Java **full-stack app**:-

- Spring Boot + MongoDB backend
- React (Vite) frontend
- CRUD operations (Add, Edit, Delete, View)

Project Breakdown

Backend

- Spring Boot API ✔
- MongoDB connection ✔
- CRUD operations ✔

This is **backend development in Java**

Frontend

- React (Vite) ✔
- API integration ✔
- Form handling ✔

This is **frontend development**

Download Java JDK

1. Go to the official Adoptium Temurin JDK website.(https://adoptium.net/)
2. Select **Windows x64** and **JDK 17 or higher** (Spring Boot 3 requires at least Java 17).
3. Click **Latest release → MSI installer** (easy for Windows).
4. Download the installer.

Step 2: Install Java

1. Run the downloaded `.msi` file.
2. Keep default options.
3. Make sure the installer **adds Java to your PATH** (checkbox).

Step 3: Verify Java installation

Open **Command Prompt** and run:

```
java -version
```

Expected output (example):

```
java version "17.0.8" 2023-07-18 LTS
```

Step 1: Create a Spring Boot Project

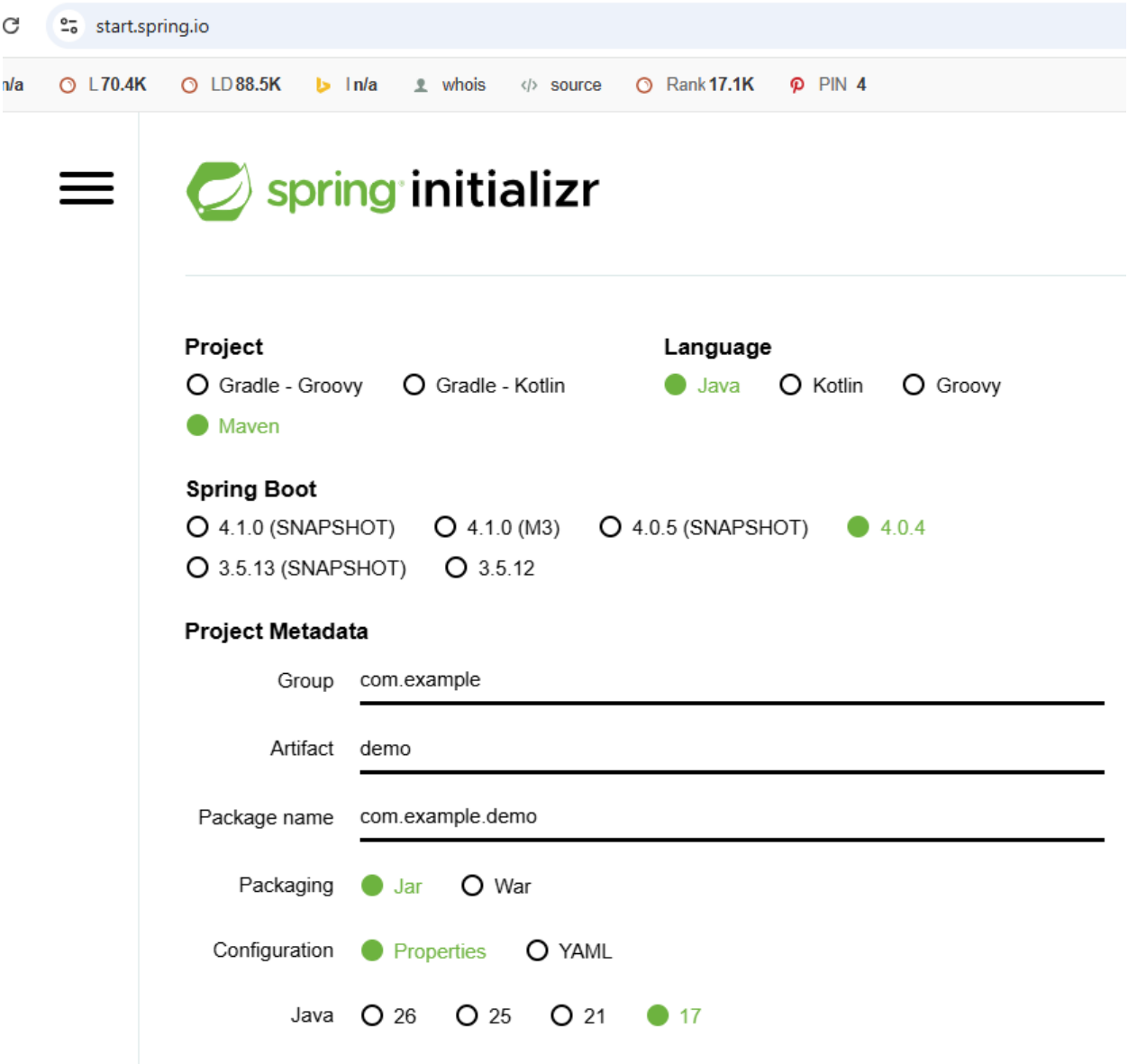
You can use Spring Initializr (https://start.spring.io/) or your IDE (like IntelliJ IDEA) to generate the project:

