Workshop in Databases:

Final Project Documentation

Adam Aharony
Adam.Aharony@gmail.com

Bar Katash
KatashBar15@gmail.com

Contents

Database Documentation

Overview

This section covers the database implementation for the project. We use MySQL as our database management system with a single schema `db04` (as required by the server) that contains both application data and user-related data.

Directory Structure

```
DB/
├── Data/ # Contains raw data files
│   └── CSV/ # Contains CSV data files (after processing)
├── Migration/ # Python scripts, Jupyter Notebooks for data migration
├── SQL/ # SQL scripts for database operations
│   ├── Initialisation.sql # Schema initialisation script for the database
│   └── Insertion/ # Data insertion scripts for the database
└── Synth/ # Synthetic data generation scripts and output files
```

Database Schema

Application Data Tables

Tables containing application data: food and beverage data, nutritional data, activity data, etc...

Tables:

1. **Category**
    - Primary food and beverage categories, its goal is to group food and beverages into categories.
    - Fields:
        - `ID` (`SMALLINT UNSIGNED`): Category ID
        - `Description` (`VARCHAR(150)`): Category description
    - Primary Key: `ID`
    - Indexes: `ID`

2. **Food_Beverages**
    - Main food and beverage items, its goal is to store the food and beverage items in the database.
    - Fields:
        - `ID` (`INT UNSIGNED`): Food/Beverage ID
        - `Description` (`VARCHAR(200)`): Food/Beverage description
        - `Extra_Description` (`VARCHAR(450)`): Extra description
        - `Category_ID` (`SMALLINT UNSIGNED`): Category ID
        - `Score` (`FLOAT UNSIGNED`): Health score
    - Primary Key: `ID`
    - Foreign Keys: `Category_ID` references `Category(ID)`
    - Indexes: `ID`
3. **Portions_Weights**
    - Provided portion sizes for foods, its goal is to store portion sizes for each food and beverage item.
    - Fields:
        - `FB_ID` (`INT UNSIGNED`): Food/Beverage ID
        - `Description` (`VARCHAR(200)`): Description of the portion size
        - `Weight` (`FLOAT UNSIGNED`): Weight of the portion size
    - Primary Key: `FB_ID`, `Description`, `Weight`
    - Foreign Keys: `FB_ID` references `Food_Beverages(ID)`
    - Indexes: `FB_ID`
4. **Nutrients**
    - Nutrient definitions and properties.
    - Fields:
        - `ID` (`SMALLINT UNSIGNED`): Nutrient ID
        - `Name` (`VARCHAR(50)`): Nutrient name
        - `Unit` (`ENUM("kcal", "g", "mg", "mcg")`): Unit of the nutrient
        - `Importance` (`FLOAT`): Importance of the nutrient, used to calculate the health score and obtained using linear regression (see Jupyter Notebook found in `Migration/Nutrient_Importance.ipynb`).
    - Primary Key: `ID`
    - Indexes: `ID`

5. **Ingredients**
    - Ingredient table, store ingredient descriptions along with their ID.
    - Fields:
        - `ID` (`INT UNSIGNED`): Ingredient ID
        - `Description` (`VARCHAR(150)`): Ingredient description
    - Primary Key: `ID`
    - Indexes: `ID`
6. **Ingredients_Values**
    - Nutrient values for each ingredient.
    - Fields:
        - `Ingredient_ID` (`INT UNSIGNED`): Ingredient ID
        - `Nutrient_ID` (`SMALLINT UNSIGNED`): Nutrient ID
        - `Value` (`FLOAT UNSIGNED`): Value of the nutrient (i.e. amount of the nutrient per 100g of the ingredient)
    - Primary Key: `Ingredient_ID`, `Nutrient_ID`
    - Foreign Keys:
        - `Ingredient_ID` references `Ingredients(ID)`
        - `Nutrient_ID` references `Nutrients(ID)`
    - Indexes: `Ingredient_ID`, `Nutrient_ID`
7. **FB_Ingredients**
    - Mapping between food and beverage items and the ingredients they contain.
    - Fields:
        - `FB_ID` (`INT UNSIGNED`): Food/Beverage ID
        - `Ingredient_ID` (`INT UNSIGNED`): Ingredient ID
        - `Weight` (`FLOAT UNSIGNED`): Weight of the ingredient in the food/beverage item
    - Primary Key: `FB_ID`, `Ingredient_ID`, `Weight`
    - Foreign Keys:
        - `FB_ID` references `Food_Beverages(ID)`
        - `Ingredient_ID` references `Ingredients(ID)`
    - Indexes: `FB_ID`, `Ingredient_ID`

