Politechnika
Wrocławska

# Sprawozdanie Aplikacja Webowa w języku C#


Autorzy:
Jakub Gil
Kacper Zomerfeld
Numery indeksów:
263455, 264314

Prowadzący:
dr inż. Aneta Górniak


28 kwietnia 2024

# Spis treści

# 1 Wstęp

Aplikacja webowa stworzona przez naszą grupę realizuję zadanie tzw. kalkulatora kalorii. Służy ona do obliczania kilokalorii spożytych w ciągu dnia, rejestrowania danych dot. spożytych produktów w specjalnym dzienniku oraz śledzenia swoich postępów w uzupełnianiu dziennego zapotrzebowania makroskładników.

## 1.1 Funkcje jakie realizuje aplikacja

- Rejestracja nowego użytkownika, obliczanie jego zapotrzebowania kalorycznego oraz preferowanych ilości makroskładniki.
- Dodawanie/usuwanie własnych produktów z bazy danych
- Dodawanie/usuwanie produktów stworzonych przez zewnętrzne API na podstawie tekstu podanego przez użytkownika
- Dodawanie/usunięcie/zmiana ilości produktu w posiłku w ciągu dnia (zakładamy trzy typy posiłków: śniadanie, obiad oraz kolację)

## 1.2 Wykorzystane technologie

Interfejs użytkownika został napisany przy pomocy frameworku React.js w połączeniu z MUI. Obsługa bazy danych odbywa się przy pomocy Entity Framework a wszystkie operacje backend'owe są realizowane przy pomocy języka C#. W projekcie wykorzystujemy bazę danych PostgreSql.

## 1.3 Uruchomienie oraz dodatki

Instrukcje dotyczące uruchomienia znajdują się w pliku Readme.md. Oprócz samej aplikacji projekt zawiera również:

- Dokumentację stworzoną przy pomocy swaggera (pod endpoint'em ../swagger-ui/index.html)
- Dokumentację w docfx
- Testy jednostkowe oraz integracyjne

# 2 Interfejs użytkownika

Strona została podzielona na kilka najważniejszych widoków dzięki czemu jej obsługa staje się o wiele łatwiejsza. Na każdej infografice czerwonymi okręgami zaznaczono najbardziej istotne funkcjonalności.

## 2.1 Błąd pobrania danych użytkownika

Jeżeli sesja backendowa nie działa poprawnie wyświetlony zostaje komunikat:


Rys. 1: Komunikat o błędzie

Strona w pięciosekundowych interwałach próbuje ponownie połączyć się z bazą danych.