Settings:

- Project: **Maven**

- Language: **Java**
- Spring Boot: Latest stable (e.g., 3.2.x)
- Dependencies:
    - **Spring Web** (for REST endpoints)
    - **Spring Data MongoDB** (for MongoDB integration)
    - **Lombok** (optional, for getters/setters)
    - **Spring Boot DevTools** (optional, for hot reload)

Download the project and open it in your IDE



And

Dependencies ADD DEPENDENCIES... CTRL + B

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Data MongoDB NOSQL
Spring Data support for MongoDB.

Spring Boot DevTools DEVELOPER TOOLS
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Lombok DEVELOPER TOOLS
Java annotation library which helps to reduce boilerplate code.

Let’s create a **ready-to-run full setup** for your React frontend and Spring Boot backend with MongoDB, including CORS and sample data. Here’s the complete guide.

---

1 Spring Boot Backend Setup

Project structure:

```
demo/
├─ src/main/java/com/example/demo/
│   ├─ DemoApplication.java
│   ├─ model/Contact.java
│   ├─ repository/ContactRepository.java
│   └─ controller/ContactController.java
│   └─ config/WebConfig.java
├─ pom.xml
```

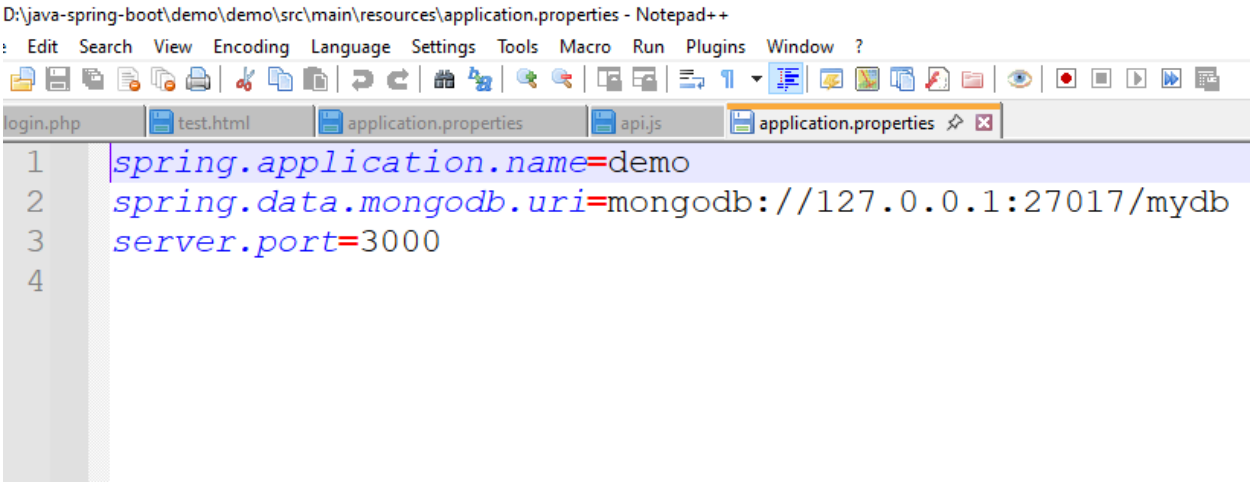
1a. pom.xml

Include Spring Boot + MongoDB + DevTools:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" ...>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.2</version>
  </parent>

  <dependencies>
    <!-- Web + REST -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- MongoDB -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-mongodb</artifactId>
    </dependency>

    <!-- DevTools for hot reload -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
```

```
      <scope>runtime</scope>
    </dependency>
  </dependencies>
</project>
```

D:\java-spring-boot\demo\demo\src\main\resources\application.properties



1b. DemoApplication.java

