Changes from the Original Proposal

One of the major changes we had to pivot from the original proposal was the idea of recipe recommendations. We found that developing a recommendation engine was a little too ambitious. However, we were able to help sort recipes based on user settings (dietary restrictions/budget/weekly calories). We essentially changed from a pseudorandom recommendation to a filtering list of possible recipes. We also decided to not include adding individual ingredients or creating recipes. This left the financial planning, nutrition tracking, and cookbook part of the application intact with less of a focus on the individual ingredients. This change was made due to the processing of our datasets, as we found that adding ingredients one by one into a recipe would be difficult to implement.

Overall we feel that the application is mostly successful in what we set out to do. The user is able to create an id, list their dietary restrictions and nutrient/budget goals, and add recipes to their calendar. Once they add recipes it will track the total price, several nutrients such as calories and protein, as well as list on the weekly diet. As such the functionality is correct, but there are undoubtedly more possible improvements we could have made to the app which we will discuss later.

We kept our data sources fairly consistent, we used the same .csv files throughout the project. However during our original data processing stage before moving to the cloud we had to perform some formatting changes in order to keep consistency among different data sets. Examples include changing empty values to n/a and moving attributes.

Changes to ER/Tables

We had to make several important changes to our table implementations in the final version. One of these was adding more attributes to tables such as time (breakfast,lunch,dinner) to the MealPlan table. As we kept developing the application we realized that we needed these attributes to accommodate the features such as the weekly calendar. Along with this we had to change many of the attributes in the nutrition table, as the data we used contained different values which didn't include things like cholesterol. To keep consistent with our datasets we decided the nutrition should just have carbohydrates, kilocalories, protein, sugar, and total fat. Consequently our Ingredient_Nutrition table differed greatly from our proposed design on the ER diagram. These changes were made going into stage 3 in our DDL commands. In our original proposed diagram the keys were not explicitly stated but we made no changes to our primary keys, we had unique ids for each instance of the tables. Finally our Ingredient_Price table was streamline to only containing the price per 100g instead of a cup and liter as well. Conversion could be made to those values so we felt that these attributes were ultimately unnecessary in the final design.

Overall we felt that the final design was more suitable as many of the table implementations in the original ER diagram had many unnecessary attributes that did not need to be stored. The data was more easy to manage and writing queries turned out to be much easier with this layout.

Functionalities + Advanced Databases

As discussed before we decided to remove the recommendation engine that would directly recommend the user recipes in favor or simply listing based on restriction. This was both more practical and easier to implement than the initial design, with the same level of functionality. We followed our proposal fairly accurately, as we added 3 functional pages(home page, user settings, meal plan) that allows the user to create an account, change settings, and add meals. We felt that the number of pages was serviceable to fulfill the functions proposed in stage 1. In regards to the meal plan page we did add a weekly calendar which you can add breakfast,lunch, and dinner to each day of the week. Since this was a plan for the week we felt we must add 3 slots since most people eat 3 meals a day, so this feature was added.

When adding the trigger and stored procedures, we decided to use our advanced queries from stage 3. Our trigger allows us to check if a recipe we decide to add contains a forbidden ingredient and if so reject it. Our stored procedure will list the recipe by calories on the home page which give a better idea of the available recipes. These helped add extra functionality to the application while fulfilling the requirements needed.

Our project runs on the Google Cloud database which runs a fully managed MySQL service. This advanced database program will essentially run the database on its own service. This made it much easier for us since it allows us to focus on the development of the application with the data being stored on a large scale without us directly managing much of it. This program also allows for scalability so adding more recipes or more data causes much less issues than otherwise.

Challenges

Here is a list of technical challenges that each group member encountered while working on the project:

Gary- There were several problems involving connecting frontend to backend, and backend to the database, such as converting date and time in Python. In addition, pinpointing the origin of errors was complicated due to the different paths involved in the coding process. There were often times where I couldn't figure out whether the problem was with the SQL query, the limited display in Javascript, or because the JSON file returned the wrong datatype.

Brian- The biggest challenge I encountered was learning to link frontend components with backend components as I was not familiar with react at all. I had to use both inspect element and prints in the flask backend to guess why the errors were happening. I also had to figure out that CORS was the issue our frontend was initially not displaying.

Evan- I encountered challenges translating syntax and code learned in the classroom to queries and code implementable on SQL. Issues with delimiters and entering the lengthy, multiline trigger and stored procedure was especially challenging. The fact that this process has to be repeated every time a bug fix was attempted made finalizing both of these much more difficult than it would be in the PrairieLearn development environment.

Joshua- One of the issues I encountered when working on the backend was understanding how to properly implement our SQL queries using python and how this would relate to the frontend. I spent some time looking through SQLAlchemy which is a toolkit for python and I also had trouble understanding how this outputs to JSON for the front end to work with.

Additional Changes + Possible Improvements

Besides the already mentioned changes in regards to functionality there were not many additional changes we had to the project. However there are many possible improvements that could be made to the application with more time allocated. Currently the method for the data to be filtered under the user's restrictions is limited and needs more sophistication. Another improvement would be to structure the meal plan for the week so that it is more intuitive and easier to look at. Our queries also could use some improvements with regards to efficiency, especially in the stored procedure which uses multiple queries over multiple massive tables.

Work Distribution:

Brian: Frontend setup (minimal UI, react project structure, pages and components). Experimental backend (User page, recipe page in checkpoint 1)

Gary: Development of backend, connection of backend and database, CRUD operations in backend

Evan: Developed trigger, stored procedure with two advanced queries, and two advanced queries for transactions

Joshua: Helped Evan develop queries for transactions, worked on backend implementation using libraries like SQLAlchemy to work on the query functionality.