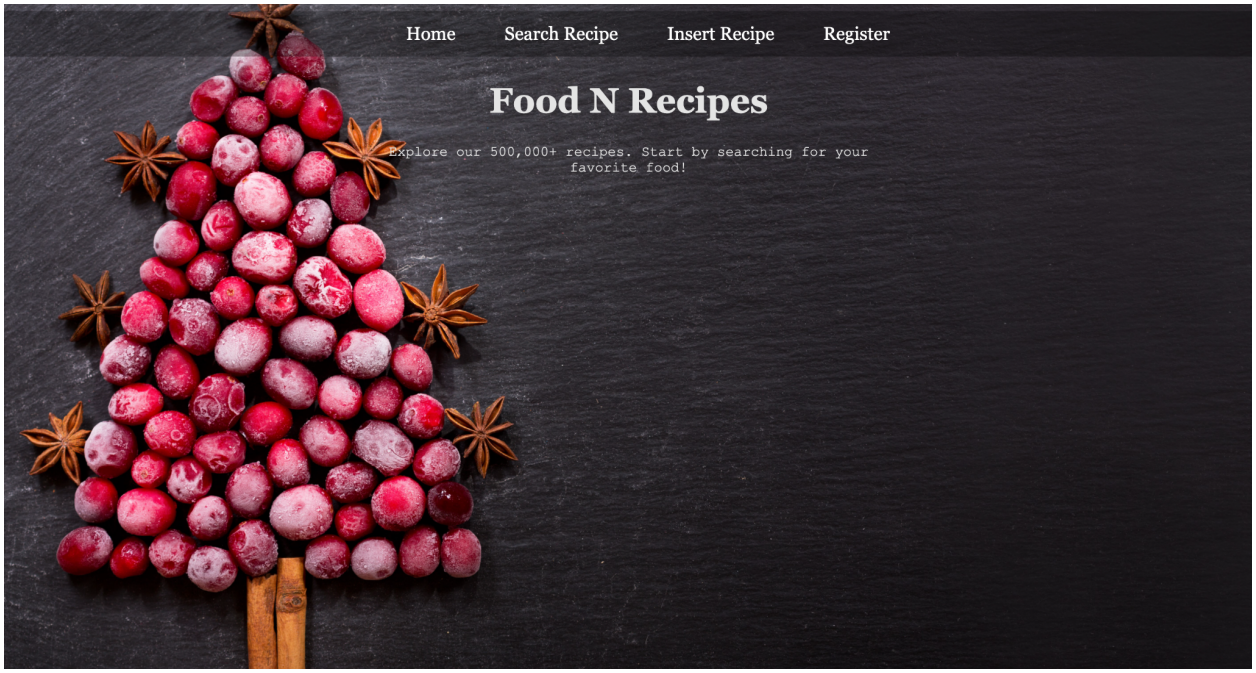
Food Recipes and Reviews Project Report

Austin Wang, Mengyu Yang



I. Introduction

For this project, we used the Food.com Recipes and Reviews dataset from Kaggle and created a website application that provides user interactions and data visualization based on the dataset. This dataset comes to our particular interest because it contains nicely-formatted, rich information about more than 500,000 recipes regarding ingredients, nutrition, instructions and more.

We would like to create a website application based on the Food.com dataset, which includes information about the recipes themselves and user reviews for those recipes. We will first import the dataset into a MySQL database. Then, we will connect the database with the front-end application.

The user of our application will be able to search for recipes by keywords, write and update their own recipes, write reviews on recipes, and delete recipes and reviews. By exploring our web application, the user will be able to find their ideal recipe by various ways of searching and share their own recipes on the site.

II. Final Implementation

A. Final Implementation: Use cases, Platform

The frontend of our application is developed using php, html, and css. The backend is developed using MySQL using InnoDB engine. The UML diagram is drawn using draw.io.

The user must register themselves in the “Register” tab and remember their user ID if they wish to insert recipes or to insert reviews into the application.

The user of our application is able to search through recipes in the database by keyword and author name. They are also able to further filter the results by limiting the number of rows to all rows, 50 rows, 100 rows, 500 rows, and 1000 rows; by time or calories; and finally, by ascending or descending order.

Once results have been fetched and displayed from a search, the user will be able to view details about the recipe, view the recipe’s reviews, and delete the recipe from the search result page.

In a recipe's details page, the user can conveniently update field values or go to the reviews of this recipe. Note that some field values, namely "Recipe ID," "Recipe Name," "Author ID," "Author Name," "Date Published," "Aggregated Rating," and "Review Count," cannot be edited.

