Test Plan Document

Group 14: Don't Eat That!

Group Members: Kenton Ma, Benny Lo, Charn Rai, Alex Macdonald, Adil Kydyrbayev

Introduction

Don't Eat That! is a web application that helps users choose healthier foods by comparing two food items of a genre to each other and suggesting the healthier one.

The web application has both frontend and backend components. The frontend is implemented using HTML, PHP, CSS, the Bootstrap framework, JavaScript and jQuery. The backend is built using PHP, the FatSecret REST API, and a MySQL database.

When a user enters two food items into the web application, a PHP script will run and query the nutritional data of each food item from the FatSecret API. The data of each food item is entered into an algorithm, which gives both foods a score based on their nutritional information. There will be two different algorithms used to determine the healthier food item. One algorithm is used for general users, who are not registered in our system. The other algorithm is specially catered to a registered user's dietary preferences (outlined in their profile). After determining the healthier food choice (choice with the higher score), the web application will direct the user to a page that will display both food items along with their corresponding nutritional data. The healthier food choice will be graphically highlighted to the user.

To become a registered user, users can make an account that will keep track of their dietary preferences. When a user logs in, the application will start a PHP session that adjusts variables used in the recommendation algorithm. This will allow the algorithm to be customized to every registered user.

Verification Strategy

To ensure the software meets the user's real needs, we have weekly in-person meetings. During these in-person meetings with the client, notes are taken as we show the GUI mock-ups, requirements documents, and demonstrate the application. Additionally, we have created a "Contact" page where users can send us a direct message if they have questions and/or concerns about our service. All messages are immediately sent to a designated email address, which will be checked regularly - ensuring the user receives a quick response.

Non-Functional Testing

Response Time:
To ensure that the website has an almost instantaneous response time, we will use an online tool called dotcom-monitor, tests the responsiveness of the web application from

different locations. Developers will also use the application on different computers, browsers, and internet networks to measure the responsiveness from the perspective of a user.

Account Storage Capacity:
It is important to know how many accounts the web application can store. To know how many accounts can be stored, developers will make an account, determine how much memory is used, and extrapolate the amount.

Safety:
To make sure our user receives the correct recommendation, we will thoroughly test our “recommendation algorithm”. If a user has a profile, the user should get a result that complies with their dietary preference(s). For example, a user’s profile indicates a preference for low sugar, then the specialized algorithm should avoid recommending foods with a high sugar content.

Security:
To make sure our user’s information are secure when they create a user profile, we will use the HTTPS protocol. To test this, we will use Wireshark to attempt to receive sensitive information. In a successful test case scenario, we should not be able to acquire the user’s sensitive information such as their dietary preferences.

Correct and reliable:
To make sure our user receives the correct recommendation, we will need test cases. The test cases will consist of an input (food item 1 and food item 2) and an expected output (food recommendation).

Usable:
To test the usability of our web application, we will measure how long it takes for a user to figure out how to query their food items. At most it should take a user 20 seconds.

Maintainable and flexible:
To test the maintainability and flexibility of our web application, we will measure how long it takes to make a build as well as the readability of our implementations. The implementations should be formatted properly.

Interoperability:
To test functionality across different browsers, we will apply our use cases to each browser and see if the same output is given.

Portability:
To test for portability, we will apply our use cases on mobile phone browsers.

Results of Testing Non-Functional Requirements

Test	Requirement Tested	Input	Expected Result	Result	Pass/Fail	Notes
1	Application response time is almost instantaneous	Apple and orange were compared on Google Chrome	~1 second response time	1.4 second response time	Pass	Tested through https://www.dotcom-tools.com/website-speed- test.aspx
2	Account storage	1 account made	Newly created account should show up on phpMyAdmin view	New account took 0.014% of server memory	Pass	
3	Safety, correct, and reliable	Apple and orange were compared on Google Chrome	Apple score: -0.158 Orange score: 0.782	Apple score: -0.158 Orange score: 0.782	Pass	100 g Results page was temporarily changed to put scores produced by the algorithm
4	Usability	3 users were shown our application with no instructions	All users can use the application within 20 seconds of exposure	All users could use the application with 10 seconds	Pass	Test sample consisted of an engineering student, an arts student, and a commerce student.
5	Maintainable and Flexible	A Shell script was made to upload files to the server Code was shown to client at weekly meetings	Script can make a build in a few minutes Client would understand code	Script could upload files and make a build in ~2 minutes Client understood the code	Pass	The code was refactored multiple times to be efficient and understandable.
6	Interoperability	Application was tested on Google Chrome, Mozilla Firefox, Microsoft Edge, Opera, and	Applications should remain consistent on all browsers	Applications remained consistent on all browsers	Pass	Application produced identical results at similar speeds on all browsers.