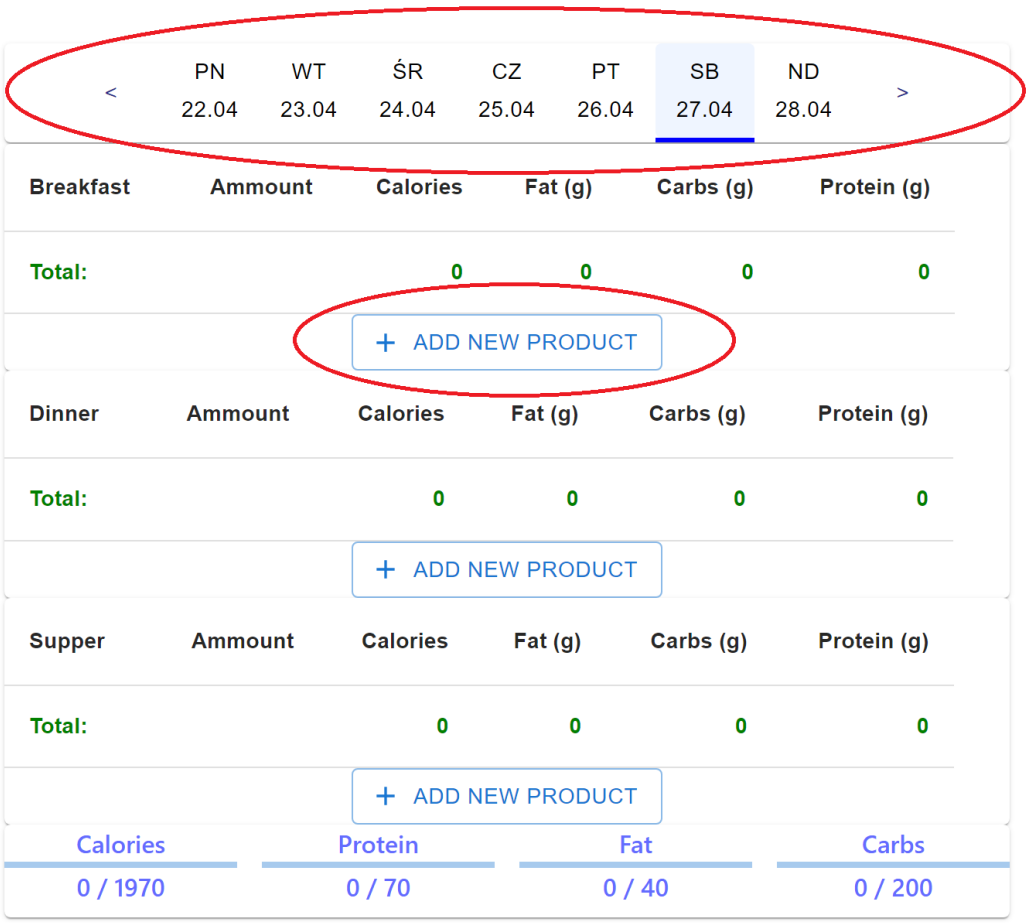
## 2.2 Rejestracja

Jeżeli użytkownik pierwszy raz używa tej strony (nie ma go jeszcze w bazie danych) w jego widoku pojawi się formularz rejestracyjny:


Rys. 2: Formularz rejestracyjny

Użytkownik musi podać niezbędne dane a następnie je zaakceptować kliknąć "Zarejestruj" aby utworzyć swoje konto.