```
package com.example.demo;

import com.example.demo.model.Contact;
import com.example.demo.repository.ContactRepository;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

    // Seed sample data
    @Bean
    CommandLineRunner initData(ContactRepository repository) {
        return args -> {
            repository.deleteAll(); // optional: clear previous
            repository.save(new Contact("Alice", "New York", "1234567890"));
            repository.save(new Contact("Bob", "Los Angeles", "9876543210"));
        };
    }
}
```

Create Contact Model

src/main/java/com/example/demo/model/Contact.java

1c. Contact.java (Model)

```
package com.example.demo.model;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "contacts")
public class Contact {
    @Id
    private String id;
    private String name;
    private String city;
    private String phone;

    public Contact() {}
    public Contact(String name, String city, String phone) {
        this.name = name;
        this.city = city;
        this.phone = phone;
    }

    // Getters & setters
    public String getId() { return id; }
    public void setId(String id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getCity() { return city; }
    public void setCity(String city) { this.city = city; }

    public String getPhone() { return phone; }
    public void setPhone(String phone) { this.phone = phone; }
}
```

Create Repository

src/main/java/com/example/demo/repository/ContactRepository.java

1d. ContactRepository.java

```
package com.example.demo.repository;

import com.example.demo.model.Contact;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface ContactRepository extends MongoRepository<Contact, String>
{}
```

Create Controller

src/main/java/com/example/demo/controller/ContactController.java

1e. ContactController.java:-

```
package com.example.demo.controller;

import com.example.demo.model.Contact;
import com.example.demo.repository.ContactRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/contacts")
@CrossOrigin(origins = "http://localhost:5173") // allow React
public class ContactController {

    @Autowired
    private ContactRepository repository;

    @GetMapping
    public List<Contact> getAllContacts() {
```

```
        return repository.findAll();
    }

    @GetMapping("/{id}")
    public Contact getContactById(@PathVariable String id) {
        return repository.findById(id).orElse(null);
    }

    @PostMapping
    public Contact createContact(@RequestBody Contact contact) {
        return repository.save(contact);
    }

    @PutMapping("/{id}")
    public Contact updateContact(@PathVariable String id, @RequestBody
Contact updatedContact) {
        return repository.findById(id).map(contact -> {
            contact.setName(updatedContact.getName());
            contact.setCity(updatedContact.getCity());
            contact.setPhone(updatedContact.getPhone());
            return repository.save(contact);
        }).orElse(null);
    }

    @DeleteMapping("/{id}")
    public String deleteContact(@PathVariable String id) {
        repository.deleteById(id);
        return "Deleted";
    }
}
```

1f. Optional: Global CORS Config(demo/config/WebConfig.java)

```
package com.example.demo.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class WebConfig {
    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                registry.addMapping("/**")
                        .allowedOrigins("http://localhost:5173")
                        .allowedMethods("GET", "POST", "PUT", "DELETE");
            }
        };
    }
}
```

1g. src/main/resources/application.properties

