

**POLITECHNIKA POZNAŃSKA**  
**WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI**  
Instytut Informatyki

**Patryk Kustoń, Łukasz Kopicki**  
**PROJEKT**

**Projekt oprogramowania dla portalu internetowego –  
aplikacja optymalizująca czas robienia zakupów**

## Spis treści

Wstęp .....	3
1. Wprowadzenie .....	5
1.1. Przedstawienie problemu.....	5
1.2. Opis podobnej aplikacji .....	5
1.3. Założenia projektowe .....	6
1.4. Wymagania funkcjonalne .....	6
1.5. Wymagania нефункционалне .....	7
2. Koncepcja .....	8
2.1. Języki programowania .....	8
2.2. Biblioteki, frameworki i protokoły komunikacyjne .....	8
2.3. Wzorce projektowe.....	8
2.4. Narzędzia programistyczne .....	8
2.5. Główne elementy architektury aplikacji .....	9
3. Implementacja .....	10
3.1. Struktura aplikacji .....	10
3.2. Tworzenie bazy danych.....	16
3.3. Funkcje umożliwiające komunikacje (fragmenty kodu) .....	16
4. Testy funkcjonalne .....	28
5. Instrukcje użytkownika .....	54
6. Podsumowanie .....	69
Bibliografia .....	70

## Wstęp

Celem projektu jest zaimplementowanie aplikacji internetowej, która będzie możliwa do używania przez przeglądarkę WWW, a jednocześnie przyjazna i łatwa do wykorzystania przez użytkownika. Finalnie powstanie kompletny i gotowy do wykorzystania produkt. Zakres aplikacji dotyczy utworzenia warstwy wizualnej (front-end) oraz logicznej, przechowującej dane (back-end) przy wykorzystaniu wybranych języków oprogramowania, narzędzi programistycznych itd.

Aplikacja ma być docelowo wykorzystywana przez użytkowników, którzy chcą na podstawie swojego planu żywieniowego, wygenerować listę zakupów. Jest to aplikacja dla osób, które odpowiednio planują zakupy na poszczególne dni – pomaga to zdecydowanie lepiej zorganizować sobie czas spędzony na zakupach przez przykładowo grupowanie poszczególnych rodzajów produktów, np. nabiał, wędliny, warzywa itd.

Najważniejszą funkcją aplikacji jest wygenerowanie listy zakupów. Na tę funkcjonalność składa się szereg innych metod niezbędnych do wykonania:

- dodawanie produktów,
- dodawanie dań,
- dodawanie posiłków,
- dodawanie dni dietetycznych.

Główne założenie projektu to zaprojektowanie i zaimplementowanie aplikacji internetowej przy użyciu odpowiednich narzędzi programistycznych.

Głównym wymaganiem projektu jest możliwość dalszej rozszerzalności funkcjonalnej skonstruowanego oprogramowania.

Dokumentacja aplikacji została podzielona na sześć głównych rozdziałów:

1. Wprowadzenie oraz założenia projektowe – opis rozwiązywanego problemu wraz z określeniem klasyfikacji oprogramowania oraz przedmiotu przetwarzania. W tym rozdziale został również zamieszczony opis podobnego oprogramowania „AnyList”. W założeniach projektowych przedstawiono cel aplikacji wraz z opisem wymagań oprogramowania. Wymagania zostały podzielone odpowiednio na funkcjonalne i niefunkcjonalne.
2. Koncepcja – w rozdziale opisano języki oprogramowania, wzorzec projektowy, narzędzia programistyczne itd. wykorzystane podczas konstruowania oprogramowania. Zawarto również informacje o rodzajach licencji każdego z wykorzystanych języków programowania i narzędzi programistycznych. W ostatnim podrozdziale przedstawiono w formie graficznej (z krótkim opisem) główne elementy tworzonej aplikacji.
3. Implementacja – w rozdziale przedstawiono strukturę aplikacji, encję odpowiadającą tabelą w bazie danych oraz funkcję umożliwiającą komunikację. Zawarto liczne fragmenty kodu i przykłady implementacyjne.
4. Testy funkcjonalne.
5. Instrukcje użytkownika.
6. Podsumowanie.

Rozdziały zawierające wprowadzenie, założenia projektowe, koncepcję i testy były tworzone wspólnie, dlatego też bezpośrednie wskazanie jednego autora nie jest możliwe. Za rozdział

implementacja, implementacje metod aplikacji dodających elementy do bazy danych, pobieranie wartości dań i posiłków oraz widoki dań, posiłków i dni dietetycznych odpowiadał Łukasz Kopicki. Za rozdział instrukcje użytkownika oraz implementacje metod aplikacji dotyczących pobierania wartości z bazy danych wraz z ich wyświetlaniem w widokach odpowiedzialny był Patryk Kustoń.

# 1. Wprowadzenie

Aplikacja ma rozwiązać problem pojawiający się podczas tworzenia dokładnej listy zakupów produktów na poszczególne dni. Oprogramowanie ma zagwarantować możliwość wyświetlania listy zakupów po wprowadzeniu odpowiednich produktów. Te z kolei można dalej przetwarzać poprzez tworzenie na ich podstawie dań składających się na posiłki będące obligatoryjną częścią dni dietetycznych składających się finalnie na listę zakupów.

Z punktu widzenia lokalizacji oprogramowania i dostępu do niego – utworzona aplikacja będzie dostępna dla użytkownika za pomocą przeglądarki internetowej.

