

14. Contact Management System

Creating a Contact Management system using the MEAN stack (MongoDB, Express.js, Angular, Node.js) is a valuable project. Here's a high-level overview and code snippets to guide you through the process:

Project Setup and Structure

Set up a new project folder and structure for your Contact Management system. Install the required Node.js packages and create a basic Angular application.

Create a new Angular application

```
ng new contact-management-app
```

- Backend (Node.js & Express.js)

Create the backend of your Contact Management system using Node.js and Express.js.

Installation of Packages

Install the necessary packages for Express.js, Mongoose (for MongoDB), and other dependencies.

```
npm install express mongoose cors multer
```

Setting up Express.js

Create your Express.js server, set up middleware, and handle routes.

- **javascript**

// server.js

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const multer = require('multer');

const app = express();
```

// Middleware

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

// Database connection

```
mongoose.connect('mongodb://localhost/contact-management-
app', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
```

```
useCreateIndex: true,  
});  
  
// Define Mongoose models for Contact data  
const Contact = mongoose.model('Contact', {  
  firstName: String,  
  lastName: String,  
  email: String,  
  phone: String,  
  address: String,  
// Add more fields as needed  
});
```

```
// Multer storage for importing contacts  
const storage = multer.diskStorage({  
  destination: (req, file, cb) => {  
    cb(null, 'uploads/');  
  },  
  filename: (req, file, cb) => {  
    cb(null, Date.now() + '-' + file.originalname);  
  },
```

```
});
```

```
const upload = multer({ storage });
```

// Routes for managing contacts

```
app.post('/api/contacts', async (req, res) => {
```

```
    // Create a new contact
```

```
    // Save the contact to the database
```

```
});
```

```
app.get('/api/contacts', async (req, res) => {
```

```
    // Retrieve a list of contacts
```

```
});
```

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

```
    // Update a contact
```

```
    // Save the updated contact to the database
```

```
});
```

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

```
    // Delete a contact
```

```
// Remove the contact from the database
});

// Route for importing contacts from a CSV file
app.post('/api/import-contacts', upload.single('file'), async (req, res) => {
    // Parse the CSV file and add contacts to the database
});
```

- **Frontend (Angular)**

Create the frontend of your Contact Management system using Angular. Design the user interface for managing contacts and importing/exporting contact lists.

Design and UI

Design the user interface for your Contact Management system using Angular components, templates, and styles.

Contact Management

Create components and forms for users to manage contacts, including specifying first name, last name, email, phone number, and address.

Import/Export

Design components for importing and exporting contact lists. Allow users to upload CSV files to add multiple contacts at once.

- typescript

```
// contact-import-export.component.ts
```

```
import { Component } from '@angular/core';
import { ContactService } from './contact.service';
```

```
@Component({
  selector: 'app-contact-import-export',
  templateUrl: './contact-import-export.component.html',
})
```

```
export class ContactImportExportComponent {
  file: File;
```

```
constructor(private contactService: ContactService) {}
```

```
importContacts() {
```

```
  this.contactService.importContacts(this.file).subscribe((data) => {
```

```
    // Handle success response
```

```
});
```

```
}

exportContacts() {
  this.contactService.exportContacts();
}

}
```

MongoDB

Create a MongoDB database to store contact data.

Putting It All Together

Integrate the frontend and backend by making API requests from Angular components to Node.js routes. Ensure that you handle contact creation, updating, and deletion properly.

Building a Contact Management system is a practical and useful project. You can further improve it by adding features like search, filtering, tagging, and contact sharing among users if needed.