Test Plan

NaBrO (Sodium Hypobromite)
Sarah George
Brandon Jones
Rajdeep Bandopadhyay
Noah Trenaman
Manvith Krishna Kandukuri

Part I. Description of Overall Test Plan

We will test each component of our system individually so that their various functions are shown to be reliable under many kinds of inputs. In addition to functionality-based tests, we will run some tests to make sure our system can handle a high load of traffic. Finally, we will have tests which integrate the various components of the system so that end users can enjoy all the features without errors. Relevant components are the Data Server, Logic Engine, End User App, User Interface

Part II. Test Case Descriptions

DS1.1	**Data Server Test 1** (`dataServerStatus`)
DS1.2	To ensure that the data server is currently accepting requests on a public URL.
DS1.3	Send HTTPS request to https://DATASERVER/status and parse response
DS1.4	No inputs, just sending a GET request from an internet-connected device
DS1.5	Output is a status indicator from database as a REST response
DS1.6	Normal
DS1.7	Blackbox
DS1.8	Functional
DS1.9	Unit test

DS2.1	**Data Server Test 2** `(dataServerLanguageSearch)`
DS2.2	To ensure that the data server can return relevant search results for a single text description. Whenever the database is modified we can re-run this test to guarantee data integrity.
DS2.3	Send HTTPS request to https://DATASERVER/search and parse response
DS2.4	For each test, send a string describing a type of food or meal to the database. No other search parameters beyond the name. Run the test for a list of food names to test coverage.
DS2.5	Output should be a ranked listed of results in which the top 5 results must include a predetermined food object that is relevant to the query.
DS2.6	Normal
DS2.7	Blackbox
DS2.8	Functional
DS2.9	Unit test

DS3.1	**Data Server Test 3** `(dataServerNutrientCombiner)`
DS3.2	To ensure that the data server can return the combined full-spectrum nutrient profiles for a list of food objects and their relative mass. This will confirm data integrity and proper additive logic.
DS3.3	Send HTTPS request to https://DATASERVER/nutrients and parse response
DS3.4	For each test, send a JSON object representing a collection of foods or meals and their relative mass.
DS3.5	Output should be a nutrient object which has macronutrients and micronutrients including vitamins and minerals, in which certain values pass specific predetermined thresholds.
DS3.6	Normal
DS3.7	Blackbox
DS3.8	Functional
DS3.9	Unit test

DS4.1	**Data Server Test 4** (`dataServerFoodRecommender`)
DS4.2	To ensure that the data server has a reliable understanding of the benefit of particular foods over other foods, given a user’s nutrient goals and targets. This will be used to benchmark our ability to make contextual recommendations.
DS4.3	Send HTTPS request to https://DATASERVER/recommend and parse response
DS4.4	For each test, send a JSON object representing a very simple nutrient goal such as to maximize protein or vitamin C.
DS4.5	Output should be a sorted list of foods in which we can check that the first foods are more dense in the desired nutrient than the following foods.
DS4.6	Normal
DS4.7	Blackbox
DS4.8	Functional
DS4.9	Unit test

DS5.1	**Data Server Test 5** (`dataServerLanguageSearchRandom`)
DS5.2	To ensure that the data server avoids sending any relevant results when the search query is about strictly non-food items.
DS5.3	Send HTTPS request to https://DATASERVER/search and parse response
DS5.4	For each test, send a string describing some everyday object or concept that is not related to any items in the database.
DS5.5	Output should be an empty list.
DS5.6	Normal
DS5.7	Blackbox
DS5.8	Functional
DS5.9	Unit test

LE1.1	**Logic Engine Test 1** `(checkRDAandRDI)`
LE1.2	To guarantee that every nutrient in our logic engine has an associated RDA (recommended dietary allowances) or RDI (reference daily intake.) We can iterate through a spec file for each nutrient and make sure we have the proper records for each.
LE1.3	Load the nutrient spec file and iterate through its fields. File is likely to be JSON or YAML.
LE1.4	No input
LE1.5	Output should be a hashmap of booleans in which we can see which nutrients have the requires values, and which do not
LE1.6	Normal
LE1.7	Whitebox
LE1.8	Functional
LE1.9	Unit test

LE2.1	**Logic Engine Test 2** `(checkSerializers)`
LE2.2	To guarantee that every object defined in our ORM (object-relational mapping) can be both serialized and deserialized correctly and without error. This is essential to communicating over a REST API and supporting different runtime environments (Python, JavaScript, Flutter.)
LE2.3	For each object defined such as Nutrient, Food, Meal, have a couple JSON examples of each as if they came from the API. Use the serializers to decode and then re-encode the file, and make sure it is identical.
LE2.4	Input is a JSON representing a given object
LE2.5	Output should be a transformed JSON that represents the same object and is equivalent
LE2.6	Normal
LE2.7	Whitebox
LE2.8	Functional
LE2.9	Unit test

WA1.1	**Web App Test 1** `(webAppAPIEndpoints)`
WA1.2	To ensure the wep app is correctly able to correctly perform all designed CRUD operations through the API and the changes are correctly reflected in the database
WA1.3	Send requests to the server for all possible endpoints including variations of parameters and ensure correct information is received
WA1.4	A list of all API endpoints and parameters
WA1.5	All operations send/receive the correct information
WA1.6	Normal
WA1.7	Whitebox
WA1.8	Functional
WA1.9	Integration test

WA2.1	**Web App Test 2** `(webAppResponsiveDesign)`
WA2.2	To ensure that the web app ui design implementation is responsive and visually appealing on all sizes of devices/screens
WA2.3	This can be done by simply resizing the window and seeing how different elements of the UI react to the changing of sizes (menus collapsing into expandable icons, grids reducing in the amount of columns, no overlapping/ overflowing elements, etc). Chrome dev tools has a variety of preset dimensions for common devices as well to test.
WA2.4	No inputs
WA2.5	UI correctly adapts to the size of the screen to make the app appealing and useable on all devices
WA2.6	Normal
WA2.7	Blackbox
WA2.8	Functional
WA2.9	Unit

WA3.1	**Web App Test 3** `(webAppBrowserSupport)`
WA3.2	To ensure that all elements of the web app are functioning and appearing consistently across a variety of popular web browsers since a lot of browsers handle different factors of rendering slightly differently and can break on a specific browser and work correctly on the rest
WA3.3	Access the web app pn a few of the top browsers and use the application like normal and access each page of the website to make sure it looks consistent and correct on all browsers
WA3.4	No inputs
WA3.5	UI will be working and consistent across all browsers
WA3.6	Boundary
WA3.7	Blackbox
WA3.8	Functional
WA3.9	Unit

Part III. Test Case Matrix

Test Case	Normal / Boundary	Whitebox/ Blackbox	Functional/ Performance Test	Unit/ Integration Test
DS 1	Normal	Blackbox	Functional	Unit
DS 2	Normal	Blackbox	Functional	Unit
DS 3	Normal	Blackbox	Functional	Unit
DS 4	Normal	Blackbox	Functional	Unit
DS 5	Normal	Blackbox	Functional	Unit
LE 1	Normal	Whitebox	Functional	Unit
LE 2	Normal	Whitebox	Functional	Unit
WA 1	Normal	Whitebox	Functional	Integration
WA 2	Normal	Blackbox	Functional	Unit
WA 3	Boundary	Blackbox	Functional	Unit