Project Write-Up: CRM Application

Architectural Choices

The CRM dashboard frontend was developed with a focus on modularity, responsiveness, and maintainability, leveraging modern web technologies.

1. Frontend: React.js (with Vite)

- **Why React?**
  React's component-based architecture ensures scalability and easy maintenance. It allows us to split the UI into reusable pieces, making development efficient and modular.
- **Why Vite?**
  Vite offers lightning-fast development builds and hot module replacement, making the development experience smoother than older bundlers like Webpack.
- **State Management:**
  Local component state and hooks (useState, useEffect) were sufficient for our use case.
- **- Bootstrap:** Integrated for styling and responsive design. Bootstrap's pre-built components and utility classes significantly accelerated development, ensuring a consistent and mobile-friendly user interface without extensive custom CSS.
- **-Component-Based Structure:** The application is broken down into logical, reusable components (e.g., `Dashboard`, `UserManagement`, `ContactManagement`, `Reporting`, `Navigation`, `UserModal`, `ContactModal`, `SuccessModal`, `ConfirmationModal`). This separation of concerns enhances code organization, readability, and maintainability.

2. Backend: Spring Boot

- **Why Spring Boot?**
  It simplifies backend development using Java and provides production-ready configurations, an embedded server (Tomcat), and RESTful API creation with ease.
- **Structure:**
  The backend follows the layered architecture:
  - Controller → handles incoming HTTP requests.
  - Service → business logic layer.
  - Repository → interacts with the database via Spring Data JPA.

3. Database: PostgreSQL

- PostgreSQL is recommended for production due to its reliability, scalability, and wide adoption.

4. Communication: REST API (JSON)

- React fetches data via Axios from Spring Boot's REST endpoints using a clean separation between frontend and backend.

5. Cross-Origin Resource Sharing (CORS):

- Configured on the backend to allow frontend (usually served from localhost:5173) to make API requests to backend (localhost:8080).

Challenges Faced & Resolutions

1. CORS Issues

- **Problem:** React frontend was unable to fetch data from the Spring Boot backend due to CORS policy restrictions.
- **Solution:** Implemented a global CORS configuration in the backend using WebMvcConfigurer.

3. Backend not starting (Spring Boot Error)

- **Problem:** mvn spring-boot:run failed with process exit code 1.
- **Solution:** Investigated logs and found an issue in entity configuration. Ensured that entity classes were annotated correctly with @Entity and that application.properties was properly configured for the database.

4. During the development of the CRM dashboard frontend, several challenges were encountered, primarily related to state management, UI responsiveness, and user interaction flows.

-Challenge 1: Managing Global Application State

Problem: As the number of users and contacts grew, and CRUD operations were introduced, managing the `users` and `contacts` arrays across different components (Dashboard, UserManagement, ContactManagement, Reporting) became complex. Passing state and update functions through multiple levels of components (prop drilling) could lead to cumbersome code.

Resolution: The core `users` and `contacts` state, along with their respective CRUD handler functions (`handleAddUser`, `handleUpdateUser`, `handleDeleteUser`, etc.), were centralized in the top-level `app/page.js` component. These functions and data were then passed down as props to the relevant child components. While this still involves some prop drilling, it keeps the data logic in a single, accessible place for this application's size. For a much larger

application, a dedicated state management library (like Zustand, Redux, or React Context API) would be considered to abstract this further.

⚙️ Setup Instructions

💻 1. Frontend (React)

```
cd my-react-app
npm install       # Installs dependencies including axios
npm run dev       # Runs the app on http://localhost:5173
```

🛠️ 2. Backend (Spring Boot)

```
cd crm-backend
./mvnw spring-boot:run      # Or use: mvn spring-boot:run
```

✅ Final Notes

- Ensure both frontend and backend run in **parallel** during local development.
- Port alignment is essential: React (5173), Spring Boot (8080).
- All APIs are hosted at /api/users with full CRUD support.
- CORS must be handled either globally (WebConfig) or using annotations like @CrossOrigin.