Quisper Documentation (Providers)

https://quisper.eu/connecting-to-quisper/providers/documentation/

For users to be able to implement a provider, they need sufficient information about how the service works. Thus, a provider:

1. MUST describe the purpose of the API in a non-technical summary, a non-technical description (targeted at potential users, e.g. healthcare providers) and a technical description. These descriptions SHOULD describe possible uses or use cases for the API.
2. SHOULD provide an image or logo pertaining to the API
3. MUST describe the API with Swagger version 2.0 or OpenAPI version 3.0 (see https://swagger.io/specification)
4. MUST document all API calls, parameters and possible responses
5. SHOULD include descriptions and possible values
6. SHOULD document the structure of the requests and responses
7. MUST provide fully functional example requests or an example client on what to call the API
8. SHOULD provide documents or URLs of publicly available documents describing the possible contents of a request and response, e.g. a reference to an ontology or thesaurus

TABLE OF CONTENTS

TABLE OF FIGURES

1. eNutri API description

Non-technical summary

The eNutri API receive a request containing dietary intake collected from a Food Frequency Questionnaire (FFQ) and returns nutrients, portions, diet quality score (DQS) and nutrition advice.

Non-technical description

eNutri FFQ

The eNutri FFQ captures dietary intakes for the last month. It was adapted from the validated Food4Me FFQ (Fallaize et al. 2014) and contains 157 food categories (e.g. white rice, bananas, coffee) that encompass all aspects of a UK diet. It also asks about dietary supplement use. Users first selects how often they consumed these foods/drinks (such as ‘not in the last month’, ‘once a month’, ‘twice a day’, etc.). Users are then asked to identify their usual portion size for each food/drink category by selecting 1 of the 3 portion images displayed on the screen or by selecting one of the buttons if their typical portion size was different to the images (e.g. more than, less than or between); in total, there are 7 portion options per food category (XS, S, S-M, M, M-L, L and XL). Sub-questions also allow differentiation between types (e.g. low salt vs. regular varieties) to increase accuracy.

Using the responses from the FFQ, mean daily dietary intakes (e.g. energy kcal/d, carbohydrate g/d, vitamin C mg/d) were calculated automatically by the eNutri app, which incorporates the nutritional compositions for each of the food categories.

Requirements

Four initial questions used to collect information about dietary requirements (meat, fish, dairy, supplements)

Frequency

This is the standard frequency question (eNutri 1 FFQ). They are necessarily followed by the portion images.

FreqOnly

This frequency question is identical to the standard frequency question, but the portion images are not presented (e.g. used for salt with an average portion size)

freqButtons and buttons

The freqButtons question has an extra frequency subquestion (“buttons”) (presenting three buttons: rarely, sometimes, mostly) with a variation of the main food item.

For example, after the normal frequency questions for the question 6.0 (Sushi (with or without fish)), the subquestion 6.1 (Did you eat sushi containing fish?) is presented and the composition is calculated as follows:

Rarely = Composition from (6.1-N) (No)
Mostly = Composition from (6.1-Y) (Yes)
Sometimes = Average between 6.1-N and 6.1-Y

The majority of the subquestions “buttons” are only used to collect the frequencies and don’t add weight (additional food item), but in a few cases the buttons can add a food item with a different food composition

freqCheckbox
In this type of question, the composition is calculated based on the average of the selected checkboxes.

Supplements
The supplements questions are collected using a specific frequency list (value/ratio per day).

"label":"No","value":"0"
"label":"1-3 doses per month","value":"0.0667"
"label":"1 dose per week","value":"0.1429"
"label":"2-4 doses per week","value":"0.4286"
"label":"5-6 doses per week","value":"0.7857"
"label":"1 dose per day", "value":"1"

NoYes and images
This type of question (No/Yes) is used after a standard “frequency” question (level 0) to check if an extra food item is added to the previous food item, generating this specific flow:

Frequency => NoYes => images => checkbox

In case the answer is “Yes”, the questions images and checkboxes are presented in order.

For example:

ID	type	text	image
12.0	frequency	Muesli & granola	Muesli
12.1	NoYes	Did you eat muesli/granola with milk (dairy or dairy-free)?	
12.2	images	Imagine you eat this portion size of muesli/granola. How much milk would you add?	Muesli with milk
12.3	checkbox	What type of milk did you add to muesli/granola?	

The types of milk are presented as checkboxes:

12.0	*Muesli & granola*
12.3-1	Whole milk
12.3-2	Semi-skimmed or 1% fat milk
12.3-3	Skimmed milk
12.3-4	Dairy-free milk

The Nutrients for this group are calculated in two steps. The first step (12.0) is identical to the standard frequency calculation. If the user selected “yes” in 12.1, then another row of nutrients is added to the nutrients sum taking into account the grams/day reported in 12.0 to calculate the amount of milk added in 12.2.

If an user eats a medium portion (48g) of “Muesli & Granola” once a day, so the consumption is 48g/day.

This individual selected the large image in 12.2 (81%), so the amount of milk added in the second step would be calculated as follows:

*(48*0.81)/(1-0.81)=204.6g*

As 81% of the bowl was milk and 19% was muesli.

The food composition of this second food item (milk) is calculated similarly to the freqCheckbox questions. If 12.3-1(whole milk) and 12.3-4 (dairy-free) are selected, the composition is the average between these two composition rows.

