CS 411 Group 070 Project Report

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).

In terms of the overall direction of our project, we did not change our vision much. We focused on providing information about allergens of food items, and basic dietary preferences (vegan, vegetarian) to users and also providing prices. The application allows users to filter built-in listed allergens (gluten, crustaceans/shellfish, fish, eggs, milk, soy, mollusks) and built-in listed dietary preferences (vegan, vegetarian, pescatarian, keto, paleo, Mediterranean, low-carb, high protein, low sodium, low fat, low sugar). We also allow users to enter custom dietary preferences. However, we moved away from providing nutritional information or macronutrients of each item because it required a more complex database that was outside our project scope. Furthermore, we believed it was redundant considering users can add preferences like low-carb, high-protein, etc.

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

Our application, Snipiddy, was able to achieve usefulness to users. Snipiddy can process an uploading image of a menu and saves users time because it helps them avoid researching each menu item. The application can quickly identify common allergens and the price of the item. Furthermore, the user can personalize their experience based on their user profiles. Also, they can filter through items based on allergens. Our application uses Optical Character Recognition (OCR) to extract the menu text of uploaded images and provide meaningful analysis. However, there are areas for improvement. Snippidy does not currently have nutrition details about each item. Although ruled out for our project's scope, it could be helpful to users who have their personal definitions of low-carb or high-protein. This feature would also provide more transparency about food item content.

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

We included two new sources of data. The first is a dataset that has a comprehensive list of food items with their corresponding allergies. We access it through Kaggle at this link: https://www.kaggle.com/datasets/boltcutters/food-allergens-and-allergies. The second new source of data includes different food dishes and their ingredients. We access it through Kaggle at this link: https://www.kaggle.com/datasets/pes12017000148/food-ingredients-and-recipe-dataset-with-images. The source of our menu data and menu item data is from the online menus of local restaurants and bars. This is mainly for testing, as any picture a user takes of a menu should also scan with our system.

We also changed the relational schema throughout our project. Our first schema was normalized and included separate tables for *User_Allergen* and *User_Diet*, whereas now, we have those stored as arrays or text inside the *profiles* table. For *Menu_Item_Ingredient*, our first schema had them linked via join tables, but our current one, which is denormalized, has them stored as text fields inside *menu_items*. Additionally, our first schema's data relationships were explicit with foreign keys and had minimal duplication, but our second schema's data relationships are implicit inside text fields and are more easily able to be duplicated across fields. In terms of query complexity and extensibility, the first schema made it easier to add new allergens, diets, and ingredients leading to accurate data, but it required more joins; the second one has easier selects, but it is harder to add data due to the requirement of updating fields and parsing text. Lastly, our first schema's consistency was enforced by foreign keys in our database, which was better for scaling larger data sets, and our second one's consistency wais left to application logic, which works for scaling smaller projects such as ours. Overall, our first schema was normalized, which is better for production and long-term scaling, but our second one is denormalized and is better for quick prototyping and launch, which is what we want right now. If we were to improve this in the future, we would likely refactor our current schema into a normalized structure.

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?

There are several differences between our original UML diagram and our revised UML diagram. First, we removed some redundancies. For instance, attributes like allergens and dietary_prefs were removed from the User table and normalized into relationships (User allergic to Allergens and follows Diet). We also removed restricted_ingredients from Diet, and restrictions are now captured in relationships. In the final diagram, we also normalized the Ingredient and Allergen Structure. Instead of ingredients being embedded in menu items, ingredients are in their own table linked to menu items. Ingredients are also separately linked to allergens. We also focused on having more consistent relationships in the diagram by using proper foreign key relationships instead of embedding lists in columns. Finally, we separated Scan and Menu_item. Our final design separates scanning from menu parsing more cleanly — the scan leads to menu items, but the menu items are standalone with their own ingredients.

Discuss what functionalities you added or removed. Why?

One additional functionality is that the user can perform the delete functionality. For example, the user can remove a specific allergy from their profile if they wrote that information in by accident, or if they no longer have that allergy. Also, users can remove menus from their profile if a restaurant changes their menu drastically, or they don't plan on returning to that restaurant. Thus, making the experience more clean and easy to use. Furthermore, users can now view the price of the item through our application. This way, users do not have to cross-reference both the actual menu and the information through our application to get the details they need when questioning if they are going to order a specific food item. Another added functionality is that users can search through items in a menu to see certain foods they might want to, or not want to eat based on their dietary preferences or allergies. The last functionality we added was storing a history of the menu scans so that users can look back at the scans in case they revisit the same restaurant. One functionality we removed is the ability for users to view the nutritional details of each item. Our team deemed this redundant since users can enter macro-specific preferences, like low carb, high protein, etc.

