Utrition: System Verification and Validation Plan

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

March 5, 2023

1 Revision History

Date	Version	Notes
11/02/2022	1.0	Completed Version 1

Contents

List of Tables

2 Symbols, Abbreviations and Acronyms

symbol	description
App	Application
CT	Cultural Test
LF	Look and Feel Test
LT	Legal Test
MG	Module Guide
MIS	Module Interface Specification
MS	Maintainability and Support
OE	Operational and Environmental Test
PR	Pull Request
PT	Performance Test
SFWRENG 4G06	Software Engineering 4G06 course
SRS	Software Requirements Specification
ST	Security Test
UH	Usability and Humanity Test
UI	User Interface
VnV	Verification and Validation

This document depicts the Verification and Validation Plan for the Utrition project. This includes verifying both implementation and documentation. The document will depict the plans for reviewing all critical documentation along with the system and unit tests to be used for code verification.

3 General Information

3.1 Summary

The purpose behind this document is to provide the testing plan that Utrition will undergo during its development period. This plan will ensure a high-quality final product. Utrition is an application that allows individuals to discover the nutritional value of the foods they are consuming and track their previously eaten meals. This software consists of three main functions that are being tested such as image upload, image processing, and displaying nutritional information.

3.2 Objectives

The objectives for Utrition's VnV plan is to confirm that the implementation and documentation is as effective as possible to produce the best final product. Firstly, the developers of Utrition would like to ensure the correctness of the application's internal processes. The team must ensure that the artificial intelligence can correctly identify the food that is depicted in the uploaded image. The output the nutritional information must also be correct. Since the application is health-based, all nutritional facts must be correct since users will be using this information to influence their eating habits. Additionally, the Utrition application must be robust in order to deal with common and uncommon errors that may potentially occur. One such error is the user uploading an image of the wrong type. All tests in this document are written to ensure that the code meets these objectives.

3.3 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)

4 Plan

This section details Durum Wheat Semolina's approaches for verifying and validating Utrition's documentation and implementation. Specific documentation to be reviewed are the SRS, Design documents (MG and MIS), and VnV Plan. Additionally, the verification and validation of Utrition's implementation is discussed. Tools being used for verification including automated testing frameworks and linters are also detailed below.

4.1 Verification and Validation Team

Each team member will be contributing to the verification and validation of Utrition's supporting documentation and implementation. Jack Theriault is the team's testing lead and can be deferred to for any queries regarding verification procedures. It is expected that all team members create the tests that correspond to their written code. In addition, each team member's specific role in the verification and validation process is given below.

Team Member	Role	Description
Alexander Moica	Dynamic Verification Lead	This role is responsible for leading the dynamic verification for Utrition's implementation.
Yasmine Jolly	Software Validation Lead	This role is responsible for leading the validation effort of Utrition.
Jeffrey Wang	Static Verification Lead	This role is responsible for leading the static verification for Utrition's implementation.
Jack Theriault	VnV Plan Verification Lead	This role is responsible for leading the verification effort of the VnV plan.
Catherine Chen	Design Verification Lead	This role is responsible for leading the verification effort of the design documents.
Justina Srebrnjak	SRS Verification Lead	This role is responsible for leading the verification effort of the SRS.

Table 1: Team Member Roles for Verification and Validation Testing

4.2 SRS Verification Plan

To verify Utrition's SRS document, two approaches will be primarily used. Firstly, ad hoc reviews by fellow classmates are done to provide Durum Wheat Semolina with feedback from an outsider perspective. The rubric for the SRS is heavily used during this process.

Once feedback from these reviews has been collected in addition to the feedback given by SFWRENG 4G06 TAs while marking version 0, the team will walk through the SRS and make any necessary changes. Since the SRS is critical when implementing Utrition, this review process will be done by all team members to ensure total coverage of possible improvements. The team will be instructed to go through all functional and non-functional require-

ments to identify any ambiguities or missing requirements. Once identified, the corresponding changes will be made. This walkthrough will be led by Justina Srebrnjak who is in charge of SRS verification.

4.3 Design Verification Plan

Utrition's design documents will be verified through a combination of peer reviews and documentation walkthroughs. Once version 0 of the design documents has been completed, students from SFWRENG 4G06 will perform ad hoc reviews based on the criteria given in the design rubric. These critiques will make Durum Wheat Semolina aware of any discrepancies found in the documentation.

Secondly, documentation walkthroughs will be conducted by all team members to verify the completeness of the design documents. The primary purpose of these walkthroughs is to verify that all functional requirements listed in the SRS document are fulfilled through the proposed design. Any missing functionality will be identified through this process and amended as needed. This verification plan will be led by Catherine Chen.

4.4 Verification and Validation Plan Verification Plan

The verification plan for the VnV plan document will consist of two techniques. Firstly, ad hoc reviews from the team's classmates in SFWRENG 4G06 will be conducted. These reviews will identify any discrepancies indicated by the VnV plan rubric. Their resulting feedback will contribute to any changes made to the document.

In addition to peer reviews, the Utrition team will participate in a walkthrough of the document while referring to the SRS. During this review, the team will verify that every requirement found in the SRS has at least one test that will test its completeness. This review will be led by Jack Theriault.

4.5 Implementation Verification Plan

Verifying Utrition's implementation will be done through a variety of dynamic and static verification techniques. In terms of dynamic methods, a variety of system and unit tests will be performed on the written code. More

information on these tests can be found in sections 5 and 6 of this document. These tests will be created by the entire team. More specifically, the people who work on a certain functionality of Utrition will be responsible for creating the corresponding test cases. These tests will be automated and run on the system whenever new code is pushed to the team's main branch on GitHub. This ensures that new code does not create any regressions in the current system. Alexander Moica will be the dynamic verification lead which entails monitoring new test case additions and ensuring full testing coverage of new code which includes edge cases and error cases.