The amount of milk in hot drinks calculation differs from that used for cereals

ID	type	text	image
74.0	frequency	Coffee	Coffee
74.1	NoYes	Did you add milk (dairy or dairy-free) to your coffee?	
74.2	images	Imagine you made your usual coffee in this mug. How much milk would you add?	Milk in coffee
74.3	checkbox	What type of milk did you add to coffee	

74.0	*Coffee*
74.3-1	Whole milk, evapourated or condensed milk
74.3-2	Semi-skimmed or 1% fat milk
74.3-3	Skimmed milk
74.3-4	Dairy-free milk

The amount of milk is calculated as direct percentage of the g/day reported for the hot drink.

eNutri DQS

The eNutri DQS is based on 11 components, 7 of which are positive (maximum scores for high intakes of fruit, vegetables, dairy products, wholegrain products, healthy fats, oily fish, and nuts/pulses) and 4 are negative (maximum scores for low intakes of free sugar, salt, alcohol, and red/processed meat). For each component, users score between 0 and 10 points (e.g. someone who eats ≥1 portion/day of nuts/pulses daily would score the maximum 10 points for this component, whereas someone who eats 0 portions/day would score 0 and those with intermediate intakes would score proportionately). Each component contributes equally to the DQS and the total score available is 110 points (scaled down to 100 for ease of interpretation), whereby the highest scores represent the healthiest diets or those of greatest diet quality. The DQS is calculated automatically by the eNutri API and used in the creation of the nutrition advice.

The nutrition advice is individually tailored to the user, with the aim of having the greatest improvement to their overall diet quality. Following FFQ assessment, the eNutri app uses an algorithm to assign each food category within the FFQ a ‘healthiness’ score, from which it identified the foods/drinks that would have the greatest/worst impact on the DQS if one additional portion is consumed daily.

Preference model

Food items are ordered according to an individuals' food preference. Food preferences are obtained using the estimates from a Latent Dirichlet Allocation model which was applied to specific 'kernel density' estimates. The kernel densities are distributions that describe each food in the FFQ in terms of up to 12 attributes (sweet, spicy, salty, bitter, smooth, crunchy, chewy, mild flavored, expensive, easy to prepare, healthy, and eat for pleasure) and they are based on food perception data collected in the UK and Germany in 2018.

Nutrition advice

The healthiness score of each food differed for each participant since it is based on their actual dietary intake.

The nutrition advice was split into 5 categories:

1. Foods to boost
2. Foods to try
3. Foods to reduce
4. Foods to keep eating
5. Foods to continue avoiding

For each category, the top 5 foods items are presented to the participant (only 3 for boost and try). For example, the categories 'foods to boost/keep eating' and 'foods to reduce' highlighted the best/worst aspects of the participant's current diet, respectively.

Alongside the food items in the healthy eating report were tips about portion sizes and ideas for incorporating these foods into their diet or suitable swaps for unhealthy foods, as well as information about the health benefits/detriment associated with eating these foods. Tips differed according to an individual's BMI (e.g. smaller portion sizes if overweight/obese) and dietary choices (e.g. vegetarian) as additional degrees of personalisation. Participants were also presented with their overall DQS out of 100 as part of their PN report at weeks 0 and 12 so that they could monitor any improvement after 12 weeks.

Technical description

2. eNutri logo

Figure 1 - eNutri logo

3. Swagger specification

The API specification is as defined in the swagger document found in the annex. See *Swagger Specification*.

4. API calls, parameters and responses

Since the code base for the eNutri model is R, the library selected to serve the necessary endpoints was Plumber. Since R is a single threaded language, so too is the REST API. As a result, only one request per API instance can be processed at any given time. The API is hosted in Azure within a Kubernetes cluster capable of being scaled as load increases and at present is load-balanced across three instances. Note, however, if the number of concurrent requests exceeds three, some of these surplus requests may result in failure (response code 504).

API calls

Although there are five exposed API endpoints, only the "advice" endpoint is currently being utilized by the eNutri web app.

Path	Method	Description
advice	POST	Returns advice on how a user's diet can be improved
nutrients	POST	Returns the nutrients breakdown based on the submitted diet
dqs	POST	Returns the diet quality score breakdown based on the submitted diet
preferences	POST	Returns a user's food preferences based on their submitted diet
status	GET	Status of API

Table 1 – API calls

API parameters

Each of the POST endpoints requires the FFQ to be supplied as JSON. If the structure of the input fails JSON validation, the issuer will receive a 500 error. This is simply due to a lack of validation on input to the model.

API responses

Response codes returned are as follows:

Response Code	Description
200	Successful request
401	Access denied due to missing subscription key
404	not found; if an endpoint is requested that is not exposed
429	Rate limit exceeded – see response for delay before retry
500	Internal Server Error – occurs when the input is invalid or there is a bug in the model

504	Gateway timeout – API gateway times out while waiting for the request to be fulfilled. This can occur if the number of concurrent requests exceeds that supported by the server and a request sits in the queue longer than the configured timeout (30 seconds)

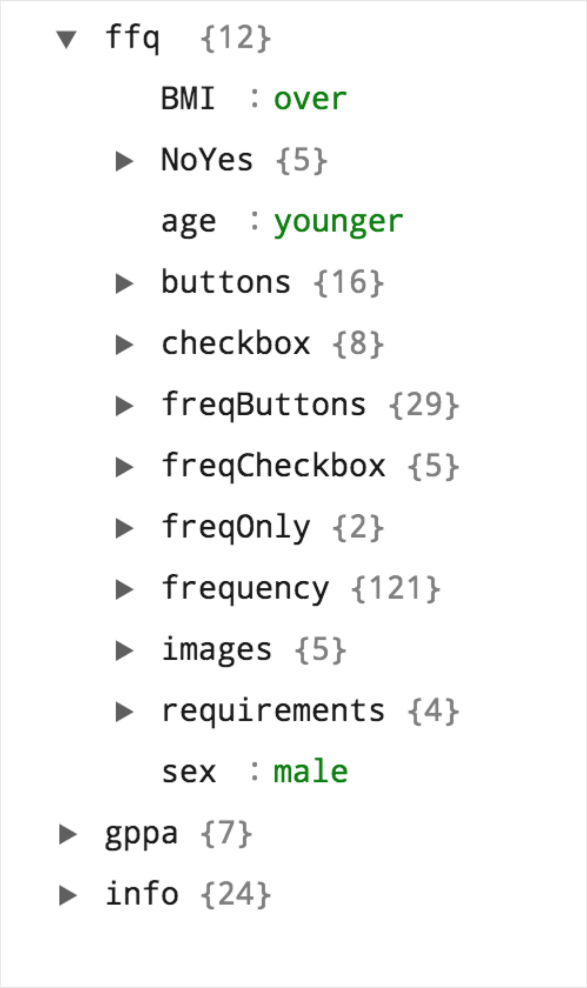
Table 2 – API response codes

5. API description and possible values

6. Structure of the requests and responses

Request structure

Each question type (frequency, freqButton, freqCheckbox, buttons, checkbox, requirement, images, NoYes) has a specific node structure.