In a recipe's review page, the user can add a review for this particular recipe. Note that in order to add a review, the user must have an author ID. If the user does not already have an author ID, they can register themselves in the "Register" tab. "Author ID" is required for a successful review, and the user can set the "Rating" via a slider, but the user may leave the review paragraph blank if they so choose.

The user is able to insert a recipe into the database. Apart from the required fields "Recipe Name" and "Author ID," if the user does not provide any value in a field, we will provide a default value for it. Like mentioned above, the user may change field values for the recipe if they so choose at a later time.

B. Final Database Design

1. Normalization Process and Functional Dependencies
   We created two megatables from two CSV files downloaded from Kaggle.com, recipe_mega for recipe information and review_mega for information about reviews on the corresponding recipes. In these two megatables, we only found one functional dependency, based on which we performed normalization on the megatables. As author_name solely depends on author_id, and no other column depends on author_id, we separated these two columns to create a new table for author information, and all the original attributes except author_name stays the same in the recipe and review tables. Since there are no other functional dependencies between non-prime attributes in our decomposed tables, the decomposed tables are in 3NF.

2. Final Database Design
   In our final implementation, we have three tables in our database—recipe, review, and author. The recipe table contains information including ingredients, nutrition, and preparation time of recipes. The review table the rating, review, and the date of reviews on recipes. The author table contains the author names.

Based on these three tables, we created the following features which total to 9 stored procedures, 2 transactions, 2 views, and 2 triggers.

First, we have 3 procedures for supporting searching for a particular recipe. The show_by_time stored procedure takes in a keyword string, an ordering (1 for ascending and 0 for descending), and an integer row limit. The procedure will return the recipes of which the recipe name contains the keyword, in the specified order and with the row limit from input. Similarly, the show_by_calories procedure supports searching for recipes by a keyword string and the returned result is ordered by calories in ascending or descending order. Also, we support searching for recipes written by a certain author with the show_for_author procedure, which takes in a name string and returns all recipes by authors whose name contains the input name string. In addition to showing details of recipes, we also support showing the reviews of a certain recipe by the show_reviews procedure.

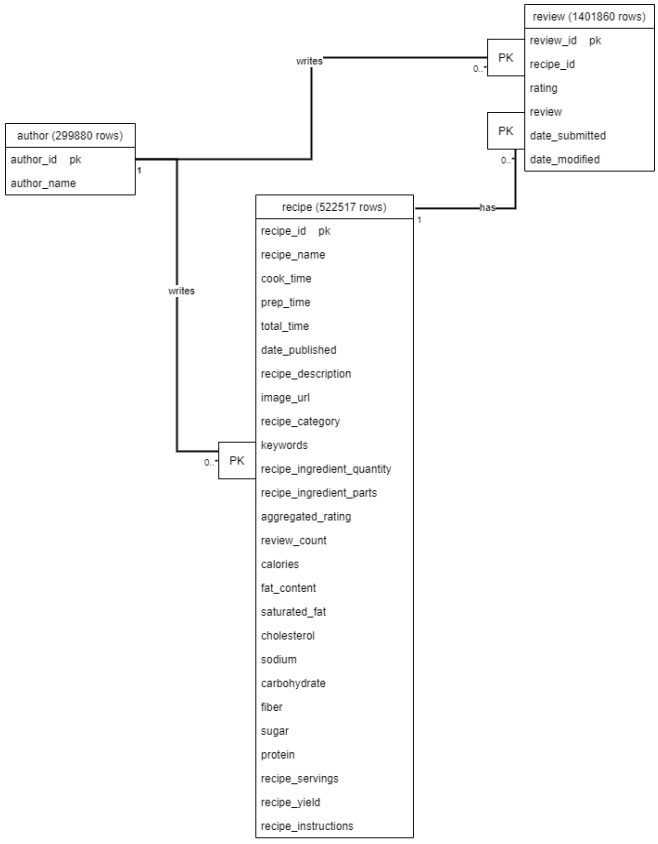
Next, we have 2 views that provide high-level presentations of the summary of our data. The trending_recipe view shows information about recipes that have high ratings and are mostly reviewed. The top_author recipe view shows the combination of the authors who write the most recipes and have high average ratings. These views inform users about the most popular recipes and authors on the website, which give them hints on what to search for.

In order to support insertions, we have 2 stored procedures and a transaction that add a new author, recipe, and review respectively into the database. The add_author procedure takes the user name string and registers the user into the author table. Here, we have a before-insert trigger that truncates the input username to 50 characters before adding it to the table. In addition, we have the add_recipe procedure that takes in values of attributes of a new recipe and adds it to the recipe table. In order to ensure data integrity, attributes date_published, aggregated_rating, and review_count do not need user input and are generated by the stored procedure instead. Last, the add_review transaction adds a new review to a certain recipe with a specified author. The transaction is implemented as a stored procedure. A transaction must be used here because after adding a new review, the aggregated rating and review count of the corresponding recipe should also be updated.

