UniCareers
Project Report

Keeron Huang, Libin Wang, Mingyan Gao, Yanzi Li[1]

[1]University of Illinois Urbana-Champaign

Keywords: Job Exploration, College Career, Education

1 | Consistency

1.1 Changes in Direction

Initially, our project was designed to create a platform that allows users, particularly students in their exploratory phase, to customize personal career paths and options. Over the course of development, we have **remained aligned** with this goal. However, **additional insights** from data preprocessing and integration prompted us to enhance some aspects of our platform. For example, we introduced features such as a **page search bar** for finding specific job roles, advanced **filtering** options to narrow down career choices based on user preferences, and a **"favorites" feature** that allows users to mark and revisit jobs or fields they are particularly interested in. These modifications aim to provide students with a more tailored and interactive experience while maintaining the flexibility of the original design.

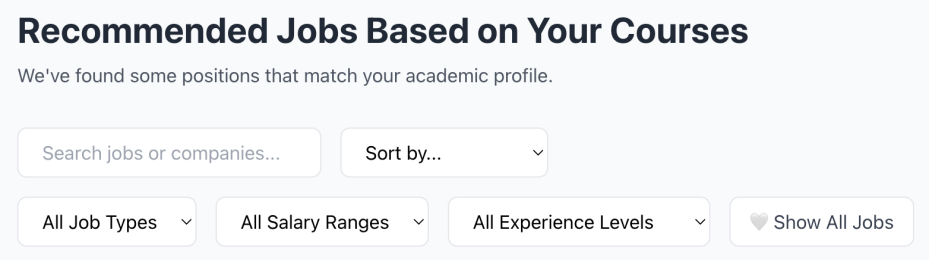

Figure 1: New Feature Example

1.2 Changes in Data

The data sources used in our project, such as SOC codes from O*NET and real-world job data from Kaggle, **adhere to the original design**. We implemented rigorous preprocessing techniques to clean and align the data with our database schema, ensuring compatibility with advanced queries. During this phase, we also restructured certain attributes to improve consistency, such as splitting salary ranges into minimum and maximum values and refining course-skill-job mappings. These refinements not only improved the accuracy of our queries but also enhanced the usability of the platform for end users.

1.3 Changes in ER Model

We do some modifications on our ER model of Stage2. We have claimed the changes in our Stage3 document. The final ER model is exactly the same as the ER model in Stage3.

- **Type of relationship:** We switch the relationships between SOC and Job entities, between Company and Job entities from 'at most one' to 'exactly one'. Since after we observe our datasets, we found that every job can match exactly one SOC code and one company.
- **Change on Job:** We delete the original 'Job title' attribute from Job entity since its information overlaps with the attribute 'role'. Also, we want to provide the users with information about average salary of each company. So we split the original 'Salary Range' attribute into two parts: 'MinSalary' and 'MaxSalary'.
- **Change on SOC:** We delete the original 'Description' attribute from SOC entity since it is a very general concept. The detailed description of a SOC job is actually within other attributes like 'Work task', 'Related Experience', et al.

Below is the new image of our ER model.

