Contact Management ERINO

Internship Task

Name: Nikhil Sri Ram Pulluri

Email: nikhilpulluri7810@gmail.com

Phone: 8317533755

GitHub Link: https://github.com/Nikhil-Pulluri/Contact-Management-ERINO

Overview

This project is a Contact Management System, which is a mini feature of a larger CRM application. It allows users to add, view, edit, and delete contacts. The system is designed to help users organize and manage client or customer contact information in an efficient and user-friendly way. The application is built with React (TypeScript) on the frontend and Express.js with MongoDB on the backend.

Tech Stack

Frontend:

- React with TypeScript
- **Material-UI (MUI)**: For form elements, tables, buttons, and layout.

Backend:

- **Express.js**: REST API for CRUD operations.
- **MongoDB**: Database for storing contact information.

Setup Instructions

Follow these steps to set up and run the project locally:

Prerequisites:

- Node.js (v16+)
- MongoDB (installed locally or available via a cloud provider like MongoDB Atlas)

1. Clone the Repository

```
git clone https://github.com/Nikhil-Pulluri/Contact-Management-ERINO.git
cd <to the project root folder>
```

2. Install the dependencies

```
// front end dependencies
cd client
npm i
```

```
// backend dependencies
cd server
npm i
```

3. To make the project run, we need to run the frontend and backend servers parallelly.

```
// front end server running
cd client
npm run dev
```

```
// backend server running
cd server
nodemon .\index.js
```

4. Access the Application in the given link below

Open your browser and go to http://localhost:3000.

Major Technical Decisions

1. **Why React with TypeScript?**
    - TypeScript adds type safety, reducing runtime errors.
    - React is versatile and efficient for building responsive user interfaces.
2. **Why Material-UI?**
    - Provides ready-to-use, customizable components.

- Speeds up development while ensuring design consistency.

3. **Why MongoDB?**
    - Flexible schema allows rapid development.
    - Suitable for handling structured and semi-structured data like contact records.

How Each Part of the App Works

Frontend (React with TypeScript)

1. **Add Contact Form**:
    - A form built using Material-UI components captures user input for new contact details (First Name, Last Name, Email, Phone Number, Company, Job Title).
    - On submission, the form sends the data to the backend API using Fetch.
2. **Contacts Table**:
    - Displays all stored contacts in a paginated and sortable Material-UI table.
    - Includes action buttons for editing and deleting each contact and the actions are collapsible in every row of the table. Just click on the down arrow for the contact action buttons.
    - User actions (edit/delete) trigger requests to the backend, ensuring the table remains synced with the database.
3. **State Management**:
    - Use local state to manage the list of contacts and reflect real-time updates when new contacts are added, or existing ones are edited/deleted.

Backend (Express.js with MongoDB)

1. **REST API**:
    - **POST /contacts**: Handles form submissions to add new contacts.
    - **GET /contacts**: Retrieves all contacts from MongoDB, with support for pagination and sorting.
    - **PUT /contacts/:id**: Updates contact details for a specific ID.
    - **DELETE /contacts/:id**: Removes contact from the database.

2. **Validation and Error Handling**:
    - Ensure all required fields are provided and valid during contact creation or updates.
    - Prevents duplicate email entries and returns meaningful error messages for invalid data.
3. **Database Integration**:
    - MongoDB stores contact data, providing a flexible and efficient schema for CRUD operations.
    - Mongoose is used to model and validate data.

Database Schema Script

```
const  Contacts = mongoose.model("contacts",{
  firstname:{
    type:String,
    required:true
  },
  lastname:{
    type:String,
    required:true
  },
  email:{
     type:String,
     required:true,
     unique:true,
  },
  phone:{
    type:String,
    required:true,
    unique:true,
  },
  company:{
    type:String,
    required:true,
  },
  jobtitle:{
    type:String,
    required:true,
  },
```

```
})
```

Challenges I faced during the development

- While creating the table structure and pagination, I have encountered a lot of errors for which I took around 2 hours. The table developed in Material UI is very complex and took a lot of time to update the things that are suitable for this Contact Management System.
- The collapsible contact action buttons of edit and delete options and their connectivity to the database in real time.
- Faced several errors in the backend. Like, that took me a while to realize where I have committed mistakes and yes, I was able to sort them out and was able to connect with the frontend using fetch.