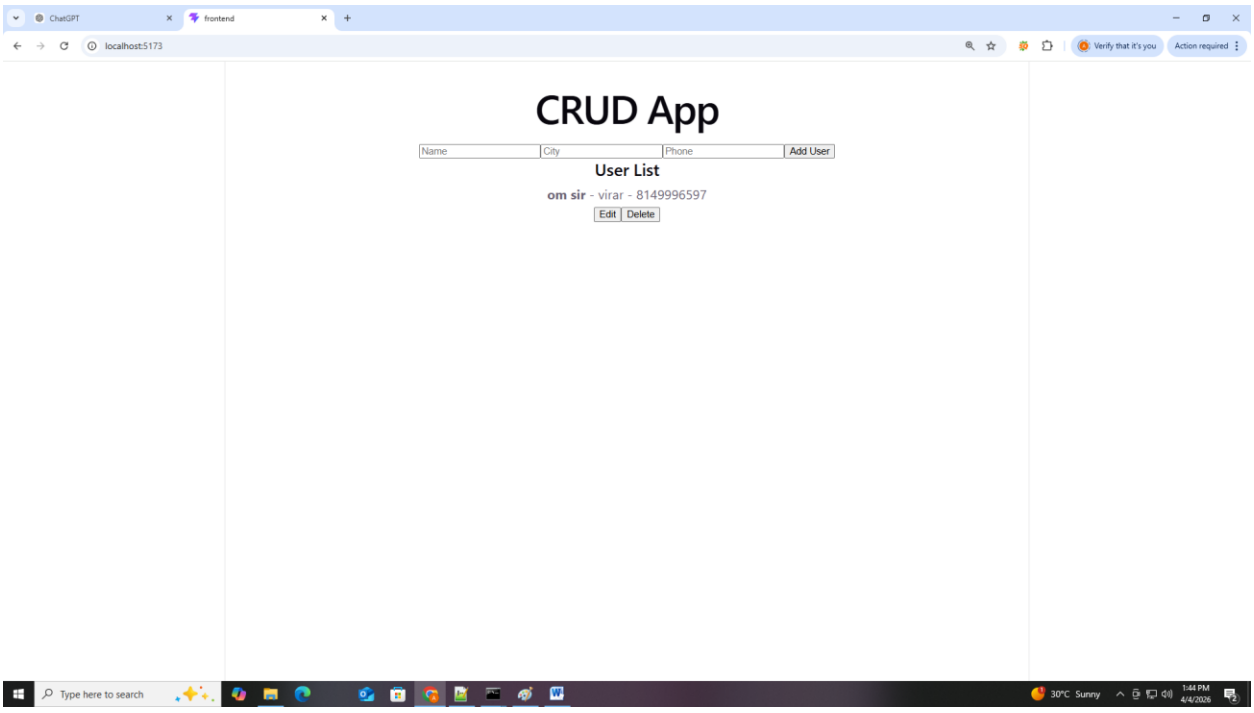
First run your backend:-

```
D:\express-js\2026\express-warmup>node app.js
Server running on http://localhost:3000
Connected to MySQL
```

And then run your frontend code:-

```
D:\express-js\2026\frontend>npm run dev

> frontend@0.0.0 dev
> vite

1:39:41 PM [vite] (client) Re-optimizing dependencies because lockfile has changed

  VITE v8.0.3  ready in 402 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

Output:-


ChatGPT
frontend
localhost:5173
Verify that it's you
Action required
CRUD App
Name
City
Phone
Add User
User List
om sir - virar - 8149996597
Edit
Delete
Type here to search
30°C Sunny
1:44 PM
4/4/2026

let's go **line-by-line** through your `App.jsx` so you fully understand what every part does.

□ `App.jsx` Explained

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

□ Imports React hooks:

- `useState` → stores data (like users, selected user)
- `useEffect` → runs code when component loads or updates

```
import {
  getUsers,
  createUser,
  updateUser,
  deleteUser
} from "./api";
```

□ Imports API functions you created:

- `getUsers()` → fetch all users
- `createUser()` → add new user
- `updateUser()` → update user
- `deleteUser()` → delete user

```
import UserForm from "./components/UserForm";
import UserList from "./components/UserList";
```

□ Imports child components:

- `UserForm` → form to add/update user
- `UserList` → display all users

```
function App() {
```

□ Main React component (root of your UI)

```
  const [users, setUsers] = useState([]);
```

□ State to store all users

- `users` → current data
- `setUsers` → function to update it
- Starts as empty array `[]`

```
const [selectedUser, setSelectedUser] = useState(null);
```

 Stores user being edited

- `null` → means no user selected initially

```
const fetchUsers = async () => {
```

 Function to get users from backend

```
  const res = await getUsers();
```

 Calls API → GET `/users`

- `res.data` will contain users from database

```
  setUsers(res.data);
```

 Updates state with fetched users → triggers UI re-render

