# Requirements, Vision, and Scope

**Project: Don’t Eat That!**

**Team: Kenton Ma, Charn Rai, Benny Lo, Alex Macdonald, Adil Kydyrbayev**

## Product Vision Statement

### Motivation

Don’t Eat That! is a web application that helps users make healthier diet choices by comparing two foods to each other and recommending the healthier option. Don’t Eat That! will be free to use. Unlike the book *Eat This, Not That!* and its corresponding mobile application, our web application contains a much larger variety of food products and allows users to compare two food products side by side. Additionally, users will be able to make a profile to store any special dietary preferences and needs. Once logged in, these dietary preferences will be used in specialized algorithms to cater to the specific needs of the user. Currently existing food comparison web applications are unable to remember the dietary preferences of regular users and provide a customized recommendation.

### Problem Statement

| | |
|---|---|
| **The problem of** | Choosing the healthiest food from two similar options |
| **Affects** | People concerned about their diet |
| **The impact of which is** | People may not consume the food options that are the most beneficial to their needs |
| **A successful solution would be** | A web application that compares two food choices and recommends the healthiest option for the user |

**Product Position Statement**

| | |
|---|---|
| **For** | Anyone who is concerned about their nutritional intake and/or has special dietary concerns/needs |
| **Who** | Want to choose healthier foods |
| **Our system** | Is a web application (all software) |
| **That** | Compares two given inputs and outputs the healthier option |
| **Unlike** | Current web applications that do not cater to special needs and current books that have a limited variety and do not compare side by side |
| **Our Product** | Allows users to create their own accounts to remember special needs, performs side by side comparisons of a wide variety of foods, and has a friendly, easy to use user interface |

**User Demographics**

*Non-registered Users:*
Non-registered users are people who use the web application “Don’t Eat That!”, without creating a user account and signing in. For non-registered users, the “default” algorithm will be used to determine the healthiest food choice. It is assumed they do not have any adverse medical condition or any particular dietary preferences.

*Registered Users:*
Registered users are people who have created an account. Once a registered user has signed in, the food recommendation algorithm will be specially tailored to their dietary restrictions and preferences.

*Developers:*
Developers will implement, extend, test, and maintain the webpage. We must be careful to write clear code, and document it properly. It is also very important to keep different subsystems/components completely separate, and loosely coupled. The behaviour of one system cannot depend on the behaviour of another, as that could lead to failure in multiple systems if one system fails.

## Feature List

The product will provide the following features:

- Compare two food items against each other and recommend the healthiest choice
- Provide accurate nutritional information about food products
- Keep track of user dietary preferences
- Use user dietary preferences to recommend the best choice for a registered user
- Allow a user to contact developers directly (via email) of any concerns when using our web application

## Constraints

The product is designed to work in all operating systems (Windows, Linux, Mac OS) and popular web browsers such as Google Chrome, Firefox, and Internet Explorer. As Don't Eat That! is a web application, a web hosting service will have to be used, such as GoDaddy. The web application will use a MySQL database, which is provided by the web host. To build the frontend (user interface), we will use HTML, CSS, Bootstrap, PHP, and JavaScript. Additionally, the application will be using a jQuery (version 1.12.4) library to help in writing JavaScript functions. PHP will also be used to connect the front and back ends by making queries to the database. The source of nutritional information will be the FatSecret REST API. The web application will make calls to the API, which will return nutritional statistics. The API is free to use and allows unlimited calls a day, which means usage of Don't Eat That! is unlimited for both general and registered users.

## Scope and Limitations

Features that will NOT be in our product:

- Will not provide recipes
- Will not cure health problems, use at your own discretion
- Will not provide fitness plans
- Will not replace a dietician
- Will not count your daily calories
- Will not replace regular visits to a health professional
- Does not replace exercise and other healthy living habits
- Does not guarantee weight loss

## Assumptions and Dependencies

We expect that HTML, CSS, Bootstrap, JavaScript, and PHP work as outlined in their documentation. The responsiveness of the web application will depend on Bootstrap, JavaScript, and PHP. The application will depend on a jQuery (version 1.12.4) library as well as a form validation and UI jQuery plugin. We will assume that GoDaddy's MySQL database and PHP interpreter included as part of their web hosting package works as advertised. We also assume that the FatSecret REST API and its methods work as advertised on the FatSecret API website.

