Smart Contact Manager - Q&A for Advisor Presentation

1. What is the Smart Contact Manager project about?

The Smart Contact Manager is a full-stack web application designed for efficiently managing personal and professional contacts. It allows users to create, organize, and manage their contact lists with features like categorization, favorites, profile images, and search functionality. The application includes user authentication, a dashboard with statistics, and an admin panel for system management.

2. What technologies did you use to build this project?

Backend:

- Node.js - JavaScript runtime environment
- Express.js - Web framework for building APIs
- SQLite - Lightweight database using better-sqlite3
- bcryptjs - For password hashing
- jsonwebtoken - For JWT-based authentication
- multer - For handling file uploads
- cors - For cross-origin resource sharing
- morgan - HTTP request logger
- winston - Logging library

Frontend:

- HTML5 - Markup language
- CSS3 - Styling
- Vanilla JavaScript - Client-side logic
- Chart.js - For data visualization in dashboard
- Font Awesome - For icons

3. Can you describe the overall architecture of the application?

The application follows a client-server architecture:

- **Frontend:** Static HTML, CSS, and JavaScript files served from the public directory
- **Backend:** Node.js/Express server handling API requests
- **Database:** SQLite database for data persistence
- **File Storage:** Local file system for uploaded profile images

The backend uses MVC-like structure with routes, models, and middleware. Authentication is handled via JWT tokens stored in localStorage.

4. What are the key features of the application?

- User authentication (signup, login, logout) with JWT
- Contact CRUD operations (Create, Read, Update, Delete)
- Contact categorization (Work and Personal)
- Favorites system for marking important contacts
- Profile image upload and management
- Dashboard with contact statistics and charts
- Search and filter functionality
- CSV import/export for contacts
- Admin panel for managing users and contacts
- Responsive design for desktop and mobile

5. How does authentication work in your application?

The application uses JSON Web Tokens (JWT) for authentication:

- Users register with email and password
- Passwords are hashed using bcryptjs before storage
- Upon login, a JWT token is generated and sent to the client
- The token is stored in localStorage and included in Authorization headers for subsequent requests
- Middleware validates the token for protected routes
- Admin routes require additional role-based authorization

6. Can you explain the database design?

The application uses SQLite with two main tables:

- **Users table:** id, name, email, password (hashed), role, created_at
- **Contacts table:** id, user_id (foreign key), name, email, phone, address, category, is_favorite, notes, profile_image, created_at, updated_at

Relationships are established through foreign keys, ensuring data integrity. The database is initialized using better-sqlite3 with prepared statements for security.

7. What challenges did you face during development?

- Implementing secure file upload with proper validation and storage
- Managing JWT token expiration and refresh logic

- Ensuring responsive design across different screen sizes
- Handling asynchronous operations in vanilla JavaScript
- Implementing role-based access control for admin features
- Optimizing database queries for better performance
- Integrating Chart.js for data visualization

8. How did you handle security in the application?

- Password hashing using bcryptjs with salt rounds
- JWT tokens for stateless authentication
- CORS configuration to prevent cross-origin attacks
- Input validation and sanitization
- Role-based authorization for admin functions
- File upload restrictions (size, type)
- SQL injection prevention using prepared statements

9. How does the admin panel work?

The admin panel is accessible only to users with 'admin' role:

- View system-wide statistics (total users, contacts, etc.)
- Manage all users (view, delete except self)
- Manage all contacts across all users (view, delete)
- Charts showing user and contact distributions

Admin access is protected by middleware that checks the user's role from the JWT token.

10. What are some potential future improvements?

- Add email integration for contact reminders
- Implement contact sharing between users
- Add more advanced search filters (date ranges, custom fields)
- Implement real-time notifications
- Add backup and restore functionality
- Integrate with external APIs (social media, email providers)
- Add unit and integration tests
- Implement pagination for large contact lists

11. How did you handle file uploads?

File uploads are handled using multer middleware:

- Profile images are stored in the uploads/ directory

- Files are validated for size (max 5MB) and type (images only)
- Unique filenames are generated using timestamps
- Images are served statically by Express
- Old images are deleted when updated or contact is deleted

12. How is the dashboard implemented?

The dashboard provides visual insights into contact data:

- Statistics: total contacts, work/personal split, favorites count
- Pie chart showing contact distribution by category using Chart.js
- Data is fetched via API endpoint and rendered client-side
- Responsive design ensures charts work on mobile devices

13. What deployment considerations did you think about?

While not deployed, considerations include:

- Environment variables for sensitive data (database path, JWT secret)
- Static file serving optimization
- Database migration scripts for production
- Logging configuration for different environments
- HTTPS configuration for secure communication
- Rate limiting to prevent abuse

14. How did you ensure the application is user-friendly?

- Intuitive navigation with consistent navbar
- Clear visual feedback for user actions
- Responsive design for mobile and desktop
- Search and filter functionality for easy contact finding
- Form validation with helpful error messages
- Loading indicators for asynchronous operations
- Consistent styling and color scheme

15. What did you learn from this project?

This project provided hands-on experience with:

- Full-stack JavaScript development
- RESTful API design and implementation
- Database design and SQL operations
- Authentication and authorization patterns

- File upload handling and storage
- Frontend-backend integration
- Security best practices
- Project structure and organization
- Problem-solving and debugging skills