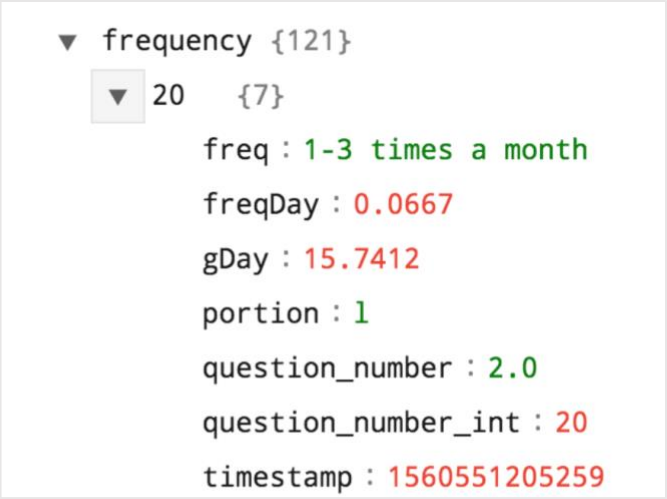


```
▼ ffq  {12}
     BMI  : over
   ▶ NoYes {5}
     age  : younger
   ▶ buttons {16}
   ▶ checkbox {8}
   ▶ freqButtons {29}
   ▶ freqCheckbox {5}
   ▶ freqOnly {2}
   ▶ frequency {121}
   ▶ images {5}
   ▶ requirements {4}
     sex  : male
▶ gppa {7}
▶ info {24}
```

Figure 2 - JSON structure of the expected request

Apart from the question type nodes, the eNutri app generates collects sex, age, BMI and a physical activity questionnaire (GPPA). Age and GPPA are not used by the current version of the eNutri API.

- sex
    - male or female
- age
    - younger: 18-39
    - mid: 40-64
    - older: ≥65 y)
- BMI
    - under: <18.0
    - normal: 18.0-24.9
    - over: ≥25.0 kg/m2)

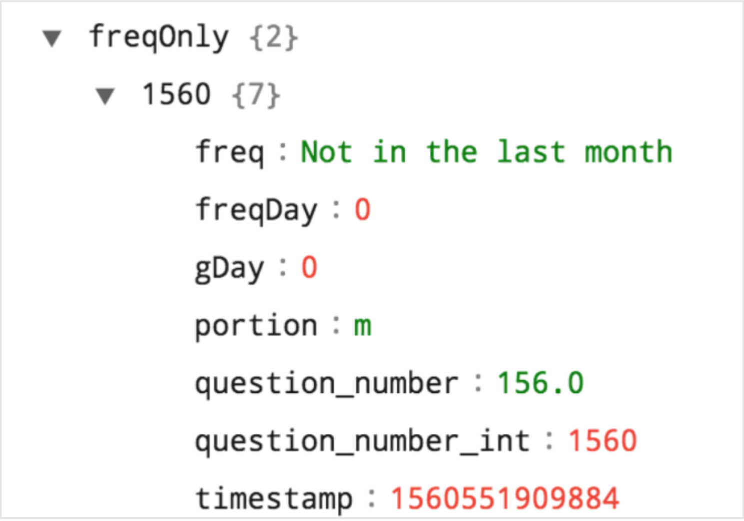


```
▼ frequency {121}
    ▼ 20   {7}
          freq : 1-3 times a month
          freqDay : 0.0667
          gDay : 15.7412
          portion : 1
          question_number : 2.0
          question_number_int : 20
          timestamp : 1560551205259
```

Figure 3 - JSON structure for question type frequency

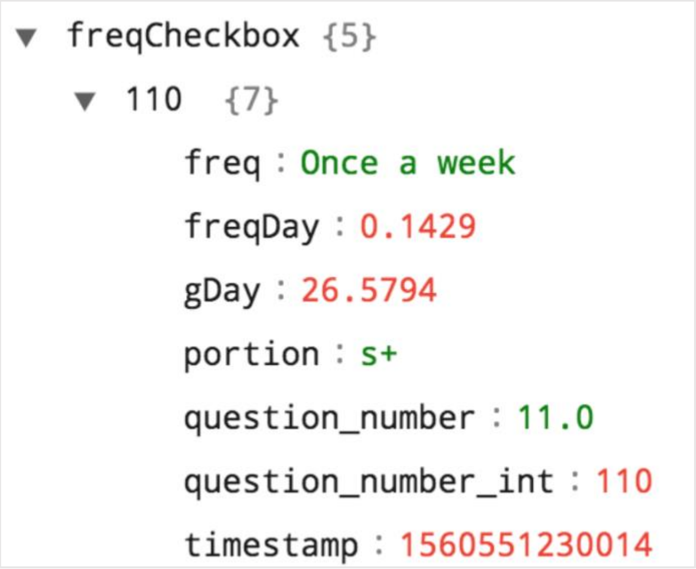
In the eNutri app, all questions are saved together with the absolute timestamp, but they are not used by the eNutri API (decision engine) during the processing.

The *question_number* contains the official question number in float (1 decimal). The *question_number_int* is the conversion of the *question_number* into integer, multiplied by 10.



