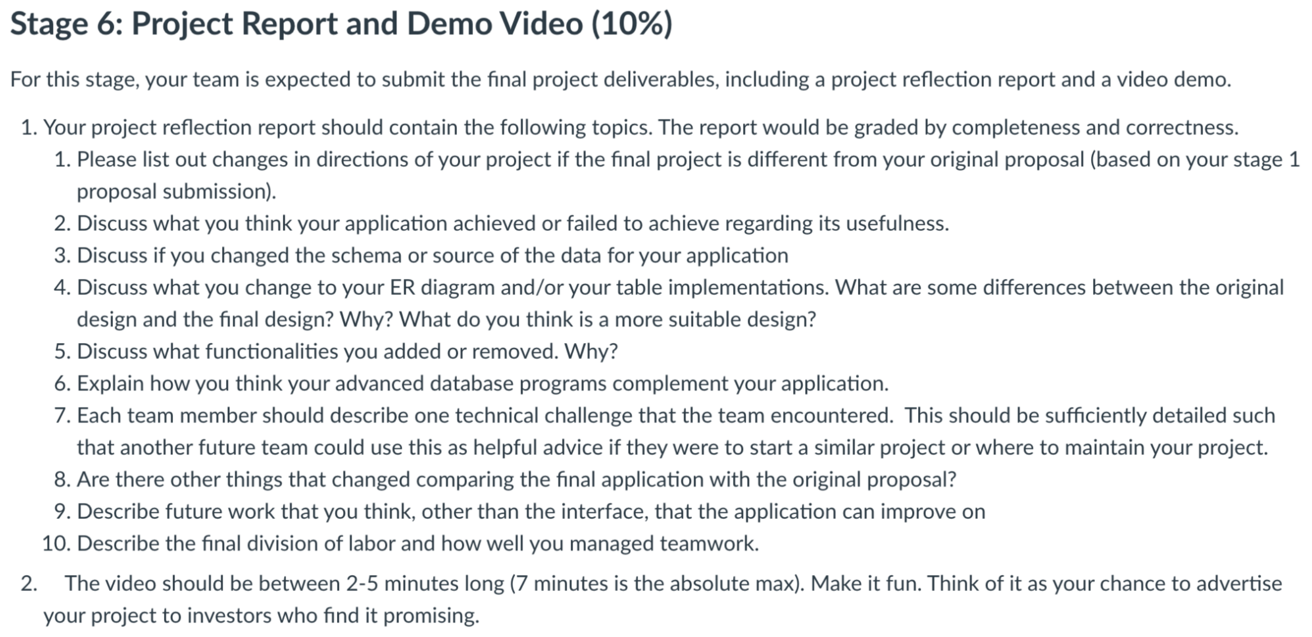

Stage 6: Project Report and Demo Video (10%)

For this stage, your team is expected to submit the final project deliverables, including a project reflection report and a video demo.

1. Your project reflection report should contain the following topics. The report would be graded by completeness and correctness.
    1. Please list out changes in directions of your project if the final project is different from your original proposal (based on your stage 1 proposal submission).
    2. Discuss what you think your application achieved or failed to achieve regarding its usefulness.
    3. Discuss if you changed the schema or source of the data for your application
    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?
    5. Discuss what functionalities you added or removed. Why?
    6. Explain how you think your advanced database programs complement your application.
    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.
    8. Are there other things that changed comparing the final application with the original proposal?
    9. Describe future work that you think, other than the interface, that the application can improve on
    10. Describe the final division of labor and how well you managed teamwork.
2. The video should be between 2-5 minutes long (7 minutes is the absolute max). Make it fun. Think of it as your chance to advertise your project to investors who find it promising.

1.

Overall, the main scope, direction, and ambitions of our project remained the same. We aimed to create a web app that will allow students at the U of I to match with other students through common interests, RSOs, major, and shared classes. We did this by generating a list of people, as matches, who could be compatible with them through our application, which the app ended up achieving. The motivation for the schema in our database also remained the same. In the end, the general idea of the project was unchanged as we were able to accomplish the goals that we set out in the original proposal.

2.

We believe our application achieved all of our goals regarding its usefulness and the features that it provides. It does all of the things that we wanted it to do in our original proposal. It allows a student to make an account and input their majors, interests, classes, and the RSOs they're in. They can edit/change any of these things whenever they want to. From there, it generates a list of potentially compatible students to match with based on these inputs. We also got the app to store the user's current classes that they are taking, which benefits students by allowing them to find study partners who also have similar interests outside of school. Our app allows this all to happen in one place by simply having the user input their preferences and then display students who share common interests, classes, or majors.

3.

The source of the data for our application is still the same as before; the app still uses the same python script to automatically generate thousands of tuples for each table in our database hosted on GCP. For the fifth stage, we decided to add the friend request feature (Requests, Friends to the Students table) as well as the GradYear attribute to our Major table. This was so we could build on the existing functionality of our application, setting the groundwork for additional features we planned to add.

4.