```
spring.application.name=demo

spring.data.mongodb.uri=mongodb://127.0.0.1:27017/mydb

server.port=3000
```

Run the project:-

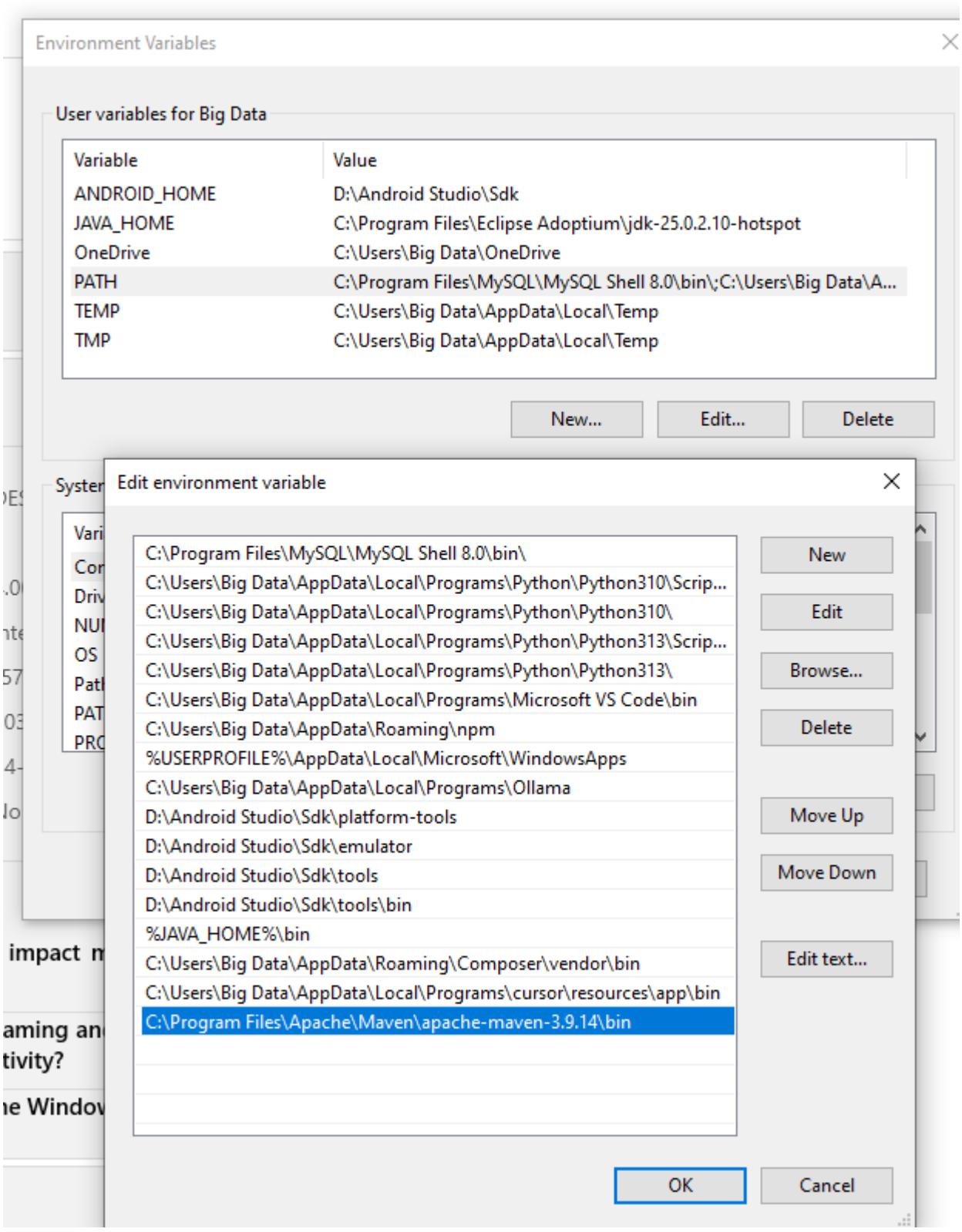
(download maven form https://maven.apache.org/download.cgi)

Download

Binary zip archive

apache-maven-3.9.14-bin.zip

and set into path in your windows pc :-


Environment Variables
User variables for Big Data
Variable
Value
ANDROID_HOME
D:\Android Studio\Sdk
JAVA_HOME
C:\Program Files\Eclipse Adoptium\jdk-25.0.2.10-hotspot
OneDrive
C:\Users\Big Data\OneDrive
PATH
C:\Program Files\MySQL\MySQL Shell 8.0\bin\;C:\Users\Big Data\A...
TEMP
C:\Users\Big Data\AppData\Local\Temp
TMP
C:\Users\Big Data\AppData\Local\Temp
New...
Edit...
Delete
Edit environment variable
C:\Program Files\MySQL\MySQL Shell 8.0\bin\
C:\Users\Big Data\AppData\Local\Programs\Python\Python310\Scrip...
C:\Users\Big Data\AppData\Local\Programs\Python\Python310\
C:\Users\Big Data\AppData\Local\Programs\Python\Python313\Scrip...
C:\Users\Big Data\AppData\Local\Programs\Python\Python313\
C:\Users\Big Data\AppData\Local\Programs\Microsoft VS Code\bin
C:\Users\Big Data\AppData\Roaming\npm
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Users\Big Data\AppData\Local\Programs\Ollama
D:\Android Studio\Sdk\platform-tools
D:\Android Studio\Sdk\emulator
D:\Android Studio\Sdk\tools
D:\Android Studio\Sdk\tools\bin
%JAVA_HOME%\bin
C:\Users\Big Data\AppData\Roaming\Composer\vendor\bin
C:\Users\Big Data\AppData\Local\Programs\cursor\resources\app\bin
C:\Program Files\Apache\Maven\apache-maven-3.9.14\bin
New
Edit
Browse...
Delete
Move Up
Move Down
Edit text...
OK
Cancel

Option 1: Using Maven directly

