System Design for Utrition

Team 16, Durum Wheat Semolina
Alexander Moica
Yasmine Jolly
Jeffrey Wang
Jack Theriault
Catherine Chen
Justina Srebrnjak

April 5, 2023

1 Revision History

Date	Version	Notes
January 18, 2023	1.0	Initial Document
April 02, 2023	1.0	Revision 1 Changes

2 Reference Material

This section records information for easy reference.

2.1 Relevant Documentation

This document references multiple other documents that are listed below:

- SRS, Semolina (2022d)
- Development Plan, Semolina (2022a)
- MG, Semolina (2022b)
- MIS, Semolina (2022c)

2.2 Abbreviations and Acronyms

symbol	description
API	Application Programming Interface
App	Application
BMI	Body Mass Index
CP	Cultural and Political Requirement
FR	Functional Requirement
LF	Look and Feel Requirement
LR	Legal Requirement
MG	Module Guide
MIS	Module Interface Specification
MS	Maintainability and Support Requirement
OE	Operational and Environmental Requirement
PR	Performance Requirement
SR	Security Requirement
SRS	Software Requirements Specification
UH	Usability and Humanity Requirement
Utrition	Explanation of program name

Contents

List of Tables

List of Figures

3 Introduction

3.1 Document Purpose

This series of design documents are comprised of the System Design document, the Module Guide, and the Module Interface Specification. The purpose of these documents is to communicate the modular decomposition of the Utrition application, and provide readers with information regarding the functions/methods of each module, as well as the relationships between modules. Information about the high-level decomposition of modules can be found in the MG. Details regarding each module's functions and methods are discussed in the MIS.

3.2 System Purpose

The purpose of Utrition is to provide users with an application that empowers them to discover the nutritional value of the foods they are consuming, and track their previously eaten meals. Utrition takes a unique approach to the traditional nutritional logging applications by providing a variety of ways for users to input their food items into the application such as through text, verbal communication, and image. Users with keyboard-typing accessibility concerns will have a nutritional logging application that they can use without external assistance.

3.3 Scope

Users will be able to input their food items to Utrition by using speech to text, typing, or uploading food images to the front-end user interface. If users would like to upload an image of the food they want nutritional information for, they would have to provide a photo of the food on the device using Utrition. This can be done by downloading an image of the food item onto the user's device (i.e. computer). After user input, Utrition will begin internal calculations to obtain nutritional information for the food item from the public Nutritionix API. The nutritional information will be provided to the front-end user interface for the user to view.

