Final Project Report: Job-Genie

Changes in Project Direction

Our initial proposal was to develop an AI-powered job recommendation system that primarily matched jobs based on keyword extraction from resumes. However, as the project evolved, we expanded the scope to include application tracking, and a more sophisticated matching system using a semantic similarity approach based on cosine similarity over word embeddings. By leveraging lexical representations of both resume and job descriptions, our model measures the contextual closeness between text segments, allowing for more intelligent matching beyond simple keyword overlaps. We also shifted our hosting strategy from Google Cloud Platform (GCP) to a local MySQL database to simplify development and testing.

Achievement and Usefulness

Job-Genie successfully provides users with a streamlined way to upload resumes, receive job recommendations, apply to jobs, track their application status, and receive updates. It significantly reduces the manual effort involved in job hunting. However, the current implementation relies heavily on contextual segment matching, which limits its precision. Our intended advanced semantic matching, still in development, would significantly improve accuracy and usefulness.

Changes to Schema or Data Sources

Initially, our schema included separate tables for resumes and skills extraction. However, for simplicity and performance during development, we integrated resumes directly into the User table as text. This decision simplified our schema significantly and enhanced query efficiency, although it limits advanced data normalization practices we initially planned.Besides consolidating the schema by embedding resume text directly into the `User` table, we enhanced the database design by introducing a trigger-based mechanism to automatically create notifications when a new application is submitted.

Although the core table structures remained stable, the addition of database triggers and stored procedures constitutes a functional extension of the schema.

These changes improved data consistency, reduced application-side overhead, and enforced critical logic directly at the database layer.

Changes to UML Diagram and Table Implementations

In the final implementation, several refinements were made to the UML diagram and table designs to improve consistency and logical flow. We established a clear and explicit relationship between the Application and Notification tables, ensuring that each application event is properly

tied to a corresponding notification. Additionally, we added a justification for maintaining the separation between JobMatch and Application, clarifying that not every recommended job necessarily results in an application event, thus preserving the modularity of user recommendations and application tracking.

The UML diagram was updated to accurately reflect these revisions, and the relational model and SQL/database schema were modified accordingly to implement these structural changes. We also revised our original assumptions and justifications to align with the new design choices, ensuring that the final system architecture is both more robust and scalable.A more suitable design balances data normalization with practical application flow. By introducing a clear relationship between Application and Notification, and keeping JobMatch independent, we preserved modularity and allowed future extensibility without overcomplicating user workflows.

Functionalities Added or Removed

We added:

- An explicit "View Uploaded Resume" functionality for users.

- We expanded the functionality of the job recommendation system by replacing the earlier basic word count and direct keyword matching method with a more sophisticated semantic similarity approach using word embeddings. Instead of relying on exact word matches between resumes and job descriptions, we now vectorize text inputs and compute cosine similarity between them, allowing for far more nuanced, context-aware matching.

We removed:

- Detailed skills extraction due to the complexity and integration time required, intending to revisit it later with proper ML integration.

Complementary Advanced Database Programs

Our use of advanced database programs, specifically MySQL stored procedures, significantly complemented the application's robustness and functionality. The ApplyForJobProcedure handled complex transaction logic, prevented duplicate applications, and automated notification creation, improving the application's integrity and user experience.

Technical Challenges Faced

Technical Challenge (Detailed by Team Member)

Nakul – Handling NULL Foreign Key Constraint Issues

One of the major challenges Nakul tackled was resolving a foreign key constraint issue between the Application and Notification tables. Initially, when an application was inserted without an existing notification, the database attempted to assign a NULL value to the NotificationID field, resulting in a foreign key violation.

To address this, we modified the schema to allow a default value of 0 for NotificationID instead of NULL. This required careful adjustments in both the SQL schema and backend logic to ensure that placeholder values were properly updated once a notification was generated.

This solution maintained referential integrity while allowing asynchronous creation of applications and notifications, a critical improvement for smooth user interactions without transaction failures.

Aaryan – Transitioning from GCP to Local MySQL Setup

Aaryan led the effort to transition the backend database from Google Cloud Platform to a locally hosted MySQL server.

This process involved installing and configuring MySQL on the local development machine, recreating all tables manually, and migrating schema designs to match our updated ER diagram. In addition, backend code had to be refactored to adjust MySQL connectors, environment configurations, and authentication methods to connect locally rather than through cloud-based endpoints.

Aaryan also had to thoroughly test the local server stability, update all database connection pooling settings, and ensure that API routes remained functional with minimal downtime.

This transition provided faster development iterations and reduced dependency on internet connectivity, significantly improving our team's efficiency.

Devansh – Handling Inconsistent Job Descriptions and Implementing ML-Based Matching

Devansh faced the challenge of inconsistencies within the Job table, particularly with the Description field across different job entries. Initially, the matching logic relied on both the Title and Description fields to connect users to suitable jobs. However, due to the irregular quality and format of job descriptions, this approach led to unreliable recommendations.

To overcome this, Devansh restructured the matching system to focus solely on the Title field, which was more consistently populated across entries. To preserve and even enhance the intelligence of the recommendations after dropping descriptions, he implemented a machine learning approach using word embeddings and cosine similarity to semantically match resumes to job titles.

This shift ensured that users still received contextually accurate recommendations while avoiding the inconsistencies and noise present in job descriptions, significantly improving system reliability.

Other Changes from Original Proposal

Originally, we aimed to integrate advanced NLP-based matching early in the development cycle. Due to complexity and time constraints, this was deferred for future improvements. Additionally, UI aesthetics evolved significantly from a basic functional interface to a professional and polished design inspired by existing market applications like ResumAI.com.

Future Work

- Integration of advanced NLP and machine learning (TensorFlow.js) for semantic job-resume matching.
- Automated extraction and utilization of skills and experience data.
- Inclusion of an administrative dashboard for managing job postings and user analytics.
- Enhanced application tracking (interviews scheduled, feedback integration).

Division of Labor and Teamwork

The project was strategically divided according to each team member's strengths to ensure efficient and parallel development:

Frontend Development: Devansh Agarwal led the development of the user interface using React, focusing on responsive design and delivering a seamless user experience.

Backend Development: Aaryan Gusain handled the server-side operations, building API endpoints with Node.js and managing communication between the frontend and database.

Database Schema and Optimization: Nakul Kuppu designed and optimized the MySQL database schema, developed stored procedures, and worked on performance tuning for faster data retrieval.

Integration and Testing: All team members collaborated to integrate the frontend, backend, and database layers. We conducted comprehensive testing to ensure system reliability and smooth user interaction.

Teamwork Approach:

- We held regular meetings to track progress and resolve issues early.
- Responsibilities were clearly defined from the start to avoid overlap and delays.
- After completing individual components, we worked together to review, improve, and integrate each part.
- Consistent communication fostered accountability, timely problem-solving, and a collaborative development environment.

This division of labor and teamwork approach allowed us to move efficiently, leverage individual expertise, and deliver a polished project.