Testing and Validation

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 (entered by the user) and recommending the healthier option. The application has 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 query the nutritional data of each food item from the FatSecret API. The data is entered into an algorithm, which gives both foods a score. There will be two different algorithms used to determine the healthier food item. The default algorithm is used for general users. A customized algorithm is available to registered users; it 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 displays both food items' nutritional data and highlights the recommendation 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 application's design and features meets the user's desires and needs, we have weekly in-person meetings. During these in-person meetings with the client, notes are taken as we demonstrate prototypes, requirements, and parts of the application that have been built. These notes are carefully reviewed and suggestions are implemented in the following sprints.

Additionally, we have created a "Contact" page where users can send the developers a direct message if they have questions and/or concerns about the application. All messages are immediately sent to a designated email address, which will be checked regularly - ensuring the user receives a quick response.

Testing Non-Functional Requirements

<u>Response Time:</u>
To ensure that the website has an almost instantaneous response time, we will use an online tool called dotcom-monitor, which 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. As this is a free web application, we can expect many users. The application must be able to store all of their accounts. To know how many accounts can be stored, developers will make an account, determine how much memory is used, and extrapolate the number of accounts that the host server can hold.

Safety:
To make sure our user receives the correct recommendation, we will thoroughly test our recommendation algorithm with multiple test cases. If a user has a profile, the user should get a result that complies with their dietary preference(s). For example, if a user has a preference for low sugar, then the application should not recommend 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 steal 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 to use multiple test cases. The test cases will consist of two inputs (food item 1 and food item 2) and an expected output (recommended food). The recommended food should be the healthier of the two inputs, which is the input with the higher score from the algorithm.

Usable:
To test the usability of our web application, we will measure how long it takes for a user to figure out how to compare their food items upon initially using the application. At most, it should take a user 20 seconds. A longer time suggests that the application is not user friendly and must be changed to increase the learnability.

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 and follow good coding standards.

Interoperability:
To test functionality across different browsers, we will apply our use cases to multiple browsers and check that the same output is given on all browsers. The functionality of the application should be the same on all browsers.

Portability:
To test for portability, we will apply our use cases on mobile phone browsers. This means the developers will play the role of users and use the application extensively on multiple phones and mobile 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	As one account took up 0.014% of server memory, the server can hold 7143 accounts, which is much more than the expected number of users
3	Correct and reliable - algorithm should produce the correct scores	Apple and orange were compared on Google Chrome	Apple score: -0.158 Orange score: 0.782 Orange should be recommended	Apple score: -0.158 Orange score: 0.782 Orange was recommended	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 - code should be easy to build and understand	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,	Applications should remain consistent on all browsers	Applications remained consistent on all browsers	Pass	Application produced identical results at similar speeds on all browsers.

		Opera, and Safari				
7	Portability	Application was tested on an Android phone on browsers: Chrome, Firefox Application was tested on an iPhone on browsers: Safari, Chrome, Firefox	Applications should remain consistent on all browsers	Applications remained consistent on all browsers	Pass	Application produced identical results at similar speeds on all browsers.
8	Safety	Usually, apples are recommended over chocolate A registered account has a preference for high sugar and will be used to compare apples to chocolate	Application should recommend chocolate over apples	Application recommended chocolate over apples	Pass	

Testing Functional Requirements

Our testing strategy for functional requirements is based on acceptance, unit, and PHP testing. After implementing a feature, we check if requirements are met. Once the tested outputs are correct and meets the specifications, we move onto the next sprint. Tests are implemented weekly to allow early detection of bugs. Bugs that are found early are easier to fix (code is still fresh in the developers' minds) and allow for more accurate scheduling (it is hard to predict how long debugging takes). Fixing bugs before writing new code also prevents bugs from affecting new features. In order to effectively track the bugs in the application, we use GitHub. If any bugs are identified, we create an issue in GitHub. Once a bug is fixed, the issue is solved - providing a detailed log of developmental obstacles. The system is also tested by randomly selected users (such as classmates) who can provide feedback on both the user interface (frontend) and performance (backend).

