NutriLog

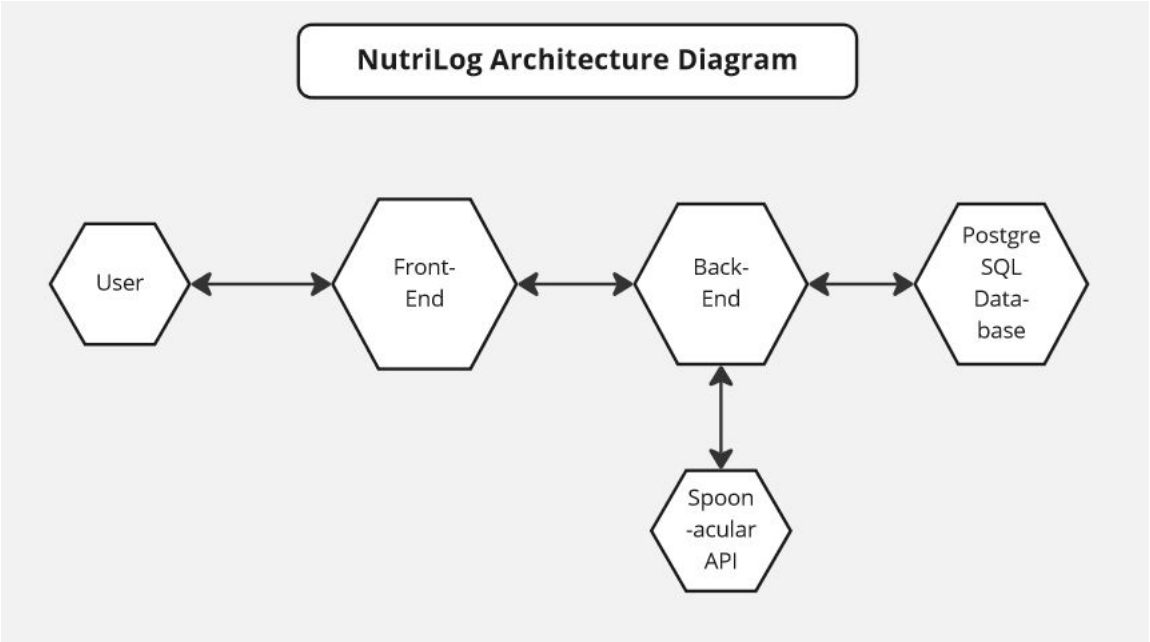
Aditya Ahuja - Maxwell Meiser - Ash Young - Saphire Young

NutriLog allows the user to track their nutrition and practice more mindful eating habits. NutriLog keeps users aware of what they are eating and helps them stay committed to their health and nutrition objectives.

Tool	*Purpose*	*Rating*
GitHub	*Repository, Project Tracking*	*5*
PostgreSQL	*Database*	*2*
VS Code	*IDE*	*5*
Figma	*Wireframing*	*3*
HTML	*UI*	*4*
EJS	*UI*	*3*
NodeJS	*Application Server*	*5*
LocalHost/Docker	*Deployment Environment*	*3*
Spoonacular	*External API*	*4*

Methodologies: Iterative development, agile programming, pair programming

Max presents


NutriLog Architecture Diagram
User
Front-
End
Back-
End
Postgre
SQL
Data-
base
Spoon
-acular
API

Covering usage edge cases - REST API

The programmer must consider every way the user can call every method

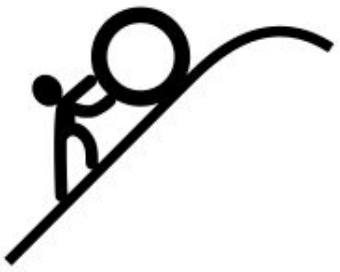
- **Preventing unintended behavior**
  - Authentication protocol
  - User input types
- **Predicting user expectations**
  - Writing flags to handle specific behavior
    - (bugfix) Navbar /library method call resets library search variable

Max

A challenge I ran into while writing API calls was that it is hard to predict every possible way/combination in which the user will utilize the methods. For example, on the library page I saved the user's last query as a session variable for the creation of more complex queries, which allowed me to sort the results of the user's recipe search. However, this had the unintended consequence of saving the user's last search query every if they left the library page, and when they returned to the library page only the results matching with the user's last search input would populate. As a user, I would expect all filters and sorting to be refreshed upon returning to the library page via the navbar. As a developer, I must anticipate this expectation. To resolve the issue, I created a flag that triggered a reset of the last query variable that is only passed to the /library method when the navbar link was clicked. From now on, I will code while thinking about user expectations.