```
mvn spring-boot:run
```

```
example :-
```

```
D:\java-spring-boot\demo\demo>mvn spring-boot:run
```

Check if backend is running

Open your browser or Postman and go to:

```
http://localhost:3000/contacts/
```

2 React Frontend Setup (Vite)

Directory structure:

```
react-frontend/
├─ src/
│  ├─ api.js
│  ├─ App.jsx
│  ├─ ContactList.jsx
│  └─ ContactForm.jsx
├─ index.html
├─ package.json
```

2a. api.js

```
import axios from "axios";


const api = axios.create({

  baseURL: "http://localhost:3000/contacts",

  headers: {

    "Content-Type": "application/json",

  },

});


export default api;
```

2b. ContactList.jsx

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

import api from "./api";


function ContactList({ onEdit, onDelete, refresh }) {

  const [contacts, setContacts] = useState([]);


  useEffect(() => {

    fetchContacts();

  }, [refresh]);
```

```
  const fetchContacts = async () => {
    try {
      const res = await api.get(""); //  FIXED (removed "/")
      setContacts(res.data);
    } catch (err) {
      console.error("Error fetching contacts:", err);
    }
  };

  return (
    <div>
      <h2>Contacts</h2>
      <ul>
        {contacts.map((c) => (
          <li key={c.id}>
            {c.name} - {c.city} - {c.phone}{" "}
            <button onClick={() => onEdit(c)}>Edit</button>
            <button onClick={() => onDelete(c.id)}>Delete</button>
          </li>
        ))}
      </ul>
    </div>
  );
}

export default ContactList;
```

2c. ContactForm.jsx

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

function ContactForm({ selectedContact, onSaved }) {
  const [contact, setContact] = useState({
    name: "",
    city: "",
    phone: "",
  });

  useEffect(() => {
    if (selectedContact) {
      setContact(selectedContact);
    } else {
      setContact({ name: "", city: "", phone: "" });
    }
  }, [selectedContact]);

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

```
};

const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    if (contact.id) {
      // □ Update
      await api.put(`/${contact.id}`, contact);
    } else {
      // □ Create (FIXED here)
      await api.post("", contact);
    }

    // Reset form
    setContact({ name: "", city: "", phone: "" });

    // Refresh list
    onSaved();
  } catch (err) {
    console.error("Error saving contact:", err);
  }
};

return (
  <div>
    <h2>{contact.id ? "Edit Contact" : "Add Contact"}</h2>
    <form onSubmit={handleSubmit}>
      <input
```

```
          name="name"
          placeholder="Name"
          value={contact.name}
          onChange={handleChange}
          required
        />
        <input
          name="city"
          placeholder="City"
          value={contact.city}
          onChange={handleChange}
          required
        />
        <input
          name="phone"
          placeholder="Phone"
          value={contact.phone}
          onChange={handleChange}
          required
        />
        <button type="submit">Save</button>
      </form>
    </div>
  );
}

export default ContactForm;
```

2d. App.jsx

```
import React, { useState } from "react";
import ContactList from "./ContactList";
import ContactForm from "./ContactForm";
import api from "./api";

function App() {
  const [selectedContact, setSelectedContact] = useState(null);
  const [refresh, setRefresh] = useState(false);

  const handleEdit = (contact) => setSelectedContact(contact);

  const handleDelete = async (id) => {
    try {
      await api.delete(`/${id}`);
      setRefresh(!refresh);
    } catch (err) {
      console.error(err);
    }
  };

  const handleSaved = () => {
    setSelectedContact(null);
    setRefresh(!refresh);
  };

  return (
    <div>
      <h1>Contact Manager</h1>
      <ContactForm selectedContact={selectedContact} onSaved={handleSaved} />
      <ContactList
        onEdit={handleEdit}
        onDelete={handleDelete}
        refresh={refresh}
      />
    </div>
  );
}

export default App;
```

---

3 Run Everything

Backend

```
cd demo
mvn spring-boot:run
```

- Runs on **http://localhost:3000**
- MongoDB must be running locally

Frontend

```
cd react-frontend
npm install
npm run dev
```

- Runs on **http://localhost:5173**

 Now you have a**full stack Contact Manager**:

- Add / Edit / Delete / List contacts
- React frontend talks to Spring Boot backend
- CORS is configured
- Sample data preloaded

If you want, I can also **give you a ZIP structure ready to unzip and run**, so you don't need to create files manually.