Final Report

Database Functionality Changes

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).

> For our project we ended up not developing a feature that would allow the user to “favorite” and save recipes. Instead, we created the functionality to automatically tag older recipes with new allergen tags if a new allergen was entered into the database. We also didn’t need to incorporate functionality to expand the recipe tabs in a separate browser window, as during the search the recipe boxes are already large enough to contain all the steps and ingredients necessary for the recipe. We also ended up not including nutritional information in the recipes, only ingredients, steps, user id, recipe id, cook time, and recipe name.

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

> When we first constructed our design, we all agreed upon the fact that the existing websites have too much useless information as well as commercial. We decided while keeping the core functionalities like search or CRUD operations, we want to keep everything else as simple as possible. We think in terms of functionalities, our website could be very useful. All the core features are derived from user stories like allergy filter, or tag filter. Besides, it could also improve over time like the allergens pool will grow as users mark more ingredients as allergens. What has failed to achieve, however, is the authentication part of the design. Right now the project grants all the users permission to perform CRUD action without permission, which is not realistic if we want to put it into a larger scale. This could also be something that we can work on in the future to improve its usefulness.

Discuss what functionalities you added or removed. Why?

> We added the functionality to have people search by tags that were applied to the recipes. This would allow for the user to find recipes where they might not have a specific keyword in mind, but they would like to find quick, easy, etc recipes. Another thing we added was the ability for people to add the allergy ingredient to the recipe. This would then find every single recipe in the database with that ingredient and then add it to the allergy table. This was the first step into

adding the allergy to the list which would then allow the user to select the ingredient when doing their search.

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

None that we can think of.

Database Design changes

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

Our original proposed data sources were either a .csv file containing 100 thousand recipes from Food.com or a JSON file containing 1 million recipes. As the project was implemented in SQL, using the 100 thousand recipe data table made most sense as we didn't have to manipulate the structure of the original data file. We ended up sticking with the Food.com data source through the end, although we only ended up using a subset of the data due to time and budget constraints (discussed further below in the Technical Difficulties section).

We did change the schema from our original design (discussed in the following subsection), but we made very few changes to the data source's schema. Any changes made to the schema provided by the source pertained to i) breaking down arrays to form one tuple for each array item or ii) removing information that wasn't required for our goals, like descriptions of the recipe.

Discuss what you changed 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?

We had to make many changes to the UML diagram and table structure between stage two and the final demo. The UML diagrams themselves are provided in the Additional Materials section. Changes consist of the following:

1. The **Equipment** table was dropped due to not having the necessary data to fill this table.
2. The **Saved** and **Users** tables both were not implemented due to time constraints, though this would still be an important piece to add to our application moving forward.
3. We renamed the **Comments** table to **Reviews** to better reflect the content of the table and the goals of the application.

4. The **Ingredients** table was changed to an attribute of **Recipes** rather than an independent table to reduce the number of joins required to perform simple searches.
5. We added three new tables: **Tags**, **Allergens**, and **Nutrition** that contain additional data about the recipes since this is information we wanted to include in our application. Allergen information was previously an attribute of ingredients, but it was much simpler to keep it in a separate table and only join with **Recipes** when the user requests to filter by allergy information. **Nutrition** ended up not being used in the final implementation of the project, though it could be introduced at a later date.
6. The **Recipes** table is the pillar of the application and has primarily changed in terms of the attributes we've chosen to include. The attributes were changed primarily because of i) data availability, ii) changes to the application objective, and iii) table design changes.

In terms of which design is more suitable, the intentional design choices we made pertain to the types of information that were best represented as attributes rather than tuples in a separate table. Recipe information determined to be more integral, such as the ingredients or recipe steps, was kept as recipe attributes to reduce the number of joins required to perform the basic functions of the application. Other information such as tags or allergy information, which are optional search filters for the application, were separate tables that could be joined to the **Recipes** table using the recipe id number as a primary key.

