# Table of Contents

# Developer Documentation

## 1. Introduction

### Overview

ProfilePro is a web-based application designed to manage professional contacts. It provides a simple and intuitive interface for users to create accounts, log in, and manage their contacts. The application is built using React.js for the frontend and Node.js with Express.js for the backend, with MySQL as the database.

### Technologies Used

- **Frontend**: React.js
- **Backend**: Node.js, Express.js
- **Database**: MySQL
- **Styling**: Tailwind CSS
- **Additional Technologies**: HTML, JavaScript

## 2. System Requirements

### Hardware Requirements

- **Minimum**: 4 GB RAM, 1 GHz Processor
- **Recommended**: 8 GB RAM, 2 GHz Processor

### Software Requirements

- **Operating System**: Windows 10 or later, macOS 10.15 or later, Linux (Kernel 3.10+)
- **Web Browser**: Latest versions of Chrome, Firefox, Safari, or Edge
- **Node.js**: Version 14.x or later
- **MySQL Server**: Version 8.x or later

### Dependencies

- **npm packages**: React, Express, MySQL, Nodemon, Dotenv, Winston, etc.

## 3. Codebase Structure

### Frontend (React.js)

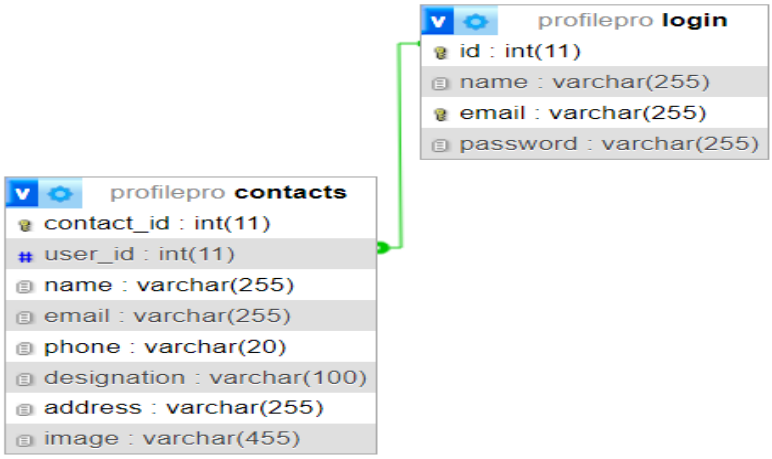
- **src/**
    - **components/**: React components for UI elements (e.g., Login, Sign-Up, Navigation pages, Contacts).
    - **Main.jsx**: Main React component that sets up routing and context providers.
    - **index.html**: Entry point for the React application.

## Backend (Node.js with Express.js)

- **routes/**
    - **server.js**: Routes for user authentication (login, registration).
    - **dataRoutes.js**: Routes for managing contacts (CRUD operations).
    - **mailRoutes.js**: Route for handling email sending functionality.
- **config/**
    - **db.js**: Database configuration and connection setup.
    - **env/**: Environment-specific configurations (e.g., production, development).
- **server.js**: Main entry point for the Express.js application.

## Database

- **Structure:**




- **contacts table:**

| Column Name | Data Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| id | int(11) | | Primary Key | No | None | Unique identifier for each user | AUTO_INCREMENT |
| name | varchar(255) | utf8mb4_general_ci | | No | None | Name of the user | |
| email | varchar(255) | utf8mb4_general_ci | Index | No | None | User's email address | |
| password | varchar(255) | utf8mb4_general_ci | | No | None | User's password (stored securely) | |