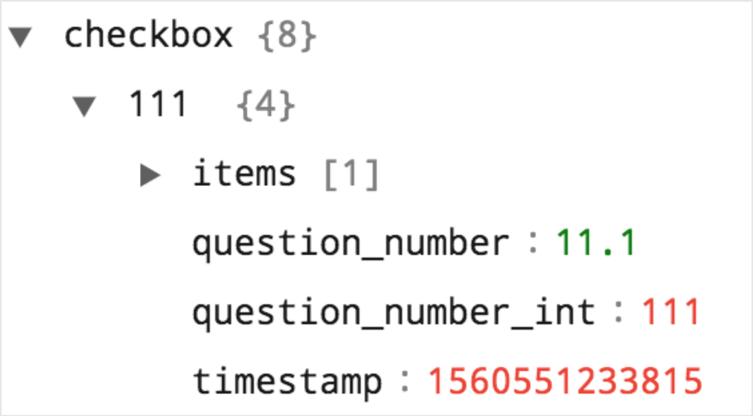
```
▼ freqOnly {2}
  ▼ 1560 {7}
      freq : Not in the last month
      freqDay : 0
      gDay : 0
      portion : m
      question_number : 156.0
      question_number_int : 1560
      timestamp : 1560551909884
```

Figure 4 - JSON structure for question type freqOnly



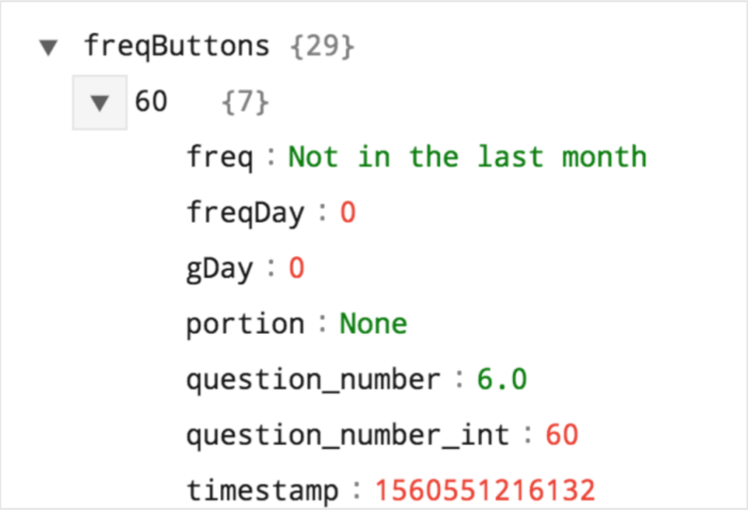
```
▼ freqCheckbox {5}
  ▼ 110  {7}
      freq : Once a week
      freqDay : 0.1429
      gDay : 26.5794
      portion : s+
      question_number : 11.0
      question_number_int : 110
      timestamp : 1560551230014
```

Figure 5 - JSON structure for question type freqCheckbox



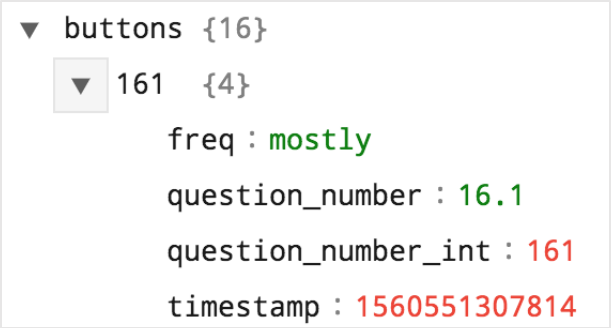
```
▼ checkbox {8}
  ▼ 111  {4}
    ▶ items [1]
      question_number : 11.1
      question_number_int : 111
      timestamp : 1560551233815
```

Figure 6 - JSON structure for question type checkbox



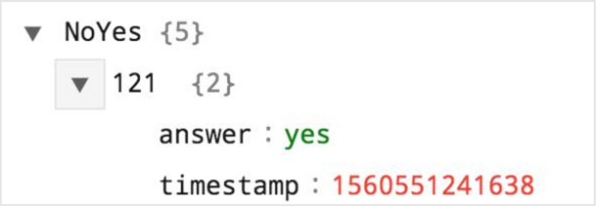
```
▼ freqButtons {29}
  ▼ 60   {7}
        freq : Not in the last month
        freqDay : 0
        gDay : 0
        portion : None
        question_number : 6.0
        question_number_int : 60
        timestamp : 1560551216132
```

Figure 7 - JSON structure for question type freqButtons



```
▼ buttons {16}
  ▼ 161   {4}
        freq : mostly
        question_number : 16.1
        question_number_int : 161
        timestamp : 1560551307814
```

Figure 8 - JSON structure for question type buttons

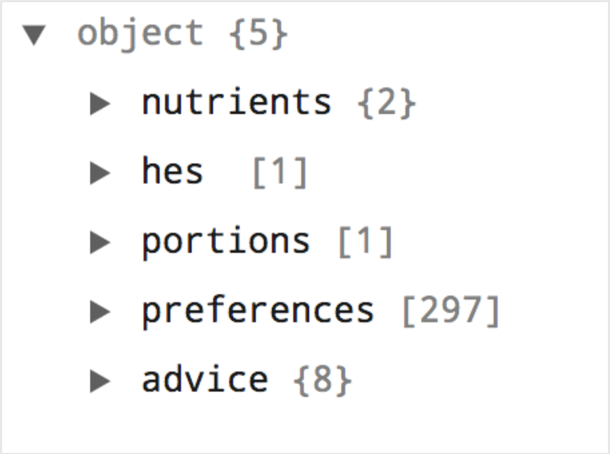


```
▼ NoYes {5}
  ▼ 121   {2}
        answer : yes
        timestamp : 1560551241638
```

