Team 065 - Theo Mamalis, Aviral Halsi, Neil Prabhu

CS 411 Project Reflection

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).**

   There were no changes in the direction of the project proposal based on stage 1.

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

   Regarding its usefulness, the application was successful in allowing home-cooks to register to the application, as well as restaurants. Also, in being able to include various types of cuisines. The application also allows for its users to check where each vendor is located. Moreover, for every restaurant, a classification system which classifies each vendor as expensive or not was made. All in all, the application extended the type of cuisines offered by including home cooks, and also allowed home cooks to showcase their cooking skills or make profit out of their cooking, in this way extending the food-service network both temporally and spatially.

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

   The source of the data was not changed, that is, it remains a mix of real and synthetic data.

   The schema was changed. In specific, a new relation called UserAuthentication was introduced which now serves as the login page for each user of the application (either a customer or a vendor). This did not exist during stage 1.

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?**

   The change in the table implementations are shown below. Strikethrough means removed, and bold means added:

   - User(UserId, PhoneNumber, Address, FirstName, LastName ~~,Orders, OrderId, Complaints~~)

- Orders(OrderId, VendorId, **UserId, Item, OrderDate, Ratings, Total, TransactionId**)
- Vendor(VendorId, PhoneNumber, Address, FirstName, LastName, VendorName, Type , ~~Menu~~)
- MenuItem(VendorId, ItemName, Price)
- Coupons(CouponId, DiscountPercentage ~~,expiryDate~~ **, Active,** ~~UserList, VendorList,~~ **UserId, VendorId**)
- **UserAuthentication(UserId, EmailAddress, Password)**

Therefore, some attributes were moved to other relations because they were related more to these relations (e.g. OrderId from stage 1 was moved to Orders), and some attributes were introduced to make the schema more realistic and also satisfy integrity constraints (E.g. UserId in the Orders table, compared to Stage 1).

The ER diagram did not change except for some attributes that were altered in each table. For example, in the Coupons table, the UserList, VendorList attributes were made UserId, VendorId respectively. This is because the former attributes were envisioned to be represented as JSON types. However, by appropriately changing the lists and making them Ids, it is easier to handle integers than JSON.

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

   One functionality we added was the ability for a user to see their order history, which should be commonplace for a restaurant-finding website. One feature we removed from the initial planning, was the ability to find nearby restaurants because it was quite difficult to try and figure out how to enable geocoding for a restaurant and a user's address. Furthermore, most of the restaurant locations we generated for the database were Chicago addresses, which would be useless for a user based in Champaign.

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

   Our trigger is actually quite useful, not only for a restaurant finding application but also for any application which contains users. Essentially our trigger determines, for any user that has created a new account, if the provided email address is already in use or not. If so, the trigger does not make changes to the database and simply returns to the front-end "This email is already in use".

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.**

   Neil - One technical challenge we faced was that one of our API requests was not working, and we figured out that you cannot send a body object along with a GET request. We were initially confused because when using postman to test API we were able to send the body with the GET request and were able to get the response. We learned this the hard way after the request kept sending an undefined object from the frontend.

   Theo - No technical challenges were encountered for the MySQL/Database part. The database was hosted on GCP (Google Cloud Platform) according to the first project video. After that, using the database using the project 1 video was straightforward. The MySQL code that was used on GCP was first written in MySQL Workbench, and from there ported to the SQL instance on GCP.

   Aviral- The main challenge faced on the backend part was hosting the API. Initially, we hosted Api on Heroku, but as of November 28, Free Heroku Postgres, free Heroku Data for Redis, and free Heroku Dynos are no longer available. Therefore, I had to find an alternative hosting solution. We ended up using the Google Cloud Platform. However, the issue in GCP was that we had to run the instance of the GCP virtual machine when we wanted to use the API. So if we close the browser or the laptop, the instance is stopped, and the API stops functioning. This made it challenging to keep the API running consistently. We used the "No-Capture" mode in the SSH terminal to keep the instance running even after closing the browser. To do this, open the SSH terminal and type the following command: gcloud compute ssh [instance-name] -- -Nf

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

   There are no other things that changed between the final application and the original proposal.

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

   In the future, the application would ideally be able to show users the menu items for a specific restaurant, place orders, and have the ability to track an order status after being placed. Furthermore, a user would be able to change their profile information, which we currently have not implemented. Lastly, one cool feature we thought of is to use machine

learning to figure out a relationship (if any) between restaurants with a high number of orders and the average price of the menu items.

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

For the final division of labor, Neil worked on the front end to create the web interface using ReactJS, Aviral worked on the back end to create the API layer using NodeJS, and Theo worked on the database management/development using SQL workbench. In the beginning, we worked separately to complete our respective tasks in the initial setup, however, once we were past the initial stages, we collaborated quite frequently to implement each feature from a full-stack perspective.