Adequacy Criterion

- Ensure at least one test exists for each method written:
    - Application must produce the correct results, such as:
        - Correct nutritional facts are being displayed
        - Correct recommendation is being made
- Ensure at least one test exists for responsiveness on both desktop and mobile:
    - The web application must be user friendly - safe, efficient, and learnable
- Ensure at least one test exists for each requirement and for each use case:
    - Everything should function as specified in the use cases, which dictate how this application will be used

Results of Testing Web Page Setup

Test	Requirement / Purpose	Input	Expected Result	Actual Result	Pass / Fail	Notes
1	To create the background of the web application with 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 have a user-friendly interface with the proper placement of the elements (header, body, and footer) and functional buttons	The actual result is the same as the expected result	Pass	The frontend is well implemented, with a visually appealing design and fully functional buttons
2	To create the appropriate function(s) for each button	Created “Index”, “About”, “How It Works”, “Contact”, “Login”, “Signup”, “Compare”, “Change Food Items”, and “Clear Choices” buttons	When a button is pressed, the appropriate function(s) should be performed .	The actual result is the same as the expected result	Pass	All the source code is in the public_html folder
3	To create a PHP script for the login button so the user can log into their account	Build a PHP script that allows a user to log into their account	User should be able to access their account if and only if the correct login information is entered	User is able to access their account upon successful log in	Pass	All the source code is in the public_html folder

Results of User Database Testing

Test	Requirement / Purpose	Input	Expected Result	Actual 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 / Purpose	Action / Input	Expected Result	Actual Result	Pass / Fail	Notes
1	To access the API and output nutritional facts for two food items requested by the user	To query the FatSecret API Find the nutritional data of two food items Return the result on the php page	To output two lines of data containing the food names and nutritional data;	The correct food items and nutritional information is returned by the API	Pass	Involved the following code files: results.php results-serving.php FoodFinder.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	Pass	We had to contact FatSecret via email 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 actual results were the same as the expected results.	Pass	The results page was temporarily changed to print all returned food items from the FatSecret API

Results of User Acceptance Testing

User	User Feedback	Effect of Feedback on Development
Omar	● Confused if supposed to enter food items, meals, or recipes ● Serving size table is more useful, put on top ● Interface is clean ● If food is not found, message should be more clear ● Compare button should be larger than Clear Choices button	● Results page only displays one table at a time; user can choose which table to display ● Made Compare button larger than Clear Choices button
Mihailo	● Not clear what the site does ● On mobile, it is not obvious you can scroll horizontally ● Include specific details why one food is healthier	● Instructions on main page were replaced with catchier, more informative instructions
Raj	● Website is simple ● Add a “Return to main menu” button, it is unclear the logo does this	● Added a “Change Food Items” button on the results page to redirect users to the main page, where they can enter new inputs
Michael	● Have a toggle switch for the two tables ● Unclear what the application does ● Make the recommended choice more obvious	● Results page only displays one table at a time; user can choose which table to display ● Instructions on main page were replaced with catchier, more informative instructions ● Recommended food product is highlighted on tables
Chen	● How It Works page has too much writing ● Results page is confusing ● Wanted autocomplete feature ● Liked countdown page (page after an input cannot be found)	● Shortened links on How It Works page ● Put tables on separate pages ● Implemented autocomplete feature
Sirine	● Explain to the user how the application works ● User interface is clear ● “Cucumber without peel” returns “Potato without peel” ● Allow the user to choose between 100 g portions or typical serving sizes ● Add explanation to explain the recommendation	● Results page only displays one table at a time; user can choose which table to display ● Implemented autocomplete feature to help users enter the correct input
Jasper	● Home page seems empty ● Queried items were different from input	● Implemented autocomplete feature to help users enter the correct input ● Redesigned home page to appear fuller
Aaron	● Redirection after invalid queries is too long	● Changed redirection time to 3 seconds

	● Required fields should have stars ● Instructions are unclear ● Explain how recommendations are made ● Display more nutritional data ● What if user enters invalid inputs? ● Display table differently ● Implement sign up verification	● Instructions on main page were replaced with catchier, more informative instructions ● If a required field is left blank, the system will prompt the user to fill it in

PHP Testing

There are three PHP unit tests. Each unit test tests the core functionality of a PHP class:

- FoodFinderTest.php
  - Tests if the FoodFinder API wrapper class is returning the right data
- FoodComparerTest.php
  - Compares two foods, and tests to see if the results is correct (the tester supplies the food and nutritional information)
- AutoCompleterTest.php
  - Tests if the AutoCompleter API wrapper class gives meaningful autocomplete suggestions

These tests can be run with the 'runUnitTests.sh' script. All these tests make use of the Atoum PHP testing framework.

In addition, there are three PHP database and login/logout tests:

- MakeAccountTest.php
  - Creates a fake account and checks the user database to verify the account exists
- LoginLogoutTest.php
  - Creates a fake user, logs in, and checks the '_SESSION' variable to see if the login script worked
  - Calls the logout script (after logging in), and checks the '_SESSION' variable to make sure it contains no information about the logged out user
- DeleteAccountTest.php
  - Deletes the account created when running MakeAccountTest.php and ensures the deleted account is not in the user database

These tests do not use any testing framework and are simple PHP scripts. To run these tests, use the script 'runDatabaseTests.sh'. These tests will only work if the tester has a properly setup config.php file, and a local instance of a MySQL database running. The MySQL database must be properly set up (see README.md).