Project Report

1. **Please list out changes in the directions of your project if the final project is different from your original proposal (based on your stage 1 proposal submission).**

   In fact, we have implemented almost every function from our original proposal, what we only didn't achieve is the visualization of ranking metric comparison.

2. **Discuss what you think your application achieved or failed to achieve regarding its usefulness.**

   We have successfully implemented CRUD functionality for universities, ranking metrics, user profiles, admissions data, and comments, ensuring comprehensive management of key platform components. Additionally, role-based access control has been integrated, allowing only administrators to update or delete university information. This distinction helps maintain data integrity and prevents unauthorized modifications.

   For Ranking Search and Filtering, we have introduced advanced features that allow users to search university rankings by keyword, country, source (e.g., QS or Times), and apply academic reputation filters (<30, 30-60, >60). These enhancements provide users with a more tailored and efficient way to explore rankings based on their specific criteria.

   For Admission Results, we allow users to post their own admission data, creating a valuable repository of real-world application insights. Users can also search the admission data by keyword, making it easy to find relevant results based on specific universities, programs, or other criteria. This feature promotes knowledge sharing and helps prospective applicants make informed decisions.

   Moreover, to promote engagement and track institutional interest, the system automatically increments a university's popularity metric each time a user creates a new comment associated with that university. This feature not only enhances user interaction but also provides valuable insights into trending universities.

   However, we have not yet achieved the ranking metric comparison between different universities with a line graph visualization. This feature is essential for providing users with a dynamic way to compare ranking trends across multiple universities visually. Implementing this capability remains a priority, as it will significantly enrich the platform's analytical and decision-making tools by offering users a clear and engaging representation of historical ranking data.

3. **Discuss if you change the schema or source of the data for your application**

   Initially, our database design plan included five tables: AdmissionData, Comment, RankingMetric, University, and User. During the implementation phase, we realized the need to store data related to users' favorites. To address this, we added a new table, Favorite, to record users' university favorites. Additionally, we made minor modifications by adding userID to the AdmissionData and Comment tables. This change was implemented to streamline data insertion and enhance database maintenance by enabling better tracking of user-related data.

   Regarding the source of data, our initial plan was to integrate existing university ranking data. We collected rankings from QS and Times, which formed the foundation of our ranking-related tables. However, as we introduced new tables, such as the Comment table, which contains fields like learningAtmosphere, library, and restaurant, we faced a challenge. These fields ideally require user-generated data, but due to current limitations, we opted to auto-generate this data as a temporary solution. While this is not ideal, it allows us to meet our application's data requirements and simulate realistic user interactions.

4. **Discuss what you change to your ER diagram and/or your table implementations. What are some differences between the original design and the final design? Why? What do you think is a more suitable design?**

   During Stage 2, we utilized a UML Diagram to design our database. Initially, our diagram included only five tables, but we later added the Favorite table. Although we originally defined a direct relationship between the User and University tables for the favorite functionality, we did not initially create a dedicated table for storing favorite data.When we reached the data insertion phase, we realized that storing user favorites was crucial. To address this, we redesigned our schema to include a Favorite table, which stores the userID and the universityID. These fields correspond to the User and University tables, respectively. This change proved to be a more suitable approach, as it aligned with the requirements of our application. For instance, our application allows users to log in, browse universities, and save their favorite universities for easier access in the future. The ability to store this data is a key feature to enhance user engagement, making it a significant addition to our database.

   Another difference between the original and final design lies in updates to table parameters. Specifically, we added a userID column to both the AdmissionData and Comment tables. These additions were made to improve database management. By associating data in these tables with a specific user via userID, we gained the ability to efficiently control each user’s data. This allowed us to perform CRUD (Create, Read, Update, Delete) operations

more effectively. This change not only enhanced the efficiency of database management but also reduced the cost of maintaining the database. By improving the ability to track and manage user-specific data, we made the database more robust and better aligned with the needs of our application.

