System Design for Utrition

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

January 18, 2023

1 Revision History

Date	Version	Notes
January 18, 2023	1.0	Initial Document

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
Utrition	Explanation of program name
SRS	Software Requirements Specification
MG	Module Guide
MIS	Module Interface Specification

Contents

1 Revision History **i**

2 Reference Material **ii**
2.1 Relevant Documentation ii
2.2 Abbreviations and Acronyms ii

3 Introduction **1**
3.1 Document Purpose 1
3.2 System Purpose 1
3.3 Scope 1

4 Project Overview **2**
4.1 Normal Behaviour 2
4.2 Undesired Event Handling 3
4.3 Component Diagram 3
4.4 Connection Between Requirements and Design 5
4.4.1 Functional Requirements 5
4.4.2 Non-Functional Requirements 6

5 System Variables **9**
5.1 Monitored Variables 9
5.2 Controlled Variables 9
5.3 Constants Variables 9

6 User Interfaces **9**

7 Design of Hardware **11**

8 Design of Electrical Components **11**

9 Design of Communication Protocols **11**

10 Timeline **12**

A Interface **13**

B Mechanical Hardware **13**

C Electrical Components **13**

D Communication Protocols **13**

E Reflection **13**

List of Tables

1 Module Timeline 12

List of Figures

1 System Context Diagram 2
2 Communication between the front-end and back-end 13

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 from the internet, taking a photo on the device using Utrition, or by uploading a photo from the user's phone to the device using Utrition. 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.

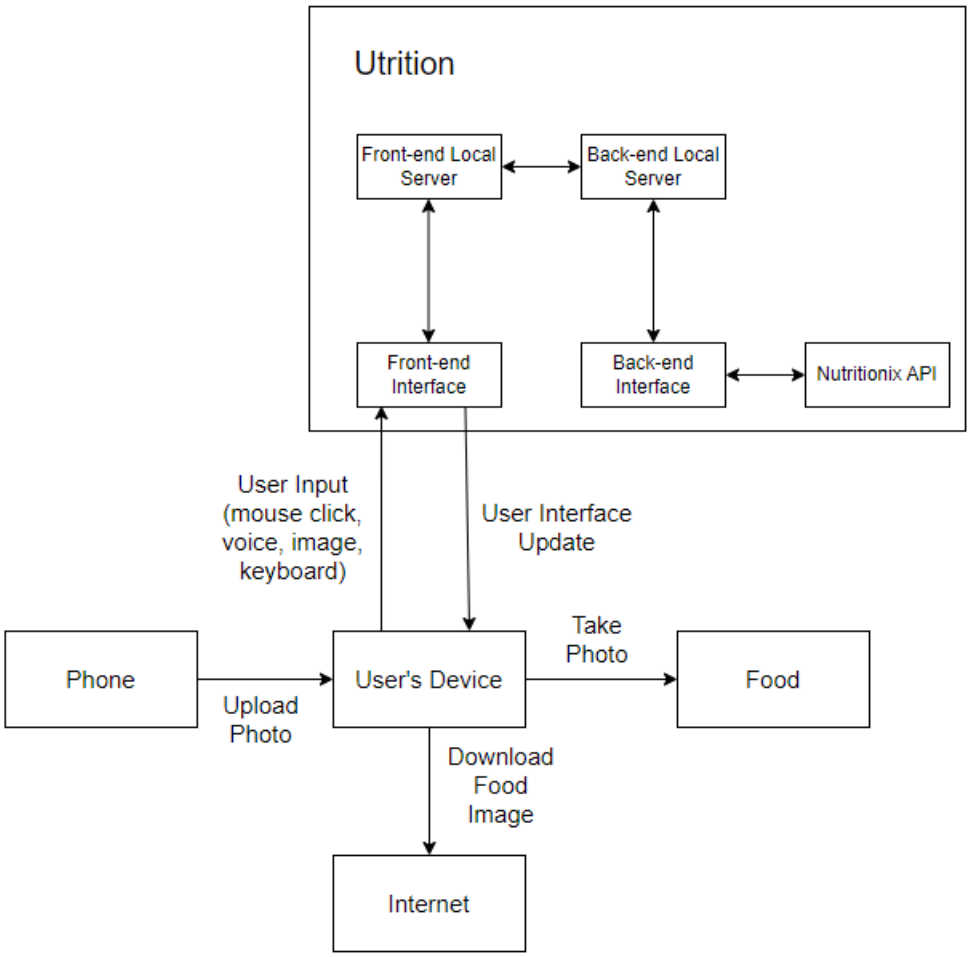


Figure 1: System Context Diagram

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 can include multiple images in their search by clicking on the upload button again.