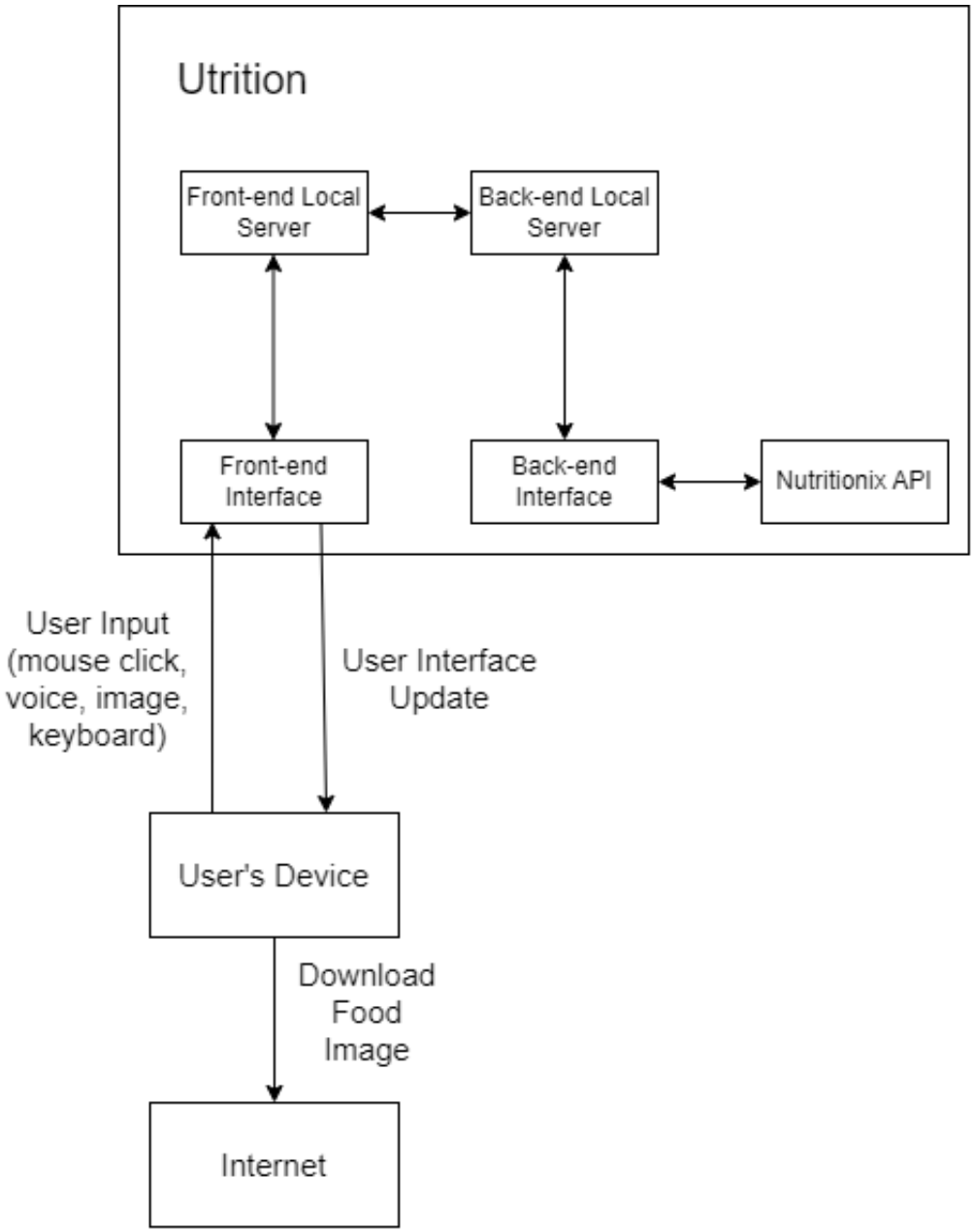


Figure 1: System Context Diagram

4 Project Overview

4.1 Normal Behaviour

In their command-line interface, users will travel to the Utrition directory. From there, users will enter commands “npm start” and “npm run start-backend” separately. Utrition’s front-

end and back-end interfaces will successfully connect to the user's localhost server. Users will be greeted with Utrition's home page. They may travel to their profile page, where they can find their past nutritional data saved in a list. On this page, users may decide to view their nutritional trends in graph form. Users may also travel to the upload page, where they can input food item(s) to find their nutritional information. Users can input food items by

- Clicking on the search bar and typing in the food items. Each food item is separated by a comma.
- Clicking on the microphone button and verbally listing the food items.
- Clicking on the upload button and then selecting an image of a food item.

Users finalize their search by clicking the search button. In return, the user will be shown the nutritional information for each food item searched. The nutritional information is acquired by utilizing the Nutritionix public API. The data is stored on the user's device, so that past nutritional trends can be viewed on their profile page.

4.2 Undesired Event Handling

A more in-depth look to Utrition's undesired event handling can be found in Utrition's Hazard Analysis document.

In all cases except for one, the Utrition front-end interface will notify the user of the issue disrupting normal use of the application. Most of these errors will be detected by the back-end interface of Utrition. As a result, the application's back-end interface will send the error message to the front-end interface through the user's local server for the front-end interface to display. Certain errors, such as incorrect image type, are detected by the front-end interface and will not have to travel through the local servers to deliver the error message to the user.

The one specific case where Utrition will not notify the user of the specific error is when Utrition closes unexpectedly. The cause of this rare issue is unknown to the developers, but the application will save its data periodically during use to ensure minimal information is lost from the crash.