5. **Discuss what functionalities you added or removed. Why?**

   Functionalities Added: Ranking Search and Filtering: Added the ability to search for university rankings by keyword, country, source (e.g., QS or Times), and Academic Rep filter (<30, 30-60, >60). Why: This functionality was added to improve the usability of the application for users who need to filter and locate specific universities based on their preferences. Filtering by key metrics such as Academic Rep ensures a targeted search experience, which can help users to locate the more exact preference they care about.Manage Ranking Data (Admin Only). We implemented a comprehensive system for administrators to manage ranking data directly through the UI. This includes the following functionalities: Add Ranking Data: A button and modal interface for adding new ranking entries. Edit Ranking Data: A modal interface that allows editing of all relevant ranking fields (e.g., universityName, source, academicRep). Delete Ranking Data: An action button to delete existing ranking entries. Why: Previously, administrators had to update the database manually, which was both time-consuming and prone to human errors. By introducing a unified interface for managing ranking data:

   Add: Simplifies the workflow for creating new entries, ensuring data consistency and reducing the likelihood of missing fields or formatting issues.

   Edit: Provides flexibility to correct or update rankings without needing direct database access, enabling more efficient handling of dynamic data changes.

   Delete: Facilitates the removal of outdated or incorrect entries, ensuring the database remains relevant and accurate. This integrated solution improves efficiency, minimizes the need for technical expertise, and maintains data integrity across the application. Validation for Adding Rankings: Implemented a pre-check to ensure the university exists in the University table before adding a ranking. Why: This prevents foreign key constraint errors and ensures database consistency.

6. **Explain how you think your advanced database programs complement your application.**

   The advanced database programs are integral to our application, enabling it to provide an efficient, and reliable user experience. These programs ensure robust data interaction by supporting features like dynamic filtering and searching, which allow users to refine ranking data based on criteria such as country, source, and academic reputation. This enhances the application's usability by making data retrieval fast and intuitive. Furthermore, the use of

paginated queries ensures that the application remains scalable and responsive, even when dealing with large datasets.

For administrators, the database programs provide comprehensive support for CRUD operations, enabling the addition, editing, and deletion of ranking data directly through the application interface. These operations are complemented by robust validation mechanisms, such as ensuring that universities exist before adding related ranking data. This safeguards the integrity of the database and prevents the entry of invalid data, while also simplifying administrative workflows by eliminating the need for manual database interventions. Performance optimization is another critical aspect of the database programs. Efficient query design, combined with the use of indexing and optimized transaction management, ensures that data retrieval and updates are fast and reliable. For instance, advanced sorting capabilities allow users to order data by metrics like academic reputation, providing a more tailored and interactive experience. Transactional integrity further guarantees that operations like data updates or deletions are completed successfully, maintaining consistency across the database.

The database programs enhance user engagement by supporting features like the ability to mark universities as favorites. By managing user-specific data efficiently, the programs enable personalization features that make the application more engaging. Additionally, features like foreign key constraints ensure that relationships between entities, such as rankings and universities, are consistently maintained, preventing issues like orphaned records.

The advanced database programs complement the application by providing a strong, efficient, and flexible data foundation. They enable the application to deliver a high-quality user experience, support administrative workflows, and maintain performance and scalability, ensuring that both end users and administrators can interact with the application seamlessly and effectively.

7. **Each team member should describe one technical challenge that the team encountered. This should be sufficiently detailed such that another future team could use this as helpful advice if they were to start a similar project or where to maintain your project.**

   yhb : One of the major challenges I encountered was implementing the "favorite" functionality, which allows users to bookmark universities of interest. Initially, our database design lacked a dedicated Favorite table, making it difficult to manage user favorites effectively. During the development phase, we realized the need for a well-defined structure to store userID and universityID associations. Without such a structure, the application frequently encountered issues like duplicate entries and synchronization errors between

the frontend and backend. The most challenging aspect of implementing this feature was handling the deletion of user favorites. Initially, when users clicked to remove a favorite, the frontend correctly reflected the change, but the backend database failed to delete the corresponding record. As a result, attempting to re-favorite the same university triggered a 500 Internal Server Error. At first, I assumed the issue was with the adding logic. However, through debugging, using console logs to report errors, and examining the database state, I discovered that the deletion SQL query was not executed properly. The biggest challenge was debugging this issue because it required inspecting multiple layers of the application to identify the root cause. My advice to teams undertaking similar database projects is to frequently use console logs to display the execution status at various stages. By verifying functionality step by step, teams can identify and resolve issues early, avoiding the time-consuming process of debugging multiple layers later in the project.