Static verification will be done primarily through coding standards and code inspections. When writing code, each team member will be responsible for abiding by the coding guidelines outlined in Utrition's Development Plan in section 7. Python development will use the linter Pylint to enforce PEP8 standards. The Google style guides for JavaScript, HTML, and CSS will need to be followed manually. Additionally, coding inspections will be done for every new PR that is made to the team's main branch in GitHub. Before any PR is merged, at least one team member will be required to review the incoming code. During these reviews, team members will ensure complete functionality fulfillment inline with documentation, proper coding style, efficient implementation choices, and consistent naming practices for files, functions, and variables. Jeffrey Wang will lead this static verification plan to ensure all coding inspections are completed and answer any questions regarding static verification.

4.6 Automated Testing and Verification Tools

The tools used for automated testing and verification have been previously outlined in the Development Plan for Utrition in sections 6 and 7.

4.7 Software Validation Plan

In order to validate Utrition's implementation, interviews will be scheduled with potential users. In these interviews, the interview conductor will walk through the functional requirements of the system with the interviewee. The interviewee will be asked for their feedback on the current requirements of the system. This feedback may include adding or removing specific functionality (i.e. altering requirements). Software validation will be led by Yasmine Jolly.

5 System Test Description

5.1 Tests for Functional Requirements

The below areas cover all the nine functional requirements outlined in the SRS, dividing the functional requirements into three distinct sections. The first section details user actions associated with image upload and encompasses FR1 and FR2. The second section details image processing done by the system which encompasses FR3, FR4, FR5, FR6, and FR7. Lastly, the third section outlines the user steps to view nutritional data which correlates to groupings of functional requirements.

5.1.1 Image Upload

This section has three tests, with each corresponding to an action the user can take with Utrition, corresponding to FR1 and FR2 in the SRS. Testing that users can successfully submit one or multiple images is important to the core functionality of the other components of the application.

Single image upload

1. one-upload-1

   Control: Functional, Dynamic, and Manual.

   Initial State: No image in the system.

   Input: An image file of type .png, .jpg, or .jpeg (images found in utrition/test/testPhotos directory).

   Output: The system accepts the image upload.

   Test Case Derivation: The system should accept an uploaded image file from the user.

   How test will be performed: A locally stored image will be uploaded to the application through the image upload prompt.

2. one-upload-2

   Control: Functional, Dynamic, and Manual.

Initial State: No image in the system.

Input: A file of extension .txt will be uploaded into the system (found in utrition/test/testPhotos directory).

Output: The system will not contain the uploaded file. An error message will be returned.

Test Case Derivation: The system should only save the inputted image if it is of the proper file extension.

How test will be performed: A file of type .txt will be saved on the tester's device. The tester will upload this file to the system.

Multi-image upload

1. multi-upload-1

   Control: Functional, Dynamic, and Manual.

   Initial State: No image in the system.

   Input: Three image files of type .png, .jpg, or .jpeg (images found in utrition/test/testPhotos directory).

   Output: The system accepts the image uploads.

   Test Case Derivation: The system should accept multiple uploaded image files from the user at the same time.

   How test will be performed: Three locally stored images will be uploaded to the application through image upload prompts.

5.1.2 Image Processing

This section contains tests that relate to the processing of an image uploaded by the user. Tests correspond to actions that will be taken by the application during image processing. Relevant functional requirements include FR3, FR4, FR5, FR6, and FR7.

Image Identification

1. image-identification-1

Control: Functional and Automatic.

Initial State: System contains an image of extension .png type.

Input: System is prompted to process the image.

Output: The type of food that is captured in the image is returned as a string.

Test Case Derivation: The system should identify the food present in an image, and return the name of the food item as a string.

How test will be performed: The system identified item will be verified that it matches the true item contained in the image.

2. image-identification-2

   Control: Functional and Automatic.

   Initial State: System contains an image of extension .jpg type.

   Input: System is prompted to process the image.

   Output: The type of food that is captured in the image is returned as a string.

   Test Case Derivation: The system should identify the food present in an image, and return the name of the food item as a string.

   How test will be performed: The system identified item will be verified that it matches the true item contained in the image.

3. image-identification-3

   Control: Functional and Automatic.

   Initial State: System contains an image of extension .jpeg type.

   Input: System is prompted to process the image.

   Output: The type of food that is captured in the image is returned as a string.

   Test Case Derivation: The system should identify the food present in an image, and return the name of the food item as a string.

   How test will be performed: The system identified item will be verified that it matches the true item contained in the image.

Call and Fetch API Response

1. api-1

   Control: Functional and Automatic.

   Initial State: System on standby.

   Input: Food item name as a string.

   Output: JSON containing nutrition facts for the inputted food item.

   Test Case Derivation: A request will be made to the Nutritionix API with the food item name, which will return a response body containing the food item's nutrition facts.

   How test will be performed: The API response will be verified that the contents are as expected.

Logging Data

1. log-data-1

   Control: Functional and Automatic.

   Initial State: System contains nutritional data of a food item.

   Input: Request to log the data to the database.

   Output: Database is updated as the system writes the data to the database.

   Test Case Derivation: Nutritional data of a food will be saved for future reference. This is done by logging the data to the database.

   How test will be performed: After the system receives the request to update the database, the database will be verified that it is updated with the new data.

Reading Data

1. read-data-1

   Control: Functional and Automatic.

Initial State: Database contains nutritional data of a food item.

Input: Request to fetch the data of a particular food item.

Output: Nutritional data of the requested food item.

