Frontend Code (ReactJS)

Install Required Dependencies
npm install @mui/material @emotion/react @emotion/styled axios react-router-dom

App.js

```
import React, { useState, useEffect } from "react";
import { TextField, Button, Table, TableBody, TableCell, TableContainer, TableHead,
TableRow, Paper, Pagination } from "@mui/material";
import axios from "axios";

const App = () => {
  const [contacts, setContacts] = useState([]);
  const [formData, setFormData] = useState({
    firstName: "",
    lastName: "",
    email: "",
    phone: "",
    company: "",
    jobTitle: "",
  });
  const [editId, setEditId] = useState(null);

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

  const fetchContacts = async () => {
    try {
      const response = await axios.get("http://localhost:5000/contacts");
      setContacts(response.data);
    } catch (error) {
      console.error("Error fetching contacts:", error);
    }
  };

  const handleInputChange = (e) => {
    const { name, value } = e.target;
    setFormData({ ...formData, [name]: value });
  };

  const handleSubmit = async () => {
    try {
      if (editId) {
        await axios.put(`http://localhost:5000/contacts/${editId}`, formData);
      } else {
        await axios.post("http://localhost:5000/contacts", formData);
      }
      setFormData({ firstName: "", lastName: "", email: "", phone: "", company: "",
```

```
jobTitle: "" });
      setEditId(null);
      fetchContacts();
    } catch (error) {
      console.error("Error saving contact:", error);
    }
  };

  const handleEdit = (contact) => {
    setEditId(contact.id);
    setFormData(contact);
  };

  const handleDelete = async (id) => {
    try {
      await axios.delete(`http://localhost:5000/contacts/${id}`);
      fetchContacts();
    } catch (error) {
      console.error("Error deleting contact:", error);
    }
  };

  return (
    <div>
      <h1>Contact Management</h1>
      <form onSubmit={(e) => e.preventDefault()}>
        <TextField name="firstName" label="First Name" value={formData.firstName}
onChange={handleInputChange} required />
        <TextField name="lastName" label="Last Name" value={formData.lastName}
onChange={handleInputChange} required />
        <TextField name="email" label="Email" value={formData.email}
onChange={handleInputChange} required />
        <TextField name="phone" label="Phone Number" value={formData.phone}
onChange={handleInputChange} required />
        <TextField name="company" label="Company" value={formData.company}
onChange={handleInputChange} />
        <TextField name="jobTitle" label="Job Title" value={formData.jobTitle}
onChange={handleInputChange} />
        <Button onClick={handleSubmit}>{editId ? "Update" : "Add"} Contact</Button>
      </form>
      <TableContainer component={Paper}>
        <Table>
          <TableHead>
            <TableRow>
              <TableCell>First Name</TableCell>
              <TableCell>Last Name</TableCell>
              <TableCell>Email</TableCell>
              <TableCell>Phone</TableCell>
              <TableCell>Company</TableCell>
              <TableCell>Job Title</TableCell>
```

```
              <TableCell>Actions</TableCell>
            </TableRow>
          </TableHead>
          <TableBody>
            {contacts.map((contact) => (
              <TableRow key={contact.id}>
                <TableCell>{contact.firstName}</TableCell>
                <TableCell>{contact.lastName}</TableCell>
                <TableCell>{contact.email}</TableCell>
                <TableCell>{contact.phone}</TableCell>
                <TableCell>{contact.company}</TableCell>
                <TableCell>{contact.jobTitle}</TableCell>
                <TableCell>
                  <Button onClick={() => handleEdit(contact)}>Edit</Button>
                  <Button onClick={() => handleDelete(contact.id)}>Delete</Button>
                </TableCell>
              </TableRow>
            ))}
          </TableBody>
        </Table>
      </TableContainer>
    </div>
  );
};

export default App;
```

Below is a complete implementation of the Contact Management System in a modular way using ReactJS for the frontend, Node.js for the backend, and MySQL for the database.