We realized that the initial schema and usable data we had was not enough for the context of this project and the goals we had in mind. For this reason, we experienced some difficulties gathering all friends/matches in one place for the user with our original schema. Without columns for students' friends or their friend requests, we didn't have a way of storing friends that the user can add from their matches. Having all the user's matches in one place and storing them as friends would be easier for them to keep track of the people they want to connect after filtering out their matches. We also wanted to allow the user to send friend requests should they want to connect with a match, but all of this wasn't possible with our original design. Therefore, we added two new columns called 'Friends' and 'Friend Requests', which are essentially a long string that contains the NetIDs of the user's friends separated by the comma. We added a column called 'GradYear' to indicate the cohort of a student. This way, the user can easily tell if a match of theirs is in the same grade which can aid them in determining if a match is compatible enough for them, and it's also factored into who the user gets matched with. With these changes, we believe this was the more suitable design for our database.

5.

Since the goal of our app is to match students, we added friend requests in order for students to match with each other. This meant we had to add two more columns for a user's current friend requests and current friends. When a user submits a request after a match, it fills out the other user's current friend requests table. This will be visible on the front end of the other user, showing that there is an incoming friend request. Upon approval, the friend request user now becomes a friend.

6.
Our team went with the stored procedure + trigger option. Initially, we tried creating a trigger that would delete all students who already graduated from all tables upon deleting an entry from the Students table. However, we felt that deleting students who already graduated wasn't necessary because it's still possible for them to remain on campus with certain situations. Also, we encountered technical errors with the message 'Can't update table 'Students' in stored function/trigger because it is already used by the statement which invoked this stored function/trigger.' This will be further covered in #7. Instead, we thought having a trigger to indicate that they're an alumni by updating a boolean value in the Majors table was a better idea overall.

Our stored procedure allowed us to have an easy way to execute multiple complex SQL queries and was very convenient in the context of our application. We could simply specify a NetID and the number of matches desired, and our stored procedure would return them. It also was perfect for returning mutual friends between the user and a match they have, which was a feature that we felt would add to the user experience. The stored procedure made our lives much easier when writing the code for the back end of our application.

7.
Georges Durand – When creating the back end for edit profile, I struggled with having the currently stored data for the user show up on the frontend. I ended up creating a GET request to GCP and used json data to populate the front end. Once a user edited their fields, clicking the submit button sends a POST request that would update the specified tables.

Luke Zhang - A technical challenge I encountered was figuring out how to fix a certain error when developing the trigger. The error was 'Can't update table 'Students' in stored function/trigger because it is already used by the statement which invoked this stored function/trigger.’ Initially, I made a trigger that deletes all students who graduated from the Students table everytime the Students table was updated. However, although the operations were different (UPDATE vs DELETE), the trigger still didn’t work and would infinitely loop, displaying that error message. I realized that the trigger cannot modify the same table where the event (INSERT, UPDATE, DELETE) was happening. As a group, we agreed that it wasn’t necessary to completely delete all students who graduated from the database. We could account for situations that alumni could still be on campus such as: working at Research Park, taking a gap year before pursuing graduate school, volunteering, or students who graduated early and are still on campus. We added an ‘Alumni’ column that’s a boolean to the Majors table with the value ‘1’ indicating if they graduated and ‘0’ if they didn’t. We wrote a trigger that would update the Majors table by setting the Alumni value to ‘1’ for every student that graduated in or before 2022 upon any UPDATE on the Students table.

Akshay Ghosh - One technical challenge that I encountered as I was demoing the app was figuring out how to concisely and efficiently display friend requests and current friends on the Home page. It was also a bit of a challenge to modify the existing schema to account for the friend request/friends capability of our application. I had to refactor a lot of existing backend and frontend code to allow the friend request feature to work properly.

James Huang - A technical challenge that I encountered was getting our stored procedure to correctly return a list of potential matches with their mutual friends. We wanted to have a condition that checked for a certain flag to decide what pool of potential matches the user wanted to use and declare cursor based off of that, but since the cursor declaration didn’t like that, we opted to modify our cursor’s subquery to include those flags. Since mysql doesn’t support storing a list and creating a concatenated list of friends seemed impractical for future use, I opted to instead just have multiple lines for each match with a separate mutual friend. Also, since mysql doesn’t support the intersect operator, I had to instead figure out a way to emulate it using other operations, which I figured out includes a subquery and an exists clause since there were many conditions we were checking on.

8.

There are no other things that changed when comparing the final application to the original proposal. All changes have already been addressed earlier.

9.

One aspect of the application we can improve on is the time it takes to fetch all the classes from the UIUC database when using Pandas. Currently, we have a web scraping function that would fetch all classes from the UIUC GPA dataset using Pandas and BeautifulSoup. The issue is that whenever the user is selecting the classes they're enrolled in, the app sometimes lags because the web scraping function runs every time the dropdown is selected. We could fix this by making a bucket in our database just for classes and fetch them from there.

Furthermore, another feature we could improve upon is the matching algorithm for the friend requests to make it more complex.This would most likely require a refactoring of our schema as well as the data that we currently have in the database.

10.

Akshay Ghosh and Luke Zhang worked primarily on the backend (SQL side of things & GCP, the database design, schema, triggers, and stored procedure), as well as getting the data and routes up and running for the Flask app in Python.

James Huang and Georges Durand worked on getting the routes for the friend requests and friend functionality and worked mostly on the front end of our application using React and MaterialUI.

As a group, we collectively agree that we managed our teamwork well and stuck to our intended division of labor as set out in our original proposal.