Figure 9 - JSON structure for question type NoYes

Response structure

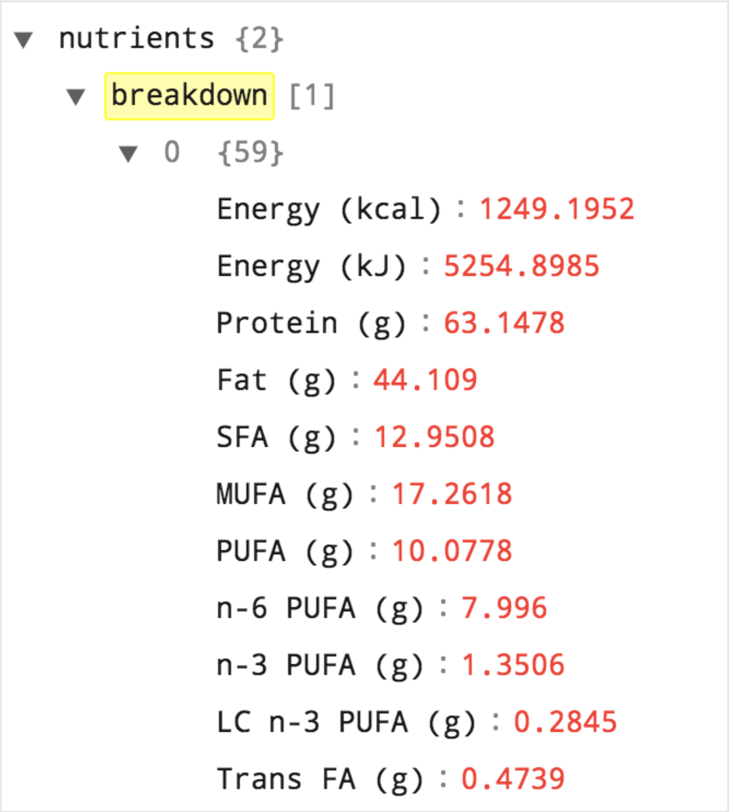
The root node will always contain five nodes: nutrients, hes, portions, preferences and advice.



```
▼ object {5}
    ▶ nutrients {2}
    ▶ hes  [1]
    ▶ portions [1]
    ▶ preferences [297]
    ▶ advice {8}
```

Figure 10 - JSON response structure

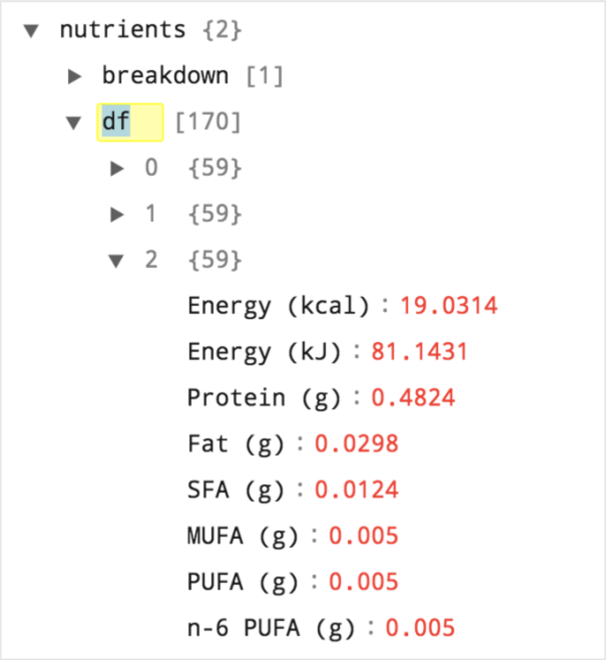
The nutrients will present the breakdown (total sum) per variable (e.g. Energy) with the related unit (e.g. kcal).



```
▼ nutrients {2}
    ▼ breakdown [1]
        ▼ 0  {59}
               Energy (kcal) : 1249.1952
               Energy (kJ) : 5254.8985
               Protein (g) : 63.1478
               Fat (g) : 44.109
               SFA (g) : 12.9508
               MUFA (g) : 17.2618
               PUFA (g) : 10.0778
               n-6 PUFA (g) : 7.996
               n-3 PUFA (g) : 1.3506
               LC n-3 PUFA (g) : 0.2845
               Trans FA (g) : 0.4739
```

Figure 11 - JSON response for nutrients breakdown

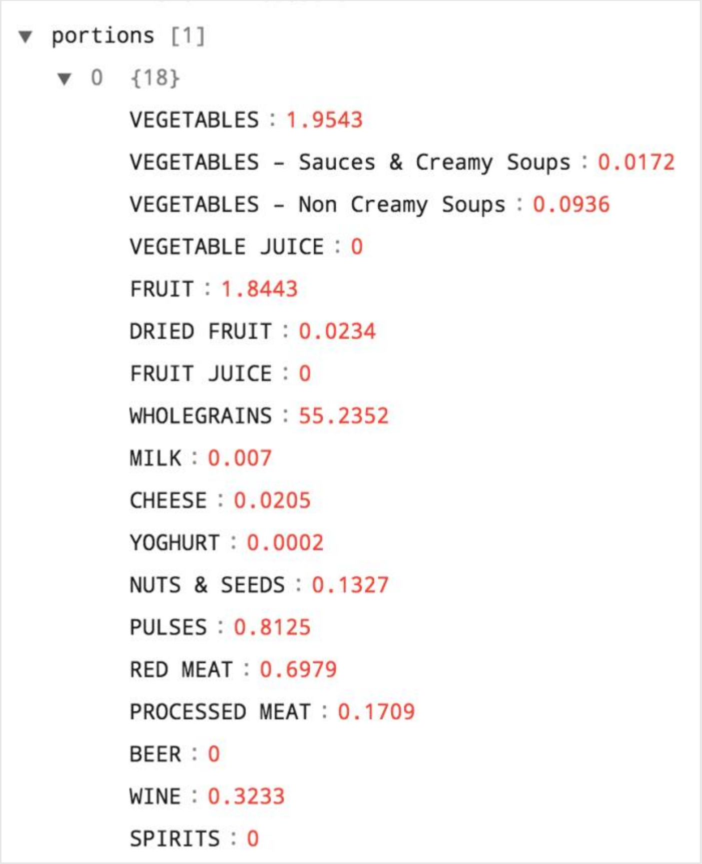
How each food item contributed to the total is shown in the *df* (dataframe) node:



```
▼ nutrients {2}
    ▶ breakdown [1]
    ▼ df  [170]
        ▶ 0  {59}
        ▶ 1  {59}
        ▼ 2  {59}
              Energy (kcal) : 19.0314
              Energy (kJ) : 81.1431
              Protein (g) : 0.4824
              Fat (g) : 0.0298
              SFA (g) : 0.0124
              MUFA (g) : 0.005
              PUFA (g) : 0.005
              n-6 PUFA (g) : 0.005
```