- **login table:**

| Column Name | Data Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| contact_id | int(11) | | Primary Key | No | None | Unique identifier for each contact | AUTO_INCREMENT |
| user_id | int(11) | | Index | Yes | NULL | Foreign key linking to login table | |
| name | varchar(255) | utf8mb4_general_ci | | Yes | NULL | Name of the contact | |
| email | varchar(255) | utf8mb4_general_ci | | Yes | NULL | Contact's email address | |
| phone | varchar(20) | utf8mb4_general_ci | | Yes | NULL | Contact's phone number | |
| designation | varchar(100) | utf8mb4_general_ci | | Yes | NULL | Contact's job title or designation | |
| address | varchar(255) | utf8mb4_general_ci | | Yes | NULL | Contact's address | |
| image | varchar(455) | utf8mb4_general_ci | | Yes | NULL | Path to the contact's image | |

# 4. Setup and Installation

## Prerequisites

- Node.js and npm installed on your system.
- MySQL Server installed and configured.

### Installation Steps

1. **Clone the Repository**:
   - git clone https://github.com/Rohitmh09/profilepro.git
2. **Navigate to the Project Directory**:
   - cd profilepro
3. **Install Dependencies**:
   - npm install
4. **Configure Database**:
   - Update config/db.js with your MySQL database credentials.
   - Run database migrations to set up the schema.
5. **Start the Server**:
   - npm start

## 5. API Documentation

### Authentication

- **/api/login**
  - **Method**: POST
  - **Request Body**: { email: "user@example.com", password: "password" }
  - **Response**: { success: true, token: "jwt-token" }
- **/api/register**
  - **Method**: POST
  - **Request Body**: { name: "John Doe", email: "john@example.com", password: "password" }
  - **Response**: { success: true, message: "User registered successfully" }

### Contact Management

- **/api/data/saveData**
  - **POST**: Add a new contact or update an existing contact.
- **/api/data/fetchData**
  - **GET**: Fetch all contacts for the logged-in user.
- **/api/data/deleteContact**
  - **DELETE**: Delete a contact.

### Email Handling

- **/api/mail/send-mail**

- **Method**: POST
- **Request Body**: { name: "General user", email: "general@example.com", phone: "1234567890" }
- **Response**: { success: true, message: "Email sent successfully" }

### Error Handling

- **400 Bad Request**: Invalid input data.
- **401 Unauthorized**: Missing or invalid JWT token.
- **404 Not Found**: Resource not found.
- **500 Internal Server Error**: Unexpected server error.

## 6. Testing

### Manual Testing

- The project was tested manually using a trial-and-error approach. This method involved actively interacting with the application to identify any issues or bugs, ensuring that all features worked as expected.

### Integration Testing

- Ensured that the frontend and backend worked seamlessly together by performing end-to-end tests during development. This was done by manually interacting with the application and observing the behavior of API integrations

### User Acceptance Testing (UAT)

- Involve end-users in testing to ensure the application meets their requirements.

## 7. Development Workflow

### Project Development

- **IDE**: Visual Studio Code is used as the main development environment.
- **File Management**: All code files are maintained locally on the development machine.
- **Backup Strategy**: Regular backups of the project are taken manually by copying project files to a secure location.

### Code Review

- **Self-Review**: Regularly review code for consistency, proper comments, and best practices.
- **Peer Review (optional)**: Code can be shared with peers for review through file sharing services before finalizing.

### Deployment

- **Local Testing**: The application is tested locally using the command npm start to ensure everything is working before deployment.
- **Manual Deployment**: Once the application is tested, it is manually deployed to the target environment (e.g., web server, hosting platform).

### Monitoring and Maintenance

- **Local Monitoring**: Logs and errors are monitored locally during testing.
- **Manual Updates**: If required, updates are manually applied by modifying the files and redeploying the project.

## 8. Glossary

- **CRUD**: Create, Read, Update, Delete - the basic operations for managing data.
- **API**: Application Programming Interface - a set of rules that allows software programs to communicate with each other.
- **JWT**: JSON Web Token - a compact, URL-safe means of representing claims to be transferred between two parties.

## 9. References

- **React.js Documentation**: React Installation Guide
- **Express.js Documentation**: Express Routing
- **MySQL Documentation**: MySQL Documentation