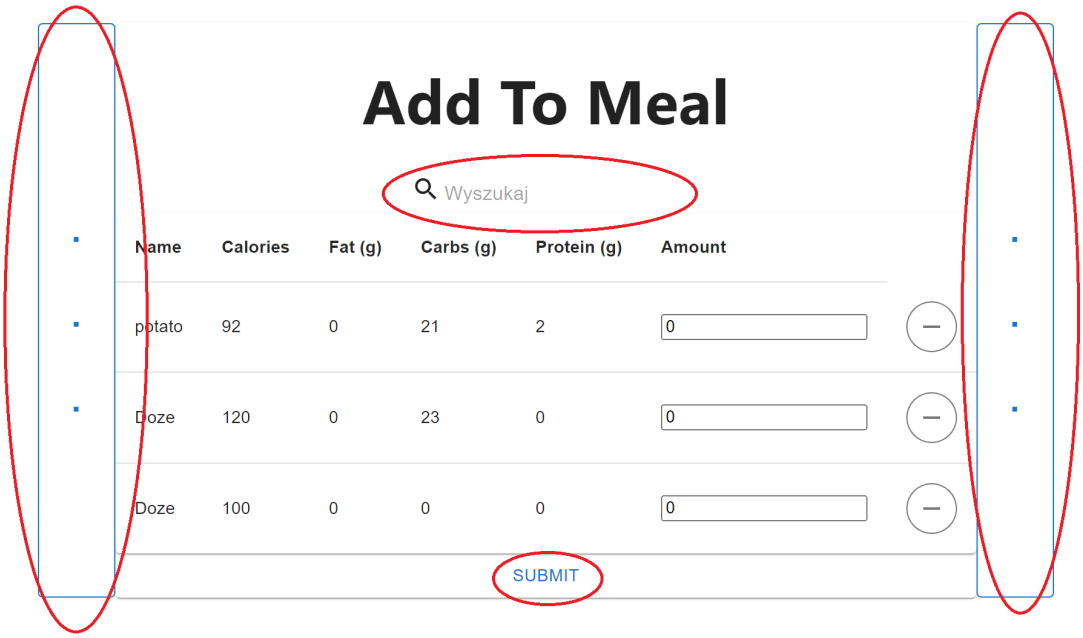
## 2.3 Strona główna




Rys. 3: Główny widok po zarejestrowaniu

Po zarejestrowaniu użytkownik otrzymuje powyższy widok. Może on zmienić dzień tygodnia (dzięki czemu może planować posiłki na kolejne dni lub zobaczyć co zjadł kilka dni temu). Użycie przycisku "Add new product" przeniesie nas do następnej strony na której będziemy mogli dodać produkty do naszego posiłku. Na samym dole strony wyświetlone są parametry dot. konkretnego dnia. Tzn ile kalorii, białka, tłuszczu i węglowodanów zostało już spożyte. Takowe podsumowanie widnieje też pod każdym posiłkiem w zakładce "Total"

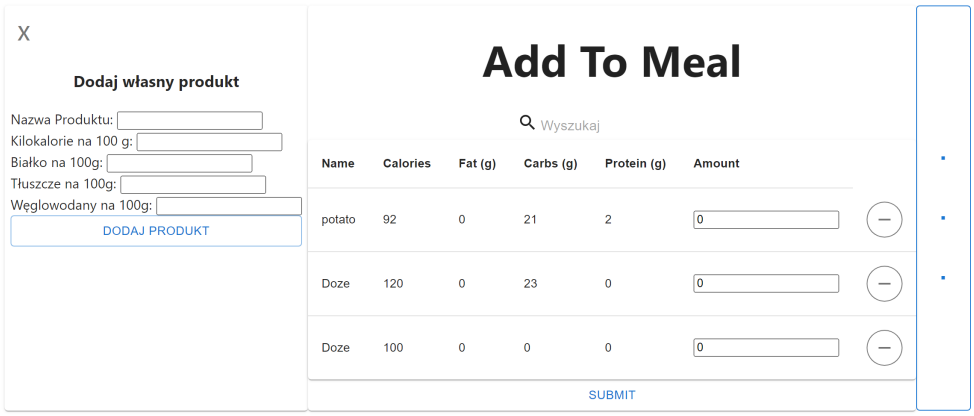
## 2.4 Dodawanie produktów do posiłku




Rys. 4: Panel dodawania posiłku

Widok ten zawiera wyszukiwarkę, pole do dodawania posiłków, przycisk "Submit" służący do dodawania produktów do posiłku. Wszystkie produkty których Ammout jest większy od 0 zostaną dodane do posiłku. Po prawej stronie od każdego produktu widnieje przycisk "-" USUWA ON PRODUKT Z BAZY DANYCH. Panel po lewej stronie zawiera formularz do dodawania nowego produktu oraz przycisk do zatwierdzania. Panel po prawej stronie posiada miejsce do wpisania nazw produktów (po angielsku). Można w nim dodać produkty korzystając z API. Z obu zakładek można wyjść wciskając "X" w lewym/prawym górym rogu.