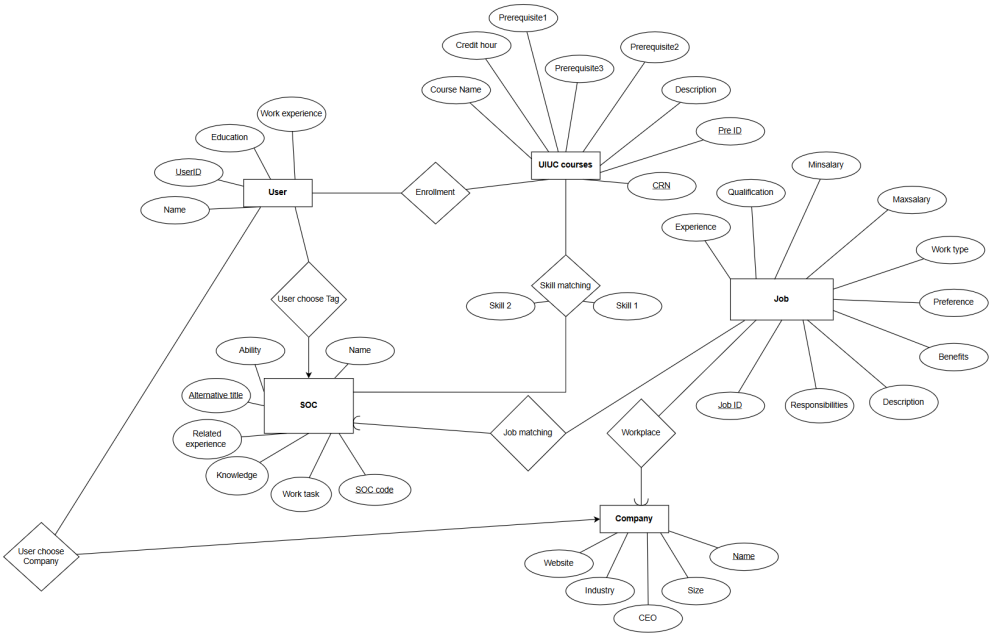

Figure 2: ER Model for Project Database

1.4 Changes in Functionalities

Compared to our proposal, our final App has the following differences in functionalities:

- **Functionalities in "Explore Special":** Our app provides a pilot feature for UIUC ECE students: the user can select Courses, from which the app provides Job information associated with the selected course combination; the user can also select a job category, from which the app provides course combinations associated with the job content in that category. Our final app successfully implemented the first feature. For the second feature we implemented the associated stored procedure, but we ended up not being able to complete the associated Back-end API and Front-end UI, thus we ended up removing this feature.
- **Functionalities in "User Profile":** In the proposal, we wrote about the problem of confidentiality of user information and suggested that we would like to perform an encryption/decryption process on the information entered by the user between entering the database. In our final app, we were not able to accomplish this because we had to do query testing directly against the UserAccount table.
- **Functionalities in "Learn Company":** We initially wanted to let the user directly select the ideal company and find out detailed job information based on the selected company. However, considering the large number of companies in the database, we made a simple leveling in the final app implementation: the user selects Industry first, and then selects a specific company under Industry.

2 | Usefulness Analysis

Our application successfully achieved many of the objectives outlined in our original proposal, though some aspects of the implementation diverged from our initial vision. Key functionalities, including profile creation and updates, interest refinement, job recommendations, and the mapping of courses to related job skills, were implemented effectively. However, the platform's design leaned more toward practical job exploration rather than the comprehensive career guidance we initially envisioned.
Instead of offering broad exploratory features aimed at addressing long-term career uncertainties or niche aspirations, the final product emphasizes structured interactions. These include matching courses to jobs and

providing specific job lists—functionalities that are highly actionable but somewhat limited in addressing less defined career goals. Moreover, although the platform was designed with a focus on career exploration rather than recruitment, some features, such as direct job recommendations, reflect a recruitment-oriented approach. This represents an opportunity for future improvement: integrating more exploratory and mentoring features to better align with the original aim of supporting college students in navigating the unique uncertainties of their career paths.
Despite these differences, our application offers significant value to students. It serves as a solid foundation for career exploration and skill mapping, providing practical tools that facilitate actionable insights. With further development, it has the potential to evolve into a more holistic career guidance platform.

3 | Workflows

3.1 UserProfile

3.1.1 CRUD Features

- `validateAndGetUser`: This function validates the user's credentials by querying the `UserAccount` table based on the provided `userId` and `name`. It retrieves user-specific details such as experience, ideal company, and education if the validation is successful. This ensures only authorized users access their account information.
- `createUser`: This function allows the creation of a new user in the `UserAccount` table. It verifies the completeness of required fields (`name`, `experience`, `idealcom`, and `education`) and prevents duplicate entries by checking existing records. A unique `userId` is assigned to each new user.
- `updateUser`: This function updates the details of an existing user in the `UserAccount` table. It ensures the `userId` exists before applying changes. It also validates the presence of required fields (`name`, `experience`, `idealcom`, and `education`) to maintain data integrity.
- `createUpdateExperienceTrigger`: This function creates a SQL trigger named `update_experience_trigger` to enforce data consistency rules. Specifically, it ensures that a user's work experience cannot be decreased during updates to the `UserAccount` table (because the work experience of a user can only increase year by year). The trigger prevents invalid updates by signaling an error when such changes are attempted.
- `deleteUser`: This function removes a user from the `UserAccount` table based on their `userId`. It first checks if the `userId` exists to prevent accidental deletions. If the user is found, their record is deleted from the database.
- `getMaxUserId`: This function retrieves the maximum `userId` from the `UserAccount` table to ensure the assignment of unique `userId`s to new users. If no records exist, it defaults to returning `0`.