4.3 Component Diagram

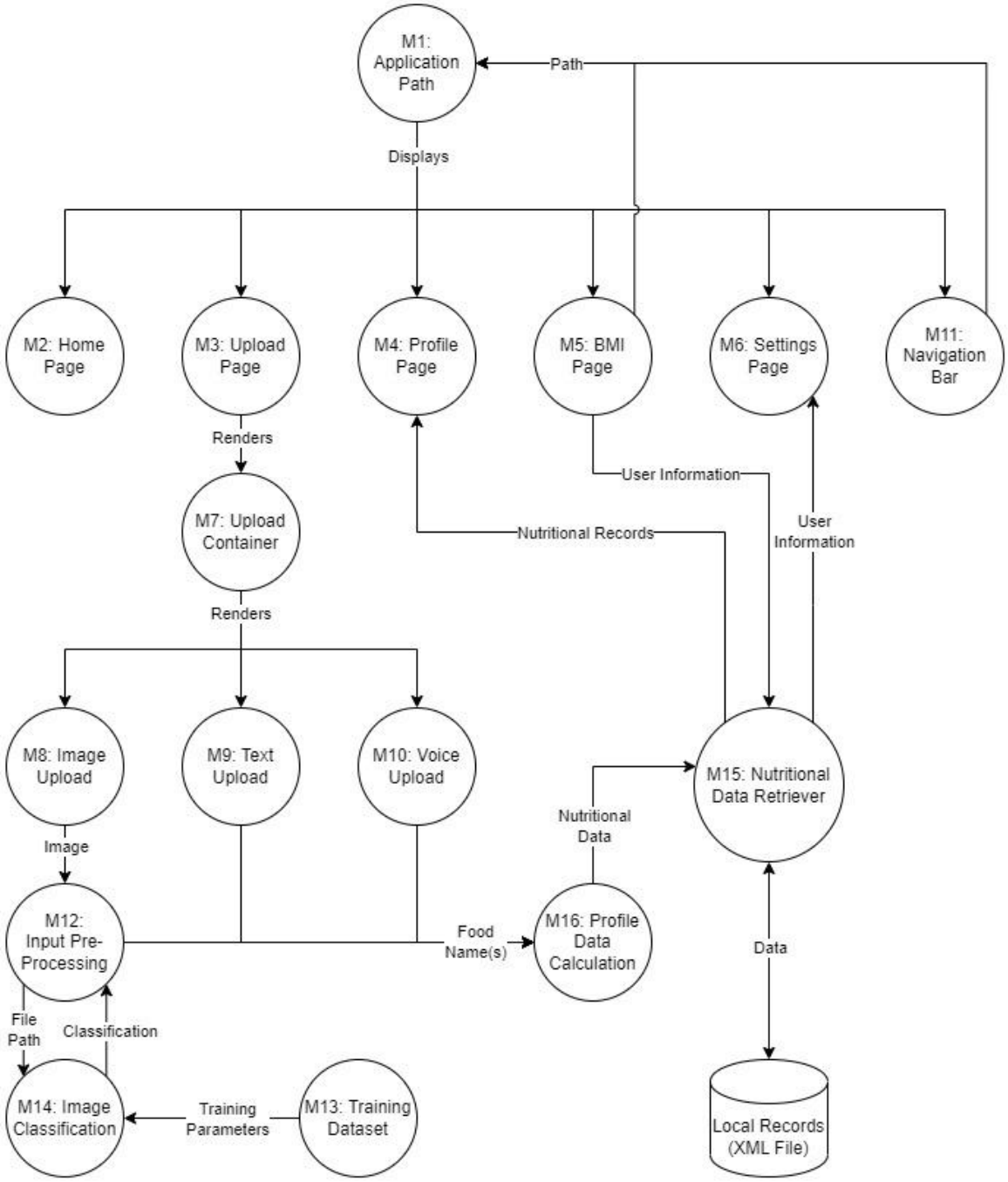


Figure 2: Component Diagram

4.4 Connection Between Requirements and Design

4.4.1 Functional Requirements

FR1. Provide the user with an interface that has the option to select the type of input as image upload. The user can then select an image file from their computer that they can submit as input.

FR2. The food item(s) uploaded to the Utrition front-end will be condensed to a 32x32 pixel image, which is then turned into a 32x32 array. The array containing data for each pixel is then transferred to the Utrition back-end. From there, the image is parsed through a TensorFlow machine learning algorithm which is trained on the CIFAR-100 dataset to determine the food item.

FR3. The Utrition back-end provides the Nutritionix API with valid headers (x-app-id, x-app-key, x-remote-user-id) that allow the use of the API (retrieval of nutrition facts).

FR4. The Utrition back-end interface sends a POST request to the Nutritionix API. The response to the POST request provides the back-end interface with a JSON file containing the food's macro-nutrients, micro-nutrients, and caloric details.

FR5. Once the Utrition back-end has received the nutritional details of the food item from the Nutritionix API, the back-end stores the nutritional information in a CSV file in the users' Utrition folder.

FR6. Once the user requests to see past nutritional data, the front-end interface sends a GET request to the front-end local server. This request is passed to the back-end local server, which then gets sent to the back-end interface. The back-end interface returns the data of all logged entries in the previously logged food items CSV file to the front-end interface (going through their respective local servers). The front-end interface displays the file's data in reverse-chronological order.

FR7. Once the Utrition back-end interface receives the JSON file from the Nutritionix API, the data is sent to the local back-end server, to the local front-end server, and then to the front-end interface. In the front-end interface, the JSON file has the "stringify" method applied to it to display the information to the user.

FR8. Provide the user with an interface that has the option to select the type of input as text upload. The user can then type in their food items that they can submit as input.

FR9. The text input field will be able to accommodate large amounts of text to be submitted where more than one food item can be specified.

FR10. Provide the user with an interface that has the option to select the type of input as voice upload. The user can then vocalize their food items that they can submit as input.

FR11. The voice input field will be able to accommodate large amounts of audio to be submitted where more than one food item can be specified.

