Department of Computer Science and Engineering
University of Notre Dame

CSE 40746 Advanced Database Projects
Spring 2023

Final Report

Group 09
Ava DeCroix
Colin McKechney
Selina Nie

Abstract
Our final web application, Irish Bites, is a campus-specific nutrition planner application designed to help students meet their health goals and manage their personal nutrition. Irish Bites allows students to quickly and easily set goals for different macro and micronutrients, log what they eat, and then see how they are progressing relative to their set goals.

Table of Contents

I. Team Organization & Group Dynamics

Division of tasks
The development team consisted of three programmers, Ava, Colin, and Selina. All teammates took a fairly equal split of the work in the feasibility, planning and database design phases of the project. For the actual building portion of the project, team responsibilities/tasks were divided roughly as follows:

- Ava: data collection and cleaning, frontend design and development
- Selina: database creation and data loading, frontend design and development
- Colin: backend development, deployment

Group Dynamics & Collaborative Workflow
The general workflow was very collaborative, and we often had long team work sessions where all team members contributed to different parts of the codebase. To facilitate group code sharing, we used a GitHub repository to store and work on all parts of the project.

Decision-Making Process
As mentioned in Section 1, our team functioned very collaboratively and generally easily arrived at a group consensus for all decisions. In situations where a decision was required, such as determining the general layout of the application, deciding on a technology stack, etc., we would generally share individual ideas and then debate pros/cons of each to find an agreeable solution.

II. Final Design

Our final design schemas are as follows:

Eatery(id, e_name, e_location)

Menu_item(item_id, item_name, eatery_id, serving_size)

Nutrition_info(item_id, eatery_id, calories, fat_g, saturated_fat_g, trans_fat_g, cholesterol_mg, carbs_g, fiber_g, sugar_mg, protein_g, sodium_mg, potassium_mg)

- Foreign keys: item_id (references item_id in Menu_item), eatery_id references id in Eatery)

Student(net_ID, first_name, last_name, pswd, salt)

Goal(s_net_id, total_cal, total_fat, total_sat_fat, total_trans_fat, total_carbs, total_fiber, total_sugar, total_protein, total_sodium, total_potassium, total_cholesterol)

- Foreign keys: s_net_id (references net_ID in Student)

Result(s_net_id, week, total_cal, total_fat, total_sat_fat, total_trans_fat, total_carbs, total_fiber, total_sugar, total_protein, total_sodium, total_potassium, total_cholesterol)

- Foreign keys: s_net_id (references net_ID in Student)

Each user has a table of the form below where Uname is replaced with the actual username (automatically generated upon signing up):

Uname(item_id, amount, item_name, calories, fat_g, saturated_fat_g, trans_fat_g, cholesterol_mg, carbs_g, fiber_g, sugar_mg, protein_g, sodium_mg, potassium_mg, menu_item)

- Foreign keys: item_id (references item_id in Menu_item), item_name (references item_name in Menu_item), calories, fat_g, saturated_fat_g, trans_fat_g, cholesterol_mg, carbs_g, fiber_g, sugar_mg, protein_g, sodium_mg, potassium_mg (reference those same attributes in Nutrition_info)

III. Technology Stack

We chose the React.js framework for our frontend, due to its versatility, excellent documentation and resources, and its comprehensive UI library Material UI. For our backend, we wrote a server and API in Rust. Our database was Oracle, as per the project parameters.

IV. Coding/Debugging

As previously mentioned, we managed our code via a GitHub repository and worked on branches to ensure we would not have issues with conflicting changes. We included a number of debugging aids in our code as we worked, including extensive use of console logging statements on the frontend and an error logging system on the server.

V. Verification that the System Meets Requirements

Our original functional and non-functional requirements are given here, along with a description of how we implemented/verified them, or if we changed them and if so why.

Functional:

- View the nutritional information for all food items available at on-campus dining locations (eg. serving size, total calories, fat (g), sugar (g), carbohydrates (g), etc.)
    - *Through the "Menus" tab, users are able to view comprehensive nutritional information from the campus dining locations, including both dining hall items and chain restaurants.*
- Set nutritional goals (eg. target daily intake of calories/ protein, grams of sugar consumed per day, etc.) according to their preference
    - *Upon entering the application, users are brought to a home page to set their nutritional goals, which can be edited at any time.*
- Add meals to their daily plan and observe how consumption of the meal affects their nutritional goals
    - *Users can add meals/food items in two different ways: by going through the menus of on-campus eateries in the "Menus" tab, or by searching directly for an item in the "Log Foods" tab.*
- View recommendations of suggested food items (based on nutritional goals) and when the meals/ food items are typically available
    - *After speaking with stakeholders and potential users, we decided to drop this particular requirement. Students expressed a distaste for the idea that the application would be "trying to force them to eat a certain way" and also told us that the hours of on-campus eateries are well-known and would be unnecessary extra information in the system.*
- Save previous meals and the corresponding nutrition information
    - *Previous meals are easily viewable on the "Plan Progress" tab, along with their nutrition information.*
- Input the nutritional information of outside meals, so users can include off-campus restaurant meals and home cooked meals and track
    - *The "Log Meals" tab has a form to input off-campus or home-cooked meals and their corresponding nutritional information.*

Non-functional Requirements

- Nutritional information for food items must be accurate and as complete as possible