```
};
```

 Ends function

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

 Runs when component loads (like "on page load")

- `[]` → empty dependency array = run only once
- Calls `fetchUsers()` → loads data initially

```
const handleSubmit = async (data) => {
```

 Handles form submission (add/update)

```
    if (selectedUser) {
```

 If a user is selected → we are editing

```
      await updateUser(selectedUser.id, data);
```

 Calls PUT `/users/:id` to update user

```
      setSelectedUser(null);
```

 Clears selection after update

```
    } else {
      await createUser(data);
    }
```

 If no user selected → create new user
Calls POST `/users`

```
    fetchUsers();
```

 Refresh user list after add/update

```
  };
```

 Ends function

```
  const handleDelete = async (id) => {
```

 Function to delete a user

```
    await deleteUser(id);
```

 Calls DELETE `/users/:id`

```
    fetchUsers();
```

 Reloads updated list after deletion

```
};
```

 Ends function

```
const handleEdit = (user) => {
```

 Function triggered when clicking "Edit"

```
    setSelectedUser(user);
```

 Stores selected user → passed to form for editing

```
};
```

 Ends function

UI Rendering Part

```
return (
  <div style={{ padding: "20px" }}>
```

 Main container with padding

```
      <h1>CRUD App</h1>
```

 Title of app

```
      <UserForm onSubmit={handleSubmit} selectedUser={selectedUser} />
```

 Renders form component
Passes:

- `onSubmit` → function to call when form submits
- `selectedUser` → data to pre-fill form (for editing)

```
      <UserList
        users={users}
        onEdit={handleEdit}
```

```
        onDelete={handleDelete}
      />
```

 Renders user list
Passes:

- `users` → list of users
- `onEdit` → edit handler
- `onDelete` → delete handler

```
    </div>
  );
```

 Ends UI

```
}
```

 Ends component

```
export default App;
```

 Makes component available to use in `main.jsx`

Big Picture Flow

1. Page loads → `useEffect` runs → fetch users
2. Users stored in state → displayed in UI
3. Add/Edit form → calls `handleSubmit`
4. Delete button → calls `handleDelete`
5. Edit button → sets `selectedUser` → form auto-fills

Simple version of React js frontend all in One inside App.jsx file code:-

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

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

  // Form state
  const [form, setForm] = useState({
    name: "",
    city: "",
    phone: ""
  });

  // Fetch users
  const fetchUsers = async () => {
    const res = await axios.get("http://localhost:3000/users");
    setUsers(res.data);
  };

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

```
// Auto-fill form whenever selectedUser changes
useEffect(() => {
  if (selectedUser) {
    setForm({
      name: selectedUser.name,
      city: selectedUser.city,
      phone: selectedUser.phone
    });
  } else {
    // Clear form when no selectedUser
    setForm({ name: "", city: "", phone: "" });
  }
}, [selectedUser]);

// Handle input change
const handleChange = (e) => {
  setForm({
    ...form,
    [e.target.name]: e.target.value
  });
};
```

```
// Add or Update user
const handleSubmit = async (e) => {
  e.preventDefault();

  if (selectedUser) {
    // UPDATE
    await axios.put(
      `http://localhost:3000/users/${selectedUser.id}`,
      form
    );
    setSelectedUser(null); // Clear selection after update
  } else {
    // CREATE
    await axios.post("http://localhost:3000/users", form);
  }

  fetchUsers(); // Refresh list
};

// Delete user
const handleDelete = async (id) => {
  await axios.delete(`http://localhost:3000/users/${id}`);
  fetchUsers();
};
```

```
// Edit user
const handleEdit = (user) => {
  setSelectedUser(user); // Form will auto-fill via useEffect
};


return (
  <div style={{ padding: "20px" }}>
    <h1>CRUD App</h1>


    {/* FORM */}
    <form onSubmit={handleSubmit}>
      <input
        name="name"
        placeholder="Name"
        value={form.name}
        onChange={handleChange}
      />
      <input
        name="city"
        placeholder="City"
        value={form.city}
        onChange={handleChange}
      />
      <input
        name="phone"
```

```
        placeholder="Phone"
        value={form.phone}
        onChange={handleChange}
      />
      <button type="submit">
        {selectedUser ? "Update" : "Add"}
      </button>
    </form>

    <hr />

    {/* USER LIST */}
    {users.map((user) => (
      <div key={user.id}>
        {user.name} - {user.city} - {user.phone}
        <button onClick={() => handleEdit(user)}>Edit</button>
        <button onClick={() => handleDelete(user.id)}>Delete</button>
      </div>
    ))}
  </div>
 );
}

export default App;
```

Let's go **line by line** and explain everything in your `App.jsx` code, in a beginner-friendly way. I'll include both **what the line does** and **why it's needed**.

---

1. Import React hooks and axios

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

- `useState` → allows you to create **state variables** in functional components (like `users` or `form`).
- `useEffect` → allows you to run **side effects** (like fetching data when the component loads).
- `axios` → a library to make HTTP requests to your backend (GET, POST, PUT, DELETE).

---

2. Start functional component

