Munich AI Nexus

1. The Overview: Setting the Scene

CURRENT ("As-Is") Stateless Flow

[UI: Static Form / Chatbot] -> [n8n Workflow Engine] -> [LLM for Structuring] -> [NocoDB Storage] & [CV Sent to User]

FUTURE ("To-Be") Stateful Flow

Diagram Sketch for "To-Be" Flow:

```
[Authenticated User] <-> [Dynamic UI (with JWT)]
-> [Secure Netlify Functions (API Gateway)] <->
[NocoDB (Users, Favorites, etc.)]
```

Top Architectural Challenges

1. **Challenge: Evolving from Stateless to Stateful Architecture.**

1. **Challenge: Designing a Secure & Scalable Permissions Model**

1. **Challenge: Ensuring GDPR Compliance with an LLM Pipeline.**

2. The Deep Dive:
The Collaborative Whiteboard

Material 1: Data Model

"As-Is" Data Model (Simplified)

- `Workers` **Table**
  - `Id`
  - `Email` (The only "identifier")
  - `FirstName`
  - `PhoneNumber`
  - ... (other CV fields)
- `Jobs` **Table**
  - `Id`
  - `CompanyName`
  - `NeededPosition`
  - ... (other job fields)

Proposed "To-Be" Data Model (For Discussion):

- `Users` **Table (New Core)**
  - `id` (Primary Key)
  - `auth0_user_id` (UNIQUE - Links to Auth0)
  - `email`
- `Workers` **Table (Modified)**
  - `id`
  - `user_id` (Foreign Key -> `Users.id`)
  - ... (CV fields)
- `User_Job_Favorites` **Table (Many-to-Many Join)**
  - `user_id` (Foreign Key -> `Users.id`)
  - `job_id` (Foreign Key -> `Jobs.id`)
- `User_Job_Applications` **Table (The Goal)**
  - `user_id` (FK -> `Users.id`)
  - `job_id` (FK -> `Jobs.id`)
  - `status` (e.g., 'submitted', 'viewed')
  - `applied_at`
  - `resume_snapshot_json` (❓ ***Discussion Point: Is this the best way to preserve the CV at the time of application?***)

Material 2: API Handoffs

1. **User Sync & Profile Linking**
    - **Endpoint:** `POST /api/sync-user`
    - **Triggered:** After a user's first login via Auth0.
    - **Action:** The function receives the user's JWT. It creates a record in our `Users` table and then searches the `Workers` table by email to link the new account to any pre-existing anonymous profile.
2. **Fetching Data with Permissions**
    - **Endpoint:** `GET /api/jobs`
    - **Triggered:** When a logged-in user views the job board.
    - **Action:** The function validates the JWT, gets the `user_id`, fetches all jobs, and then annotates each job with user-specific data (e.g., `isFavorited: true`) before returning the list.
3. **Secure, State-Changing Actions**
    - **Endpoint:** `POST /api/jobs/{jobId}/apply`
    - **Triggered:** When a user clicks "Apply".
    - **Action:** The function validates the JWT to confirm the user's identity, creates the `resume_snapshot`, and writes a new record to the `User_Job_Applications` table. **(❓ *Discussion Point: How should we structure the permissions check for the future 'Employer' role who needs to view this application?*)**

Scan me!

www.sohnus.de

info@sohnus.de