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.

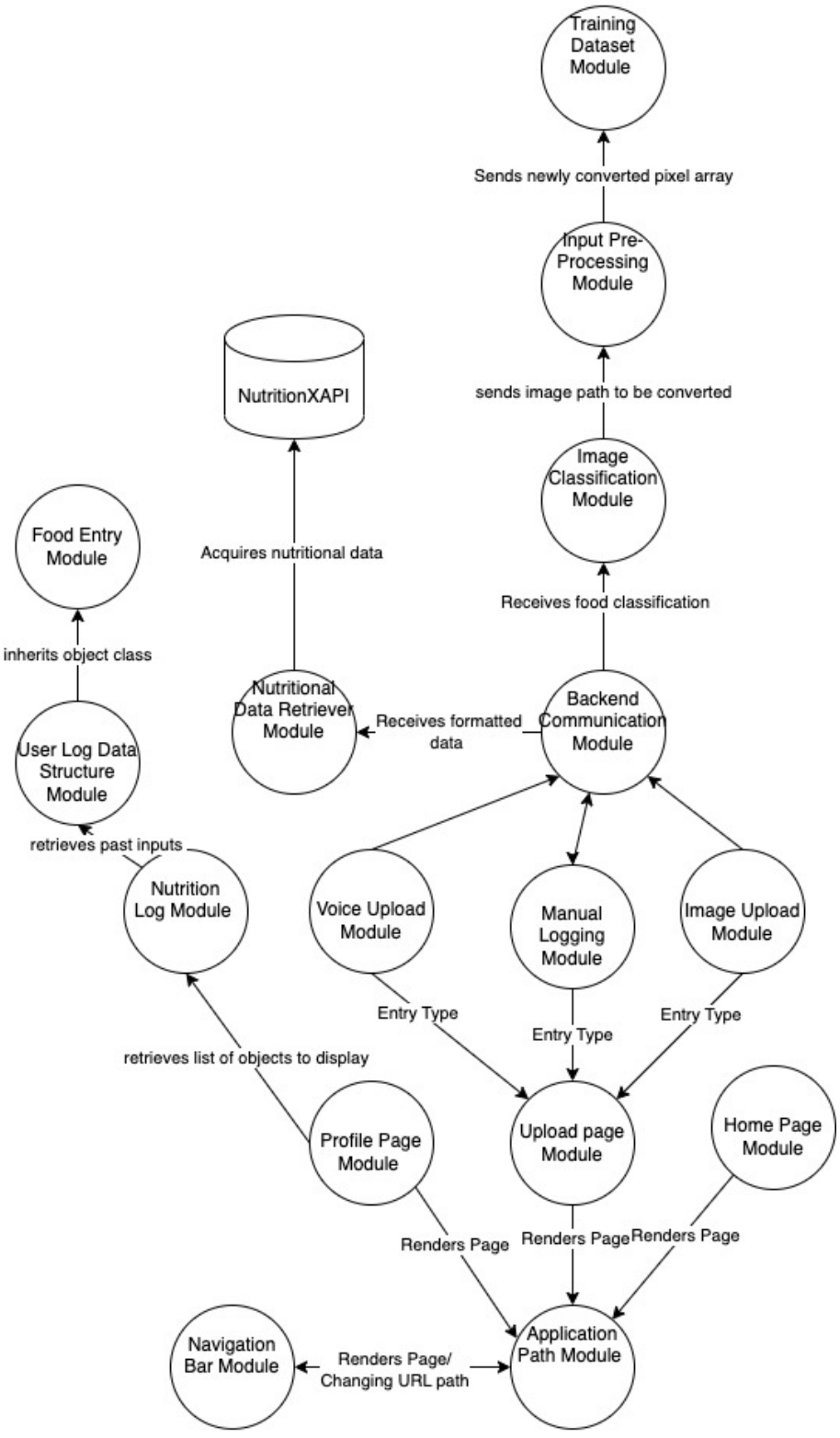
4.2 Undesired Event Handling

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


Training Dataset Module
Sends newly converted pixel array
Input Pre-Processing Module
sends image path to be converted
Image Classification Module
Receives food classification
Backend Communication Module
NutritionXAPI
Acquires nutritional data
Nutritional Data Retriever Module
Receives formatted data
Food Entry Module
inherits object class
User Log Data Structure Module
retrieves past inputs
Nutrition Log Module
Voice Upload Module
Manual Logging Module
Image Upload Module
Entry Type
Entry Type
Entry Type
retrieves list of objects to display
Profile Page Module
Upload page Module
Home Page Module
Renders Page
Renders Page
Renders Page
Navigation Bar Module
Renders Page/ Changing URL path
Application Path Module

4.4 Connection Between Requirements and Design

4.4.1 Functional Requirements

FR1. Provide user a screen with the option to click a button, which then leads the user to upload an image.

FR2. Provide the user a button to click on that allows them to enter another food item. After the second food item is entered, the button remains for more food items to be entered.

FR3. 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.

FR4. 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.

FR5. 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.

FR6. 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 as a JSON file on the users' Utrition folder.

FR7. 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 JSON files in the previously logged food items folder to the front-end interface (going through their respective local servers). The front-end interface displays the JSON files' data in reverse-chronological order.

FR8. 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.

FR9. Once the user requests to see the past nutritional data graph, the front-end interface switches to a different user interface screen. From there, 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 JSON files in the previously logged food items folder to the front-end interface (going through their respective local servers). The front-end

interface parses the data by using ChartJS to display a graph of past nutritional trends to the user.

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 FR6'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 an appropriate symbol next to it.

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. No calculations are done while retrieving nutritional facts.

PR4. The submitted image format is checked by the front-end interface If the image format is not ".png", ".jpg", or ".jpeg", an error message is returned to the user.