Xingyuan Liu: The most challenging technical issue I encountered was implementing keyword search functionality for admission data sharing. Users may input search terms related to different fields. For example, a user may want to access admission data for a specific university, program, or admission season. Simple SQL queries targeting particular fields couldn't provide sufficient flexibility for this requirement. Therefore, I implemented a solution that treats each input word as a separate keyword and performs fuzzy searching on these keywords through a stored procedure. While this approach successfully returns results based on keyword input, there's room for improvement in the word division process. For example, currently "Computer Science" is treated as two separate keywords "Computer" and "Science," which can lead to some irrelevant results. Future teams can build upon this keyword search implementation and develop an enhanced solution for word tokenization that better handles user input.

ZhichengTong:One of the major technical challenges I encountered during this project was ensuring data consistency between the University table and the RankingMetric table, which are linked via a foreign key relationship. Specifically, when adding a new ranking to the RankingMetric table, the corresponding university must already exist in the University table. Failure to ensure this results in a foreign key constraint error, which can disrupt the workflow of administrators trying to add rankings. The foreign key constraint between RankingMetric.universityName and University.universityName ensures data integrity but introduces complexity in maintaining and managing dependent data. For example: If an administrator attempts to add a ranking for a university that does not exist in the University table, the operation fails. The administrator must manually verify the existence of the university before proceeding, which adds friction to the data entry process. This issue became evident during the development and testing phase when administrators

reported errors while attempting to add rankings without corresponding university entries. To address this challenge, I implemented the following measures: Pre-check for University Existence: Before adding a new ranking, the system first checks whether the university exists in the University table. If the university does not exist, the API returns a user-friendly error message prompting the administrator to add the university first.

Yuansui Xu: One of the most challenging problems I encountered during this project was ensuring that a university name entered when creating a new comment already existed in the database. This validation was critical to prevent comments from being associated with non-existent universities. Initially, I attempted to handle this by performing a direct lookup in the database before saving the comment. However, this approach led to performance bottlenecks when handling large datasets and introduced inconsistencies in validation across different parts of the application.To solve this problem, I redesigned the process to eliminate the need for the user to manually enter the university name. Instead, the application retrieves the list of existing universities and presents it to the user for selection. By choosing a university from this predefined list, the risk of associating comments with non-existent universities is eliminated. This approach also improves the user experience by reducing the potential for errors and streamlining the process. The implementation ensures data integrity and consistency while minimizing database lookups during the creation of new comments.

8. **Are there other things that changed comparing the final application with the original proposal?**

   Yes, there were several changes made to the final application compared to the original proposal as the project evolved.The original database design was simplified, but as the project progressed, additional constraints like foreign keys were implemented to maintain data integrity. For instance, a foreign key constraint was added between the RankingMetric and University tables to ensure that ranking data is linked only to existing universities. This was not explicitly outlined in the proposal but became a crucial part of the development process.The original proposal did not include real-time updates. However, in the final application, administrators could see changes reflected immediately after adding, editing, or deleting data, enhancing the workflow efficiency and usability.The original proposal did not explicitly mention role-based access control. In the final application, user roles were introduced to differentiate between regular users and administrators. Features like "Favorite Universities" were tailored for regular users, while administrators were granted access to advanced actions like adding, editing, and deleting

ranking data. Error handling was significantly improved compared to the proposal. For example, validations were added to ensure that universities exist in the database before ranking data is added. Similarly, error messages were displayed as user-friendly modal alerts instead of being logged to the console, providing better feedback to users and administrators.

9. **Describe future work that you think, other than the interface, that the application can improve on**

   I think we can expand the functionalities of the existing system. To recommend universities similar to users' favorite universities is a good choice. Users with similar preferences often show interest in related universities. Therefore, we can recommend potential good-fit universities to users based on the user behavior data. For example, we can integrate a graph database (Neo4j) to implement the recommendation system, as it can find relationships between users and universities easily. We can recommend universities or programs based on users' actions, such as suggesting universities that are frequently collected by users who have similar collection patterns. This will help users discover appropriate and suitable universities and programs more efficiently.

10. **Describe the final division of labor and how well you managed teamwork.**

    We generally divided our work according to functionalities. The detailed division is listed below.

    Xingyuan Liu: Established the full-stack architecture, implemented complete CRUD functionalities for both user management and admission data parts, and finished the recommendation part that is based on admission data.

    Hongbin Yang and Zhicheng Tong: Implemented CRUD functionalities for university ranking and user favourite(user can collect university in their profile).

    Yuansui Xu: Implemented CRUD functionalities concerning University Info and comments.

    We divided tasks according to specific functional needs. Afterwards we implemented these functionalities and merged our code on github. We have a meeting weekly, aligning our progress and discussing challenges.