Test Case Derivation: The system shall be able to fetch previously recorded data.

How test will be performed: After the system receives the request to fetch the data of a food item, the returned data will be verified that it matches with the saved data in the database.

2. read-data-2

   Control: Functional and Automatic

   Initial State: Database contains nutritional data of 5 different dates.

   Input: Request to fetch the data from the last 3 most recent dates.

   Output: A list of the nutritional data from the last 3 most recent dates.

   Test Case Derivation: The system shall be able to fetch previously recorded data. The system shall be able to fetch data from a range of dates.

   How test will be performed: The nutritional data from the 3 most recent dates will be verified if it matches the data in the database.

3. read-data-3

   Control: Functional and Automatic

   Initial State: Database contains nutritional data.

   Input: Request to fetch the data of a particular food item that is not logged in the database.

   Output: No data is returned.

   Test Case Derivation: The system shall be able to fetch previously recorded data. This is an edge case for if the system is requested to read data from the database that does not exist. In this case, no data will be returned.

   How test will be performed: The output will be verified to be empty.

5.1.3 Nutritional Facts Display

This section has three tests, with each corresponding to an action the user can take with Utrition. Each test corresponds to (FR3, FR4, FR5, FR6, FR8), (FR7, FR8), and (FR7, FR8, FR9) respectively in the SRS. These tests will test the actions the user can take after uploading an image, which are comprised of viewing past and present nutritional data in textual and graphical formats.

Viewing the nutritional data of an uploaded food image

1. current-nutrition-1

   Control: Functional, Dynamic, and Automatic

   Initial State: Image(s) exists in system

   Input: Input signal from the image upload process

   Output: The nutritional data of the food

   Test Case Derivation: Primary black-box test for our system as a whole. The system will take the uploaded image(s), analyse them for the food they display, cross-reference them with a nutrition database, and display the information to the user.

   How test will be performed: The program will automatically test that the resulting nutrition data of a pre-uploaded image file matches that of the nutrition database entry for the food.

Viewing textual nutritional data of logged foods

1. past-nutrition-text-1

   Control: Functional, Dynamic, and Automatic

   Initial State: Logged foods exist in system

   Input: Request to view nutritional data of logged foods as text

   Output: The nutritional data of past logged foods in a textual format

   Test Case Derivation: The system will keep a record of all analysed foods for the user, allowing the user to request to see the nutritional

data of their past uploads. This data can be displayed in a human readable textual format.

How test will be performed: The program will automatically test that a file of pre-logged foods can be processed to return their corresponding nutritional data as text.

Viewing graphical nutritional data of logged foods

1. past-nutrition-graph-1

   Control: Functional, Dynamic, and Automatic

   Initial State: Logged foods exist in system

   Input: Request to view nutritional data of logged foods as graph

   Output: The nutritional data of past logged foods in a graphical format

   Test Case Derivation: The system will keep a record of all analysed foods for the user, allowing the user to request to see the nutritional data of their past uploads. This data can be displayed in a graphical format.

   How test will be performed: The program will automatically test that a file of pre-logged foods can be processed to return their corresponding nutritional data in the form of a graph.

5.2 Traceability Between Test Cases and Functional Requirements

Requirement #	**Description**	**Test ID(s)**
FR1	The user must have the ability to upload a digital image of a standard image type to the system.	one-upload-1 one-upload-2
FR2	The user shall have the ability to upload multiple digital images of standard image types to the system.	multi-upload-1
FR3	The system will be able to identify the type of food that is captured in an image.	image-identification-1 image-identification-2 image-identification-3 current-nutrition-1
FR4	The system will be able to make an API call to an external database of nutrition facts for a variety of foods, including their macro-nutrients, micro-nutrients, and caloric details.	api-1 current-nutrition-1
FR5	The system will be able to retrieve the nutrition facts for a specific food.	api-1 read-data-1 current-nutrition-1
FR6	The system will log the nutritional data of a food.	api-1 log-data-1 current-nutrition-1
FR7	The user will be able to access the nutritional data of previously logged foods.	read-data-1 read-data-2 read-data-3 past-nutrition-text-1 past-nutrition-graph-1
FR8	The system will display the nutritional information of a food to the user.	current-nutrition-1 past-nutrition-text-1 past-nutrition-graph-1
FR9	The system will display the history of logged nutritional data in a graph.	past-nutrition-graph-1

Table 2: Traceability between System Tests and Functional Requirements

5.3 Tests for Nonfunctional Requirements

5.3.1 Look and Feel Testing

1. LF-1

   Type: Functional, Dynamic, and Manual.

   Initial State: A developer from Durum Wheat Semolina has launched the Utrition app.

   Input/Condition: The developer measures the distance between the displayed user interface components.

   Output/Result: The distance between user interface components must surpass 20 pixels.

   How test will be performed: Each developer launches the Utrition app and opens the Pixel Measuring Tool. The developer will use the tool by viewing a page on Utrition, then clicking "Print Screen" on their keyboard. The developer then pastes the screenshot into the tool. The developer clicks the two spots he would like to measure distance from.

5.3.2 Usability and Humanity Testing

1. UH-1

   Type: Functional, Dynamic, and Manual.

   Initial State: The user is viewing a page of the user interface.

   Input/Condition: The user clicks on the "Menu" button located at the bottom of the user interface.

   Output/Result: The user will be brought to the main menu screen.

   How test will be performed: A developer of Durum Wheat Semolina will access every page listed in "Initial State" and will attempt to reach the main menu in 2 or less clicks from the respective page.

2. UH-2

   Type: Functional, Dynamic, and Manual.

   Initial State: The user has previously entered 5 items in Utrition and is now viewing the main menu.

