JobbAI: AI-Powered Job Matching for German University Students

Author Names: Jiaqi Wen, Arina Kuznetsova
Repository: https://github.com/wenjiaqi8255/job-hunter

Contents

---

1. Introduction

German university students and graduates face two critical challenges in job searching.

Information distortion chain

Business need → HR understanding → job description → student interpretation

Real example: A startup needed "mobile UI skills + basic coding ability" but the job posting emphasized "Angular, Go, Huggingface AI frameworks" while burying the actual requirement in one line about "mobile application development." As a result, the true requirements are hard to spot, which makes it especially timely to create a well-fitted application.

Modern entry-level positions increasingly demand hybrid skill sets, blurring traditional role boundaries. A "Product Manager" might require design thinking, data analysis, and technical implementation. Students struggle to decode which listed requirements are essential versus aspirational, especially when descriptions use gentle phrasing for strict requirements ("German preferred" = "German required").

Manual search inefficiency

Traditional job hunting requires students to manually review hundreds of postings, spending 2-3 hours crafting each application without understanding true compatibility. This time-intensive process often leads to random applications rather than strategic matching.

AI addresses both challenges: semantic analysis decodes hidden requirements and cultural context, while automated matching eliminates manual screening inefficiency.

1.1 Team workload distribution

Because the team consisted of 2 people, the division of labor was straight-forward. The backend development was led by Jiaqi, with Arina focused on frontend and design aspects. Eventually, both of us were complementing each other's work in minor bug fixes, branch support, and design refinements. Our communication was impressively smooth, with each person knowing and fulfilling their responsibilities, oftentimes tacitly. Thus, our work distribution can be characterized as follows:

- **Jiaqi Wen (lead developer)**: system architecture, backend development (Django), AI integration (Gemini API), semantic anomaly detection, authentication system
- **Arina Kuznetsova (UI/UX developer)**: frontend templates, user interface design, styling consistency, responsive design, navigation patterns

1.2 AI usage declaration

Our work combined human-led decisions (mostly conceptual developments and ideas) and applied AI support. For example, stages such as system architecture, business logic, and technology choices were human input. Moreover, human-driven problem solving included authentication refactors and semantic analysis evolution, with debugging being mainly manual with some AI help.
On the other side, AI-assisted implementation included Django boilerplate and generating view functions. The template structures were initially generated using AI, though all were later human-reviewed and refined. AI was also involved in enhancing documentation for grammar checking and clarity improvement.

---

2. Background

German job market context

It's crucial to understand the candidate selection process most companies employ nowadays, as most of JobbAI features were built to suit it. Our team has involved real HR specialists to interview for such requirements and insider tips. For example, the applicant tracking system (ATS) plays a big role in job matching as well as CV and cover letter generation. Such systems primarily rely on keyword matching. Given that, JobbAI should generate documents with the exact wording from the job postings. Although unlike solely keyword-based platforms, it also

accounts for cultural nuances, language requirements, and hidden job posting meanings specific to the German market.

Technical information

As for technical foundations, **Django** was implemented for robust ORM, built-in authentication, and rapid MVP development; **Supabase** (open-source database infrastructure built on PostgreSQL) acts as OAuth provider and job data source; and **Google Gemini API** helps with cost-effective semantic analysis with strong multilingual support. The core innovation combines semantic analysis with cost optimization through a simulation framework that enables development without API costs while maintaining production capability.

The project evolved from statistical TF-IDF approaches to transformer-based semantic understanding, enabling recognition that "Backend Engineer" and "Server-side Developer" represent identical concepts.

---

3. Methodology

3.1 System architecture

During the development, our team stuck to the KISS principle, especially in early stages. The architecture shown in *Figure 1* consists of a Django-based web application structured in three layers: a frontend layer (Django templates + CSS), an application layer (Django views + services), and integrations with both external services and local storage. External services include Supabase and the Gemini API (used in text analysis and document generation), while local storage relies on an SQLite database managing user, job, and application models (e.g., profiles, sessions, matched jobs, cover letters, resumes).