Figure 12 - JSON response with nutrients dataframes

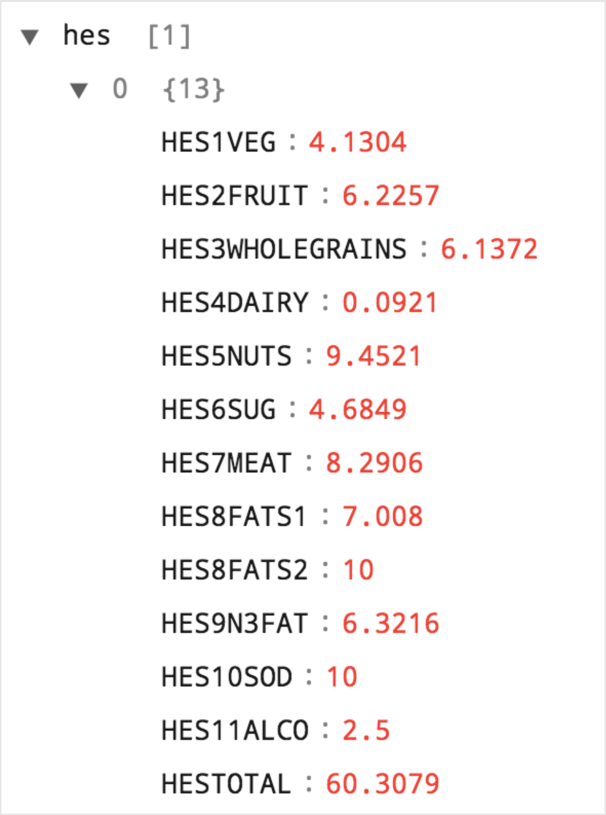
The total per portion is returned within the *portions* node:



```
▼ portions [1]
    ▼ 0  {18}
          VEGETABLES : 1.9543
          VEGETABLES - Sauces & Creamy Soups : 0.0172
          VEGETABLES - Non Creamy Soups : 0.0936
          VEGETABLE JUICE : 0
          FRUIT : 1.8443
          DRIED FRUIT : 0.0234
          FRUIT JUICE : 0
          WHOLEGRAINS : 55.2352
          MILK : 0.007
          CHEESE : 0.0205
          YOGHURT : 0.0002
          NUTS & SEEDS : 0.1327
          PULSES : 0.8125
          RED MEAT : 0.6979
          PROCESSED MEAT : 0.1709
          BEER : 0
          WINE : 0.3233
          SPIRITS : 0
```

Figure 13 - JSON response with the portions

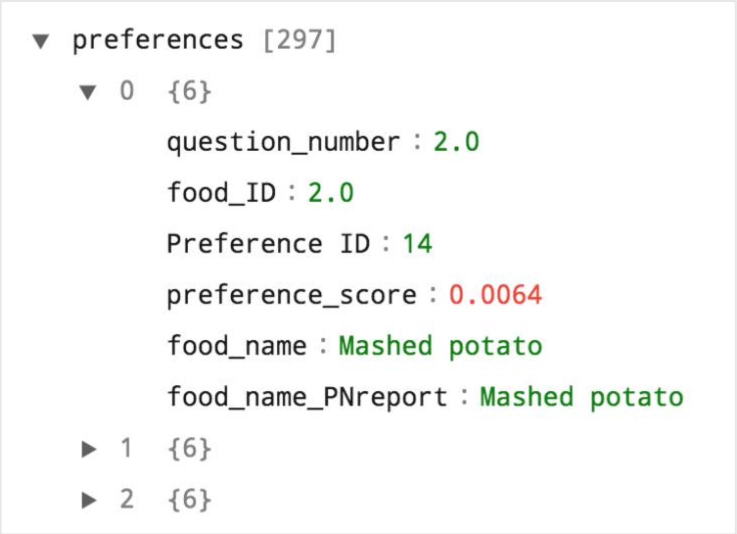
The DQS is returned within the *hes* (heathy eating score) node. The *HESTOTAL* is the overall DQS score.



```
▼ hes  [1]
   ▼ 0  {13}
        HES1VEG : 4.1304
        HES2FRUIT : 6.2257
        HES3WHOLEGRAINS : 6.1372
        HES4DAIRY : 0.0921
        HES5NUTS : 9.4521
        HES6SUG : 4.6849
        HES7MEAT : 8.2906
        HES8FATS1 : 7.008
        HES8FATS2 : 10
        HES9N3FAT : 6.3216
        HES10SOD : 10
        HES11ALCO : 2.5
        HESTOTAL : 60.3079
```

Figure 14 - JSON response with the DQS (HES)

The preference scores per food item are presented in the *preferences* node:

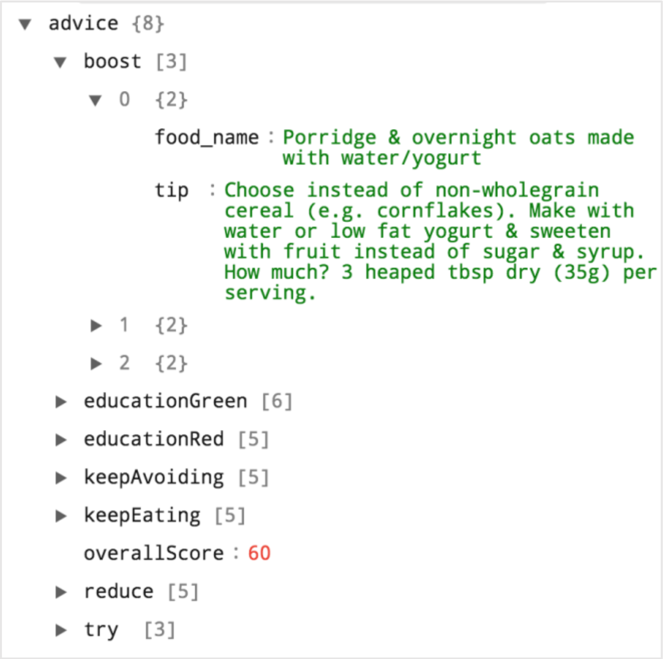


```
▼ preferences [297]
   ▼ 0  {6}
        question_number : 2.0
        food_ID : 2.0
        Preference ID : 14
        preference_score : 0.0064
        food_name : Mashed potato
        food_name_PNreport : Mashed potato
   ▶ 1  {6}
   ▶ 2  {6}
```