   Input: The user makes the request to view past nutritional data.

Output: The user views a list displaying all 5 food items with their respective details in: food name, calories, fat, sodium, proteins, carbohydrates, sugars, and date entered.

How test will be performed: A developer of Durum Wheat Semolina will open Utrition on their personal device and access the main menu. The developer will upload an image of different food items 5 times, with a minimum 1 minute time difference between each input. The uploaded images of the food items will be randomly selected by the developer in the utrition/test/testPhotos directory. The developer navigates to the past inputted foods area and views the list displaying the 5 different food items and their respective details.

3. UH-3

   Type: Functional, Dynamic, and Manual.

   Initial State: The user has previously entered 5 items in Utrition and has requested to view their past inputted food items.

   Input/Condition: The user navigates to the screen where they can view their past nutritional data in a chart format.

   Output/Result: The user views a chart displaying all 5 food items with their respective details in: food name, calories, proteins, carbohydrates, sugars, and date entered.

   How test will be performed: A developer of Durum Wheat Semolina will open Utrition on their personal device and access the main menu. The developer will upload an image of different food items 5 times, with a minimum 1 minute time difference between each input. The uploaded images of the food items will be randomly selected by the developer in the utrition/test/testPhotos directory. The developer navigates to the past inputted foods area where they can view their statistics as a chart. The developer views a chart displaying 5 different food items and their respective details.

4. UH-4

   Type: Functional, Dynamic, and Manual.

   Initial State: Utrition's main menu is opened on the user's personal device.

Input/Condition: The user uploads a photo of food.

Output/Result: The user views the food's nutritional information.

How test will be performed: Each developer of Durum Wheat Semolina will load Utrition on to their personal device. Each developer will ask 2 people aged 14 or older to upload and retrieve the nutritional data of the food item "pineapple.jpg" found in the utrition/test/testPhotos directory. The developer will track the time it takes for each test. 90% of the panel must complete the task in under 2 minutes.

5. UH-5

   Type: Functional, Dynamic, and Manual.

   Initial State: The user loads into Utrition's main menu.

   Input/Condition: The user views every page of the user interface.

   Output/Result: The user will see the respective symbol associated with every mention of calories, fat, sodium carbohydrates, sugar, and protein.

   How test will be performed: On each screen, including the main menu, a developer on Durum Wheat Semolina will check if there is a symbol associated with every mention of calories, fat, sodium carbohydrates, sugar, and protein. The developer will open Utrition on their personal device. The developer will upload a .png image of food found in the utrition/test/testPhotos directory. The developer will view the nutritional information venture through each page to confirm the correct symbols appear.

6. UH-6

   Type: Functional, Dynamic, and Manual.

   Initial State: The user has previously entered 2 items in Utrition and is now viewing the main menu.

   Input/Condition: The user views every page of the user interface.

   Output/Result: The user does not see any backend calculations.

   How test will be performed: On each screen, a developer on Durum Wheat Semolina will check if any backend calculations are displayed to the user. The developer will open Utrition on their personal device

and access the main menu. The developer will upload an image of two different food items, with a minimum of 1 minute between the inputs. The uploaded images of the food items will be randomly selected by the developer in the utrition/test/testPhotos directory. The developer will view all pages to ensure no backend calculations are visible.

7. UH-7

   Type: Functional, Dynamic, and Manual.

   Initial State: The user opens Utrition with a stable internet connection.

   Input/Condition: The user views every page of the user interface.

   Output/Result:

   - Utrition is available to be used by users at all times.
   - The user does not hear Utrition generate any sound.

   How test will be performed: On each screen, a developer on Durum Wheat Semolina will check if a sound plays. The developer will open Utrition on their personal device. The developer will upload an image of food. The uploaded image of the food item will be randomly selected by the developer in the utrition/test/testPhotos directory. The developer will view the nutritional information. The developer then will view the past inputs and the nutritional information chart.

8. UH-8

   Type: Static and Manual.

   Initial State: A developer on Durum Wheat Semolina views Utrition's GitHub page.

   Input/Condition: The developer checks every directory for an audio file.

   Output/Result: The developer removes all audio files found in Utrition's GitHub.

   How test will be performed: The developer loads into Utrition's GitHub and views the files in every directory. For each directory the developer will check for the following file types:

   - MP3

- AAC
- Ogg Vorbis
- FLAC
- ALAC
- WAV
- AIFF
- DSD
- PCM

Semi-structured interviewing will also be conducted to further receive insights and feedback regarding Utrition's overall implementation.

5.3.3 Performance Testing

1. PT-1

   Type: Functional, Dynamic, and Manual.

   Initial State: The user has previously entered 10 items in Utrition.

   Input/Condition: The user views every page of the user interface.

   Output/Result: The user will be able to access every page of the user interface in 2 seconds or less.

   How test will be performed: A developer on Durum Wheat Semolina will measure the time it takes for each new screen to load using a stopwatch. The developer will open Utrition on their personal device. The developer will upload an image of different food items 10 times, with a minimum 1 minute time difference between the inputs. The uploaded images of the food items will be randomly selected by the developer in the utrition/test/testPhotos directory. The developer will navigate to each screen, timing each transition.

2. PT-2

   Type: Functional, Dynamic, and Manual.

   Initial State: The user loads into Utrition.

   Input/Condition: The user uploads 3 images simultaneously.

   Output/Result:

- The user is able to view the identification for all 3 food items in 10 or less seconds.
- The user is able to view the nutritional information for all 3 food items in 5 or less seconds.

How test will be performed: A developer on Durum Wheat Semolina will open Utrition. The developer will upload three images of food found in the utrition/test/testPhotos directory. The developer clicks to view the foods' nutritional information. The developer measures the amount of time it takes for Utrition to notify the user of the name of the identified food items. The developer measures the amount of time it takes for the system to change from the food identification interface to the nutritional information interface.