## Use Cases:

### 1: Compare two food items

| | |
|---|---|
| Primary actor | General user |
| Stakeholders | General user, FatSecret API |
| Preconditions | 1) User is not logged in |
| Main Success Scenario | 1) User enters two food products into the input fields<br>2) User submits choices for comparison<br>3) The system displays two tables with nutritional information - one with normalized serving sizes of 100 g and the other with actual serving sizes.<br>4) The system displays two recommendations - one based on normalized serving sizes of 100 g and one based on actual serving sizes |
| Extensions and Alternative Flows | <u>The system cannot find the queried food item(s):</u><br>1) System notifies users that the choices are unavailable<br>2) System returns user to home page<br><u>The user only fills out one input field:</u><br>1) The home page prompts user to enter a second food choice<br><u>The user does not fill out any input fields but submits choices for comparison:</u><br>1) The home page prompts user to enter two food choices<br><u>The user enters the same food for both food choices:</u><br>1) The home page prompts user to enter two food choices<br><u>The user enters a food choice that is in the API database but is spelled wrong:</u><br>1) If the typo is not too outrageous, the API will still be able to return the food item<br>2) If the typo is too outrageous, the system will not be able to find the queried food items - see <u>The system cannot find the queried food item(s)</u> |

**2: Register Account**

| | |
|---|---|
| Primary actor | General user |
| Stakeholders | General user, database manager |
| Postcondition | User will be able to access their account information |
| Main Success Scenario | 1) The user enters their name<br>2) The user enters their username<br>3) The user enters their password<br>4) The user selects their meal preferences<br>5) The user submits account registration information<br>6) The system redirects the user to the home page |
| Extensions and Alternative Flows | <u>Handle Invalid Name:</u><br>1) If at 1. in 'Register Account', the name is found to be invalid:<br>2) The system displays the message "Name must contain only alphabetical letters."<br>3) The use case restarts at 1. in 'Register Account'<br><u>Handle Invalid Username:</u><br>1) If at 2. in 'Register Account', the username is found to be invalid:<br>2) The system displays the message "Username must be 3-16 characters long and contain only numbers and/or letters."<br>3) The use case restarts at 2. in 'Register Account'<br><u>Handle Invalid Password:</u><br>1) If at 3. in 'Register Account', the password is found to be invalid:<br>2) The system displays the message "Password must be at least 6 characters long and contain only alphanumeric characters."<br>3) The use case restarts at 3. in 'Register Account'<br><u>Handle Empty Input:</u><br>1) If at 1., 2., or 3. in 'Register Account', the name, username, or password is found to be empty upon form submission:<br>2) The system displays at least one of the following messages: |

| | |
|---|---|
| | a) “Please enter your name.”<br>b) “Please enter your username.”<br>c) “Please enter your password.”<br>3) The use case restarts at the first error message (1. 2. Or 3.) in ‘Register Account’<br><u>Handle Pre-existing Username</u><br>1) If at 2. in ‘Register Account’, the username is found to be already taken:<br>2) The system display the message “Username already exists.”<br>3) The use case restarts at 2. in ‘Register Account’ |

**3: Delete Account**:

| | |
|---|---|
| Primary actor | Registered user |
| Stakeholders | Registered user, database manager |
| Precondition | User has logged in with a registered account |
| Postcondition | User will not be able access their account information (the system does not contain that user's information anymore) |
| Main Success Scenario | 1) User initiates account deletion process by activating the "Delete Account" feature<br>2) The system displays a pop-up prompting the user to confirm account deletion process<br>3) The user initiates another account deletion process (confirms deletion via system prompt)<br>4) The system redirects the user to the home page |
| Extensions and Alternative Flows | <u>Handle Cancellation of Deletion Process</u><br>1) If at 3. in 'Delete Account', the deletion prompt is found to be cancelled:<br>2) The use case restarts at 1. in 'Delete Account' |

**4: Edit account information**

| | |
|---|---|
| Primary actor | Registered user |
| Stakeholders | Registered user, database manager |
| Precondition | User has logged in with a registered account |
| Postcondition | The user's meal preferences and/or password will be updated in the database depending on the user's inputs. |
| Main Success Scenario | 1) The user enters their new password<br>2) The user sets new meal preferences<br>3) The user submits updated account information<br>4) The system redirects user to the home page |
| Extensions and Alternative Flows | <u>The user submits updates without making any modifications</u><br>1) The system redirects user to the home page<br><u>The user modifies only their password or only their meal preferences</u><br>1) The system will make changes according to what the user has indicated<br>2) The user will be redirected to the home page<br><u>The user enters new invalid password</u><br>2) The system requests user to re-enter valid password at least 6 characters long containing only alphanumeric characters |