8. **FB_Values**
    - Mapping between food and beverage items and their nutrient values.
    - Fields:
        - `FB_ID` (`INT UNSIGNED`): Food/Beverage ID
        - `Nutrient_ID` (`SMALLINT UNSIGNED`): Nutrient ID
        - `Value` (`FLOAT UNSIGNED`): Value of the nutrient (i.e. amount of the nutrient per 100g of the food/beverage item)
    - Primary Key: `FB_ID`, `Nutrient_ID`
    - Foreign Keys:
        - `FB_ID` references `Food_Beverages(ID)`
        - `Nutrient_ID` references `Nutrients(ID)`
    - Indexes: `FB_ID`, `Nutrient_ID`
9. **RDI**
    - Recommended Daily Intake values per nutrient.
    - Fields:
        - `Nutrient_ID` (`SMALLINT UNSIGNED`): Nutrient ID
        - `Value` (`FLOAT UNSIGNED`): Value of the nutrient (i.e. amount of the nutrient per 100g of the food/beverage item)
    - Primary Key: `Nutrient_ID`
    - Foreign Keys: `Nutrient_ID` references `Nutrients(ID)`
    - Indexes: `Nutrient_ID`
10. **Activities**
    - Physical activities and their calorie burn rates.
    - Fields:
        - `ID` (`SMALLINT UNSIGNED AUTO_INCREMENT`): Activity ID
        - `Description` (`VARCHAR(150)`): Description of the activity
        - `CaloriesPKG` (`FLOAT UNSIGNED`): Calories burned per 1kg of body weight
    - Primary Key: `ID`
    - Indexes: `ID`

User Data Tables

Tables containing user data: user information, logged meals.

Tables:

11. **User**
    - User profile information.
    - Fields:
        - `ID` (`INT UNSIGNED AUTO_INCREMENT`): User ID
        - `Email` (`VARCHAR(50)`): User email
        - `First_Name` (`VARCHAR(50)`): User first name
        - `Last_Name` (`VARCHAR(50)`): User last name
        - `Password` (`VARCHAR(50)`): User password
        - `DOB` (`DATE`): User date of birth
        - `Gender` (`ENUM("M", "F")`): User gender
        - `Weight` (`SMALLINT UNSIGNED`): User weight
    - Primary Key: `ID`
    - Indexes: `ID`
12. **User_Food_Beverages**
    - Logged meals per user.
    - Fields:
        - `User_ID` (`INT UNSIGNED`): User ID
        - `FB_ID` (`INT UNSIGNED`): Food/Beverage ID
        - `Portion_Weight` (`FLOAT UNSIGNED`): Weight of the portion size
        - `Date_Time` (`DATETIME`): Date and time of the meal
    - Primary Key: `User_ID`, `FB_ID`, `Date_Time`
    - Foreign Keys:
        - `User_ID` references `User(ID)`
        - `FB_ID` references `Food_Beverages(ID)`
    - Indexes: `User_ID`, `Date_Time`

Data Sources

Main Dataset

The main dataset is the 2021-2023 FNDDS dataset, which is given by the US Department of Agriculture and contains food and beverage data, nutritional data, etc...

