CS 411 Project Report

Team 22 - Rahul Bansal, Emily Chen, Meghana Gopannagari, Dan Nguyen

Project Direction:

In terms of the direction of our project, Table for Two, the application turned out the way we intended for the most part. We intended for our app to be a way for people to match with other users with similar preferences and explore restaurants in New York City, which is exactly what our final product turned out to be.

Project Usefulness:

In regards to the app's usefulness, Table for Two was able to accomplish the goal of encouraging users to explore new restaurants with other people with common interests. Users are able to create profiles and input preferences on the app, which will then match them with another user and a restaurant in NYC that matches their preferences. Additionally, matches can be refreshed until the user is satisfied with their match. Then, a reservation will be made at the restaurant on the date and time noted on the match. This experience successfully encourages users to step out of their comfort zone by meeting new people and enjoying new dining options.

We initially wanted to implement more advanced features, such as embedding a map into the website that would show where the restaurant is located relative to the user, adding a feedback system to allow users to rate their experience rather than simply scraping the data off a dataset, and creating a chat box for the matched users to communicate with each other. However, we instead decided to allocate more time towards ensuring that the core of our application and its interface with the backend database worked effectively. If we were to continue working on this project, we would also develop a more thorough system that saves user matches even when they log out of the application.

Schema/Data Sources:

The schema in our final product is different from what was initially proposed, since we had a better understanding of the specifics as we continued working. In our initial proposal, we had only three tables in mind: Restaurants/Reviews, User Information, and User Login. In our final product, we had Restaurants, Reviews, User, Matches, and Reservation tables. We added more tables because we realized how to organize our data better and improved upon our understanding of relations. The data source remained the same, as we ended up using the NYC Restaurant Food Order & Delivery Dataset from Kaggle for the Reviews and Restaurant tables. However, we ended up auto-generating more data than expected to fulfill the 1000+ rows requirement. We did this for the User and Restaurant tables.

ER Diagram Design:

Since our end product was similar to our initial design, the ER diagram we created was an effective design for the construction of the necessary tables and their relationships with each other. The only attribute that we changed was replacing the user's maximum budget from a VARCHAR, with the options being $, $$, or $$$, to their maximum budget as a number. This change allowed our queries to better match the user's budget with others, rather than internally developing an arbitrary method for converting the user's budget into one of three categories. The most noticeable difference between the ER diagram and our implementation was in the construction of the tables. While our diagram simply illustrated the data stored in each table and their relationships to each other, filling the tables with the necessary data proved to be another challenge. Specifically, we had to calculate the average restaurant rating using the Reviews table, just so that we could correctly populate the Restaurant table entries with the correct information.

Functionalities:

Most of the functionalities that we intended the app to include were implemented. This includes adding new users through the sign up process (includes preferences); creating, reading, and deleting matches; creating, reading, and updating restaurant ratings; creating and reading reservations; and creating and reading reviews. Additionally, users are able to refresh the match to receive a new one. We did not have the functionality of deleting reservations as mentioned in our original proposal, but this is because we found that since users could regenerate matches until they were happy with the match and date/time of the dinner, there was no need to delete reservations. We figured that the choice of accepting a reservation should be finalized, since the user is free to find a new match if they are not satisfied with any aspect of their match.

Advanced Database Programs:

The core of our application was our advanced queries that matched a user to another user based on their preferences, and matched the pair to the restaurant that best suited them. Rather than trying to find a user and restaurant that exactly matched the user's preferences, we designed our queries in a way that guaranteed that a match would be made and that the user/restaurant match would be the best possible one. Furthermore, our trigger prevented the user from accidentally submitting their information twice, and thus, being matched to themselves. This increased the quality and usability of the application. Finally, the transaction and stored procedure, which uses another two of our advanced queries allowed the user to see the best review of the restaurant as well as its average rating. This is useful information that a user would likely want to know.