Frontend Code (ReactJS)
First, create a React app using npx create-react-app contact-management.

1. Install Required Dependencies

```
npm install @mui/material @emotion/react @emotion/styled axios react-router-dom
```

2. Create the App.js Component

```
import React, { useState, useEffect } from "react";
import { TextField, Button, Table, TableBody, TableCell, TableContainer, TableHead,
TableRow, Paper, Pagination } from "@mui/material";
import axios from "axios";

const App = () => {
  const [contacts, setContacts] = useState([]);
  const [formData, setFormData] = useState({
    firstName: "",
    lastName: "",
    email: "",
    phone: "",
```

```
    company: "",
    jobTitle: "",
  });
  const [editId, setEditId] = useState(null);

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

  const fetchContacts = async () => {
    try {
      const response = await axios.get("http://localhost:5000/contacts");
      setContacts(response.data);
    } catch (error) {
      console.error("Error fetching contacts:", error);
    }
  };

  const handleInputChange = (e) => {
    const { name, value } = e.target;
    setFormData({ ...formData, [name]: value });
  };

  const handleSubmit = async () => {
    try {
      if (editId) {
        await axios.put(`http://localhost:5000/contacts/${editId}`, formData);
      } else {
        await axios.post("http://localhost:5000/contacts", formData);
      }
      setFormData({ firstName: "", lastName: "", email: "", phone: "", company: "",
jobTitle: "" });
      setEditId(null);
      fetchContacts();
    } catch (error) {
      console.error("Error saving contact:", error);
    }
  };

  const handleEdit = (contact) => {
    setEditId(contact.id);
    setFormData(contact);
  };

  const handleDelete = async (id) => {
    try {
      await axios.delete(`http://localhost:5000/contacts/${id}`);
      fetchContacts();
    } catch (error) {
      console.error("Error deleting contact:", error);
```

```
    }
  };

  return (
    <div>
      <h1>Contact Management</h1>
      <form onSubmit={(e) => e.preventDefault()}>
        <TextField name="firstName" label="First Name" value={formData.firstName}
onChange={handleInputChange} required />
        <TextField name="lastName" label="Last Name" value={formData.lastName}
onChange={handleInputChange} required />
        <TextField name="email" label="Email" value={formData.email}
onChange={handleInputChange} required />
        <TextField name="phone" label="Phone Number" value={formData.phone}
onChange={handleInputChange} required />
        <TextField name="company" label="Company" value={formData.company}
onChange={handleInputChange} />
        <TextField name="jobTitle" label="Job Title" value={formData.jobTitle}
onChange={handleInputChange} />
        <Button onClick={handleSubmit}>{editId ? "Update" : "Add"} Contact</Button>
      </form>
      <TableContainer component={Paper}>
        <Table>
          <TableHead>
            <TableRow>
              <TableCell>First Name</TableCell>
              <TableCell>Last Name</TableCell>
              <TableCell>Email</TableCell>
              <TableCell>Phone</TableCell>
              <TableCell>Company</TableCell>
              <TableCell>Job Title</TableCell>
              <TableCell>Actions</TableCell>
            </TableRow>
          </TableHead>
          <TableBody>
            {contacts.map((contact) => (
              <TableRow key={contact.id}>
                <TableCell>{contact.firstName}</TableCell>
                <TableCell>{contact.lastName}</TableCell>
                <TableCell>{contact.email}</TableCell>
                <TableCell>{contact.phone}</TableCell>
                <TableCell>{contact.company}</TableCell>
                <TableCell>{contact.jobTitle}</TableCell>
                <TableCell>
                  <Button onClick={() => handleEdit(contact)}>Edit</Button>
                  <Button onClick={() => handleDelete(contact.id)}>Delete</Button>
                </TableCell>
              </TableRow>
            ))}
          </TableBody>
```

```
        </Table>
      </TableContainer>
    </div>
  );
};

export default App;

Backend Code (Node.js)
------------------------------
Install Required Dependencies

npm install express mysql cors body-parser

const express = require("express");
const cors = require("cors");
const bodyParser = require("body-parser");
const mysql = require("mysql");

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

const db = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "password",
  database: "contacts_db",
});

db.connect((err) => {
  if (err) {
    console.error("Error connecting to database:", err);
  } else {
    console.log("Connected to MySQL database.");
  }
});

app.get("/contacts", (req, res) => {
  db.query("SELECT * FROM contacts", (err, results) => {
    if (err) {
      res.status(500).send(err);
    } else {
      res.json(results);
    }
  });
});

app.post("/contacts", (req, res) => {
  const { firstName, lastName, email, phone, company, jobTitle } = req.body;
```

```
  db.query(
    "INSERT INTO contacts (firstName, lastName, email, phone, company, jobTitle)
VALUES (?, ?, ?, ?, ?, ?)",
    [firstName, lastName, email, phone, company, jobTitle],
    (err) => {
      if (err) {
        res.status(500).send(err);
      } else {
        res.send("Contact added.");
      }
    }
  );
});

app.put("/contacts/:id", (req, res) => {
  const { id } = req.params;
  const { firstName, lastName, email, phone, company, jobTitle } = req.body;
  db.query(
    "UPDATE contacts SET firstName=?, lastName=?, email=?, phone=?, company=?,
jobTitle=? WHERE id=?",
    [firstName, lastName, email, phone, company, jobTitle, id],
    (err) => {
      if (err) {
        res.status(500).send(err);
      } else {
        res.send("Contact updated.");
      }
    }
  );
});

app.delete("/contacts/:id", (req, res) => {
  const { id } = req.params;
  db.query("DELETE FROM contacts WHERE id=?", [id], (err) => {
    if (err) {
      res.status(500).send(err);
    } else {
      res.send("Contact deleted.");
    }
  });
});

app.listen(5000, () => {
  console.log("Server running on http://localhost:5000");
});


Database Setup (MySQL)
----------------------------------------
CREATE DATABASE contacts_db;
```

```
USE contacts_db;

CREATE TABLE contacts (
  id INT AUTO_INCREMENT PRIMARY KEY,
  firstName VARCHAR(50),
  lastName VARCHAR(50),
  email VARCHAR(100),
  phone VARCHAR(15),
  company VARCHAR(100),
  jobTitle VARCHAR(100)
);
```