3. PT-3

   Type: Functional, Dynamic, Automatic

   Initial State: Nutritional data of a food item is contained in the system.

   Input/Condition: The user requests to display the nutritional information of a food item.

   Output/Result: The system will generate the HTML body to be displayed on the user interface.

   How test will be performed: The test will measure if this interaction was completed in 5 seconds or less.

4. PT-4

   Type: Functional, Dynamic, and Manual.

   Initial State: The user loads into Utrition.

   Input/Condition: The user uploads an image of food.

   Output/Result: The food is correctly identified, and the correlated nutritional facts are correct.

   How test will be performed: A developer on Durum Wheat Semolina will open Utrition and upload the images of the following foods separately (images found in utrition/test/testPhotos directory):

   - Uncooked Pork

- Corn
- Lettuce
- Beef
- Penne Pasta
- Rice
- Milk
- Butter
- Wheat Bread
- Orange Juice

The developer will check the accuracy of each identified food item and their respective nutritional facts. 70% of the images must be identified correctly. 90% of the nutrition facts for the correctly identified foods will be given correctly.

5. PT-5

   Type: Functional, Dynamic, and Automatic.

   Initial State: System contains a set of images; all with foods, and of extension .jpg, .png, or .jpeg.

   Input/Condition: System prompted to process the image, for each image within the provided set (images found in utrition/test/testPhotos directory).

   Output/Result: For each image, the system should identify the food present in an image and return the name of the food item as a string.

   How test will be performed: The test will check if among all the provided images, more than 90% of the images have been assessed correctly.

6. PT-6

   Type: Functional, Dynamic, and Automatic.

   Initial State: System contains a set of images; some with and others without foods, and of extension .jpg, .png, or .jpeg.

Input/Condition: System prompted to process image, for each image within the provided set (images found in utrition/test/testPhotos directory).

Output/Result: The system should identify the food present in an image, and return the name of the food item as a string, and prompt the user if no food is detected.

How test will be performed: The test will check if among all the provided images, more than 70% of the images have been assessed correctly.

7. PT-7

   Type: Functional, Dynamic, and Manual.

   Initial State: The user loads into Utrition.

   Input/Condition: The user uploads 4 photos of different food items.

   Output/Result: The user is notified that they are not able to proceed with viewing the identified foods or their nutritional information.

   How test will be performed: A developer on Durum Wheat Semolina will open Utrition and attempt to upload four images of random food items found in the utrition/test/testPhotos directory. The developer is notified with an error informing them that only 3 images can be uploaded at once.

8. PT-8

   Type: Static and Manual.

   Initial State: A developer on Durum Wheat Semolina opens their internet browser.

   Input/Condition: The developer is able to access Utrition's GitHub and download the repository on to their personal device.

   Output/Result: Utrition is able to be downloaded.

   How test will be performed: The developer will access Utrition's GitHub on their personal device's web browser. The developer will click on "Code", and then "Download ZIP". The developer will verify that Utrition has been downloaded on to their device.

9. PT-9

   Type: Functional, Dynamic, and Manual.

   Initial State: The user loads into Utrition.

   Input/Condition: The user uploads a very large photo (more than 30MB) for Utrition to identify.

   Output/Result: The user is notified that they are not able to proceed with viewing the identified foods or their nutritional information.

   How test will be performed: A developer on Durum Wheat Semolina will open Utrition and upload "art.png" found in the utrition/test/testPhotos directory. The developer is notified that they cannot proceed with viewing the nutritional information unless they decrease the uploaded image's file size.

10. PT-10

    Type: Functional, Dynamic, and Manual.

    Initial State: The user loads into Utrition.

    Input/Condition: The user uploads an unintelligible image for Utrition to identify.

    Output/Result: The user is notified that Utrition is not confident in the system's identification. The system will suggest uploading another image.

    How test will be performed: A developer on Durum Wheat Semolina will open Utrition and upload "desk.jpg" found in the utrition/test/testPhotos directory. The developer is notified that the machine learning algorithm is not confident in the identification of the food, and that they should upload another image.

11. PT-11

    Type: Functional, Dynamic, and Automatic.

    Initial State: The system is identifying food objects within the user's uploaded image(s).

    Input/Condition: Uploaded image.

    Output/Result: An error message is displayed prompting the user of a failed identification.

How test will be performed: The test will check if an error message element has been rendered.

12. PT-12 Type: Functional, Dynamic, and Manual.

    Initial State: The user loads into Utrition without connecting to the internet.

    Input/Condition: The user uploads an image for Utrition to identify.

    Output/Result: The user is notified that the system is unable to retrieve nutritional information because they are not connected to the internet.

    How test will be performed: A developer on Durum Wheat Semolina will open Utrition upload a random working image found in the utrition/test/testPhotos directory. The developer is notified that they are not connected to the internet after 20 seconds.

13. PT-13 Type: Functional, Dynamic, and Manual.

    Initial State: The user loads into Utrition.

    Input/Condition: The user uploads an image for Utrition to identify.

    Output/Result: The user is notified that the system has identified the food item, but could not find the food's nutritional data.

    How test will be performed: A developer on Durum Wheat Semolina will open Utrition and upload "dragonfruit.jpg" found in the utrition/test/testPhotos directory. The developer is notified that they have uploaded an image of a "Dragon Fruit", but the nutritional data cannot be found.

14. PT-14

    Type: Functional, Dynamic, and Manual.

    Initial State: The user opens Utrition without previously inputting any food items.

    Input/Condition: The user attempts to view their past inputs' nutritional information as text and as a chart.