Figure 15 - JSON response with the preference score

The *advice* node contains all the information needed to create the personalised nutrition (PN) report. Its nodes are related to the advice sessions describe in the first chapter.

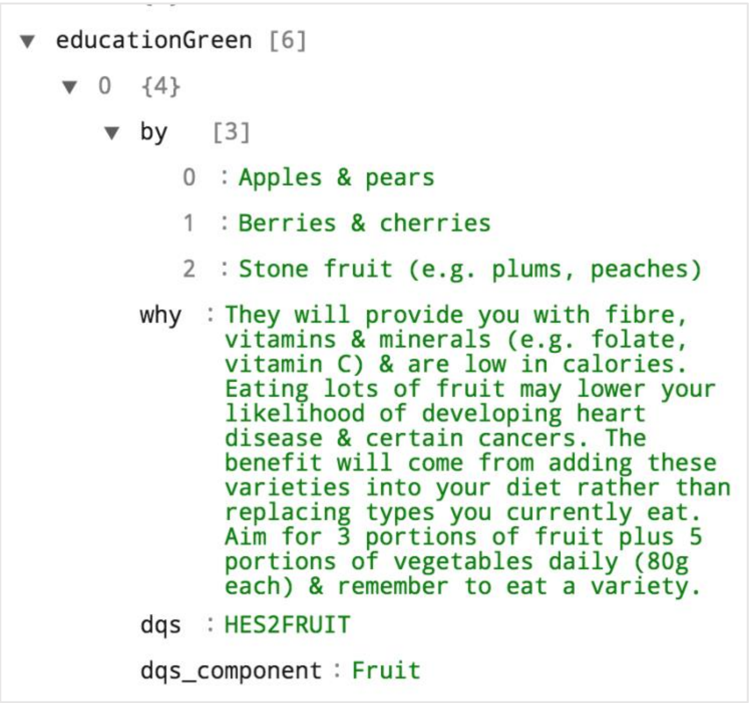
The following image shows a node with a ‘food to boost’ and related tip. The *overallScore* is a copy of the HESTOTAL.



```
▼ advice {8}
  ▼ boost [3]
    ▼ 0  {2}
         food_name : Porridge & overnight oats made
                     with water/yogurt
         tip  : Choose instead of non-wholegrain
                cereal (e.g. cornflakes). Make with
                water or low fat yogurt & sweeten
                with fruit instead of sugar & syrup.
                How much? 3 heaped tbsp dry (35g) per
                serving.
    ▶ 1  {2}
    ▶ 2  {2}
  ▶ educationGreen [6]
  ▶ educationRed [5]
  ▶ keepAvoiding [5]
  ▶ keepEating [5]
    overallScore : 60
  ▶ reduce [5]
  ▶ try  [3]
```

Figure 16 - JSON response with food to boost

In the same way, the following JSON structure shows three foods to keep eating and the reason *why*.

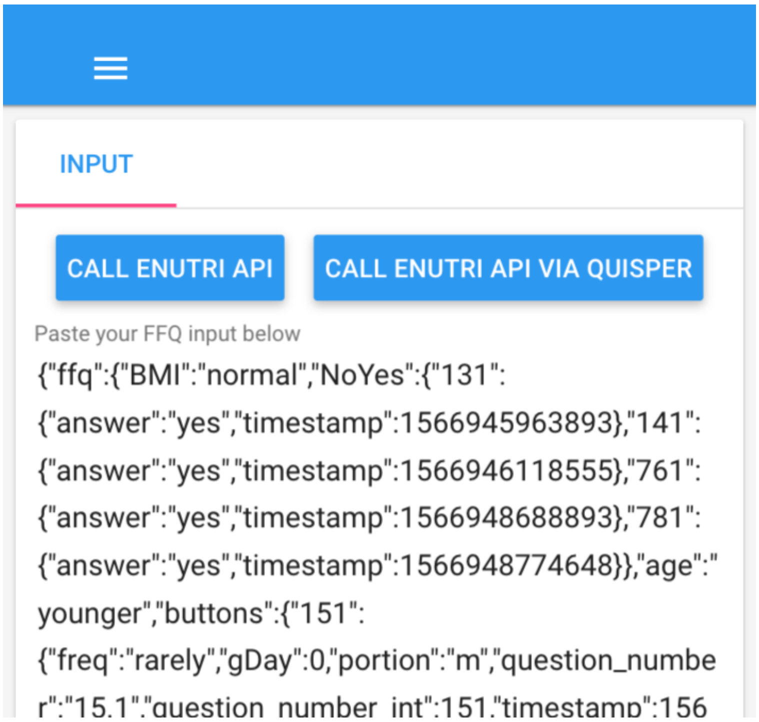


```
▼ educationGreen [6]
  ▼ 0  {4}
    ▼ by    [3]
         0  : Apples & pears
         1  : Berries & cherries
         2  : Stone fruit (e.g. plums, peaches)
      why  : They will provide you with fibre,
             vitamins & minerals (e.g. folate,
             vitamin C) & are low in calories.
             Eating lots of fruit may lower your
             likelihood of developing heart
             disease & certain cancers. The
             benefit will come from adding these
             varieties into your diet rather than
             replacing types you currently eat.
             Aim for 3 portions of fruit plus 5
             portions of vegetables daily (80g
             each) & remember to eat a variety.
      dqs  : HES2FRUIT
      dqs_component : Fruit
```