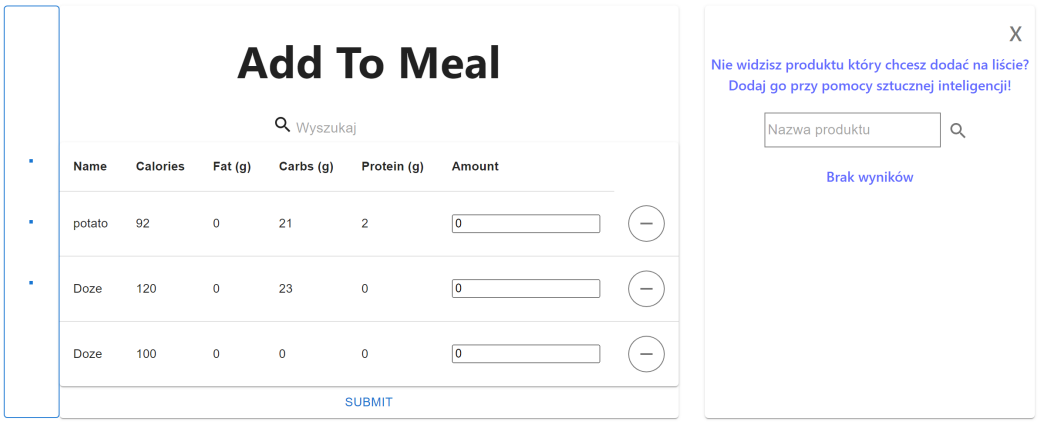
### 2.4.1 Dodawanie produktów do bazy danych przy pomocy formularza



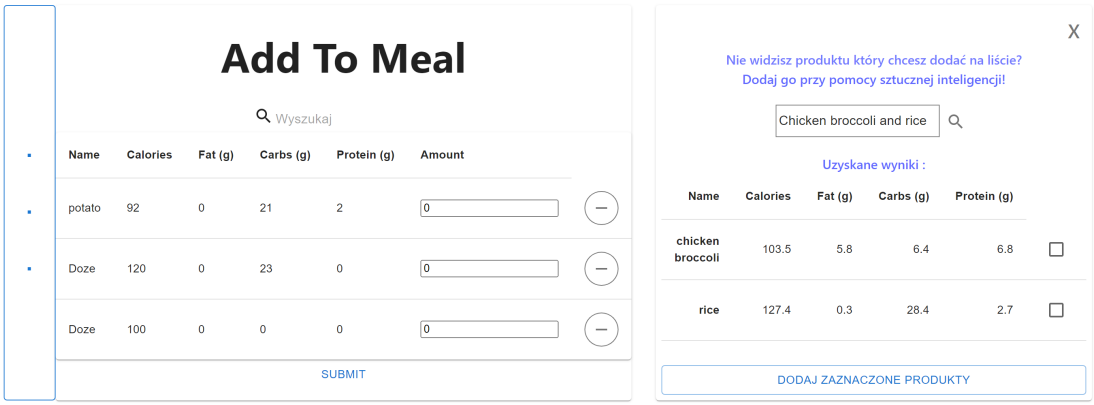

Rys. 5: Panel lewy, dodawanie ręczne

### 2.4.2 Dodawanie produktów do bazy danych przy pomocy zewnętrznego API

Aby dodać produkt używając zewnętrzengo API należy ustawić zmienną środowiskową o nazwie: APIKEY na wartość "L2WD+ooD3s7t5QPcO7NqDg==jfhdVSZtAuUcuCyT". Ewentualnie wygenerować własny klucz API na stronie:https://calorieninjas.com/




Rys. 6: Panel prawy przed podaniem nazwy




Rys. 7: Panel prawy po wpisaniu wartości

Dzięki kwadradom po prawej stronie możemy dodać tylko te produkty możemy dodać tylko te produkty które zostały poprawnie wyszukane.