    Output/Result: The user views an empty list with column entries: food name, calories, proteins, carbohydrates, sugars, and date entered. The

user will see an empty canvas for the chart, and text detailing that there are no previous nutritional logs.

How test will be performed: A developer of Durum Wheat Semolina will open Utrition on their personal device and view nutritional data from past inputs. A message displays "There are no previous food items recorded" and an empty chart.

15. PT-15 Type: Functional, Dynamic, and Manual.

    Initial State: A user loads into Utrition.

    Input/Condition: The user uploads a PDF document as an image.

    Output/Result: The user is notified that they are not able to proceed with viewing the identified foods or their nutritional information.

    How test will be performed: A developer on Durum Wheat Semolina will open Utrition upload the image "waffle.pdf" to Utrition from the utrition/test/testPhotos directory. The developer is notified that they cannot upload the file since Utrition does not support ".pdf" file types.

5.3.4 Operational and Environmental Testing

1. OE-1

   Type: Static and Manual.

   Initial State: A user aged 14 or older has read Utrition's "README.md" located at the bottom of the GitHub page.

   Input/Condition: The user installs Utrition on their personal device.

   Output/Result: The intended demographic is able to install Utrition on their computer.

   How test will be performed: Each developer on Durum Wheat Semolina will test 5 individuals aged 14 and above to install Utrition. The test passes if 95% of users can install Utrition.

2. OE-2

   Type: Static and Manual.

   Initial State: A user visits Utrition's GitHub page.

   Input/Condition: The user checks that the latest change to Utrition's GitHub is not after April 30th, 2022.

Output/Result: Utrition will not have any updates after the final release.

How test will be performed: A developer on Durum Wheat Semolina will visit Utrition's GitHub page bi-yearly to ensure no updates are made to Utrition's GitHub.

5.3.5 Maintainability and Support Testing

1. MS-1

   Type: Functional, Dynamic, and Manual.

   Initial State: The developer is in the utrition/src/database directory.

   Input/Condition: The developer switches the database file found in the database directory for another database.

   Output/Result: Backend developers are able to upload new data to Utrition's database.

   How test will be performed: The developer travels to the utrition/src/database directory on their device. The developer replaces the database found in the directory with "testdatabase.jpg", which can be found in the utrition/test/testPhotos directory. The developer launches Utrition on their device to see that the system still runs.

2. MS-2

   Type: Functional, Dynamic, and Manual.

   Initial State: The user has installed Utrition on to their Windows, macOS, or Linux device.

   Input/Condition: The user is able to upload an image of food and view all pages of the user interface.

   Output/Result: All of Utrition's functionality can be accessed by Windows, macOS, and Linux devices.

   How test will be performed: Developers of Durum Wheat Semolina will install Utrition on one of each Windows, macOS, and Linux device. The developer will open Utrition on the device and upload a random image of a food item found in the utrition/test/testPhotos directory. The developer sees their foods' nutritional information. The developer can view past inputs' nutritional data and chart.

5.3.6 Security Testing

1. ST-1

   Type: Functional, Dynamic, and Manual.

   Initial State: The user is viewing the nutritional facts for a food item they uploaded.

   Input/Condition: After idling for 3 minutes, the user closes Utrition and reopens it.

   Output/Result: The user will open Utrition and will view the nutritional facts of the image in the past inputs section.

   How test will be performed: A developer on Durum Wheat Semolina will open Utrition and will upload "waffle.jpg" found in the utrition/test/testPhotos directory. The developer views the nutritional facts of a waffle, and then idles for 3 minutes. After the elapsed time, the developer closes their instance of Utrition and opens it again. The developer can view the nutritional facts of the waffle in the past inputs section.

2. ST-2

   Type: Static and Manual.

   Initial State: A Durum Wheat Semolina developer visits Utrition's GitHub page.

   Input/Condition: The developer checks the userdata directory for any files.

   Output/Result: The userdata folder will be empty on GitHub, so Utrition will only contain personalized data once downloaded on a user's device.

   How test will be performed: The developer loads into Utrition's GitHub and goes into the utrition/src/userdata directory. The developer ensures that no userdata is preloaded into Utrition's GitHub.

5.3.7 Cultural Testing

1. CT-1

   Type: Functional, Dynamic, and Manual.

   Initial State: Utrition's main menu is being displayed to a user.

Input/Condition: The user is told by a Durum Wheat Semolina developer to find culturally insensitive material.

Output/Result: The user will be questioned after 5 minutes of searching if they have found any culturally insensitive material while using Utrition.

How test will be performed: Each developer will approach 5 peers with Utrition's main menu open on the developer's device. The developer will give full control over to their peers and will tell their peers to search for culturally insensitive material. After 5 minutes are up, the developer will ask their peers if they have found any culturally insensitive material. This test passes if 95% of users could not find any culturally insensitive material.

5.3.8 Legal Testing

1. LT-1

   Type: Static and Manual.

   Initial State: A developer of Durum Wheat Semolina views Utrition's GitHub repository.

   Input/Condition: The developer inspects all code to ensure Utrition is following Canada's Anti-Spam Legislation Requirements for Installing Computer Programs, Canada's Guide for Publishing Open Source Code, Google's HTML/CSS Style Guide, and Google's guide for material design.

   Output/Result: Utrition is in compliance with all the standards laid out by the SRS document.

   How test will be performed: The developer goes into the utrition/src directory, and then opens the link to one of the following guides:

   - Canada's Anti-Spam Legislation Requirements for Installing Computer Programs
   - Guide for Publishing Open Source Code
   - Google JavaScript Style Guide
   - Google HTML/CSS Style Guide
   - Material Design Guide

The developer will go into each module found in the utrition/src directory, and will inspect the code to ensure that each rule outlined in Canada's or Google's guide is strictly adhered to.