PR5. The submitted image size is checked by the front-end interface. If the image size is larger than 30MB, an error message is returned to the user.

PR6. After an image upload, the front-end interface counts the amount of images uploaded by the user. Once the number of images exceeds 3, an error message is displayed to the user and the ability to proceed with the food-identification system is removed.

PR7. Once the food image is sent to the back-end system, the machine learning algorithm begins a real-time stopwatch. After every iteration of the machine learning algorithm, the timer is checked. If the timer surpasses 10 seconds, an error message is sent to the back-end local server, which is sent to the front-end local server in order to then be sent to the front-end interface. The front-end interface displays the error message.

PR8. Once the back-end determines the food item, a POST request is sent to the Nutrionix API. The Nutritionix API will send back a response, and the response is verified by the Utrition back-end system. If the response is "Too many requests", the user cannot submit more food items. Once an error message is detected by the Utrition back-end system, an error message is sent to the back-end local server, which is sent to the front-end local server in order to then be sent to the front-end interface. The front-end interface displays the error message.

PR9. Once the back-end determines the food item, a POST request is sent to the Nutrionix API. The Nutritionix API will send back a response, and the response is verified by the Utrition back-end system. If the response is empty, the food item was not found in the Nutrionix API. The response is sent to the back-end local server, which is sent to the front-end local server in order to then be sent to the front-end interface. The front-end interface displays the error message.

PR10. After the back-end interface determines that the past nutritional logs folder is empty, the back-end returns an error message to the front-end interface's GET request through their respective local servers. The front-end interface displays the error message to the user.

PR11. If the back-end interface cannot find the past nutritional logs folder because it has been moved/deleted, the back-end returns an error message to the front-end interface's GET request through their respective local servers. The front-end interface displays the error message to the user.

PR12. The front-end system sends a request to the front-end local server, which travels to the back-end local server to the back-end system. If the back-end system cannot find any stored data, an error message is sent back to the local back-end server, which travels to the local front-end server to the front-end system. The error message is displayed to the user in the front-end interface.