The dataset is obtained from **here**, and is a collection of the following Excel files:

- `2021-2023 FNDDS At A Glance - FNDDS Ingredients.xlsx`
- `2021-2023 FNDDS At A Glance - FNDDS Nutrient Values.xlsx`
- `2021-2023 FNDDS At A Glance - Foods and Beverages.xlsx`
- `2021-2023 FNDDS At A Glance - Ingredient Nutrient Values.xlsx`
- `2021-2023 FNDDS At A Glance - Portions and Weights.xlsx`

Additional Datasets

- RDI values tables, given by the US Food and Drug Administration (FDA) and obtained from **here** and **here**. The tables were parsed manually to create the `RDI` table.
- Exercise dataset found in `Data/exercise_dataset.csv` and obtained from **here**.
- OpenFoodFacts dataset compressed using `xz` and found in `Data/en.openfoodfacts.org.products.tsv.xz` and obtained from **here**.

Data Migration Scripts

Main Data Processing Scripts

In the `Migration/` folder, you can find the following scripts:

- `Category.py` – Processes the `2021-2023 FNDDS At A Glance - Foods and Beverages.xlsx` file and creates the `Category` table.
- `FB_Ingredients.py` – Processes the `2021-2023 FNDDS At A Glance - FNDDS Ingredients.xlsx` file and creates the `FB_Ingredients` table.
- `FB_Values.py` – Processes the `2021-2023 FNDDS At A Glance - FNDDS Nutrient Values.xlsx` file and creates the `FB_Values` table.
- `Food_Beverages.py` – Processes the `2021-2023 FNDDS At A Glance - Foods and Beverages.xlsx` file and creates a part of the `Food_Beverages` table.
- `Ingredients.py` – Processes the `2021-2023 FNDDS At A Glance - FNDDS Ingredients.xlsx` file and creates the `Ingredients` table.
- `Ingredients_Values.py` – Processes the `2021-2023 FNDDS At A Glance - Ingredient Nutrient Values.xlsx` file and creates the `Ingredients_Values` table.

- `Nutrients.py` – Processes the `2021-2023 FNDDS At A Glance - FNDDS Nutrient Values.xlsx` file and creates a part of the `Nutrients` table.
- `Portions_Weights.py` – Processes the `2021-2023 FNDDS At A Glance - Portions and Weights.xlsx` file and creates the `Portions_Weights` table.

Additional Data Processing Scripts

In the `Migration/` folder, you can find the following files:

- `Activities.py` – Processes exercise dataset found in `Data/exercise_dataset.csv` and creates the `Activities` table.
- `Nutrient_Importance.ipynb` – Processes OpenFoodFacts dataset found in `Data/en.openfoodfacts.org.products.tsv.xz` and modifies the `Nutrients` and `Food_Beverages` tables to include the nutrient importance values and nutritional scores.

Synthetic Data Generation

In the `Synth/` folder, you can find the following files:

- `User.py` – Generates synthetic user profiles and outputs them to a CSV file.
- `User_Food_Beverages.py` – Generates synthetic food consumption records and outputs them to a CSV file.

Database Initialisation

The initialisation process follows these steps:

13. Create schema `db04` with appropriate character encoding (utf8_bin)
14. Run table creation script from `SQL/Initialisation.sql`
15. Run synthetic data generation scripts from `Synth/`
16. Run data insertion scripts from `SQL/Insertion/`
    1. Data insertion script
    2. Synthetic data insertion script

SQL Queries Documentation

Food & Beverage Queries

Search Foods

```sql
SELECT *
FROM Food_Beverages
WHERE LOWER(Description) LIKE %s
```

Searches for food items where the description contains the given search term. Before the query is executed, the search term is converted to lowercase to ensure case-insensitive matching.

Search Food Category

```
SELECT Category.Description
FROM Category, Food_Beverages
WHERE Food_Beverages.ID = %s
AND Food_Beverages.Category_ID = Category.ID
```

Retrieves the category description for a specific food item.