Moreover, we have a stored procedure for updating recipes and a transaction for updating reviews. The update_recipe procedure takes in the new information about an existing recipe (expect date_published, aggregated rating, and review_count) from user input and updates the attributes. Similar to adding a review, the update of a review is performed in a transaction because the aggregated rating of the corresponding recipe must also be updated. The transaction is implemented within the update_review procedure.

Finally, we also support deleting a recipe or a review. The delete_review stored procedure not only deletes a certain review but also updates the aggregated rating and review count of the corresponding recipe. For deleting a recipe, we first use a before-delete trigger to delete all the reviews on the recipe and then use the delete_recipe stored procedure to delete the recipe itself.

3. UML with Cardinality
   Cardinalities: 522517 rows in recipe, 1401860 rows in review, and 299880 rows in author.



C. Data Used for Testing

We retrieved our initial data set from a search on Kaggle.com according to our interest as well as required metrics (more than 30 attributes, more than 100mb in terms of file size). This database contains 522,517 recipes from 312 different categories and 1,401,982 reviews from 271,907 users.

To test the functionalities of our application, we need to perform many modifications to the database, especially when testing updates and deletes. First, whenever we finish writing a new procedure, trigger, view, or transaction, we would test in MySQL that it was working as intended before trying to utilize these functions in the frontend code. Some functions work as intended on the first try so we did not need to enter a lot of new data entries, in which case two calls to that specific function in MySQL would be sufficient for testing. Other times, however, we need to make many attempts before we finally fix the function, so we would enter upwards of 10 data entries in MySQL. All in all, we have entered around 100 data entries in MySQL alone. We have, however, deleted some of these entries in our final submission because they were only for debugging and would be unnecessary for the final presentation.

Then, after we made sure that the function works in MySQL, we would call it in our frontend codebase. We did not count how many new entries we entered here for testing because there were too many to count, but it certainly would be greater than the amount of data entries we made in testing in MySQL.

D. Summary of Implemented Use Cases

Our application allows the user to search, insert, update, and delete food recipes and search, insert, and delete reviews of the recipes.

III. Illustration of Functionality

Below are the walkthroughs for primary functionalities and use cases.

Setup

1. Database
    a. Run /db/database_receation_use.sql
    b. Run /db/add_SP.sql

c. Run /db/delete_SP.sql
d. Run /db/search_SP.sql
e. Run /db/update_SP.sql
f. NOTE: If you need to recreate the database for any reason (including dropping tables), remember to rerun the stored procedures, transactions, views, and triggers.

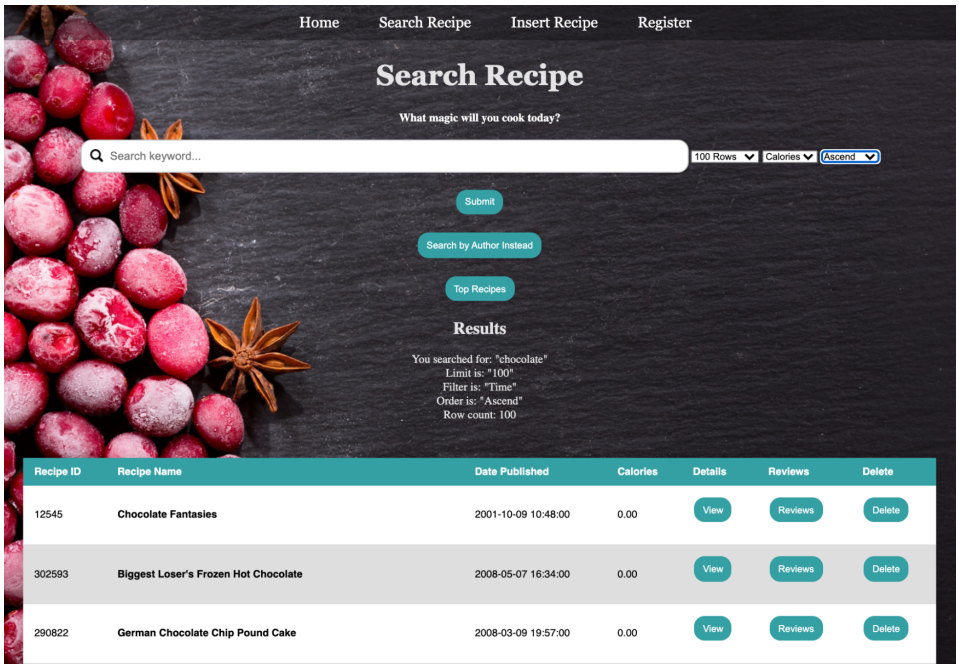
2. Application
   a. Copy /application to MAMP/WAMP's htdocs

Add Author (User is recommended to start with this step)

1. Navigate to "Register" → Enter a username and hit "Submit" → A message on the next page shows the author_id of the new user. Please take down this number. (This information is important because your user iD is required when a user wants to add a new recipe or review).