Advanced Database Programs

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

It complements our application from two perspectives which are user experience and project scalability. For our trigger, it allows the user to leave the number of step fields blank while inputting recipes and it will automatically calculate the value for them as they click create. This reduces the work that needs to be done in users' ends. For our stored procedure, it automatically marked all the allergen information if a new recipe has an allergen that was never seen before (not in the allergy table yet). This makes our project more scalable because we could not have all allergens in the world at once when we set up our database. Having the stored procedure that handles the unseen allergens helped us to improve our database overtime.

Future Directions

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

Something that we can improve on for the application is adding more features into the application. The biggest one would be for users to create accounts where we could then store user information and other things such as favorite recipes or even a tab that shows all their submitted recipes. Upon creation of an account it would give the user a userID, and then we could remove it from the form and users would not have to worry about filling out unnecessary forms.

Other features might include a more sophisticated way to match the allergen input to allergen information. Currently, the database is updated by performing a LIKE '%allergen%' match on the ingredients list, which is not sophisticated enough to capture the fact that a dairy allergy could encompass many things. We could also add a section for nutrition to each recipe in order to give users a better idea of how healthy a certain recipe is, or whether it contains high or low amounts of any specific nutrients they need to keep track of.

Technical Challenges

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.

Bhaven

Formatting the data in django was tough. We were able to do the query to get the results back from the database but we were unable to use it in the way we wanted to. It would just vomit all the data in one giant array which is not what we wanted and there was no way to individually select which column of data we wanted. We had to read the django documentation very carefully along with some stackoverflow posts to eventually get the data formatted correctly. Then we were able to get each column individually and then create the boxes which had all the relevant information for each individual recipe.

Daqian

Connecting from our project to GCP was quite challenging. We wanted to use Django at first. However, the tutorials online on connecting from Django to GCP are outdated. We then switched to Node.js because we went through the setups in one of our project workshops. Meanwhile Bhaven introduced the mysql connector library in python, which eventually helped us set up the connection from our project to the database in Django again.

Elaina

Uploading the data using GCP was a challenge. The examples from class used .sql files while our data was stored in .csv files and the process to upload a .csv file to the mysql database in GCP was surprisingly more complex than uploading .sql files seemed to be. The GCP documentation regarding this issue was either outdated or simply not helpful – this was a common occurrence when trying to figure out how to do things in GCP. Ultimately, it was much easier to upload the data to the server using MySQL Workbench, as the interface made it easier to figure out how to upload the tables and allowed us to define the schema and data types very easily. Unfortunately, we had already spent quite a significant portion of our GCP budget with the time spent struggling with figuring out how to use it. I liked using GCP so each member could access the server remotely, but I would recommend using other tools like MySQL Workbench to interact with the server!

Michael

It was challenging to figure out how to send a query from Django using python functions and MySQL connector to the database. We also had to figure out how to pass the values we sent into a function to the actual SQL query so that it could be inserted into the string the query is stored as, and it took a while to figure out how to format the string so that this data would actually be properly sent. Specifically, we had to decide whether we wanted to use f string, str.format, and %. We ended up using %, and that came with its own challenges. We found that using anything other than %s for a sql query wouldn't work even if the value being entered into the query was an integer, so it took a while to figure out which proper version of % we needed to use.

Division of Labor

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

Frontend Team

- Bhaven:
    - Python script to generate allergy table
    - UML first draft design
    - Setting up Django Project
    - Creating pages using html, css
    - Setting up the forms and connecting the forms to the database
    - Trigger
    - Stored Procedure
    - Data Formatting
    - Formatting of recipes upon search

- Elaina:
    - UML first draft design
    - Setting up database in GCP

        - Allocating GCP access to team members
        - UML second & final draft design
        - Connecting Django project to GCP through Django Models
        - Setting up the forms and testing query interaction
        - Stored Procedure first draft
        - Connecting form input to database