Search Food Name

```
SELECT Description
FROM Food_Beverages
WHERE Food_Beverages.ID = %s
```

Retrieves the description of a specific food item.

Calculate Food Calories

```
SELECT (%s * FB_Values.Value / 100) AS Value
FROM FB_Values
JOIN Nutrients ON FB_Values.Nutrient_ID = Nutrients.ID
WHERE FB_Values.FB_ID = %s AND Nutrients.Name = 'Energy'
```

Calculates calories for a given food weight (first parameter) and food ID (second parameter).

Get Food Portions

```
SELECT Description, Weight
FROM Portions_Weights
WHERE FB_ID = %s
```

Retrieves standardised portion sizes for a specific food item.

Get Food Ingredients

```
SELECT i.ID, i.Description,
       %s * fb.Weight / (SELECT SUM(Weight)
                         FROM FB_Ingredients
                         WHERE FB_ID = fb.FB_ID) AS Normalized_Weight
FROM FB_Ingredients AS fb
JOIN Ingredients AS i ON fb.Ingredient_ID = i.ID
WHERE fb.FB_ID = %s
ORDER BY fb.Weight DESC
```

Retrieves ingredients and their normalised weights for a given food item.

Get RDI Percentages

```
SELECT
    Nutrients.Name,
    %s * (FB_Values.Value / 100) / RDI.Value AS Percentage
FROM
    FB_Values
    JOIN Nutrients ON FB_Values.Nutrient_ID = Nutrients.ID
    JOIN RDI ON Nutrients.ID = RDI.Nutrient_ID
WHERE
    FB_ID = %s AND RDI.Value > 0
ORDER BY
    FIELD(Nutrients.Unit, 'kcal', 'g', 'mg', 'mcg') ASC,
    Percentage DESC
```

Calculates the percentage of Recommended Daily Intake (RDI) for each nutrient in a food item.

Get Nutrient Distribution

```
SELECT
    cte.Nutrient_Name,
    (
        cte.total_std_amount
            / SUM(cte.total_std_amount)
            OVER (PARTITION BY cte.FB_ID)
    ) as Percentage
FROM
(
    SELECT
        fb.FB_ID,
        iv.Nutrient_ID,
        n.Name AS Nutrient_Name,
        n.Unit,
        SUM(
            CASE n.Unit
                WHEN 'g'   THEN fb.Weight * iv.Value * 1000
                WHEN 'mg'  THEN fb.Weight * iv.Value
                WHEN 'mcg' THEN fb.Weight * iv.Value / 1000
                ELSE 0
            END
        ) AS total_std_amount
    FROM FB_Ingredients AS fb
    JOIN Ingredients_Values AS iv
        ON fb.Ingredient_ID = iv.Ingredient_ID
    JOIN Nutrients AS n
        ON iv.Nutrient_ID = n.ID
    WHERE fb.FB_ID = %s
    AND n.Unit <> 'kcal'
    GROUP BY
        fb.FB_ID,
        iv.Nutrient_ID,
        n.Unit
) AS cte
ORDER BY
    cte.total_std_amount DESC
LIMIT %s
```

Calculates the percentage distribution of nutrients in a food item, standardising units for comparison.

Search Nutrients Per Ingredient

```
SELECT
    Nutrients.Name,
    (Value * %s / 100) AS Value,
    Unit
FROM
    FB_Values JOIN Nutrients
        ON FB_Values.Nutrient_ID = Nutrients.ID
WHERE
    FB_ID = %s
ORDER BY
    FIELD(Nutrients.Unit, 'kcal', 'g', 'mg', 'mcg') ASC,
    Value DESC
```

Retrieves nutrient values for a given food item by weight.

Get Nutrition Score

```
SELECT Score * 100
FROM Food_Beverages as fb
WHERE fb.ID = %s
```

Retrieves the nutrition score for a given food item.