3.2 Company

3.2.1 IdealCompany → Jobs

For this section, we use functions including `getAllIndustries`, `getCompaniesByIndustry`, `getCompanyByName`, `getAllCompanies`, and `getCompanyJobs` to fulfill our goals.

- `getCompanyJobs`: This function retrieves job opportunities linked to the user's selected ideal company. It queries the `Job` table and aggregates details such as average experience, qualifications, salary range, and company preferences based on the `Idealcom` field from the `UserAccount` table. This helps users explore roles within their dream company.

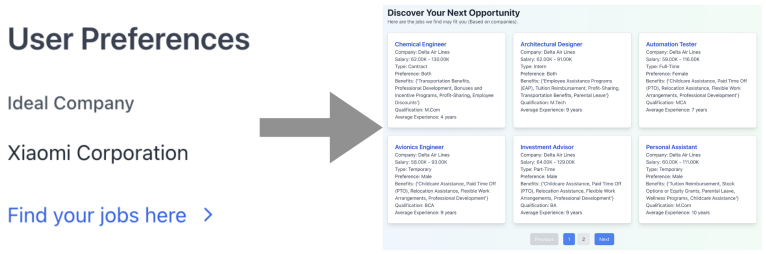

Figure 3: IdealCompany → Jobs Workflow

3.2.2 Customize Company Selection

This functionality allows users to explore and select companies based on industry or other criteria.

- `getAllIndustries`: This function retrieves distinct industries from the `Company` table, enabling users to explore sectors of interest.
- `getCompaniesByIndustry`: By selecting an industry, this function retrieves a list of companies belonging to that industry, allowing users to refine their choices.
- `getCompanyByName`: This function provides detailed information about a specific company, such as its size, CEO, industry, and website. It helps users make informed decisions.
- `getAllCompanies`: This function retrieves all available companies, optionally with pagination (via limit and offset). It provides users with a comprehensive overview of potential company options.

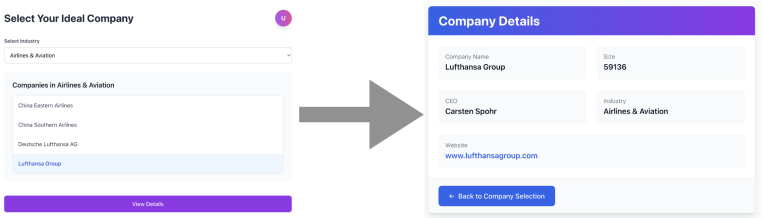

Figure 4: Customize Company Selection Workflow

3.3 TagName

3.3.1 TagName Selection

- `getSOCDetails`: This function retrieves detailed information about Standard Occupational Classification (SOC) codes based on the specified `level` and an input SOC prefix (`inputSOC`). The function operates as follows:
  - Determines the SOC level condition:
    1. Level 1: SOC codes ending in ”0000”.
    2. Level 2: SOC codes ending in ”00” but not ”0000”.
    3. Level 3: SOC codes ending in ”0” but not ”00”.
    4. Level 4: SOC codes not ending in ”0”.
  - Extracts the appropriate SOC prefix from the input SOC based on the specified level.

- Queries the SOC table to fetch details such as SOC_code, Name, RelatedEx, Knowledge, and Ability.

The results are filtered and ordered by SOC_code for structured presentation.

3.3.2 Fetching & Recommendation Page

- initializeTagNameToJobProcedure: This function initializes a stored procedure named TagNameToJob in the database. The procedure performs the following actions:
  1. Updates the TagCode field in the UserAccount table for a specific user, identified by the input InputUserId.
  2. Retrieves job-related data linked to the user's updated TagCode. The results include:
     - JobId, Role, and Type.
     - Aggregated metrics such as average experience, qualifications, salary range, and company details.
     - Additional information such as Preference and Benefits.
  3. Groups the data by job-related attributes for a concise and organized result set.