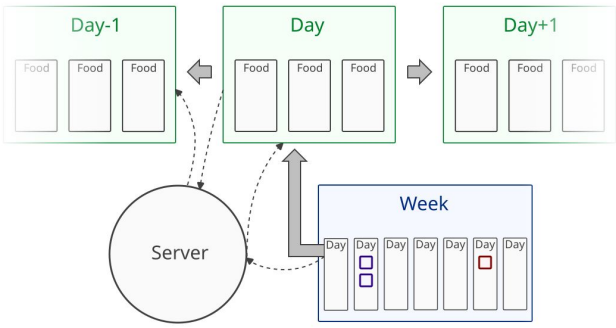
Challenges with Account Creation, Profile and Login

- **Building a regEX for creating password**
  - Identifying criteria for a valid password
  - Building and Testing regEX for each password
- **Passing data between web pages**
  - Passing username between login page and profile page
- **Creating multiple actions on a single button click in profile page**
  - Form action and button onclick()
  - Bug fix in addProfile route where profile was not updating
  - Calling a webpage from within the javascript using href



Challenges For Home and Calendar

- **Date functionality**
  - Allowing moving to next/previous day or week in day meal log and calendar pages (respectively)
  - PostgreSQL date handling is highly idiosyncratic
  - Conversions must be frequently made between different date representations
- **Underestimating scope**
  - Additions of simple functionality seemed appropriate/necessary, but required additional infrastructure.
- **Passing data between pages in general**
  - Reinforced that server renders HTML page – HTML is markup language, not coding language. Client can only access server functionality via requests. Different intuition for code structure required!

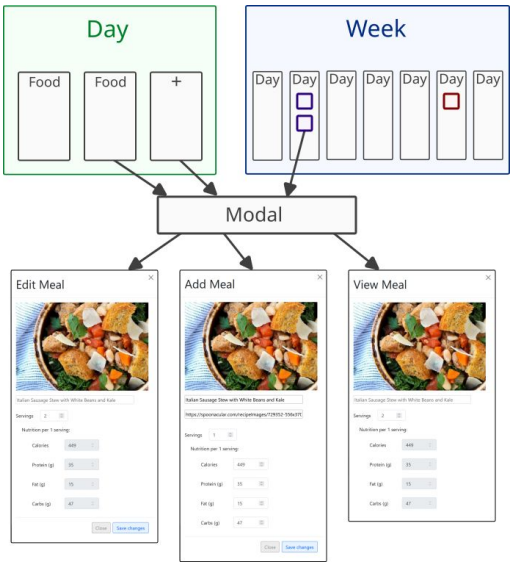


- Date functionality:
  - Allowing moving to next/previous day or week in day meal log and calendar pages (respectively) - page must send data that allows for new page to be rendered. At first calculated the next day/week to request client side, but had to shift this server-side due to requirement that dates are calculated by PostgreSQL. This is because…
  - PostgreSQL date handling is highly idiosyncratic - For example, Sunday of a given week is both day 0 and day 7. 1-7 is sequential Monday through Sunday, but 0-6 are what it returns (0-1 jumps backward a week).
  - Conversions must be frequently made between different date representations - week/dayofweek/year vs month/dayofmonth/year; date formatting. Different pages need to display/use different date elements, which had to be broken into parts and reassembled
- Underestimating scope:
  - Additions of simple functionality seemed appropriate/necessary, but required additional infrastructure. - For example, home page was meant to just show the current day's log, but from a UI perspective, it was best to use it to show previous days, and once it did that, it seemed important to be able to move between dates from there. I also didn't realize how much trouble dates would give me, especially on Calendar. Across the board, the details made implementation more complex than it originally seemed; new to-dos related to features seemed to generate themselves.
- Passing data between pages in general:
  - Reinforced that server renders HTML page; HTML is markup language, not

- coding language. Client can only access server functionality via requests. Different intuition for code structure required! -Javascript code (embedded or otherwise) runs only when page is rendered. Functions can only be called in response to events. Interaction between features was a lot more complicated than monolithic application, and implementing complex behavior took more thought. Overall, the devil was in the details, and things just took a lot more time than expected.

Challenges With the Meal Modal

- **Passing information to and from modal**
  - Appropriate library data and log data needs to be handed off to modal depending on how it is launched
  - Nutrition information types for display is passed to modal – no code modification required if changed!
  - Different data is sent back to server from modal depending on user interaction
- **User input dynamically alters modal appearance and behavior**
  - Autocomplete must create drop down and populate modal when selection is made
  - Option to store to library must appear when appropriate
  - Fields/buttons must be deactivated/invisible if editing is not allowed
- **Reuse of modal for add/edit/view of meals**
  - Code must be written to allow for same modal to be used for all three cases



- Passing information to and from modal
  - Appropriate library data and log data needs to be handed off to modal depending on how it is launched - not just sending right data when rendering to page; data has to be parsed into strings for parameters for function calls from event handlers
  - Nutrition information types for displayed is passed to modal – no code modification required if changed! - record metadata passed to modal page on include
  - Different data is sent back to server from modal depending on user interaction - event-based modification of submit button action to make different server requests requiring different handling
- User input dynamically alters modal appearance and behavior
  - Autocomplete must create drop down and populate modal when selection is made - handling removal of drop down when selection is made difficult because event-handler for drop down deletes itself.
  - Option to store to library must appear when appropriate - different reasons (interaction mode, recipe already in library, etc)
  - Fields/buttons must be deactivated/invisible if editing is not allowed - (same as above)
- Reuse of modal for add/edit/view of meals
  - Code must be written to allow for same modal to be used for all three cases - helper functions written to allow different onclick functions to tailor appearance/behavior. Avoid hard coding of case dependent functionality in modal page.

Demo