Run the Application:

Start the backend server:

```
cd server
npm start
```

Start the frontend application:

```
cd client
npm start
```

Access the application in your browser at http://localhost:3000.

Project Description

The Contact Management System is a mini CRM feature that allows users to manage their contacts effectively. Key features include:

Add New Contact: Users can add new entries with essential details like Name, Email, and Job Title.
View Contacts: A table displays all contacts with sorting and pagination for easy navigation.
Edit Contacts: Users can update contact information with a simple form.
Delete Contacts: Remove outdated or incorrect entries to maintain data accuracy.
Major Technical Decisions:

Used React.js and Material-UI for a clean, responsive, and consistent UI.
Leveraged Node.js and Express.js for backend API handling.
Chose MySQL for relational database operations due to its reliability and ease of use.
Challenges and Solutions
Challenge: Handling real-time form updates while editing a contact.

Solution: Implemented state management using React's useState and ensured controlled inputs for real-time updates.
Challenge: Ensuring smooth pagination and sorting for large datasets.

Solution: Utilized Material-UI's Table component with built-in pagination and

implemented server-side sorting logic for efficiency.
Challenge: Preventing duplicate contacts in the database.

Solution: Added backend validation using a unique constraint on the email field and provided user-friendly error messages.
Challenge: Integrating the database schema with frontend validation.

Solution: Maintained consistent validation rules between the frontend and backend to ensure data integrity.