PR13. Once the back-end determines the food item, a POST request is sent to the Nutrionix API. The Nutritionix API will send back a response, and the response is verified by the Utrition back-end system. If the response is a timed-out error, the user has an issue with their internet connection. Once an error message is detected by the Utrition back-end system, the message is sent to the back-end local server, which is sent to the front-end local server in order to then be sent to the front-end interface. The front-end interface displays the error message.

PR14. The learning rate of the machine learning algorithm is set to 0.005.

PR15. 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.

PR16. Utrition is a local web application.

PR17. After an image upload, the front-end interface counts the amount of images uploaded by the user. Once the number of images exceeds 3, an error message is displayed to the user and the ability to proceed with the food-identification system is removed.

PR18. Utrition is a local web application.

PR19. Utrition is available on GitHub.

OE1. Utrition is available on GitHub.

OE2. The installation guide on GitHub is written in simple English.

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 an installation guide, and a guide on how to use the web application on the Utrition GitHub.

MS3. This design decision is covered by PR4's design decision.

SR1. Utrition is available on GitHub.

SR2. This design decision is covered by FR6's design decision.

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 HTML 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

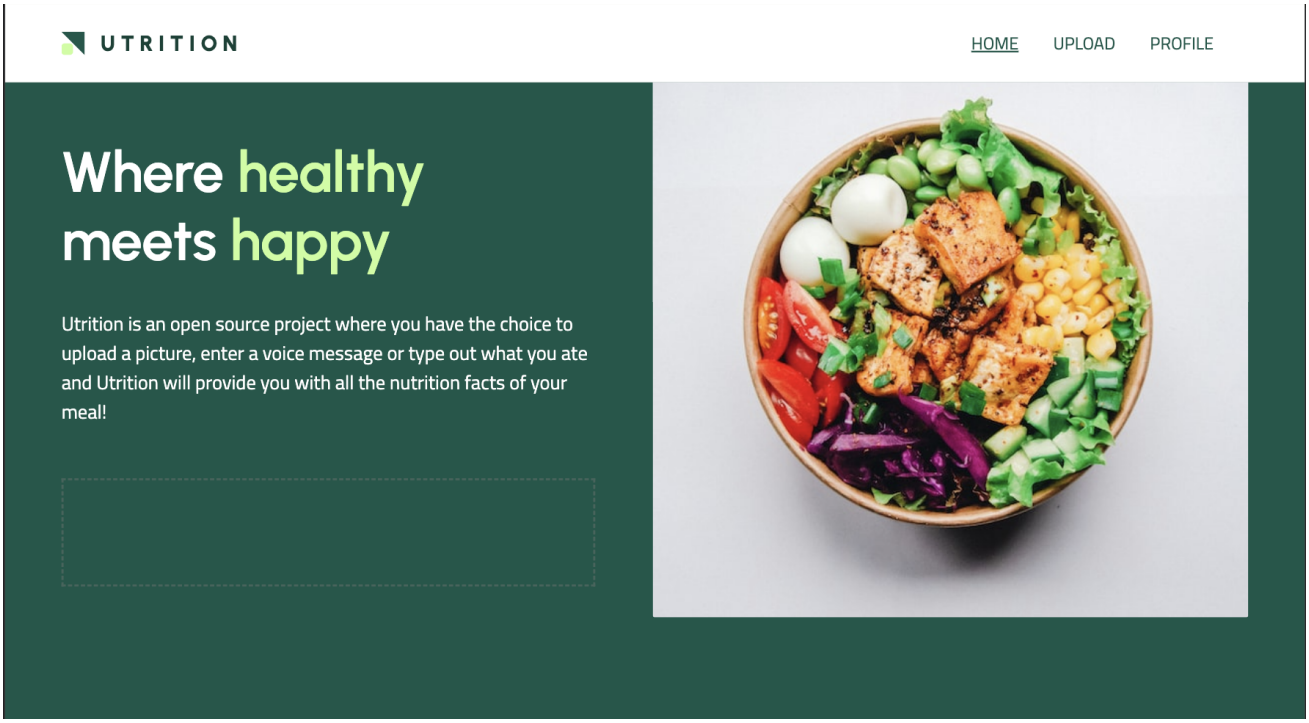
5.2 Controlled Variables

N/A

5.3 Constants Variables

N/A

6 User Interfaces



UTRITION

HOME UPLOAD PROFILE

HOW IT WORKS

UPLOAD YOUR MEAL

Choose one of the options below

Upload image | Upload Text | Upload Voice

Section

UTRITION

HOME UPLOAD PROFILE

Your most eaten food was Grapes!