Backend Team

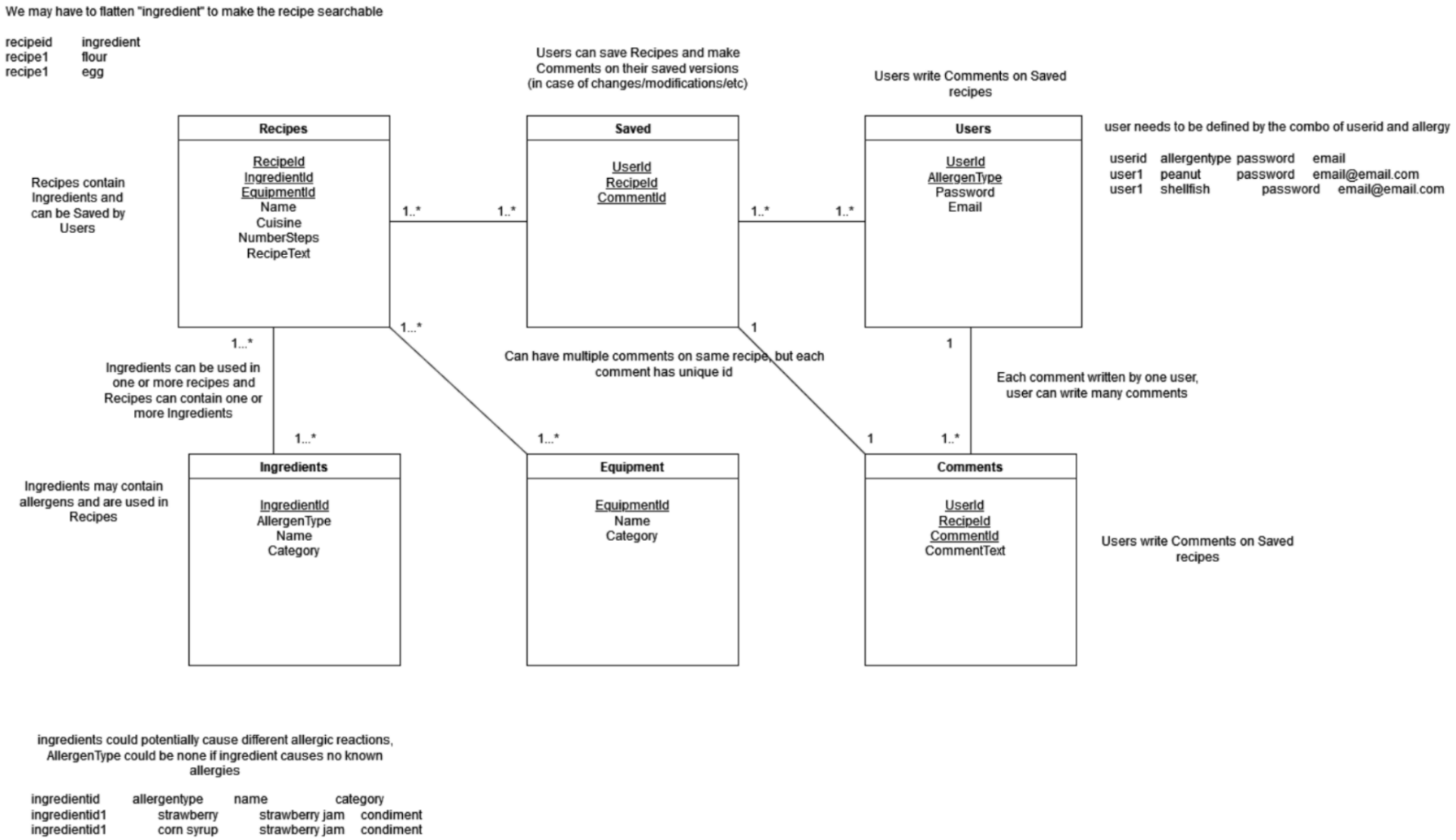
- Daqian
    - Python script to generate tags table
    - UML first draft design
    - Keyword Search Query
    - Two Advanced Queries
    - Trigger first draft
    - Stored Procedure Debugging

- Michael
    - UML first draft design
    - Create, Read, Update, Delete queries
    - CRUD debugging
    - stored procedure debugging

The team met approximately once a week as a group for the first two or so months. After this point, around the midterm demo, it made more sense for backend and frontend teams to break off and work on their respective pieces. Following this, subsets of the team met as needed to perform the tasks required for the final demo, video, and report.

Additional Materials

A: The original UML diagram provided in Stage 2



B: Final UML Diagram