2. LT-2

   Type: Functional, Dynamic, and Manual.

   Initial State: A Durum Wheat Semolina developer has completed a module for Utrition, and has Pylint installed on their device.

   Input/Condition: The developer runs Pylint on their code.

   Output/Result: Pylint will notify the developer if there needs to be changes to the code to ensure Utrition adheres to the PEP8 Python coding conventions.

   How test will be performed: The developer will open their command prompt and travel to the directory with the finished code (utrition/src) by using "cd". The developer runs Pylint on the finished code by typing "pylint finishedmodule.py" in their command prompt. The name "finishedmodule.py" is a placeholder for any future modules that Utrition will implement in the future. The command prompt will notify the developer to make any necessary changes.

5.4 Traceability Between Test Cases and Non-Functional Requirements

Table 3: Traceability between System Tests and Non-Functional Requirements

Requirement #	**Description of Fit Criterion**	**Test ID(s)**
LF1	The distance between user interface components must exceed 20 pixels.	LF-1
UH1	The user will be able to navigate to the main menu from any page in 2 clicks.	UH-1
UH2	The user must be able to view every past inputted food item.	UH-2, UH-3
UH3	90% of a test panel of the target demographic should be able to upload their food and retrieve the corresponding nutritional facts in under 2 minutes.	UH-4
UH4	Every clickable object will have a symbol associated with it. Calories, fat, sodium, carbohydrates, sugar, and protein will have a symbol associated next to it.	UH-5
UH5	All backend calculations, such as identifying the uploaded food and retrieving the nutritional information, will not be displayed to the user.	UH-6
UH6	There will be no sound used by the Utrition web application.	UH-7, UH-8
PR1	The time between the user's click and displaying the new UI will not exceed 2 seconds.	PT-1
PR2	After uploading a picture, Utrition will always display the identified food item within 10 seconds in a test with multiple images.	PT-2

Requirement #	Description of Fit Criterion	Test ID(s)
PR3	After identifying the correct food item, Utrition will always display the nutritional facts within 5 seconds in a test with multiple images.	PT-2, PT-3
PR4	Error message should prompt user upon upload of an abnormal image.	PT-15
PR5	Error message should prompt user upon upload of an image larger than 30MB.	PT-9
PR6	Error message should prompt user upon upload of more than 3 images.	PT-7
PR7	Error message should prompt user upon failure to identify any food items in the uploaded image.	PT-10, PT-11
PR8	Error message should prompt user to try again upon failure to request nutritional info regarding food items in the uploaded image.	PT-12
PR9	Error message should notify user that the nutritional information pertaining to their food item does not exist.	PT-13
PR10	Error message should notify user upon failure to access user's past nutritional info.	PT-14
PR11	Error message should notify user if user's past nutritional info has been deleted.	PT-14
PR12	Error message should prompt user if user's past nutritional info cannot be depicted on a chart.	PT-14
PR13	Request will time out after 20 seconds, and user will be prompted that their device is not connected to the internet.	PT-12

Requirement #	Description of Fit Criterion	Test ID(s)
PR14	In a test with multiple images, Utrition will identify and display the correct food with an average accuracy of 70%.	PT-4, PT-6
PR15	In a test with multiple images, Utrition will display the correct nutritional information for an identified food 90% of the time.	PT-4, PT-5
PR16	Since Utrition is a local web app, the user will always be allowed to use Utrition unless their device or internet connection is down.	PT-4
PR17	The system will not allow more than 3 photos to be uploaded at once.	PT-2, PT-7
PR18	Utrition is a local web app and does not depend on a server.	PT-8
PR19	Utrition will be available to use on GitHub for an indefinite amount of time.	PT-8
OE1	Users will be able to find and pull the repository on to their personal device.	PT-8
OE2	95% of a test panel with the intended demographic will be able to install Utrition after reading the installation guide.	OE-1
OE3	Utrition's GitHub repository will receive no updates after final implementation.	OE-2
MS1	Any backend developer is able to successfully upload new data to Utrition's machine learning model at any time during development.	MS-1

Requirement #	Description of Fit Criterion	Test ID(s)
MS2	All users will be able to view the support manuals on how to install and use Utrition.	OE-1
MS3	100% of the tested Windows, macOS, and Linux devices should be able to use all functionality provided for Utrition.	MS-2
SR1	Anyone will be able to pull the open-source project from GitHub.	PT-8, OE-1
SR2	Utrition will save user data every 3 minutes throughout the application runtime.	ST-1
SR3	Utrition will not contain personalized data from one user on another user's system.	ST-2
CP1	95% of the test panel agrees that Utrition does not provide any culturally insensitive content to the user.	CT-1
LR1	No malicious software will be present in Utrition's GitHub repository.	LT-1
LR2	Durum Wheat Semolina will not commit infractions when following Canada's guide.	LT-1
LR3	Pylint will be used throughout development to ensure every pull request contains properly formatted code.	LT-2
LR4	Before every pull request, code will be reviewed to ensure no violations of the standards occur.	LT-1
LR5	Utrition's user interface will be reviewed to ensure all fonts and colours adhere to the style guide.	LT-1

6 Unit Test Description

6.1 Unit Testing Scope

6.2 Tests for Functional Requirements

6.3 Tests for Nonfunctional Requirements

6.4 Traceability Between Test Cases and Modules

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.