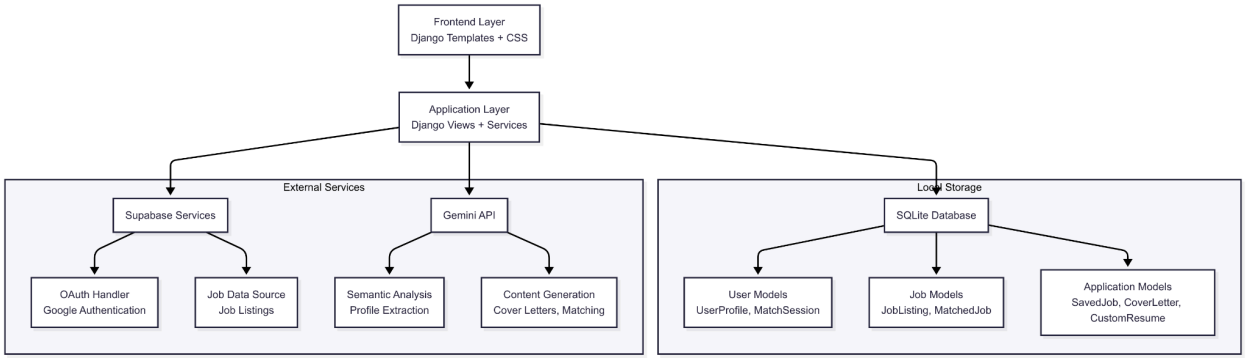

Figure 1: JobbAI system architecture

3.2 Data architecture

Core models:

- `UserProfile`: CV content, preferences, structured profiles
- `MatchSession`: individual matching attempts with skills/preferences
- `JobListing`: job postings with descriptions and metadata
- `MatchedJob`: AI scores, reasoning, insights per job-session pair
- `SavedJob`: application tracking with status progression
- `CoverLetter`: AI-generated application materials

The above models enable the app to store user data, run AI-powered matching, and track application progress. Local storage ensures fast access and persistence, while Supabase connects the app to external authentication and job sources, forming a pipeline between user input, AI insights, and actionable job applications.

3.3 AI integration and cost management

Dual-mode execution system:

The function in *Figure 2* ensures flexibility by switching between simulated outputs during development and real AI responses in production. This allowed for testing job-matching workflows quickly without relying on the live AI model, while still enabling full AI-powered analysis under deployment.

```
def _execute_ai_task(task_name, prompt_func, parser_func, sim_func,
sim_args):
    if settings.USE_AI_SIMULATION:
        return sim_func(*sim_args)
    return parser_func(model.generate_content(prompt).text)
```

Figure 2: dual-mode system function (matcher.gemini_utils._execute_ai_task)

3.4 Core matching process

The matching process starts when the AI first extracts structured data from a user's CV, including skills, work history, and language abilities. It then compares this profile against job postings to generate match scores with detailed reasoning, factoring in German market specifics like language requirements, cultural expectations, and work style preferences. This ultimately gives users both fit insights and personal application advice.

3.5 Advanced feature: semantic anomaly detection

Semantic anomaly detection in JobbAI is the process of identifying job postings or data points that deviate in meaning or context from expected patterns, rather than just in surface-level wording.

JobbAI builds a baseline vector space of skills, roles, and industries using LLM-generated vocabularies and embeddings from SentenceTransformer. A parallel process separates job content, compares role/industry similarity, and applies Euclidean distance to detect anomalies, i.e. jobs that don't make sense compared to typical expectations for that role.

Example: if most postings for "Data Analyst" mention SQL, Python, and reporting, but one job labeled "Data Analyst" instead requires "welding" and "mechanical repair," the anomaly detection pipeline would flag it as semantically inconsistent.

3.6 Authentication system

Hybrid OAuth flow:

The authentication flow in *Figure 3* shows how a user logs into JobbAI with Google via Supabase OAuth. The user initiates login, Django redirects through Supabase to Google for consent, and Google returns an authorization code. Supabase exchanges the code for tokens, then redirects back to Django, which creates or retrieves the local user and establishes a session. This flow combines external OAuth security with local data sovereignty.