FR12. The audio input that the user submits will be translated to text using "react-hook-speech-to-text" package.

FR13. A "reset" button will be created on the voice input interface to allow the user to clear their previous audio input.

FR14. The Utrition back-end interface sends a POST request to the Nutritionix API containing all the text inputted by the user (from text input or voice input). The Nutritionix API is able to parse the text for food items. The response to the POST request provides the back-end interface with a JSON file containing each identified food and the corresponding macro-nutrients, micro-nutrients, and caloric details.

FR15. The front-end interface will contain buttons for the user to switch between each upload option.

FR16. Once the user requests to see past nutritional data found on the "Profile" page, the front-end interface sends a GET request to the front-end local server. This request is passed to the back-end local server, which then gets sent to the back-end interface. The back-end interface returns a JSON-formatted string that contains the user's most eaten food item to the front-end interface (going through their respective local servers). This food item is found by finding the mode of all the food entries saved in the past food entries CSV file.

FR17. Once the user requests to see past nutritional data found on the "Profile" page, the front-end interface sends a GET request to the front-end local server. This request is passed to the back-end local server, which then gets sent to the back-end interface. The back-end interface returns a JSON-formatted string that contains the total calories inputted for the current day to the front-end interface (going through their respective local servers). This caloric value is found by adding all the calories within food entries in the past food entries CSV file that have the current date.

FR18. Once the user requests to see past nutritional data found on the "Profile" page, the front-end interface sends a GET request to the front-end local server. This request is passed to the back-end local server, which then gets sent to the back-end interface. The back-end interface returns a JSON-formatted string that contains a list of JSON formatted strings with the date, total calories per day, and all the foods eaten on that day to the front-end interface (going through their respective local servers). This list is constructed by iterating over all the food entries in the past food entries CSV file. During this iteration, a list will be constructed to keep track of all foods consumed on one day and a float sum will add all calories for one day.

FR19. The back-end interface will be able to access the CSV file directly. This will allow calculations for the total calories per day and creating list subsets for the foods consumed on a certain day.

FR20. A navigation bar is provided at the top of every page.

FR21. Once the user is located on the "Profile" page, a "delete" button will be available for each past food entry input. When a user confirms they want to delete a specified entry, the front-end interface sends a POST request to the front-end local server. This request is passed to the back-end local server, which then gets sent to the back-end interface. The back-end interface will delete the identified food entry from the past food entries CSV file.

FR22. A front-end interface will be provided to the user where they can input their personal statistics. This includes their age, weight, height, gender, and activity level.

FR23. Once the user inputs their personal statistics and presses a "submit" button, a POST request is sent to the front-end local server. This request is passed to the back-end local server, which then gets sent to the back-end interface. The back-end interface will save all user statistics in a JSON file that can be accessed for future calculations.

FR24. The user will be able to resubmit their user statistics after they have initially been saved. This is done by filling out the input fields with different values and clicking the "submit" button. The remaining process follows the FR23 design decision.

FR25. Once the JSON file with user statistics is available, the backend-interface will be able to access these values directly. Simple calculations can be done to find the user's BMI and recommended calories.

FR26. Once the user is located on the "Profile" page, the BMI related calculations will be displayed. This is done when the front-end interface sends a GET request to the front-end local server. This request is passed to the back-end local server, which then gets sent to the back-end interface. The back-end interface will return a JSON-formatted string that contains the user's BMI and recommended calories.

FR27. Once the user is located on the "Profile" page, the front-end interface sends a GET request to the front-end local server. This request is passed to the back-end local server, which then gets sent to the back-end interface. The back-end interface creates a graph of the date versus the total calories for that each date. This is done using the python package "plotly". The graph is saved as .png image that the front-end interface can access directly.

FR28. Once the user is located on the "Profile" page and clicks the "delete" button for a past food entry, a confirmation button will be displayed to the user. The specified entry will only be deleted once the user has given confirmation since deletion is an irreversible action.

4.4.2 Non-Functional Requirements

LF1. Front end developers will make a conscious effort not to place different components near each other. If components are determined to be close in proximity by the developers, a pixel measuring tool will be used to determine the distance between components.

UH1. A navigation bar is provided at the top of every page.

UH2. This design decision is covered by FR5's design decision.

UH3. Utrition will provide steps on how to use the product on its home page. Utrition will not provide the user with many available actions, so it becomes clear to individuals how to use the product.

UH4. Nutritional information (calories, fat, sodium, carbohydrates, sugar, and protein) will have appropriate units.

UH5. Only important nutritional information is displayed to the user (calories, fat, sodium, carbohydrates, sugar, and protein) and back-end calculations are not displayed to the user. Front end reformatting of information will not be displayed to the user as well.