You average calories consumed over the past 7 days was 2093 cal!

PAST MEALS

Jan 23 **Grapes -**
Calories - 200 per 100 grams Fats - 0 g Cholesterol - 0 g Protein - 1 g Fiber 1.4 g

Jan 23 **Pear-**
Calories - 100 per 100 grams Fats - 0 g Cholesterol - 0 g Protein - 1 g Fiber 1.4 g

Jan 23 **Hamburger -**
Calories - 1500 per 100 grams Fats - 0 g Cholesterol - 0 g Protein - 1 g Fiber 1.4 g

Jan 23 **Grilled Cheese**
Calories - 350 per 100 grams Fats - 0 g Cholesterol - 0 g Protein - 1 g Fiber 1.4 g

My Calroic Intake

DATE	Total Calories	What you ate...
Jan 17th	2100	Pancakes, Apple, Hamburger, Pasta
Jan 18th	2200	Oatmeal, Hamburger, Grilled Cheese
Jan 19th	1900	Pancakes, Apple, Hamburger
Jan 20th	2205	Grilled Cheese, Pear, Pancakes, Chicken Breast
Jan 21st	1995	Cereal, Sandwitch, Pasta
Jan 22nd	2100	Apple, Oatmeal, Pasta, Pea Chicken Breast,
Jan 23rd	2150	Grapes, Pear, Hamburger, Grilled Cheese

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

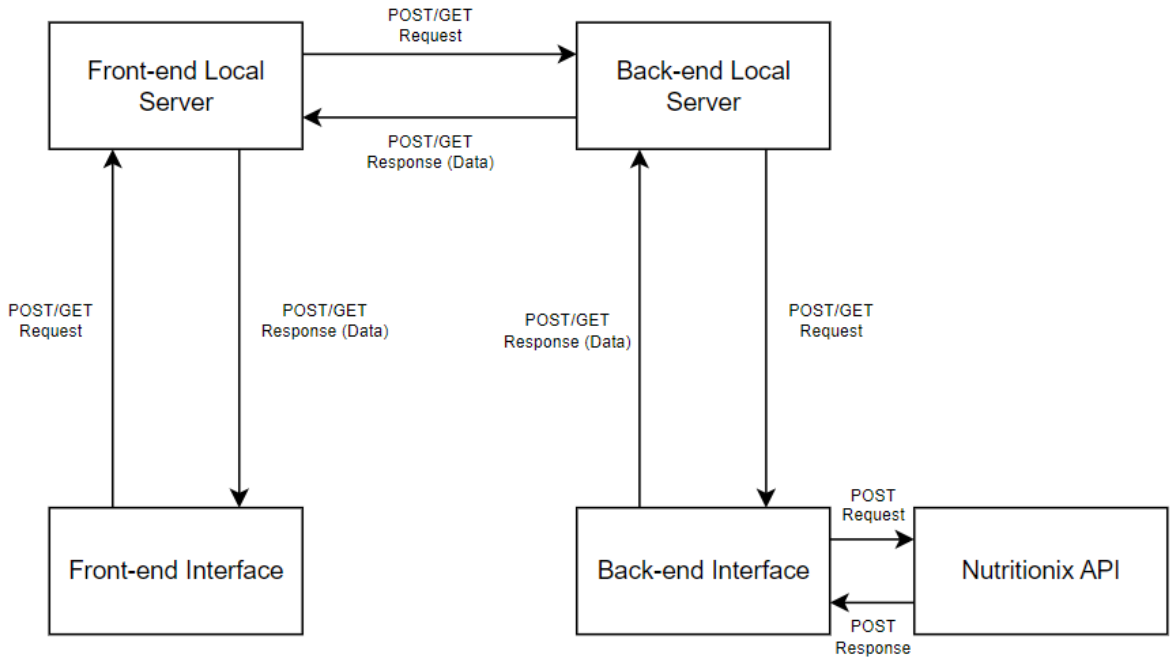


Figure 2: Communication between the front-end and back-end

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.