		Safari				
7	Portability	Application was tested on an Android phone on browsers: Chrome Application was tested on an iPhone on browsers: Safari	Applications should remain consistent on all browsers	Applications remained consistent on all browsers	Pass	Application produced identical results at similar speeds on all browsers.

Functional Testing Strategy

Our testing strategy is mostly based on the acceptance test, after implementing each use case we check if the requirements were met. Once the output is showing correctly and meet the specifications, then we move forward to the next sprint. Tests are implemented weekly because we make modifications on the existing codes. In order to track the system bugs, we use the GitHub. If any bugs are identified, we create an issue in the GitHub and fix them, by solving the issues.

Adequacy Criterion

- Make sure that at least one test exists for each method written:
    - The reasoning behind this is to make sure our scripts are producing the correct result (e.g. correct food recommendation)
- Make sure that at least one test exists for responsiveness on both desktop and mobile:
    - The reasoning behind this is to make sure the web application is user friendly
- Make sure that at least one test exists for each requirement and for each use case:
    - The reasoning behind this is to make sure that everything functions as the user expected it to.

Results of Testing Web Page Setup

Test	Requirement Purpose	Input	Expected Result	Result	Pass/Fail	Notes
1	To create the background of the web application with non-functional buttons	PHP - design the content of the web page, CSS - modification of the interface, Bootstrap - modification of the web page elements (e.g. buttons)	To get the user-friendly interface with the proper placement of the elements (header, body, and footer)	The expected result is achieved	Pass	The frontend is well implemented
2	To create the appropriate action to be performed for each button	Created Index, About, How It Works, Contact, Login, Sign Up, Compare, and Clear Choices button	When the button is pressed, to implement the appropriate functions assigned to each button .	The results are the same as what is expected	Pass	All the source codes is in the public_html folder
3	To create PHP scripts to each Log in button, so the user can enter to their accounts	On top of the PHP document to create a script that prompts the user to enter a username and password	In case of properly provided data, to give an access to the personal user profile, otherwise, to show a message that either of the data was entered wrong	User is able to access their account upon successful log in	Pass	All the source codes is in the public_html folder

Results of User Database Testing

Test	Requirement/Purpose	Input	Expected Result	Result	Pass / Fail	Notes
1	User database has to be able to handle new accounts	Make a new account on the application and use phpMyAdmin to see if account has been created with correct attributes	phpMyAdmin should show the created account with the correct attributes	The actual result is the same as expected	Pass	The source code may be found in the php folder (makeAccount.php)
2	User can delete the created account	Log into a previously created account and delete it using the Manage Account page	phpMyAdmin should show that the deleted account is no longer in the database	The actual result is the same as the expected result	Pass	The source code may be found in the php folder (deleteAccount.php)
3	User is able to edit account information	Log into a previously created account and edit dietary preferences using the Manage Account page	phpMyAdmin should show that the modified account's attributes in the database reflect the changes	The actual result is the same as the expected result	Pass	The source code may be found in the php folder (accountUpdate.php)

Results of FatSecret REST API Testing

Test	Requirement	Action / Input	Expected Result	Actual Result	P/F	Notes
1	To access the API and output nutritional facts for two food items requested by the user	To access the database; Using the search query find the corresponding data (i.e. calories, sugar, protein) of two food items; Return the result on the php page;	To output two lines of data containing the food names and nutritional data;	When users type the food items they want to compare, the scroll box with the matching items will appear.	P	The code source may be found in the docs folder (foodquery.php)
2	To make multiple API requests at the same time	Have all developers use the web application simultaneously	The web application should work as normal for all developers	The web application worked as normal for all developers as expected	P	We had to contact FatSecret to request removal of throttling limits
3	To ensure the API returns the first item with valid serving units	Query the input “tart” and print all results onto the web page	The printed results should show multiple results, some with invalid serving units such as “1 tart”. The application should display the data of the first result with valid serving units.	The results were as expected.	P	The results page was temporarily changed to print all returned food items from FatSecret.