UH6. No sound files will be found in Utrition's GitHub repository.

PR1. Each screen loads in their personal set of functions. Each screen will not load in many functions.

PR2. The machine learning algorithm sets the maximum amount of iterations to be 1000 for each image.

PR3. The Nutritionix API will provide correct results to the back-end system, which then gets sent to the front-end interface through their respective local servers unless the user does not have an internet connection or an obscure food item is entered.

PR4. Utrition is a local web application that can be run by an infinite amount of people at the same time. There is no server to restrict user access.

PR5. Once Utrition is fully developed with no code errors, any person will be able to access and run Utrition. The application will be indefinitely available on GitHub.

OE1. Utrition is available on GitHub.

OE2. The user guide will be indefinitely available on GitHub. The document will be written in simple English phrases for all people aged 14 and older to understand.

OE3. No developers will make changes to Utrition after publication.

MS1. The CIFAR-100 dataset can be swapped with another appropriate dataset by placing it under the src/utrition-backend/ML/ folder.

MS2. Developers will upload a user guide which includes installation and usage instructions on the Utrition GitHub.

MS3. The application will be available on all Windows, macOS, and Linus devices.

SR1. Utrition and all source code files are available as a public repository on GitHub.

SR2. After every food input, Utrition will save the item(s) and corresponding nutritional data in a CSV file.

SR3. Utrition only stores previously searched food items' nutritional information on the users' Utrition folder. Utrition does not communicate with other instances of Utrition on other devices.

CP1. Developers will not include any potentially insensitive content.

LR1. Developers will not include any malicious software in Utrition.

LR2. Development of Utrition will be done on GitHub.

LR3. GitHub's CI/CD pipeline will be used with Pylint. Developers will review the results of the Pylint test to determine if coding conventions are not being followed properly.

LR4. Front-end developers code in ReactJS and CSS using Google's style guide. A review is performed by the front-end developers before the final launch of Utrition.

LR5. Front-end developers design the interface using Google's style guide. A review is performed by the front-end developers before the final launch of Utrition.

