# Code Structure Documentation

## Overview

Our application is built with JS + React for its frontend, and Python + Flask for its backend.

## Backend

### Directory Structure

app/  
├── public/ **# Static files for the frontend**  
├── ActivitiesConnection.py **# Activity database operations**  
├── app.py **# Main Flask application**  
├── FBConnection.py **# Food and Beverage database operations**  
├── fbValues.py **# Functions to be used in routes for food and beverage data**  
├── requirements.txt **# Python dependencies**  
├── search.py **# Functions to be used in routes for search functionality**  
├── user.py **# Functions used in routes for user data modification and meal logging**  
├── UserConnection.py **# User database operations**  
└── utilities.py **# Utility functions**

### Core Components

1. Database Connections - Three main connection classes handle database operations:
   * **FBConnection**: Manages food and beverage related queries
   * **UserConnection**: Handles user-related operations
   * **ActivitiesConnection**: Manages physical activity queries
2. Utility Functions - Located in utilities.py, provides core database functionality:
   * Connection pool management
   * Query execution helpers
   * Error handling

### API Endpoints

#### Food and Beverage Endpoints

| Endpoint | Method | Description |
| --- | --- | --- |
| /api/search | GET | Search foods by name |
| /api/fb/category/<id> | GET | Get category information for a specific food/beverage |
| /api/fb/calories/<id> | GET | Get calorie information with optional weight parameter (*default: 100g*) |
| /api/fb/<id> | GET | Get food/beverage information for a specific food/beverage |
| /api/fb/ingredients/<id> | GET | Get ingredient breakdown with optional weight parameter (*default: 100g*) |
| /api/fb/top\_values/<id> | GET | Get top nutritional values for a specific food/beverage |
| /api/fb/values/<id> | GET | Get detailed nutritional information with optional weight parameter (*default: 100g*) |
| /api/fb/portions/<id> | GET | Get available portion sizes for a specific food/beverage |
| /api/fb/rdi/<id> | GET | Get RDI (Recommended Daily Intake) percentages with optional weight parameter (*default: 100g*) |
| /api/fb/<id>/score | GET | Get nutritional score for a specific food/beverage |

#### User Management Endpoints

| Endpoint | Method | Description |
| --- | --- | --- |
| /api/sign-up | POST | Create new user account |
| /api/login | POST | Authenticate user and receive JWT token |
| /api/user/weight/<id> | GET | Get user’s weight |
| /api/user/<id>/fb-history | GET | Get user’s food/beverage logging history |
| /api/user/<id> | PATCH | Update user profile information |
| /api/user/<id>/add-meal | POST | Log a new meal |
| /api/user/<id>/score | GET | Get user’s health score |
| /api/user/<id>/relative-score | GET | Get user’s relative health score |

#### Activities Endpoints

| Endpoint | Method | Description |
| --- | --- | --- |
| /api/activities/<calories> | GET | Get activities that roughly burn the requested amount of calories adjusted to the user’s weight (*default: 80kg*) |

### Database Classes

#### FBConnection (Food and Beverage Connection)

A class to manage database connections and execute queries related to food and nutrition data. Handles connections to the MySQL database and provides methods to search various tables for information such as food descriptions, categories, nutrients, and portions.

##### Methods

* \_\_init\_\_(): Initialises connection pool to database
* search\_foods(food\_name: str): Search foods by name
* search\_food\_category(food\_id: int): Get category description for a food item
* search\_fb\_name(food\_id: int): Get food description by ID
* search\_food\_calories(food\_id: int, weight: float = 100.0): Calculate calories for given weight
* search\_portions(food\_id: int): Get available portion sizes and weights
* search\_ingredients(food\_id: int, weight: float = 100.0): Get ingredient breakdown with normalised weights
* search\_rdi\_per\_nutrient(food\_id: int, weight: float = 100.0): Calculate RDI percentages for each nutrient
* search\_nutrient\_percentage\_per\_ingredient(food\_id: int, max\_ingredients: int = 65): Calculate nutrient percentages per ingredient
* search\_nutrients\_per\_ingredient(food\_id: int, weight: float = 100.0): Get nutrient values for given weight
* get\_nutrition\_score(food\_id: int): Calculate overall nutritional score

#### UserConnection

A class to manage user-related interactions with the database. Provides methods for user creation, retrieval, updates, and meal logging.

##### Methods

* \_\_init\_\_(): Initialises connection pool to database
* create\_user(email: str, first\_name: str, last\_name: str, password: str, dob: str, gender: str, weight: int): Register new user
* get\_user(email: str, password: str): Authenticate user
* get\_user\_weight(user\_id: int): Get user’s weight
* get\_full\_name(user\_id: int): Get user’s full name
* update\_user(user\_id: int, first\_name: str = None, last\_name: str = None, weight: int = None): Update user profile
* insert\_meal(user\_id: int, fb\_id: int, portion: float, date\_time: str): Log meal
* show\_meals(user\_id: int): Get user’s meal history
* get\_user\_static\_nutrition\_score(user\_id: int): Calculate user’s health score
* get\_user\_relative\_nutrition\_score(user\_id: int): Calculate health score relative to other users

#### ActivitiesConnection

A class to manage database connections and execute queries related to physical activities. Handles connections to the MySQL database and provides methods to search activities based on calorie expenditure and user weight.