  The stored procedure simplifies querying and provides relevant job information directly based on user-specific tags.

- callTagNameToJobProcedure: This function invokes the TagNameToJob stored procedure using the provided inputUserId and inputTagName. It retrieves job information tailored to the user based on their TagCode. The results include:
  - Job details such as JobId, Role, and Type.
  - Aggregated data including AvgExperience, MinSalary, MaxSalary, and Qualification.
  - Associated company information such as CompanyName, Preference, and Benefits.

  The function processes the procedure's output to format the results into a structured list of job objects, making it suitable for application use.

3.4 Courses

3.4.1 Customize Courses Selection

- transactionEnrollAndFetchJobs: This function performs a transactional operation that handles course enrollments and fetches relevant job opportunities for a given user. It ensures data consistency by utilizing database transactions. The function operates as follows:
  1. Initiates a transaction using BEGIN.
  2. Clears all existing course enrollments for the specified user (inputUserId) by deleting records from the Enrollment table.
  3. Inserts new enrollments for the user based on the provided list of course reference numbers (crns).
  4. Retrieves job opportunities related to the user's enrolled courses by:
     - Identifying SOCodes (Standard Occupation Codes) associated with the user's enrolled courses through the SkillMatching and Enrollment tables.
     - Filtering SOCodes that require at least two skills (Skill1 and Skill2) to qualify.
     - Matching these SOCodes to job roles via the AlterTable and Job tables.
  5. Aggregates and returns job details, including:
     - JobId, Role, and Type.
     - AvgExperience, Qualification, and Salary Range (minSalary, maxSalary).
     - CompanyName, Preference, and Benefits.
  6. Commits the transaction using COMMIT. If an error occurs, it rolls back the transaction using ROLLBACK to maintain data integrity.

  The results are sorted by maxSalary in descending order, prioritizing higher-paying job opportunities.

4 | Technical Challenges

4.1 Connecting GCP virtual machine to local

Authored by ***Libin Wang***

In order to create a secure isolated code environment, we chose to create VM instances in GCP and install Node.js, Express.js and other dependencies in it. This has the added benefit of allowing direct access to the Stage3 database from the VM's terminal via the MySQL Client and gcloud, making it easier for us to test back-end related SQL queries.
The technical challenge was therefore how to connect to the VM remotely locally: our group chose to connect via ssh keys and integrate the VM into Vscode.

- Firstly, generate ssh key on local computer. The details can be seen from *here*
- Secondly, open Vscode and "Remote-SSH: Add New SSH Host": **ssh USERNAME@EXTERNALIP -i privatesshkeypath**.
- Finally, choose "Remote-SSH: Connect to Host..." to connect the virtual machine just configured

After coming into the virtual machine, every user will have a directory named by the username, in which you only have access to modify. In this case, we should create an shared-access directory, by typing **"sudo chmod -R 777 ./directory"**. It is also recommended to use an script(.sh) file to real-time monitor changes in the directory and granting permissions to all users.

4.2 Backend and Frontend Consistency

Authored by ***Keeron Huang***

Stage 4 involves the implementation of numerous techniques and substantial efforts in developing both backend and frontend codebases. However, maintaining consistency between these components can be challenging, particularly when one advanced query is developed by different team members. To address this issue in future iterations, tasks should be organized by specific procedures or queries. This approach will ensure uniformity in variable names, function usage, and route linking, thereby enhancing both consistency and efficiency across the project.
Our bug logs include multiple instances where variable names did not match yet still compiled, parameters in `services.ts` were assigned different types, and backend functions inadvertently used identical names with different meanings. These inconsistencies not only introduce potential errors but also make the codebase harder to maintain and understand. By implementing standardized naming conventions and clearly defining function purposes, we can minimize such issues and improve overall code quality.

4.3 Writing and Debugging Complex Stored Procedures

Authored by ***Mingyan Gao***

One of the most significant technical challenges I encountered during the project was writing and debugging complex stored procedures for database operations. Stored procedures are essential for encapsulating business logic within the database, improving performance, and reducing network overhead. However, their complexity increases with advanced requirements like aggregating data across multiple tables, implementing conditional logic, and ensuring transactional consistency.
The challenge was twofold:

- Writing a correct stored procedure that meets the requirements.
- Debugging the procedure to identify and fix logic errors, especially when the database does not provide comprehensive debugging tools.