Search Recipe by Keyword

1. Navigate to "Search Recipe" → Enter keywords in the textbox → Choose row limit, order by time/calories, ascending/descending (default is unlimited rows, order by time, descending) → hit "Submit" → result will show below.
2. For each recipe in the search result, the user is able to view recipe details and view reviews.
   (e.g. enter "chocolate", select "100 Rows", "Calories", "Ascend")



Search Recipe by Author

1. Navigate to “Search Recipe” → Hit “Search by Author Instead” → On the next page, search for recipes by entering an author name → result will show below. (e.g. enter “SweetsLady” → 75 results)

View Recipe Details

1. After getting a search result, hit “View” on one of the rows → User will be navigated to a new page showing the detailed information about the recipe → Hit “Reviews” to view reviews.
2. To update recipe, note that there are textboxes after attributes on the recipe details page, and these are the attributes that the user can choose to update. (e.g. look at the “View” for Homemade Yellow Cake and Variations: Recipe ID is 208645; Author ID is 240552, etc.)
3. To delete recipe, hit “Delete” on the bottom of the recipe details page.

Update Recipe

1. Refer to step 2 in view recipe details and enter new information into one or more of the textboxes on the recipe details page → Hit “Update” on the bottom of the page → Success message on the next page → Hit “Return to Recipe” to return to the recipe details page.
   (e.g. for Homemade Yellow Cake and Variations, enter “1” for Recipe Servings, hit “Update” then “Return to Recipe”, Recipe Servings should have been changed to 1)

View Reviews

1. After getting a search result, hit “Reviews” on one of the rows (or hit “Reviews” on a recipe details page) → Reviews will show on the next page → Now, the user can hit “Return to Recipe” to go to the page with recipe details or hit “Add Review” to add review.
2. For each review on the reviews page, the user can view review details or delete review.
   (e.g. the first review for Homemade Yellow Cake and Variations has a 5-rating and starts with “I made this yesterday and talk about…”)

View Review Details

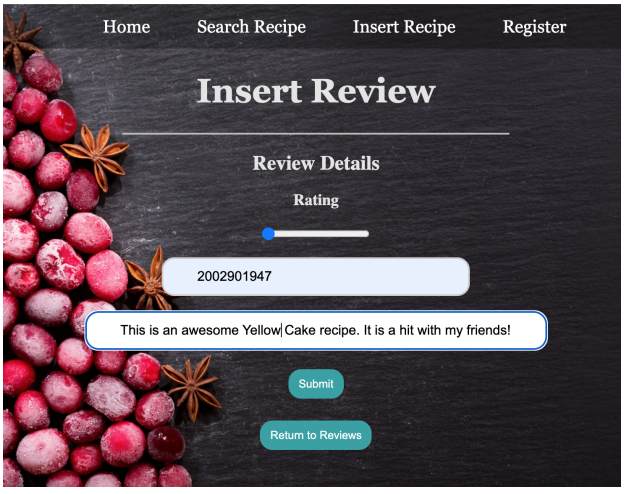
1. On the reviews page of a recipe, hit “View” on one of the rows → details of the review will show on the next page → Hit “Reviews” to return to the reviews page.
2. On the review details page, the user can update review.
   (e.g. try viewing reviews for Homemade Yellow Cake and Variations)