```
function App() {
```

- Defines the main React component. Everything inside this function is the **logic and UI** of your app.

---

3. Users state

```
  const [users, setUsers] = useState([]);
```

- `users` → stores the list of users fetched from the backend.
- `setUsers` → function to update the `users` state.
- `useState([])` → initializes it as an empty array.

---

4. Selected user state

```
  const [selectedUser, setSelectedUser] = useState(null);
```

- `selectedUser` → stores the **user currently being edited**.
- `setSelectedUser` → updates it.
- Initially `null` → no user is selected.

---

5. Form state

```
  const [form, setForm] = useState({
    name: "",
```

```
  city: "",
  phone: ""
});
```

- `form` → stores **current values in the input fields**.
- `setForm` → updates form values.
- Initially empty strings → no input.

---

6. Fetch users function

```
const fetchUsers = async () => {
  const res = await axios.get("http://localhost:3000/users");
  setUsers(res.data);
};
```

- `fetchUsers` → gets all users from backend.
- `axios.get("http://localhost:3000/users")` → sends GET request.
- `res.data` → contains the array of users from backend.
- `setUsers(res.data)` → updates the `users` state with backend data.

---

7. Fetch users on component mount

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

- Runs `fetchUsers()` **once when component first loads**.
- `[]` → dependency array. Empty array = run only once.

---

8. Auto-fill form when editing

```
useEffect(() => {
  if (selectedUser) {
    setForm({
      name: selectedUser.name,
      city: selectedUser.city,
      phone: selectedUser.phone
    });
  } else {
    setForm({ name: "", city: "", phone: "" });
  }
}, [selectedUser]);
```

- Runs every time `selectedUser` changes.
- If a user is selected → fill the form with their data.
- If no user is selected → clear the form.

9. Handle input changes

```
const handleChange = (e) => {
  setForm({
    ...form,
    [e.target.name]: e.target.value
  });
};
```

- `e` → event from input fields.
- `e.target.name` → name of the input (`name`, `city`, or `phone`).
- `e.target.value` → current value typed by the user.
- Updates only the input that changed, keeping others intact (`...form`).

---

10. Add or update user

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

- Prevents the page from **reloading** on form submit.

```
  if (selectedUser) {
    await axios.put(
      `http://localhost:3000/users/${selectedUser.id}`,
      form
    );
    setSelectedUser(null);
  } else {
    await axios.post("http://localhost:3000/users", form);
  }
```

- If a user is selected → send **PUT request** to update user.
- If no user selected → send **POST request** to create new user.
- `setSelectedUser(null)` → clears selected user after update.

```
  fetchUsers();
};
```

- Refresh the users list after adding or updating.

---

11. Delete user

```
const handleDelete = async (id) => {
  await axios.delete(`http://localhost:3000/users/${id}`);
  fetchUsers();
};
```

- Sends **DELETE request** to backend with user ID.
- Refreshes the list after deletion.

---

12. Edit user

```
  const handleEdit = (user) => {
    setSelectedUser(user);
  };
```

- Sets the user to `selectedUser`.
- The form auto-fills because of the **useEffect** watching `selectedUser`.

---

13. Render JSX

```
  return (
    <div style={{ padding: "20px" }}>
      <h1>CRUD App</h1>
```

- Wraps the app content and gives padding.

---

14. Form JSX

```
      <form onSubmit={handleSubmit}>
        <input name="name" placeholder="Name" value={form.name}
onChange={handleChange} />
        <input name="city" placeholder="City" value={form.city}
onChange={handleChange} />
        <input name="phone" placeholder="Phone" value={form.phone}
onChange={handleChange} />
        <button type="submit">{selectedUser ? "Update" : "Add"}</button>
      </form>
```

- Controlled inputs → value comes from `form`.
- `onChange` updates `form` state.
- Button text changes based on whether editing or adding.

---

15. List of users

```
<hr />
{users.map((user) => (
  <div key={user.id}>
    {user.name} - {user.city} - {user.phone}
    <button onClick={() => handleEdit(user)}>Edit</button>
    <button onClick={() => handleDelete(user.id)}>Delete</button>
  </div>
))}
```

- Maps over all users → displays name, city, phone.
- Each user has **Edit** and **Delete** buttons with correct handlers.

---

16. Export component

```
export default App;
```

- Makes the component usable in `main.jsx` (entry point).

□ Summary of Key Concepts

1. **State management** → `users`, `form`, `selectedUser`.
2. **Side effects** → fetching data (`useEffect`).
3. **Controlled inputs** → form values come from state.
4. **CRUD operations** → using axios (`GET`, `POST`, `PUT`, `DELETE`).
5. **Auto-fill form** → triggered by `selectedUser` changes.