5 System Variables

5.1 Monitored Variables

N/A

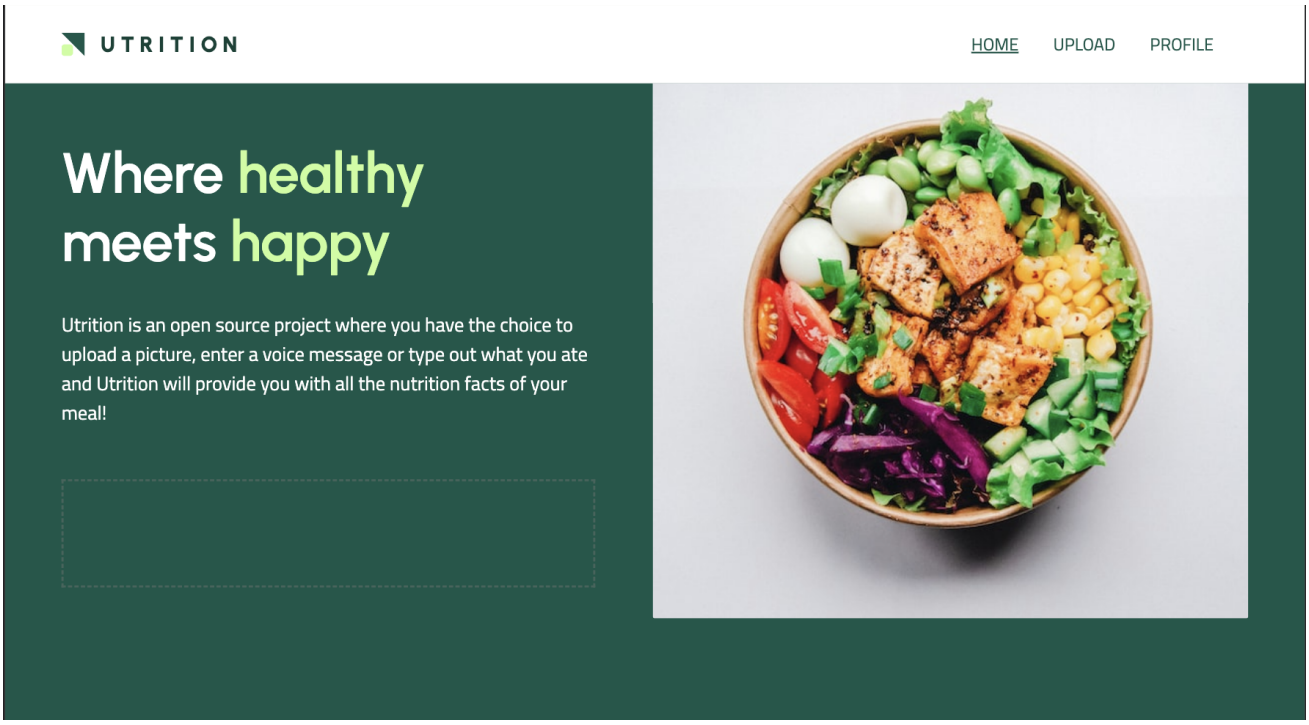
5.2 Controlled Variables

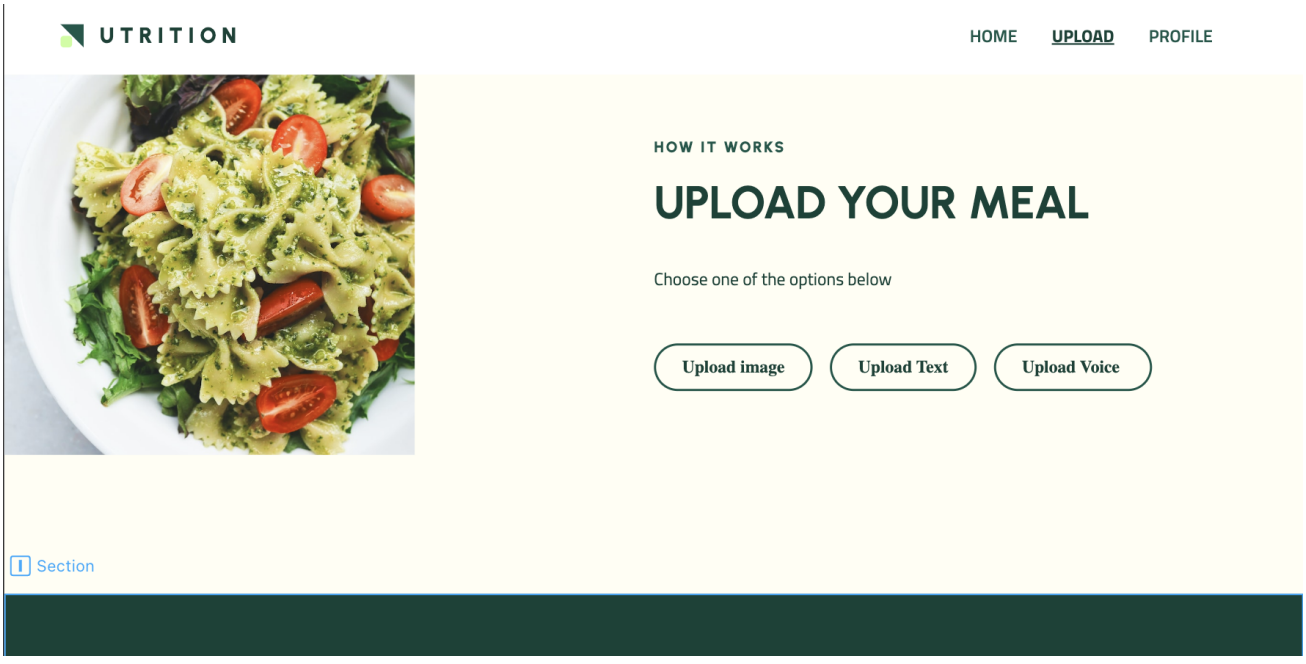
N/A

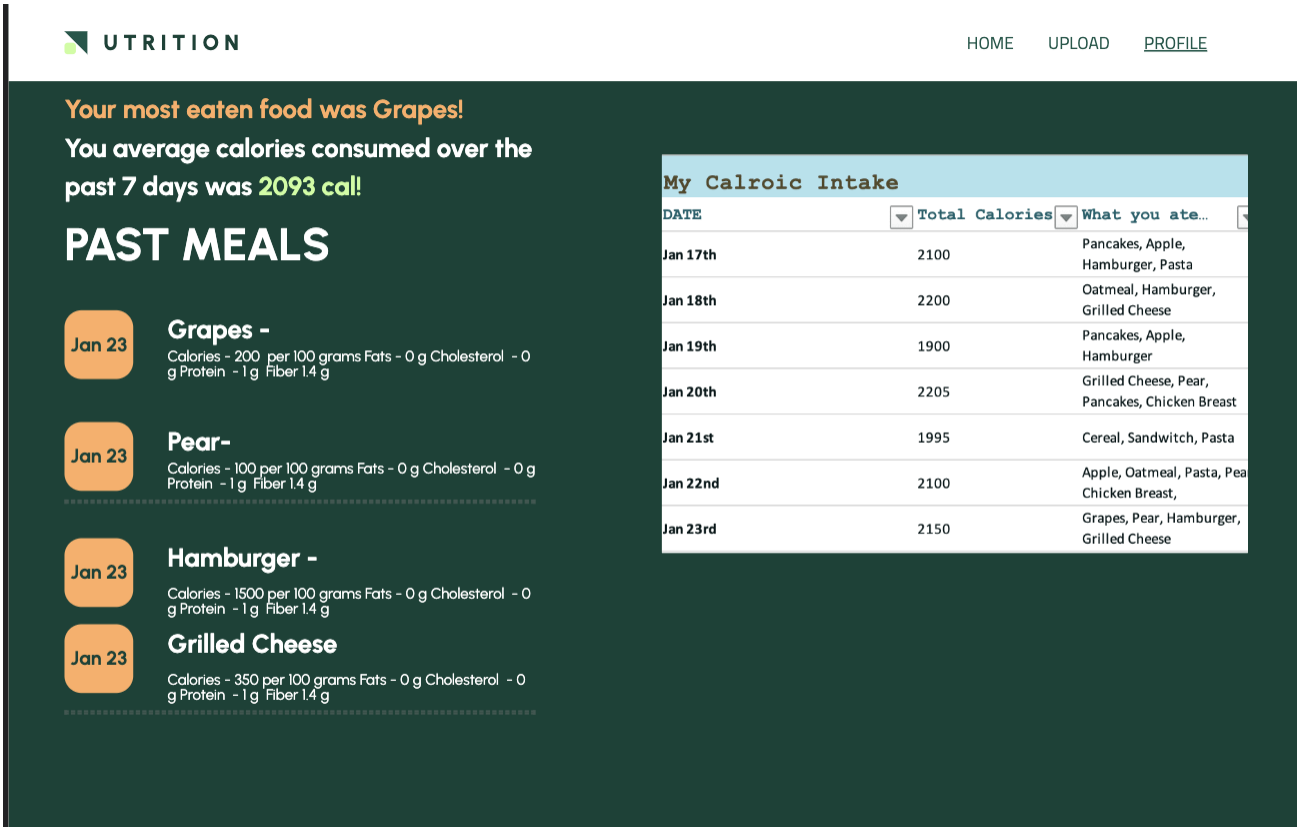
5.3 Constants Variables

N/A

6 User Interfaces







7 Design of Hardware

N/A

8 Design of Electrical Components

N/A

9 Design of Communication Protocols

To use Utrition, users must run commands “npm start” and “npm run start-backend”. The “npm start” command allows for the front-end user interface to be displayed and connects the front-end interface to the user’s local server (http://localhost:3000). The “npm run start-backend” command creates an unseen back-end interface on the user’s local server (http://localhost:5000). Since the front-end and back-end connect to the same local server with different ports, communication is done by the front-end sending GET and POST requests to the front-end local server, which is then read by the connecting back-end local server. The back-end interface receives the GET or POST request from the back-end local

server and begins working on the given task. Once the task has been completed, the back-end returns data through the local server to reach the front-end (which is waiting for the back-end's response). For example, if the request from the front-end is to find nutritional data for a food item, the front-end will send a GET request to the