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

One of our advanced queries aggregates the number of menus that the user has scanned to display the total number of menus that they have scanned. This is a small UI improvement that allows users to easily get an idea of how many restaurants they have visited and scanned menus so far instead of scrolling through all of their scans. We also have advanced queries that allow users to filter results and search for specific results when viewing all of the items of a specific menu. It also returns the number of items for each menu.

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.

One technical challenge we faced was normalizing our data sources. Our data sources came in different formats, like CSV, and HTML, and each had different indexes and naming conventions. We had to normalize them and relate them to the scanned items the uploaded image contained. To create a consistent format so everything could translate easily, we implemented exact matching with fuzzy search. If future teams run into this issue, we advise them to spend time designing a standard schema for the internal data. It may also be helpful to write scripts to format incoming data sources as soon as possible.

The second technical challenge was writing the queries themselves. Some of our queries involved JOIN with multiple relations, applying filtering logic, figuring out indexes, and more. For example, in one advanced query, adding an index didn't improve performance because the query conditions did not match the index structure. We had to learn how to select indexes to better leverage them and optimize our query. If future teams face this issue, we advise them to test queries early with smaller data sizes. EXPLAIN ANALYZE can also help see the query plan. Furthermore, remember that adding indexes is not enough, they must be structured to leverage indexes properly. In general, it is important to simplify complex queries to improve clarity for all team members.

The third technical challenge was calling and processing responses from external API endpoints. For the OpenAI endpoint, we had to perform prompt engineering to get structured and usable responses. We also had to convert the responses, which came as raw strings, into JSON responses. On the Supabase side, we had to call API endpoints to perform create, read, update, and delete (CRUD) operations on scanned menu data. So, we had to carefully structure requests and manage asynchronous data updates. If future teams run into this issue, we would remind them to not assume outputs from models like OpenAI will be perfectly formatted. Decide and build in validation and parsing layers. If using Supabase, define clean and consistent API call structures. Try to include error handling for database operations to prevent data inconsistencies.

The fourth technical challenge was using OCR to scan menu images. The raw extracted text was not always clear or perfectly segmented. Menu pictures may have had typos, misread characters, inconsistent line breaks, blurry sections, and unfamiliar formatting. We were not always able to split the menu into item names, descriptions, ingredients, prices, etc. automatically. If future teams expect to use OCR, we suggest making sure they assume OCR output to be messy. It might be beneficial to build some post-processing steps to clean up and organize the text.

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

Originally, we planned to use NextJS for the frontend and FastAPI in Python for the backend. We also planned to use MySQL for the backend. Furthermore, we planned to use OCR and Augmented retrieval generation (RAG) search to identify item contents. Our team is still using NextJS for the frontend. However, we are no longer using FastAPI in Python to handle API requests. Now, we are using Supabase with SQL for the backend because it has a hosted SQL database and built-in APIs for CRUD operations. This helped reduce our backend complexity. We are also only using OCR through OpenAI so we did not build our machine learning model. Furthermore, we are no longer using RAG search because it proved redundant when we were using allergen look-up and menu text.

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

Our application can be expanded and could include lots of new functionality. The top priority would be to include broader nutritional information, like in the original project vision. This way users can focus on each menu item and analyze the macronutrient content.
Another feature we could implement in the future would be to display a map of nearby restaurants that fit dietary preferences. The map could display different markets above restaurants to flag to users if the restaurant has vegan, vegetarian, gluten-free options, etc. Another idea would be to include a history of the places the user has visited so they can keep track of restaurants that had options for their dietary preferences.

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

- Design how backend interacts with frontend, OCR, LLM, DB, design API routes, set up tests. User creation and authentication. - Sebastian

- Implement API endpoints like signup, login, menu upload and process into text, and work on advanced queries. - Aruv
- Choose DB system and design DB diagram and schema, integrate with API routes, edit demo video - Deepika
- Find data sources, UI/UX, Integrate OCR with backend (setup pipeline), integrate frontend with backend - Madhumitha

We managed teamwork by setting up regular meetings leading up to each stage deadline. We had strong communication by talking about our plans within our text group chat, as well as during these meetings. We also used a personal GitHub repository as well as the CS 411 repository for version control and easy access to the programs and documents. We made sure that all group members understood what the overall goals were for the project, and that everyone had an equal say in what our final product for each stage release would look like. Overall, we feel that we collaborated well and fairly.