**5: Compare two food items with an emphasis on user-specific dietary preferences**

| | |
|---|---|
| Primary Actor | User with special dietary preferences |
| Stakeholders | User with special dietary preferences, FatSecret API |
| Preconditions | 1) User must have a registered account and entered his dietary preferences into their account information |
| Main Success Scenario | 1) The user submits two food choices<br>2) The system customizes the recommendation algorithm to produce a special recommendation that best fits their meal preferences<br>3) The system displays two tables with nutritional information - one with normalized serving sizes of 100 g and the other with actual serving sizes.<br>4) The system displays two recommendations - one based on normalized serving sizes of 100 g and one based on actual serving sizes |
| Extensions and Alternative Flows | <u>The system cannot find the queried food item(s):</u><br>1) System notifies users that the choices are unavailable<br>2) System returns user to home page<br><u>The user only fills out one input field:</u><br>1) The home page prompts user to enter a second food choice<br><u>The user does not fill out any input fields but submits choices for comparison:</u><br>1) The home page prompts user to enter two food choices<br><u>The user enters the same food for both food choices:</u><br>1) The home page prompts user to enter two food choices<br><u>The user enters a food choice that is in the API database but is spelled wrong:</u><br>1) If the typo is not too outrageous, the API will still be able to return the food item<br>2) If the typo is too outrageous, the system will not be able to find the queried food items - see <u>The system cannot find the queried</u> |

| | |
|---|---|
| | food items(s) |

**6: Submit a message to the developers**

| | |
|---|---|
| Primary Actor | User, developer |
| Stakeholders | User, developer |
| Main Success Scenario | 1) User fills out all the required input boxes, entering their first name, last name, email, and message<br>2) User sends the message<br>3) The system automatically redirects the user back to the home page |
| Extensions and Alternative Flows | User does not fill out every input box<br>1) A prompt is given to the user to fill out the empty input box |

## Non-functional Requirements

Performance Requirements:

The web application should have an almost instantaneous response time. Server requests should not have to wait longer than 1-2 seconds to receive the results of their queries, as that would be too long for a comfortable user experience. Many commonly used web applications have near instantaneous response times, so it would be reasonable of our users to expect a similarly fast response.

- Compare two food items against each other and recommend the healthiest choice: should be almost instantaneous (no longer than 1-2 seconds) for server to respond
- Use user dietary preferences to recommend the best choice for a registered user: should be almost instantaneous (no longer than 1-2 seconds) for server to respond
- Allow a user to contact developers directly (via email) of any concerns when using our web application: should be almost instantaneous (no longer than 1-2 seconds) for the application to send the user's message

Safety Requirements:

This web application could be detrimental to a user's health if the wrong recommendations are made. For example, if a product high in sugar is recommended to a user with diabetes, the user's condition could worsen. To prevent such accidents from

occurring, it is fundamental that custom recommendations are made with an algorithm that correctly uses the user's dietary preferences.

The user's health could also suffer if their registered account information is lost and recommendations are made without their individual concerns. This risk can be minimized by effective database maintenance. Furthermore, user passwords should be encrypted, to prevent malicious parties from obtaining user passwords and altering user preferences.

While this web application is designed with accuracy and customizability in mind, it is not meant to be a replacement for a health professional. To prevent users from getting the wrong idea and using this system in place of a health professional, a disclaimer will be placed on all pages.

Security Requirements:

Users can make an account which will keep track of their meal preferences. As their accounts can contain private information, the application must prevent unauthorized access to accounts. Thus, every account will require a password to access. The web application must also ensure that passwords are encrypted via the bcrypt hashing function upon account creation or when a user updates their account password. Furthermore, the application should ensure that queries made from user input are handled properly to prevent SQL injection.

It is important that when users send personal data to the web application, their data is received only by the application. To secure user data, our web application will use Secure Sockets Layer (SSL) security technology, provided by GoDaddy. SSL is used to establish an encrypted link between the web client (user) and the web server (host). A SSL certificate will be used to establish a secure connection. The SSL certificate has a key pair consisting of a public 2048 bit key and a private 2048 bit key, which are used to encrypt/decrypt messages.

The SSL certificate is also used to convince users that they are sending data to the correct server. Web browsers give visual symbols, such as a green lock or green address bar, to show the user that the connection is secured. If a user uses the web application and sees the SSL trusted certificate, it means the connection between the user and the server is secured, giving the user a peace of mind.