local server. The back-end reads this GET request from their local server (because they are located at the same address) and sends a POST request to the Nutritionix API. In return, the Nutritionix API will deliver the data back to the back-end. From there, the back-end returns the Nutritional data to the front-end by using the local server.

10 Timeline

Module	**Main Developer**	**Due Date**
Training Dataset	Alex Moica	Completed
Input Pre-Processing	Alex Moica	Completed
Image Classification	Alex Moica	Completed
Nutritional Data Retriever	Catherine Chen	Completed
Voice Upload	Catherine Chen	Completed
Application Path	Jeffrey Wang	Completed
Navigation Bar	Jeffrey Wang	Completed
Image Upload	Jeffrey Wang	Completed
Back-end Communication	Justina Srebrnjak	Completed
Food Entry	Justina Srebrnjak	January 25th, 2023
Upload Page	Yasmine Jolly	January 25th, 2023
Profile Page	Jack Theriault	January 28th, 2023
Manual Logging	Yasmine Jolly	January 29th, 2023
User Log Data Structure	Justina Srebrnjak	February 1st, 2023
Nutrition Log	Jack Theriault	February 4th, 2023
Home Page	Yasmine Jolly	February 4th, 2023
Extensive Manual Testing Of Each Module	Catherine Chen	February 5th, 2023

Table 1: Module Timeline

References

Durum Wheat Semolina. Development plan. https://github.com/jeff-rey-wang/utrition/blob/39b1a9bda7ec90db43221e17b035e57b7fa29650/docs/DevelopmentPlan/DevelopmentPlan.pdf, 2022a.