    - *A comprehensive review of campus nutrition information was conducted, and the results were input into CSV format and then checked and extensively cleaned to ensure data completeness and accuracy.*
- User accounts must be secure and password-protected
    - *We implemented a username-password login authentication system so that users have access to their own plans and history only.*
- The system must be highly usable (according to usability heuristics) and not add significant cognitive load to the meal decision process.
    - *We used the MUI library to implement a streamlined, simple design aesthetic. After finalizing the frontend design, we conducted a heuristic evaluation using Nielson's 10 Usability Heuristics to ensure that there were no major usability issues with our interface.*
- The system must be scalable enough to support a high volume (1000+) of concurrent visits during peak meal times
    - *We are unable to test this directly as we do not have 1000 different clients from which to test, but believe that our system would support this.*
- The system should respond within ~1 second for all user interactions (navigating to a new page, updating nutritional goals according to a new meal being added, etc.)
    - *After extensive testing and interaction, all interactions have a response time of well under 1 second (except in cases of network issues such as spotty internet connectivity).*
- The system should be accessible from both laptops and mobile devices
    - *Where possible, we used reactive web programming principles to ensure that critical components are usable from a mobile device browser.*

VI. Testing Strategies and Results

Unit testing

We conducted extensive unit testing of all functions in the code.

Integration testing

The bulk of our testing time was spent on testing the integration between the frontend HTTP client requests and the backend API handling. For each new endpoint and request, we would conduct multiple rounds of testing to ensure that requests were being sent and handled properly and that the correct data was being sent/received in both directions.

Functional testing

We performed a test case for each of our final functional requirements to ensure that the application was indeed able to complete those tasks.

Interface testing

We had several testing sessions to just “play with” the software interface, clicking buttons and trying actions to determine if there were any issues. When we did encounter issues, we would then locate the source in the code and proceed accordingly.

VII. Programming Issues

Generally, our project went very smoothly and we were able to accomplish most tasks with small amounts of debugging and testing. We did not have any major programming issues. We had two minor challenges with our database: auto populating item IDs via a sequence and auto creating a log table for each new user, but we were able to work to solve these fairly quickly and implement them in our final codebase.

VIII. Interface & Human-Centered Design

Before beginning front-end development, we spoke with students about what types of features they would like to see in a meal planning/nutrition app to find initial ideas of interface design and organization. After considering these and discussing as a team, we developed the final site architecture and page structure that we believed would be simplest for users to understand and navigate. We focused our interface design on human-centered design principles, and included a prototyping process to get early feedback and refine our design choices. This included paper prototypes, wireframes, and finally a bare-bones digital prototype to help us finalize things like page layouts. We then implemented our interface using the MUI library in React to be as simple and clean as possible. We chose table views for most of the nutritional information since this is a view that users are most likely familiar with for nutrition information, and is also well-suited for storing historical data like the meals a student has eaten in a given week. Where useful, we also included strong signifiers for actions the users could take, such as changing button colors on highlight to indicate clickability.

VIII. Deployment

We plan to deploy our application on either an Apache or NGINX.

IX. Validation & User Testing

After finalizing our interface design and fully connecting the front and backend, we asked actual students to try using our application to 1) validate it against their needs and confirm our findings in the initial feasibility study and 2) to catch any remaining usability concerns. We received a generally positive response from users of the system, who indicated that they would very much like to use a system like this as they feel that current options for nutrition tracking and planning are not well-suited to the on campus eating venues. One design error that was caught during this phase of testing was the fact that there was originally no way to delete items from a user’s log once they had logged them, which was frustrating for students who wanted to use the tool to plan ahead (e.g. logging the meals they were going to eat later that day) rather than to track past meals. We implemented this change in our final interface design.

X. Conclusion

Learnings and Insights

Over the process of this project, our concept of the fundamental ER-diagram and relationships between the different tables required to power a nutrition application shifted greatly. Through the process of designing the database, finding, cleaning, and collecting the data, designing the interface, and implementing everything into a full web application, we gained valuable experience in database programming and have a stronger understanding of the complexity of full-stack web development. Some specific insights we took away from the project are as follows:

- **Look at your actual data before trying to figure out schemas and ER-diagrams**
    - Had we actually looked carefully at the nutrition data and eatery data before initially trying to figure out our database design, we would have been better able to determine relationships between tables and fully understand what attributes and tables we would need and why.
- **Talk to actual users early and often**
    - By talking with actual potential users, we were able to build a simplified application that actually addresses user needs, instead of simply building features that no one would use that might clutter up the workflow.
- **Automation isn't always better**
    - Our data collection process was quite arduous, as we were combining data from many different sources in multiple formats. We had initially wanted to try and do this using web scraping and automated scripts, but decided that the cleaning process to deal with the scraped data would actually be more difficult than just collecting a good portion of it by hand. Had we just jumped to automation in this case, we likely would have created a lot of extra work for ourselves.
- **GitHub is your friend, use the tools!**
    - Using GitHub extensively and following good version control practices made it very easy for us to divide work, seamlessly integrate asynchronous work, and view and edit the contributions of others.

Potential Next Steps

If we were to continue developing this application and adding new features, we believe that the following would be interesting continuations:

- Allowing users to “favorite” frequent meals and add them via a shortcut

- More complicated analytics on past plan performance, such as aggregated metrics of progress over time
- Including the data from common ND off-campus restaurants, such as those on Eddy Street or places like Nick's Patio
- Including common alcoholic beverages and other game-day related food items
- Continuous collection of food items from the Dining Halls and on campus restaurants - because DH meals are on a rotational basis, and items are constantly being added or removed from menus, the data we have collected may not be the most up to date