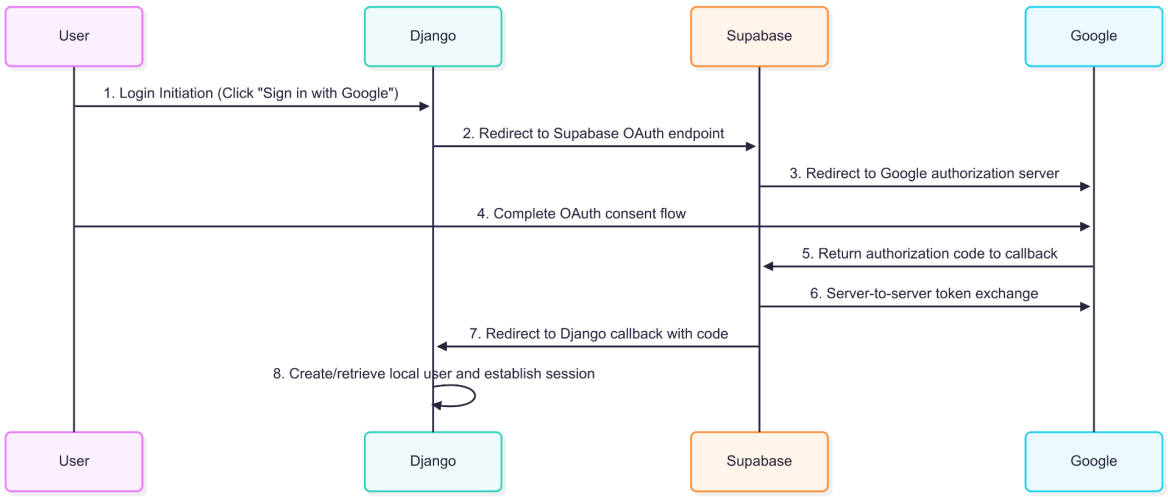

Figure 3: Authentication flow

---

4. Discussion and outlook

Result example (user path)

A student uploads their CV into JobbAI and starts a new matching session. The system scans job postings and highlights a role "Founding Product Designer" at telli in Berlin, as seen on *Figure 4*. The app shows a 90% match score, signaling strong alignment with their profile. It

recognizes that the on-site preference matches the student's availability, and points out that their AI/ML experience adds value in this context. Finally, it gives individual advice, suggesting the student emphasize their product strategy skills with an AI-first focus, turning raw job data into clear, actionable next steps.

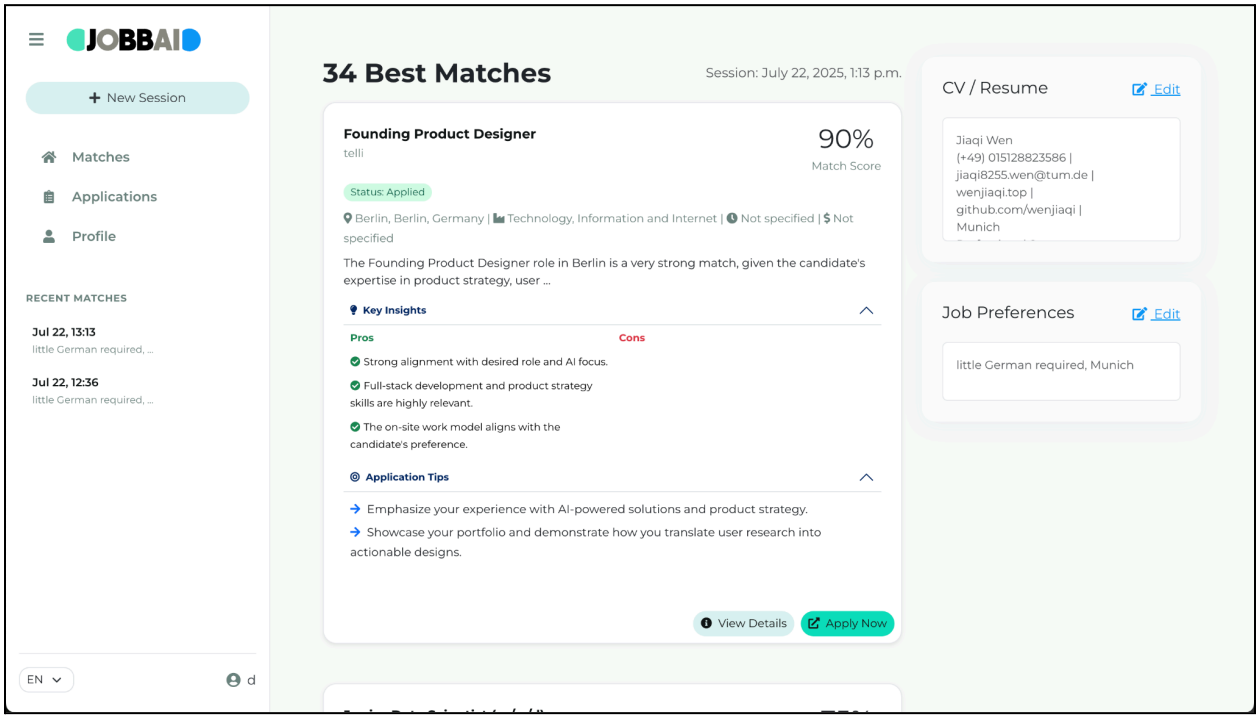

Figure 4: Match analysis example

Technical accomplishments

- **Cost effectiveness:** a big advantage we've used was cost-effective development. The above-discussed simulation framework enables full functionality testing without API costs.
- **Semantic evolution:** text analysis methods progressed from simple word frequency (TF-IDF) to transformer-based contextual understanding.
- **Target specialization:** JobbAI includes cultural context integration and location preference analysis, according to the target user.
- **Scalable architecture:** we kept a clean separation (in particular views) allowing for future enhancements
- **User experience:** given the complexity of the app, the interface is understandable and easy to navigate