##### Methods

* \_\_init\_\_(): Initialises connection pool to database
* search\_activities\_calories(calories: int, weight: int = 80, max\_activities: int = 4): Find activities that roughly burn the specified amount of calories, adjusted for user weight

### Authentication

The application uses JWT (JSON Web Token) for authentication:

1. **Token Generation**
   * Created upon successful login
   * Contains user email as identity
   * Configurable expiration time
2. **Token Usage**
   * Include in Authorisation header
   * Format: Bearer <token>

### Deployment

#### Requirements

* Python
* MySQL
* Required Python packages (see requirements.txt):
  + mysql-connector-python
  + flask
  + flask-cors
  + flask-jwt-extended

#### Setup Steps

1. Install dependencies
2. Initialise database
3. Start Flask application

## Frontend

### Directory Structure

client/  
├── public/ **# Static files for the frontend**  
├── src/  
│ ├── FB/ **# Food & Beverages components**  
│ │ ├── CalorieBurnTime.js **# Activity suggestions based on calories**  
│ │ ├── CaloriesChart.js **# Doughnut chart for calorie breakdown**  
│ │ ├── FBInfo.js **# Food and beverage information**  
│ │ ├── FoodAndBeveragesList.js **# List of search results**  
│ │ ├── FoodAndBeveragesPage.js **# Food details page**  
│ │ ├── HealthScore.js **# Food health score display**  
│ │ ├── MealIngredients.js **# Ingredient breakdown table**  
│ │ ├── MealValues.js **# Nutritional values table**  
│ │ ├── PortionPicker.js **# Portion size selector**  
│ │ └── RdiChart.js **# RDI percentage bar chart**  
│ ├── User/ **# User-related components**  
│ │ ├── HealthScale.js **# User health score comparison**  
│ │ ├── Login.css **# Login page styles**  
│ │ ├── Login.js **# User login form**  
│ │ ├── MealHistory.js **# User's meal logging history**  
│ │ ├── SignUp.js **# User registration form**  
│ │ ├── UpdateProfile.js **# User profile update form**  
│ │ ├── UserHealthScore.js **# User health score display**  
│ │ ├── UserMealInput.js **# Meal logging form**  
│ │ └── UserPage.js **# User profile page**  
│ ├── App.css **# Main application styles**  
│ ├── App.js **# Main application component**  
│ ├── FeaturePage.js **# Features showcase page**  
│ ├── HomePageFooter.js **# Footer component**  
│ ├── index.css **# Global styles**  
│ ├── index.js **# Application entry point**  
│ ├── Navbar.js **# Navigation component**  
│ └── SearchMeals.js **# Search interface component**  
├── .env **# Environment variables**  
├── package.json **# Project dependencies and scripts**  
└── README.md **# Project documentation**

### Core Components

1. **App.js**
   * Main application container
   * Handles routing with React Router
   * Manages user authentication state
   * Contains all route definitions and their corresponding components
2. **FeaturePage.js**
   * Displays a grid of feature cards
   * Shows key features like “No duplicate foods”, “Up-to-date”, etc.
3. **HomePageFooter.js**
   * Footer component with social media links
4. **index.js**
   * Application entry point
5. **Navbar.js**
   * Top navigation bar component
   * Handles user authentication state
   * Contains logo/brand name “MealMeter”
   * Dynamic rendering of login/logout buttons
   * Profile navigation functionality
6. **SearchMeals.js**
   * Search interface for meals

### Food & Beverages (FB) Components

Here’s a breakdown of the JavaScript components in the FB directory:

1. **CalorieBurnTime.js**
   * Displays activities that burn calories according to user’s weight and target calories
2. **CaloriesChart.js**
   * Shows calorie breakdown by nutrient categories
   * Displays total calories in center of chart
3. **FBInfo.js**
   * Displays food/beverage information in the header
   * Shows category and name of the item
4. **FoodAndBeveragesList.js**
   * Displays search results for foods
   * Shows “No Matching Results” when empty
   * Navigates to detailed view on click
5. **FoodAndBeveragesPage.js**
   * Contains three tabs: Information, Nutrition Results, Health Score
   * Manages portion size state
6. **HealthScore.js**
   * Displays food health score (0-100)
7. **MealIngredients.js**
   * Shows ingredient breakdown table
   * Updates based on portion size changes
8. **MealValues.js**
   * Displays nutritional values table: value and unit
9. **PortionPicker.js**
   * Allows selection of predefined portions
   * Supports custom portion input
10. **RdiChart.js**
    * Shows RDI (Recommended Daily Intake) as bar chart
    * Displays 100% threshold line
    * Updates based on portion size changes

#### Requirements

* Node.JS
* NPM
* React
* Other required Node.JS packages (see package.json, install with npm install):
  + @emotion/react
  + @emotion/styled
  + @mui/base
  + @mui/icons-material
  + @mui/material
  + @mui/x-date-pickers
  + @mui/lab
  + axios
  + chart.js
  + chartjs-plugin-annotation
  + cra-template
  + dayjs
  + react
  + react-chartjs-2
  + react-dom
  + react-hook-form
  + react-router-dom
  + react-scripts

## Machine Learning Algorithm

### Nutrient Importance Analysis

The nutrient importance analysis performed in Nutrient\_Importance.ipynb is done to determine the importance of different nutrients in predicting food healthiness. The key steps are:

1. **Data Preparation**
   * Load food and nutrient data from the OpenFoodFacts dataset. This dataset is used since it contains food nutrient values alongside their nutritional scores.
   * Handle missing values
2. **Statistical Analysis**
   * Use regression analysis to determine nutrient importance weights
   * Evaluate the statistical significance of each nutrient’s contribution
   * Key nutrients analysed include macronutrients, vitamins, minerals, etc…
3. **Result Processing**
   * Aggregate importance weights by nutrient ID
   * Sort nutrients by their relative importance
4. **Data Export**
   * Export nutrient importance weights to the Nutrients table in the database
   * Calculate the nutritional score for each food item based on the nutrient importance weights, and export to the Food\_Beverages table
   * Results are renormalised on a scale of 0-1 to ensure consistent scoring across different food items, and to avoid problems with domain shift