Durum Wheat Semolina. Module guide. https://github.com/jeff-rey-wang/utrition/blob/39b1a9bda7ec90db43221e17b035e57b7fa29650/docs/Design/MG/MG.pdf, 2022b.

Durum Wheat Semolina. Module interface specification. https://github.com/jeff-rey-wang/utrition/blob/39b1a9bda7ec90db43221e17b035e57b7fa29650/docs/Design/MIS/MIS.pdf, 2022c.

Durum Wheat Semolina. System requirements specification. https://github.com/jeff-rey-wang/utrition/blob/39b1a9bda7ec90db43221e17b035e57b7fa29650/docs/SRS/SRS.pdf, 2022d.

A Interface

B Mechanical Hardware

N/A

C Electrical Components

N/A

D Communication Protocols

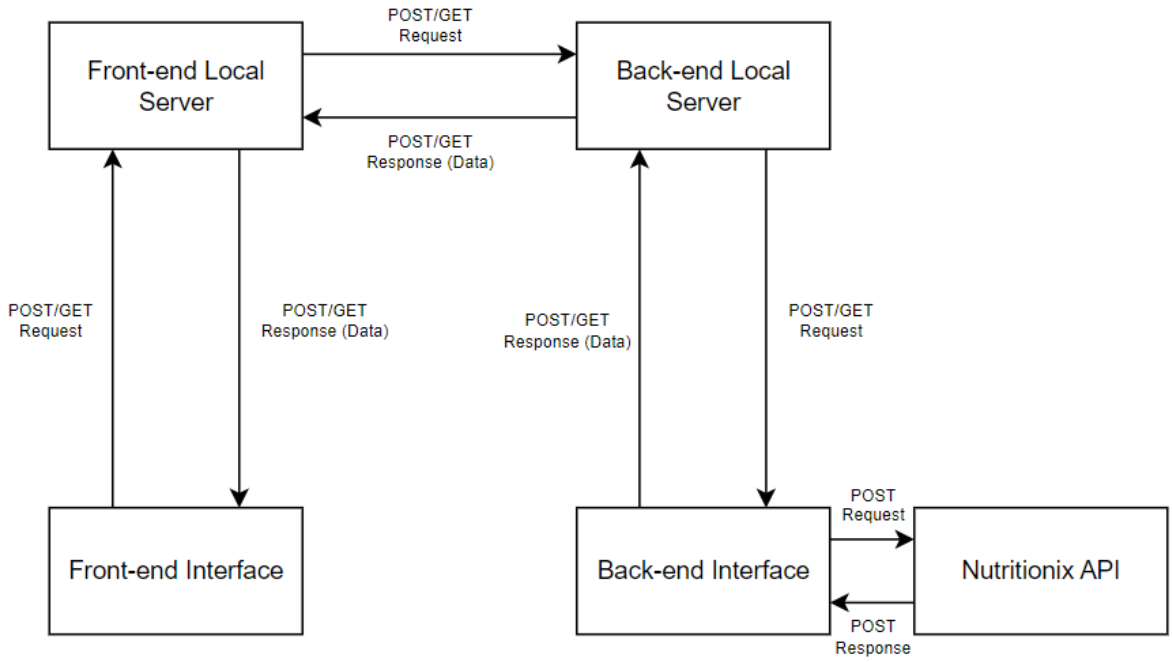


Figure 3: Communication between the front-end and back-end

E Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design. Please answer the following questions:

1. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)

2. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select documented design? (LO_Explores)

The Utrition development team has identified some shortcomings of our project. Firstly, Utrition utilizes the CIFAR-100 dataset to identify food items with TensorFlow, a machine learning algorithm. However, the CIFAR-100 dataset only contains 5 identifiable food items. With unlimited resources, the Utrition team would find a larger dataset containing a bigger variety of food items. With this larger dataset, our application would be able to identify many more food items with higher confidence. In addition to the improved machine learning technology, the application's aesthetic display could be improved to be more visually appealing and intuitive. Most notably, the front-end developers would add charts and diagrams to display a user's previous meals rather than listing them.

The original design for Utrition was to identify a food item from one form on input. This method required uploading an image that would utilize machine learning for identification. This input method was chosen since uploading an image is simple and doable for all age groups. However, the trade-off was that complex food items, such as full meals (not only simple ingredients), would be impossible for the algorithm to identify given our training dataset. As a result, users would quickly get frustrated with the near-useless application. Therefore, the developers behind Utrition decided that it was best to provide more searching options (textual and verbal input) to the user even if it came with the price of adding more clutter to the user interface. The developers did not simply want to add an option to type the food item, as those with accessibility concerns may struggle to type the food item correctly. Therefore, the developers decided it was best to provide the user with all three input options, so the users could decide what method was best for them and their goals.