To address this challenge, I followed these strategies:

1. **Incremental Development:** I broke down the stored procedure into smaller, manageable steps. Each step was tested individually using `SELECT` statements and temporary tables to verify the correctness of intermediate results.
2. **Extensive Logging:** Within the procedure, I used temporary tables or output `SELECT` statements to log intermediate variables, query results, and conditional branches. This approach helped trace where the logic deviated from expectations.
3. **Mock Data:** To test edge cases, I created mock data in a development environment that simulated real-world scenarios. This allowed me to evaluate the procedure's performance and correctness under various conditions.

Future teams working on similar projects are advised to:

- Develop stored procedures incrementally, testing each step thoroughly before moving on to the next.
- Familiarize themselves with their database's debugging and error-handling capabilities.
- Maintain clear documentation of procedure logic and expected inputs/outputs for easier maintenance and collaboration.

Writing and debugging stored procedures was a valuable learning experience that underscored the importance of planning, testing, and clear documentation in database development.

4.4 Connecting Frontend and Backend

Authored by ***Yanzi Li***

A key challenge in our project was ensuring seamless communication between the frontend and backend systems. This integration is critical as it allows user actions on the frontend to trigger backend processes and return appropriate responses. The challenge was rooted in mismatched data formats and inconsistent API specifications. To address these challenges, our approach involved a combination of structured API documentation, frontend-backend synchronization, and robust error-handling mechanisms.

- We established a clear API contract by documenting all backend endpoints, including their required inputs, expected outputs, and potential error codes. This documentation ensured that both the frontend and backend teams adhered to consistent data formats.
- We implemented logging mechanisms to debug API calls during development. For example, network requests and responses were logged on both frontend and backend sides to identify and resolve any inconsistencies.

As a result of these steps, we successfully integrated the frontend and backend, ensuring a smooth user experience. For future teams, it is highly recommended to invest in early API documentation, use tools like Postman for endpoint testing, and employ consistent logging practices to debug integration issues effectively.

5 | Future Works

5.1 TagName - Job

To enhance the accuracy and comprehensiveness of the job recommendations associated with specific tags, it is essential to expand the dataset for job information. Currently, our database contains 4,818 rows of job entries, which limits the ability to map some TagName entries to specific job roles. By incorporating additional data sources and increasing the volume of job-related information, we aim to address this limitation and provide users with a broader range of career options.

5.2 Course - Job

Further development is required to strengthen the relationship between courses and job recommendations. This includes:

- Conducting deeper research into the connections between user profiles, particularly their enrollments, and corresponding job opportunities. This can help identify more precise job matches based on the skills acquired through specific courses.
- Developing a feature for recommending courses based on user-selected jobs. This will allow users to identify the courses they should take to qualify for their desired roles.
- Enhancing the interactions between the user's enrollment history and job recommendations to provide a dynamic, personalized experience that evolves as users update their academic or professional profiles.
- Expanding the dataset for courses to include more majors and disciplines, ensuring the platform can support a diverse range of academic and career paths.

By addressing these areas, the platform can provide a more comprehensive and personalized experience, bridging the gap between academic planning and career exploration.

6 | Work Division

The responsibilities for the project were divided among team members based on their expertise and the project's requirements:

- Keeron Huang: GitHub & GCP management, backend development for ST4 CP1, and frontend implementation for ST4 CP2.
- Libin Wang: GCP maintenance, Frontend development for both ST4 CP1 and ST4 CP2.
- Mingyan Gao: Routing for ST4 CP1 and ST4 CP2, and backend development for ST4 CP2.
- Yanzi Li: Routing for ST4 CP1 and ST4 CP2, and backend development for ST4 CP2.

This clear division of labor allowed each team member to focus on specific aspects of the project, ensuring efficient progress and high-quality results. Teamwork was managed effectively through regular meetings, task tracking on GitHub, and consistent communication to address challenges and align on goals. Each member contributed their expertise, which helped maintain a balanced workload and cohesive development process throughout the project.