User Queries

Create User

```
INSERT INTO User (Email, First_Name, Last_Name, Password, DOB, Gender, Weight
)
VALUES (%s, %s, %s, %s, %s, %s, %s)
```

Creates a new user record.

Get User

```
SELECT * FROM User WHERE Email = %s AND Password = %s
```

Retrieves user information with authentication.

Get User Weight

```
SELECT Weight FROM User WHERE ID = %s
```

Retrieves user weight.

Get Full Name

```
SELECT First_Name, Last_Name FROM User WHERE ID = %s
```

Retrieves user full name.

Update User

```
UPDATE User
SET First_Name = %s, Last_Name = %s, Weight = %s
WHERE ID = %s
```

Updates user profile information.

Log Meal

```
INSERT INTO User_Food_Beverages (User_ID, FB_ID, Portion_Weight, Date_Time)
VALUES (%s, %s, %s, %s)
```

Records a meal consumed by a user.

Get User Meals History

```
SELECT FB.description, User_Food_Beverages.Portion_Weight, User_Food_Beverage
s.Date_Time
FROM User_Food_Beverages
JOIN Food_Beverages FB ON User_Food_Beverages.FB_ID = FB.ID
WHERE User_ID = %s
ORDER BY Date_Time DESC
```

Retrieves a user's meal history, ordered by date and time.

Calculate User Nutrition Score

```
SELECT AVG(FB.Score * UFB.Portion_Weight) / AVG(UFB.Portion_Weight) AS Score
FROM User_Food_Beverages UFB
JOIN Food_Beverages FB ON UFB.FB_ID = FB.ID
WHERE UFB.User_ID = %s
```

Calculates a user's nutrition score based on their meal history.

Calculate Relative Nutrition Score

```
WITH User_Scores AS (
    SELECT UFB.User_ID, AVG(FB.Score * UFB.Portion_Weight) / AVG(UFB.Portion_
Weight) AS Score
    FROM User_Food_Beverages UFB
    JOIN Food_Beverages FB ON UFB.FB_ID = FB.ID
    GROUP BY UFB.User_ID
),
User_Relative_Scores AS (
    SELECT User_ID, Score, PERCENT_RANK() OVER (ORDER BY Score ASC) AS Relati
ve_Score
    FROM User_Scores
)
SELECT Relative_Score
FROM User_Relative_Scores
WHERE User_ID = %s
```

Calculates a user's nutrition score percentile rank compared to the score of other users.

Activity Queries

Search Activities by Calories and Weight

```
SELECT ID, Description, (CaloriesPKG * %s) AS Calories_Burned
FROM Activities
ORDER BY ABS(CaloriesPKG * %s - %s) ASC
LIMIT %s
```

Finds the activities that burn the closest amount of calories to the user's provided calorie target, adjusted for user weight.

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

17. 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
18. 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:

19. **Token Generation**
    - Created upon successful login
    - Contains user email as identity
    - Configurable expiration time
20. **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

21. Install dependencies
22. Initialise database
23. 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

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

Food & Beverages (FB) Components

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

30. **CalorieBurnTime.js**
    - Displays activities that burn calories according to user's weight and target calories
31. **CaloriesChart.js**
    - Shows calorie breakdown by nutrient categories
    - Displays total calories in center of chart
32. **FBInfo.js**
    - Displays food/beverage information in the header
    - Shows category and name of the item
33. **FoodAndBeveragesList.js**
    - Displays search results for foods
    - Shows "No Matching Results" when empty
    - Navigates to detailed view on click
34. **FoodAndBeveragesPage.js**
    - Contains three tabs: Information, Nutrition Results, Health Score
    - Manages portion size state
35. **HealthScore.js**
    - Displays food health score (0-100)
36. **MealIngredients.js**
    - Shows ingredient breakdown table
    - Updates based on portion size changes
37. **MealValues.js**
    - Displays nutritional values table: value and unit
38. **PortionPicker.js**
    - Allows selection of predefined portions
    - Supports custom portion input
39. **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