# 3 Połączenie z bazą danych i logika biznesowa

## 3.1 Konfiguracje Aplikacji

Połączenie z bazą danych jest realizowane przy pomocy connection string. Jest on niezbędny, ponieważ dostarcza wszystkie informacje potrzebne do nawiązania połączenia z bazą danych. Bez niego aplikacja nie wie, jak i gdzie się połączyć. Jest to jak adres do miejsca, do którego chcemy dotrzeć, razem z informacjami potrzebnymi do uwierzytelnienia się na miejscu.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "CaloriesCounterAPIContext": "server=localhost;username=admin;password=admin;database=CaloriesCounterAPIContext
  }
}
```

Rys. 8: Konfiguracja ConnectionStringa do bazy danych

DbContext jest kluczową częścią Entity Framework (Core), reprezentującą sesję z bazą danych. To w nim definiujemy, jakie modele będą używane w aplikacji, jakie relacje istnieją między nimi, oraz gdzie ma znajdować się baza danych, do której się łączymy.

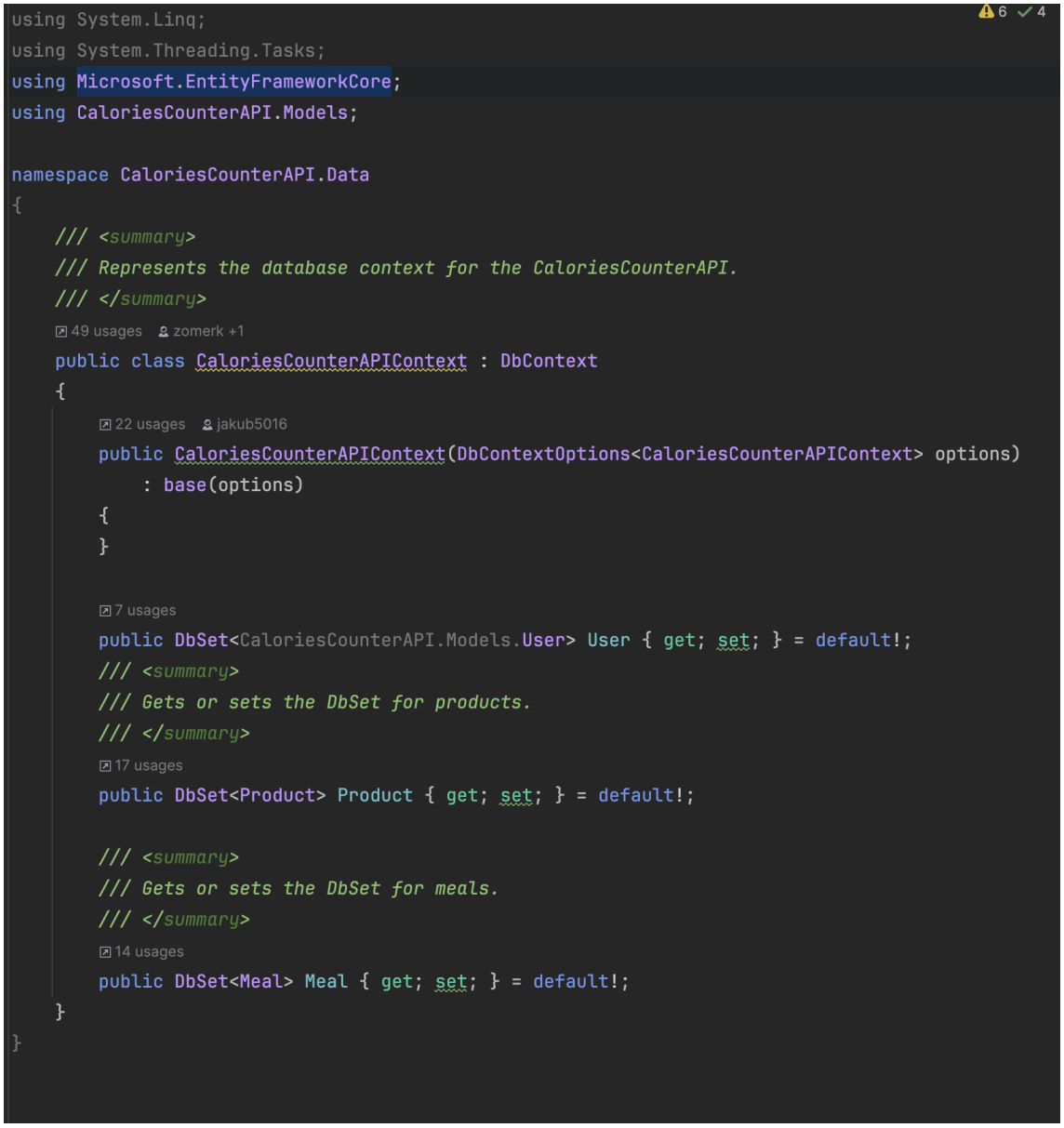
DbContext:

1. Reprezentuje sesję z bazą danych: Jest jak brama do interakcji z bazą danych.

2. Zarządza połączeniem, konfiguracją i śledzi zmiany w modelach.

3. Zapewnia dostęp do obiektów bazy danych: Umożliwia wykonywanie operacji CRUD (Create, Read, Update, Delete) na modelach.

4. Ułatwia konfigurację i migracje: DbContext pozwala na definiowanie konfiguracji baz danych i zarządzanie migracjami, co ułatwia utrzymanie spójności schematów bazy danych.




Rys. 9: Konfiguracja DbContext

## 3.2 Modele baz danych

### 3.2.1 Meal

Ta klasa to model reprezentujący posiłek który składa się z produktów. Obejmuje różne właściwości, które opisują posiłek, takie jak jego unikalny identyfikator, typ, data, produkty, które zawiera, oraz wartości odżywcze, takie jak kalorie, tłuszcz, węglowodany i białka. Zawiera również metodę do obliczania wartości odżywczych na podstawie zawartych w niej produktów.

Omówienie struktury klasy Klasa Meal składa się z kilku kluczowych części:

1. **Id**: Unikalny identyfikator posiłku. Jest to wartość, która pozwala na jednoznaczną identyfikację posiłku w bazie danych lub w systemie.

2. **Type**: Rodzaj posiłku, reprezentowany przez enum MealType. Może to być śniadanie, obiad, lub kolacja.

3. **Date**: Data posiłku. Używa typu DateOnly, co przechowuje tylko datę bez informacji o czasie.

4. **Products**: Lista produktów zawartych w posiłku. Używa typu ICollection, co oznacza, że może zawierać różną liczbę produktów.