Figure 17 - JSON response with foods to keep eating

7. Examples

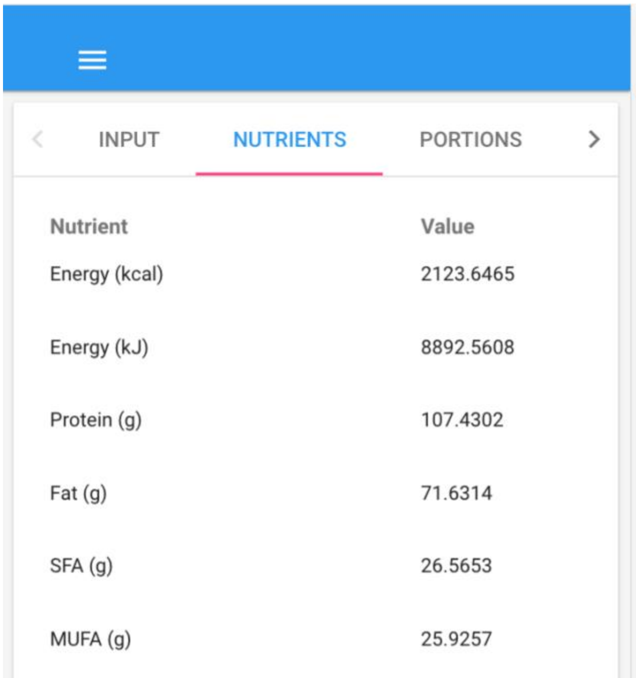
A user interface for testing the API calls was developed and it is available on https://enutri.de/#!/api



{"ffq":{"BMI":"normal","NoYes":{"131":{"answer":"yes","timestamp":1566945963893},"141":{"answer":"yes","timestamp":1566946118555},"761":{"answer":"yes","timestamp":1566948688893},"781":{"answer":"yes","timestamp":1566948774648}},"age":"younger","buttons":{"151":{"freq":"rarely","gDay":0,"portion":"m","question_number":"15.1","question_number_int":151,"timestamp":156

Figure 18 - User Interface for testing the eNutri API

The API response is formatted, and the elapsed time is shown in the response.



Nutrient	Value
Energy (kcal)	2123.6465
Energy (kJ)	8892.5608
Protein (g)	107.4302
Fat (g)	71.6314
SFA (g)	26.5653
MUFA (g)	25.9257

Figure 19 - User Interface showing the formatted nutrients response

Request example

The request example is available on the following url:
https://eatwelluk.org/assets/api_uk_sample_request.json

Response example

The request example is available on the following url:
https://eatwelluk.org/assets/api_uk_sample_request.json

Client call example (JavaScript)

```
var callAPI = function(ffqObj){
    console.log(ffqObj);
    $(function() {
        var params = {
            // Request parameters
        };

        $.ajax({
            url: "https://api.agrimetrics.co.uk/enutri/advice" + $.param(params),
            beforeSend: function(xhrObj){
     // Request headers
            xhrObj.setRequestHeader("Ocp-Apim-Subscription-Key","your-key-here");
            xhrObj.setRequestHeader("Content-Type","application/json");
            xhrObj.setRequestHeader("Ocp-Apim-Trace",true);
            xhrObj.setRequestHeader("cache-control","no-cache");
            },
            type: "POST",
            // Request body
            data: JSON.stringify(ffqObj),
        })
            .done(function(data) {
                console.log(data);
            })
            .fail(function() {
                alert("error");
            });
    });
};
```

8. Public documentation

9. Annexes

Swagger Specification (YAML)

The following swagger conforms to the Swagger V2.0 specification.

```yaml
swagger: '2.0'
info:
  title: E-Nutri API
  version: $version
  description: E-Nutri
  termsOfService: 'https://developer.agrimetrics.co.uk/terms/'
  contact:
    email: help@agrimetrics.co.uk
externalDocs:
  description: API Overview
  url: 'https://developer.agrimetrics.co.uk/docs/apis'
host: api$env.agrimetrics.co.uk
basePath: /enutri
schemes:
  - https
produces:
  - application/json
paths:
  /status:
    get:
      description: Returns something for status monitoring
      operationId: get-status
      summary: status
      responses:
        '200':
          description: Default response.
      produces:
        - application/json
  /nutrients:
    post:
      description: Return a nutrient breakdown based on diet.
      operationId: post-nutrients
      summary: nutrients
      parameters:
        - name: jsonObject
          in: body
          schema:
            $ref: '#/definitions/JsonObject'
          description: outcome
      consumes:
        - application/json
      responses:
        '200':
          description: Default response.
      produces:
        - application/json
  /dqs:
    post:
      description: Return a dietary score based on diet.
      operationId: post-dqs
      summary: dqs
      parameters:
        - name: jsonObject
          in: body
          schema:
            $ref: '#/definitions/JsonObject'
          description: outcome
      consumes:
        - application/json
      responses:
        '200':
          description: Default response.
      produces:
        - application/json
  /preferences:
    post:
      description: Return a set of preferences based on diet.
      operationId: post-preferences
      summary: preferences
      parameters:
        - name: jsonObject
          in: body
          schema:
            $ref: '#/definitions/JsonObject'
          description: outcome
      consumes:
        - application/json
      responses:
        '200':
          description: Default response.
      produces:
        - application/json
  /advice:
    post:
      description: Return dietary advice based on diet.
      operationId: post-advice
      summary: advice
      parameters:
        - name: jsonObject
          in: body
          schema:
            $ref: '#/definitions/JsonObject'
          description: outcome
      consumes:
        - application/json
      responses:
        '200':
          description: Default response.
      produces:
        - application/json
definitions:
  JsonObject:
    type: object
tags: []
```