7 Appendix

7.1 Symbolic Parameters

N/A

7.2 Usability Survey Questions

N/A

Appendix — Reflection

Skills

Usage of Pytest framework

Utrition's testing process will include unit tests, integration tests, and functional tests as a part of the verification and validation process. Pytest is a framework that is used for writing simple and scalable test cases for databases, APIs, and user interfaces. This framework can be used for anything from simple unit tests to complex functional tests. To master the use of Pytest, a team member can study the Pytest documentation, reference guides, and refer to the Pytest support community posts when necessary. Following tutorials and how-to guides can also prove to be helpful when learning how to use this framework.

CI/CD with automatic test cases in GitHub

During the development phase, new code should be continuously tested in order to mitigate the introduction of bugs. GitHub Actions is an effective tool that can be leveraged in order to automate testing every time new code is added to the repository. By automating test cases, unit tests can be executed whenever new pull requests are open to ensure new additions have not broken any existing functionality. To implement this, team members can consult online articles on the use of these CI/CD tools, or watch a video walkthrough on how to add automated testing to an existing GitHub project.

Usage of Jest framework

Developers of Durum Wheat Semolina must be able to test the frontend development of Utrition which is made using JavaScript, HTML, and CSS. Jest is a testing framework that can write tests specifically for JavaScript, HTML, and CSS that can be run from the command line The team has decided to use this framework to ensure the correctness and usability of the code for Utrition. This tool will be able to automatically catch errors in the code so that developers do not need to manually go over the code themselves. To master using the Jest framework, team members can watch online tutorials or read Jest supporting documentation.

Knowledge of and performing effective static code inspections

This skill describes the ability to understand the necessary components for a static code inspection and be able to perform one effectively. These inspections are done for all new code attempting to merge into the main branch of the team's GitHub through a PR. When static code inspections are not done properly, sloppy code will enter the main project which can cause errors as implementation progresses. Some approaches for learning what should be included in a code inspection and how to perform said inspection is by reading documentation on what should be done during a code inspection and asking more experienced team members of Utrition for guidance.

Installing and using Pylint

When coding in Python, we do not have a visible compiler to catch syntax errors and bugs prior to running our program as we might have in Java or Haskell. Pylint is required for catching errors like this, as well as for ensuring consistent and syntactically correct code that follows established coding practices. Pylint can be installed through a **pip install pylint** command in the terminal, and can be used through the command **pylint filename.py** to evaluate a file. Pylint can be set up to ignore or only use certain coding rules in its extensive coding practice ruleset, making it customizable to our own specific uses. This skill can be developed by reading the documentation of Pylint and its syntax rules to see which ones are relevant to Utrition. Practicing coding with Pylint can also help acclimatize a team member and help develop good habits for using Pylint before pushing changes.

Knowledge of Google Style Guides for JavaScript, HTML, and CSS

Frontend developers of Durum Wheat Semolina need to understand and implement the Google style guides for frontend technologies in order for all Utrition code to be consistent. By following these style guides, developers do not need to take extra time attempting to decipher unorganized code. Once the developers understand the Google style guides, they will be able to do static code reviews of the written modules. Therefore, they will now be equipped to complete the LT-1 test for the validation and verification of the project. One method for learning the style guides is by approaching the task like a school test. The person learning should take notes on the guide, and to quiz themselves at random intervals. Another way of acquiring the

knowledge of the style guides is to use online video resources to help teach them the material.

Teammate Specific Learning Approaches

Alexander Moica

Alexander would like to write code with greater consistency in its syntax and that follows established coding standards, even those he does not know about. To help him develop these skills and habits, he will incorporate Pylint into his existing projects and create personal coding projects that make use of Pylint from the very start. Alexander will learn to master using Pylint through practice since he learns fastest by making mistakes and remembering them in the future.

Yasmine Jolly

Yasmine has had no previous experience working with the Jest testing framework and requires this knowledge for the testing of Utrition. She will be watching video tutorials on how to use and write effective test cases in Jest. This method of learning was chosen because Yasmine finds it effective to learn by watching examples. With videos, she can pause and rewind steps that she did not fully understand.

Jeffrey Wang

Jeffrey has sparsely worked with automated testing frameworks during his previous work experience, and seen the ways it has helped to prevent the introduction of bugs. As a result, he is interested in leveraging CI/CD tools to automate test cases, and set up mitigations to prevent the introduction of bugs during the development of Utrition. In order to complete this successfully, Jeffrey plans to read online tutorials about the potential ways to automate test cases using GitHub Actions. He has chosen this strategy because the tutorials will walk him through the process step-by-step which he finds most effective for learning.

Jack Theriault

Jack has chosen to develop his knowledge in the Google style guides. Jack has made this choice because he has never used any frontend coding technologies such as JavaScript, HTML, or CSS. Since he is using these languages for the first time, it is best that he develops proper coding style while working in the new languages. To become familiar with the style guides, Jack will read through the guides and quiz himself at regular intervals. He has chosen this learning strategy because he usually learns this way during school and has found it effective.

Catherine Chen

Catherine has had limited experience when writing Python tests using Pytest. She'd like to develop her scalable test writing skills, which include unit tests and functional tests. Her goal is to become familiar with the Pytest framework in order to achieve her goals. Catherine has chosen to read Pytest documentation to learn this skill because of how easily she can access supporting documentation is online.

Justina Srebrnjak

Justina has identified that she struggles with performing static code inspections on other team members' code. More specifically, she is confused about what should be inspected during these reviews (ex. code functionality, potential errors, coding conventions). To master this skill, Justina will seek guidance from fellow teammates who have performed many code inspections in the past (from co-op experiences). To learn most effectively, she will watch one of her teammates perform a static code inspection of another team member's work. She has chosen this strategy because she finds it helpful to watch an example be completed where she can take notes on important steps in the process. With these notes, she will replicate and refer to them when she performs her own reviews.