Software Quality Attributes:

As this web application is likely to be used by users with health concerns, it is imperative that the application is:

- Correct and reliable: the application should always recommend the healthiest choice for the user so the user does not eat something that could worsen their condition. Careful consideration must be taken in designing the algorithm which recommends the healthier food choice.
- Usable: The application should be easy to use so users are not confused and misinterpret results. The user should intuitively be able to use and understand the website without having to read a set of instructions.
- Maintainable and flexible: The structure of the source code directory should follow certain rules, with certain types of files being placed in certain folders. Php scripts are placed in a “php” folder, php classes are placed in a “classes” folder, javascript is placed in a “js” folder, etc. The quality of the code is also important. Repeated code is to be avoided, and code must be correctly formatted. Indentations and spacing should be consistent throughout all code to make it more maintainable. The logic implemented by the code, especially the php scripts and classes should be clear to anyone reading it (even without comments).
- Portability: the application should display properly on all browsers/platforms. The application has been tested on a Samsung Galaxy S4 (Android), Iphone 6 (iOS), an Ipad, a Samsung Galaxy Tab A tablet, and on various laptops (Linux, Mac OS X, Windows). It has also been tested on Safari, Chrome, and Firefox. In all these cases, the user interface displayed properly. Furthermore, the web application functioned as designed and displays results properly.

*Note: Old backlog descriptions are in red, and backlog changes are blue

## Product Backlog*:

| Priority | Item # | Description | Estimate (Hours) | Estimated By | Remaining Hours By Sprint Start | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | 1 (Starts Sept 26) | 2 (Starts Oct 10) | 3 (Starts Oct 24) | 4 (Starts Nov 7) | 5 (Starts Nov 21) |
| Very High | 1 | Product Vision | 5 | KM | 5 | 0 | 0 | 0 | 0 |
| | 2 | Set up a website, buy a domain name and server | 2 | AM | 2 | 0 | 0 | 0 | 0 |
| | 3 | UI - Display two boxes on the website, allow user to enter two items, then display a result | 30 | AK | 30 | 20 | 5 | 0 | 0 |
| | 4 | Food DB Design - Set up database for food items, with accurate nutritional facts FatSecret API Integration | 30 | CR | 30 | 20 | 5 | 0 | 0 |
| | 5 | Query design - Food database has to be able to process queries Query Design for FatSecret API | 30 | KM | 30 | 20 | 5 | 0 | 0 |
| | 6 | Algorithm Design - Create algorithm to calculate healthiest food choice | 15 | BL | 15 | 15 | 10 | 0 | 0 |
| High | 7 | User DB Design - Set up database for user accounts to store ~~search history and~~ nutritional preferences | 20 | CR | 20 | 15 | 5 | 0 | 0 |
| | 8 | Query design - User database has to be able to handle new | 20 | AM | 20 | 20 | 10 | 0 | 0 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | accounts, log ins, and log outs | | | | | | | |
| | 9 | Security - Protect from SQL injection and other security threats (ask Farshid or Sathish about possible threats) | 15 | AM | 15 | 15 | 15 | 10 | 0 |
| Medium | 10 | UI - Display list of similar, healthier food items after a single item is entered Did not do | 20 | KM | 20 | 20 | 20 | 10 | 0 |
| | 11 | User interface - Add drop down menus, loading icon with Javascript | 10 | AM | 10 | 10 | 8 | 0 | 0 |
| Low | 12 | DB Design - Set up database for restaurants and entrees Did not do | 20 | BL | 20 | 20 | 20 | 15 | 10 |
| | 14 | User may log in using other social media accounts Did not do | 20 | AK | 20 | 20 | 20 | 20 | 20 |
| | 15 | DB Design - Allow users to enter new food choices (how do we check if it's valid?) Can send emails | 10 | BL | 10 | 10 | 10 | 10 | 10 |
| | | | | TOTALS | 247 | 205 | 133 | 65 | 40 |

## Product Backlog Changes:

- Item #4 (FatSecret API Integration): We chose the FatSecret API over designing our own database containing nutritional information of foods because the API accomplishes the same goal and it requires less database management/maintenance. Additionally, delegating queries to the API will reduce the load on our website because we are not querying the web hosted databases provided by GoDaddy.

- Item #5 (Query Design for FatSecret API): Changing item #4, requires changing item #5.
- Item #7 (User DB Design): The user's food search history is not needed in determining the healthier food choice.
- Item #10,12,14: These items are not implemented because it could clutter the UI, making it less user-friendly. We are aiming for simplicity.
- Item #15 (Can send emails): The user is able contact the developers with questions, comments or concerns they have with using the web application. Feedback from the emails can help improve further iterations of the product.