Limitations

- Job data requires external API integration for production
- Heavy AI service dependency creates cost/availability risks
- German-English context limits international applicability

- Web interface needs mobile optimization

Future roadmap

In the short term, the focus is on building partnerships with job platforms, improving the mobile experience, and strengthening analytics. In the medium term, the plan is to develop offline semantic baselines, add multi-language support, and introduce more advanced application tracking. The ultimate goal is to create a proprietary knowledge base for the German market, form partnerships with universities, and train custom AI models.

Additionally, we've considered how such a product could be reversed into a B2B scheme, as a way to monetize it with small to medium sized businesses. This idea came from an interview with an HR specialist of a small consulting firm, who mentioned it as a market gap.

Lessons learnt

We saw firsthand how an iterative approach and choosing practical technologies worked far better than aiming for theoretical perfection. For AI, using simulation frameworks proved essential to build reliably without over-dependence on live models. We also learned that domain expertise is critical for creating AI applications that have real value. Finally, keeping costs low with free or affordable tools while leaving room to scale made the project sustainable, especially in a student setting.

---

5. References

1. Django Software Foundation. (2024). Django Documentation. https://docs.djangoproject.com/
2. Supabase Inc. (2024). Supabase Authentication. https://supabase.com/docs/guides/auth
3. Google AI. (2024). Gemini API Documentation. https://ai.google.dev/docs
4. Reimers, N., & Gurevych, I. (2019). Sentence-BERT. arXiv preprint arXiv:1908.10084
5. Becker, Sophie. (2023, December 14). *What Is Supabase?* Bold Tech (tools.dev). Bold Tech Blog.

---

6. Appendix

A. Database Entity Relationships

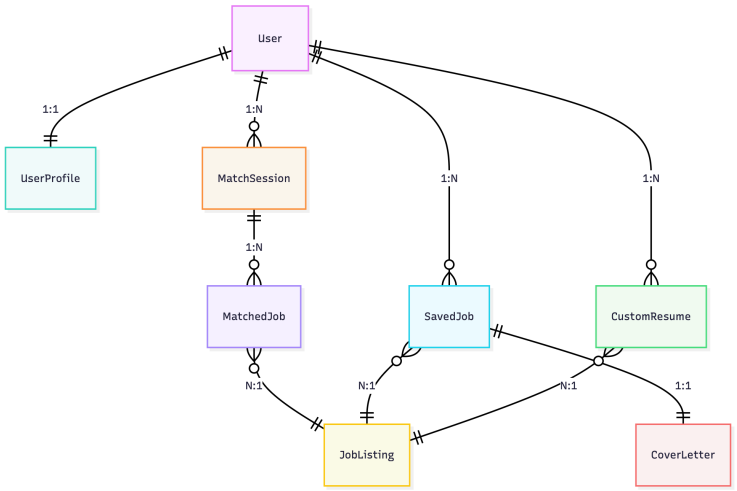

Figure 5: Database entity relationship diagram

Key relationships:

- **User ↔ UserProfile (1:1):** each authenticated user has exactly one profile containing CV data and preferences
- **UserProfile ↔ MatchSession (1:N):** users can initiate multiple matching sessions with different criteria
- **MatchSession ↔ MatchedJob (1:N):** each session generates multiple job matches with scores and insights
- **JobListing ↔ MatchedJob (1:N):** jobs can appear in multiple matching sessions for different users
- **User ↔ SavedJob (1:N):** users can save multiple jobs for application tracking
- **SavedJob ↔ CoverLetter (1:1):** each saved job can have one AI-generated cover letter

UserProfile model fields:

- Primary fields: user (OneToOne), user_cv_text, cv_file, user_preferences_text
- Automatically extracts structured profile data using AI analysis
- Supports file upload for PDF CV processing

MatchSession model features:

- Represents individual matching attempts with timestamp tracking
- Stores both raw input and structured profile JSON
- Enables historical analysis and re-matching capabilities

Job application workflow:

- JobListing → MatchedJob → SavedJob → CoverLetter

- Each stage captures specific user interactions and AI-generated content
- Status tracking enables comprehensive application management

B. Technical configuration

Environment variables:

- `USE_SIMULATION_ENV`: development mode toggle
- `GEMINI_API_KEY`: production AI integration
- `SUPABASE_URL/KEY`: OAuth and data services

Deployment: poetry dependencies, SQLite→PostgreSQL migration, static file configuration