Test Plan Document

Group 14: Don't Eat That!

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

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 front-end and back-end components. The front-end is implemented using PHP, CSS, Bootstrap, JavaScript and jQuery. On the other hand, the back-end is implemented using PHP and a MySQL database.

When the user enters two food items into the web application, a PHP script will run and query the data (i.e. calories, sugar, protein) of each food item. The data of each food item is entered into an algorithm. There will be two different algorithms used to determine the healthier food item. One algorithm is used for generic purposes i.e. the user is not registered in our system. The other algorithm is specially catered to the user's dietary restrictions outlined in their profile. After determining the healthier food choice, 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.

Verification Strategy

To make sure 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, or the software itself. Additionally, we have created a "Contact" page where users can send us a direct message if they have questions regarding our service. For example, if no entries from the database match the user's food choice(s), he/she may send a message requesting to add that particular food item into the database. All messages are sent to our email address, which will be checked regularly, so the user will receive a quick response.

Non-Functional Testing and Results

Response time:
To make sure that the website responds in less than 5 seconds, we will use an online tool called *dotcom-monitor*, which allows us to test the responsiveness of the web application from different locations. To make sure that our web application can support up to 100 users, we will need to analyze how much data is used in creating a user profile. After identifying the data size for one user profile, we can extrapolate the minimum size of the database. For example, each user profile will use up 100 kilobytes, so if we have 100 users, then we will need a database size of 10 megabytes.

Safety:
To make sure our user receives the correct recommendation, we will have to test our "recommendation algorithm". If a user has a profile, the user should get a result that complies with their dietary restriction(s). For example, a user has created a profile and the profile indicates that they prefer a low sugar diet. If the user were to input a food item high in sugar and a food item with relatively lower sugar, then the system should recommend the food item lower in sugar.

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 (e.g. first/last name, age, height etc.).

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.

Test #	Requirement Purpose	Action / Input	Expected Result	Actual Result	P/F	Notes
1.	Web application response time is less than 5 seconds	Browser: Google Chrome donteatthat.ca	Response time < 5 seconds	1.4s	P	Tested through https://www.dotcom-tools.com/website-speed-test.aspx

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.

Test Cases and Results

Web page setup test:

Test #	Requirement Purpose	Action / Input	Expected Result	Actual Result	P/F	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	P	In terms of the page design, everything is set properly
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	P	All the source codes may be found in the docs 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	For now, we only implemented a login and password fields. When the Login button is pressed, the pop-up message will show up, but user is not able to enter to the personal profile	F	The implementation of the given test case is still in progress.

Food Database and Food Database Query test:

Test #	Requirement Purpose	Action / Input	Expected Result	Actual Result	P/F	Notes
1.	To store a table with a food nutritional facts into the database	Converted excel document to .csv (comma-separated values) file and further converted into .sql file.	To have a fully uploaded table inside the database	The table was stored in the database, being separated into several portions and uploading each one independently	P	After uploading all the separate tables, everything was inserted into one table
2.	To check the connection between the web page and database by using the PHP script	Wrote the PHP script to display the connection status	If the connection is successful, PHP notifies that the web page is connected to the database, otherwise, outputs the error statement.	Connection was set successfully	P	The code may be found in the docs folder (foodquery.php)
3.	To access the database 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)

Query design - User Database Set up test:

Test #	Requirement Purpose	Action / Input	Expected Result	Actual Result	P/F	Notes
1.	User database has to be able to handle new accounts	To create the PHP script that stores the Username and Password entered by the user; To insert the stored data into user database	To observe the entered data in the user database;	The actual result is the same as expected	P	The source code may be found in the docs folder (makeaccount.php)
2.	User can delete the created account	To create a PHP script that allows users to enter to their accounts and delete it, so all the records done by the user will be destroyed	The user database query finds all the data related to the deleted user account and deletes the whole row	The given implementation still in progress The actual result will be the same as the expected result	F	Implementation is progress
3.	User is able to edit account information	To create a PHP script that allows users to enter their accounts and edit it, so all the records done by the user will be changed	The user database query finds all the data related to the selected user account and modifies the values that were changed by the user	The given implementation still in progress The actual result will be the same as the expected result	F	Implementation is progress