5. **AmountOfProduct**: Lista ilości każdego produktu w posiłku. Jest to lista wartości liczbowych, które określają ilości produktów w gramach.

6. **KcalForMeal, FatForMeal, CarbsForMeal, ProteinForMeal**: To właściwości do przechowywania wartości odżywczych posiłku, takich jak kalorie, tłuszcz, węglowodany i białka.

7. **Metoda CalculateKcalForMeal** Ta metoda oblicza wartości odżywcze posiłku na podstawie listy produktów i ich ilości. Rozpoczyna od zresetowania wartości odżywczych do zera. Następnie sprawdza, czy lista produktów i lista ilości nie są puste oraz czy mają równą liczbę elementów. Jeśli te warunki są spełnione, metoda przechodzi przez każdy produkt i oblicza wartości odżywcze w oparciu o ilości. Jeśli wszystko pójdzie dobrze, zwraca true; w przeciwnym razie zwraca false.

### 3.2.2 Product

Klasa Product reprezentuje produkt. Jest to model, który zawiera informacje o pojedynczym produkcie, takie jak jego unikalny identyfikator, nazwa oraz zawartość kalorii, tłuszczu, węglowodanów i białka. Dodatkowo, klasa zawiera kolekcję posiłków, w których dany produkt jest używany.
Omówienie struktury klasy Klasa Product składa się z kilku kluczowych części:

1. **ID**: Unikalny identyfikator produktu. To klucz podstawowy (primary key), który pozwala na jednoznaczną identyfikację produktu w bazie danych.

2. **Name**: Nazwa produktu. Jest to ciąg znaków, który określa nazwę produktu, np. "Banana", "Apple", czy "Chicken". Nazwa produktu musi być unikalna.

3. **Kcal**: Zawartość kalorii w produkcie. Wartość wyrażona jako liczba całkowita, która określa, ile kalorii ma produkt w 100 gramach.

4. **Fat**: Zawartość tłuszczu w produkcie. Wartość wyrażona jako liczba całkowita, która określa, ile kalorii ma produkt w 100 gramach.

5. **Carbs**: Zawartość węglowodanów w produkcie. Wartość wyrażona jako liczba całkowita, która określa, ile kalorii ma produkt w 100 gramach..

6. **Protein**: Zawartość białka w produkcie. Wartość wyrażona jako liczba całkowita, która określa, ile kalorii ma produkt w 100 gramach.

### 3.2.3 User

Klasa User w naszej aplikacji pozwala na określeie zapotrzebowania kalorycznego jest to klasa tylko poglądaowa jest ona tworzona automatycznie, nie ma możliwości dodania swoich parametrów. Zawiera różne właściwości opisujące użytkownika, takie jak identyfikator, waga, wzrost, płeć, wiek, a także wyliczane wartości, takie jak wskaźnik metabolizmu podstawowego (BMR) oraz poziom kaloryczny utrzymania (calorie maintenance level).
Omówienie struktury klasy Klasa User składa się z kilku kluczowych części:

1. **Id**: Unikalny identyfikator użytkownika.

2. **Weight**: Waga użytkownika, wyrażona w kilogramach.

3. **Gender**: Płeć użytkownika. Użycie typu bool wskazuje na prostą reprezentację płci (1 dla mężczyzn, 0 dla kobiet).

4. **Height**: Wzrost użytkownika, podawany w centymetrach.

5. **Age**: Wiek użytkownika, jako liczba całkowita.

6. **BMR**: Wskaźnik metabolizmu podstawowego (Basal Metabolic Rate), który określa ilość kalorii potrzebną do utrzymania podstawowych funkcji życiowych.

7. **KcalMaintainces**: Poziom kaloryczny utrzymania (calorie maintenance level), czyli ilość kalorii potrzebna do utrzymania aktualnej wagi.

8. **Metoda calculateMaintainceAndBmi** BMR jest podstawową miarą metabolizmu użytkownika, wyrażającą ilość kalorii potrzebną do utrzymania podstawowych funkcji ciała w stanie spoczynku.

## 3.3 Kontrolery

Kontroler to komponent odpowiedzialny za obsługę żądań HTTP, interakcję z modelem (danymi i logiką biznesową), oraz wybór odpowiedniej odpowiedzi do zwrócenia klientowi (frontendowi). Kontroler działa jako pośrednik między klientem (na przykład aplikacją webową) a serwerem, zarządzając logiką i przepływem danych. W naszym programie posiadamy 3 kontrollery

1. **MealsController** posiada on nastepujące endpointy:
   (a) **GET /api/Meals**, pobiera wszystkie posiłki z bazy danych.
   (b) **GET /api/Meals/Date**, pobiera posiłki na podstawie daty.
   (c) **PUT /api/Meals/id**, modyfikuje posiłek.
   (d) **POST /api/Meals**, tworzy nowy posiłek.
   (e) **DELETE /api/Meals/id**, usuwa posiłek.
   (f) **DELETE /api/Meals/mealId/Products/productId**, usuwa produkt z posiłku.
   (g) **PATCH /api/Meals/id/AppendProduct/productId/quantity**, dodaje produkt do posiłku.
2. ProductController posiada on nastepujące endpointy:
   (a) **GET /api/Product**, pobiera wszystkie produkty z bazy danych.
   (b) **GET /api/Product/id**, pobiera produkt na podstawie identyfikatora.
   (c) **POST /api/Product**, dodaje nowy produkt do bazy danych.
   (d) **POST /api/Product/search**, wyszukuje produkty na podstawie ciągu znaków w nazwie.
   (e) **DELETE /api/Product/id**, usuwa produkt na podstawie identyfikatora.
3. UserController
   (a) **GET /api/User**, Pobiera jednego użytkownika z bazy danych.
   (b) **POST /api/User**, Tworzy nowego użytkownika.
   (c) **DELETE /api/User/id**, Usuwa użytkownika na podstawie identyfikatora.
4. NutritionAPi
   - **GET /api/NutritionAPI/nutrition** ,pobiera informacje o wartości odżywczej na podstawie zapytania z zewnetrzengo API.