Technical Challenges:

1. Rahul: The most tortuous problem that I faced was ensuring the user matching query that I created would be guaranteed to find a match while also ensuring that the match is the best possible. This required me to explore the capabilities of MySQL beyond just what we have learned in class, such as through the use of case statements. Since these complicated queries used information from multiple tables that were highly integrated with each other through the use of foreign keys, I had to carefully join the tables in a way that would maintain the correct correlation between the data in different tables. To do this, I often used natural join and would refer back to the ER diagram we created to check that I was joining tables on the foreign key connecting the data in two tables together.
2. Emily: A technical challenge that I faced while working on this project was having to adapt the frontend and designs to match the functionality of our backend. While we developed the application, our database design and corresponding SQL queries were constantly evolving. Thus, some of the functionality of the original designs I made had to be scrapped or adapted to account for these changes. Even following our planning process and into our development process, the frontend code had to be rewritten to match the capabilities of our database design. This process required a lot of creativity and flexibility throughout the project's completion.
3. Meghana: The most difficult aspect of this project was the backend. Many of the aspects of our application required having multiple queries after one another, but the query function used in NodeJS is an asynchronous function, so the results do not always come in order. As a result, I had to spend a lot of time debugging code and modifying it to handle this asynchronous behavior. One recommendation that I have is to create multiple

API endpoints rather than having one endpoint run multiple queries. This way you can wait for the connections to complete and send a request to the next endpoint only when you are ready. Alternatively, you can nest the connections together or use callback functions on the backend, but this gets more complicated and more difficult to debug.

4. Dan: One technical challenge we encountered was being able to run the app on different machines. Each of our systems had different software requirements, so we ran into problems getting MySQL Workbench, Visual Studio Code, and the app's dependencies to function the same way. To combat this issue, I recommend working together to make sure that everyone is on the same page each step of the way when running code and installing software and packages. In the worst-case scenario, I recommend finding a backup plan, such as having at least two out of the four systems that can run the app, especially for aspects of the project that rely on smaller teamwork, for more accessible and convenient testing.

Changes from the Original Proposal:

In our original proposal, we intended for there to implement a creative component to the app, which would be a chat feature. We wanted to make our app unique and interactive compared to other restaurant finders by allowing users on the app to interact with one another and facilitate conversations. This would allow users to get to know each other better before going to the restaurants, or even allow users to find different restaurants together. Unfortunately, we were unable to add this feature in time, but it would definitely enhance the user experience.

Future Work:

Although the application was successful, there are still many features that can be enhanced. One feature that can be added is the ability for users to leave a review on the restaurant after dining. In our project, we used a real dataset of reviews to make our Reviews table, and from there, we generated the average rating and found the top reviews for each restaurant. However, we did not give the user the ability to create their own reviews. Therefore, in the future, giving the option for users to input a rating and review that would actually be added to the Reviews table would make the app more current as users continue to use it.

Additionally, a modification that could be made to the app is to allow the users to either keep the restaurant or the person from the match for each potential match. Currently, the app is set up such that when making a match, the person and the restaurant are chosen together, and when generating a new match, both the restaurant and person will likely be new. However, we could add a feature such that if you are happy with either the user or the restaurant from the match, you can choose to keep it and then randomly generate the other aspect of the match.

Some other features we could add are a built-in map to find the address of the restaurant and a chat feature to communicate with the other user before dining. Overall, with these changes, the app would be enhanced and more interactive.

Teamwork:

For our teamwork, we found it was easiest to divide the tasks based on each person's strengths. This turned out to be an effective method since there was a pretty even amount of work done by each person for each stage. We met up at least twice for each stage and maintained communication via messaging to ensure that everyone was aware of their responsibilities and could provide updates.

This was our final breakdown for the final product:

Rahul: SQL Query Development and Testing

Emily: Frontend/UI Design

Meghana: Backend setup, Connecting Frontend and Backend

Dan: SQL Query Development and Testing