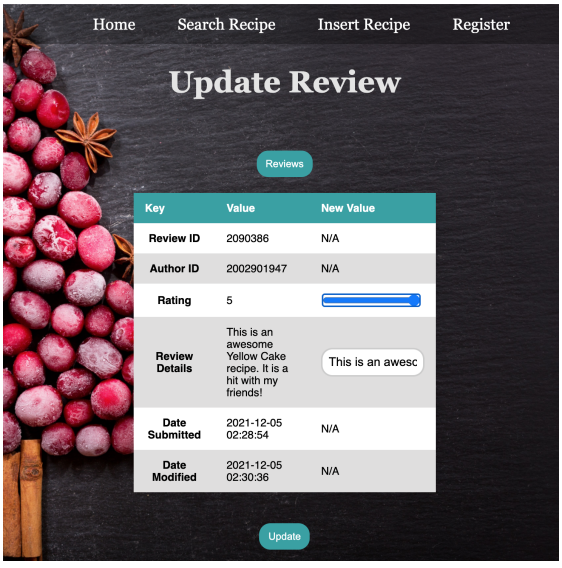
Add Review

1. On the reviews page of a recipe, hit “Add Review” → enter information into the textboxes, slide to choose the rating, and hit “Submit” (If the user has not registered, an error message will appear → Navigate to “Register” to add author and then try to add review) → Success message will show on the next page → Hit “Return to Reviews” to return to the reviews page.
   (e.g. first note that the aggregated rating for Homemade Yellow Cake and Variations is 5 → add a review for Homemade Yellow Cake and Variations; enter your user ID and shift the rating slider all the way to the left (which is 0) ; then submit → now aggregated rating is 4 and you can find your review on the reviews page)

Update Review

1. On the review details page, enter new information in the textboxes and hit “Submit” to update the review → Success message shows on the next page → hit “Return to Recipe” to return to the recipe details page.
   (e.g. find your review in the review list for Homemade Yellow Cake (should be the last one, so scroll all the way down) and Variations and hit “View” → shift the rating slider all the way to the right (which is 5) and hit “Update” → now your rating becomes 5)





Delete Review

1. On the reviews page of a recipe, hit “Delete” on one of the rows to delete the review → Success message will show on the next page → Hit “Return to Reviews” to return to the reviews page;
   (e.g. try deleting the review you just created)

Delete Recipe

1. On the recipe details page of a recipe, hit “Delete” on the bottom → Automatically navigate to the “Search Recipe” page.
   (e.g. try deleting Homemade Yellow Cake and Variations → then try searching for Homemade Yellow Cake and Variations → no result)

Add Recipe

1. On the recipe details page, enter new information into the textboxes → hit “Submit” (If the user has not registered, an error message will appear → Navigate to “Register” to add author and then try to add a review) → Automatically navigate to “Insert Recipe” page and success message will show on the top.
   (e.g. try adding a recipe → enter “Homemade Yellow Cake and Variations” as name and enter your user ID; other information is optional → hit “Submit” → search for Homemade Yellow Cake and Variations → 1 result returned)

View Top Recipes

1. Navigate to “Search Recipe” → Hit “Top Recipes” → Trending recipes will show on the next page → Hit “View” to view details of each recipe.
   (e.g. try to view recipes in the top recipes; the first one should be “Reverse-Seared” Steaks)

View Top Authors

1. Navigate to “Search Recipe” → Hit “Search by Author Instead” → Hit “Top Authors” → Top authors will show on the next page.
   (e.g. the first author should be Annacia)

IV. Summary Discussion

One of the most challenging parts in the beginning of the project was learning PHP syntax and how to use it with MySQL. During one of our project working sessions, we spent about 2 hours trying to debug why our call to a stored procedure wasn't executing on the frontend but was working correctly in MySQL. We solved this problem after asking Dr. Singh about it. It turned out that we have to use PDO instead of MySQLi when we are executing our query in PHP because in conn.php, we specified that we will be using the PDO API. This was something that was completely unfamiliar to us.

Another challenge was coming up with a reasonable and easy-to-use user interface. Initially, we had a lot of tabs in the navigation bar, such as "Delete Recipe" and "Update Recipe." It was not long before we realized that this kind of structure would make an incredibly bad user experience, since, for instance, the user would need to remember what the recipe's recipe_id is in order to delete or update something. Therefore, instead of the separate tabs, we added "delete" and "update" buttons in the search result or details page so that the user can easily modify the recipes or reviews without needing to remember their specific IDs.

One very unexpected challenge was that Mengyu had to get a new laptop during the course of our project because her old laptop's battery was apparently on the verge of explosion. This resulted in some hold-ups because she needed to get her WAMP and MySQL set up again. Nonetheless, because we shared our code using GitHub, she was able to retrieve previous work progress relatively smoothly and get the application up and running again fairly quickly.

In the beginning, we wanted to implement visualization of the database, but we ran out of time so we did not end up realizing this feature.

We shared responsibilities evenly depending on the task at hand. We were able to coordinate this very well, because we often met up to work on the project together right after classes on Tuesdays or Thursdays. Austin worked more heavily on the frontend, while Mengyu worked more heavily on the backend. But when it came to debugging, the two of us looked through the code together, thus we both gained incredibly valuable webdev and database management knowledge by the end of the project.