## 1.1. Przedstawienie problemu

Problemem jest uciążliwe poruszanie się po sklepie z powodu nieposegregowanej listy zakupów. Przykładowo dzieje się tak, gdy listy dietetyczne dostarczone przez dietetyków nie posiadają posegregowanych produktów według odpowiednich kategorii

Dane wejściowe:

- produkty,
- dania (składające się z produktów).
- posiłki (składające się z dań),
- dni dietetyczne (składające się z posiłków).

Dane wyjściowe:

- lista zakupów.

Aplikację można zaliczyć do dziedziny oprogramowania specjalistycznego z tego względu, że wspomaga funkcjonowanie użytkownika, który jest np. na określonej diecie – ułatwia korzystanie z usług dietetyka.

Oprogramowanie specjalistyczne jest grupą skupiającą aplikacje, które mają różnorodne zastosowania. Ich przykładowe przeznaczenie to [5]:

- zarządzanie projektami,
- skład tekstu (ang. *desktop publishing* - DTP),
- cele administracyjne, np. ewidencja gości hotelowych,
- zarządzanie dokumentacją medyczną,
- zaawansowana obróbka grafiki, filmów oraz muzyki.

## 1.2. Opis podobnej aplikacji

Podobną aplikacją jest aplikacja „AnyList” [6]. Jest to aplikacja, która pozwala na organizację zakupów spożywczych użytkownika. Podstawą funkcjonowania aplikacji jest tworzenie list zakupów i odpowiednie dodawanie produktów do utworzonych list. Dodatkowo istniejący kalendarz umożliwia zaplanowanie posiłków na konkretnie dni, wszystko jest ze sobą ściśle połączone i odpowiednio współgra.

Zaletami aplikacji „AnyList” są:

- intuicyjność aplikacji,
- możliwość personalizacji (wygląd),
- łatwy dostęp - potrzebny smartfon i Internet.

Wady aplikacji „AnyList” to:

- dostępność wersji komputerowej (MAC, PC) dopiero przy wykupieniu wersji „premium”,
- ograniczone funkcje przy wersji podstawowej.

### **1.3. Założenia projektowe**

Aplikacja jest przeznaczona dla osób, które chcą robić zakupy w uporządkowany i jak najszybszy sposób. Ma umożliwić dodawanie produktów, dań, posiłków, dni dietetycznych oraz list zakupowych. Utworzone listy zakupów mają posiadać podział produktów na kategorie z określoną ilością (gramy lub mililitry). Podział na kategorie ma odzwierciedlać podstawowe działy, które znajdują się w większości hipermarketów np. owoce i warzywa, mięso, nabiał.

Gramatura składników znajdująca się w listach zakupowych dotyczy części rynkowych. Części rynkowe to produkty odważone zanim zostaną poddane obróbce np. gotowaniu, obieraniu. Przykładem jest waga ziemniaków ze skórką przed ugotowaniem. Zalecane jest ważenie produktów przed spożyciem, aby nie popełnić błędów w oszacowaniu proporcji. Skład poszczególnych dań dotyczy jednej porcji, chyba że podano inaczej.

### **1.4. Wymagania funkcjonalne**

Wymagania funkcjonalne to specyficzne funkcje, które określają, co aplikacja ma osiągnąć. Oprócz tego obejmują obliczenia, szczegóły techniczne, manipulację i przetwarzanie danych [12]. Wymagania funkcjonalne:

1. Możliwość tworzenia produktów.
  2. Możliwość tworzenia dań na podstawie produktów.
  3. Sumowanie kalorii wszystkich produktów składających się na danie.
  4. Możliwość tworzenia posiłków na podstawie dań.
  5. Sumowanie kalorii wszystkich produktów składających się na posiłek.
  6. Możliwość tworzenia dodawania dni dietetycznych na podstawie posiłków.
  7. Możliwość tworzenia list zakupowych (bez ograniczeń) na podstawie dni dietetycznych.
  8. Sumowanie wag wszystkich produktów składających się na listę zakupów.
  9. Możliwość wyświetlania posegregowanych względem kategorii list zakupowych.
1. Możliwość tworzenia produktów.
    - a. Aplikacja umożliwia tworzenie produktów.
  2. Możliwość tworzenia dań na podstawie produktów.
    - a. Aplikacja umożliwia tworzenie dań z wcześniej utworzonych produktów.
  4. Możliwość tworzenia posiłków na podstawie dań.
    - a. Aplikacja umożliwia tworzenie posiłków z wcześniej utworzonych dań.
  6. Możliwość tworzenia dni dietetycznych na podstawie posiłków.
    - a. Aplikacja umożliwia tworzenie dni dietetycznych z wcześniej utworzonych posiłków.
  7. Możliwość tworzenia list zakupowych (bez ograniczeń) na podstawie dni dietetycznych.
    - a. Aplikacja umożliwia tworzenie list zakupowych bazując na wcześniej utworzonych dniach dietetycznych.
  9. Możliwość wyświetlania posegregowanych list zakupowych.

a. Aplikacja umożliwia wyświetlenie list zakupowych, posegregowanych według kategorii produktów oraz z sumowanymi wagami poszczególnych produktów.

### **1.5. Wymagania нефunkcjonalne**

Wymagania нефunkcjonalne to udokumentowane potrzeby aplikacji informatycznej dotyczące oczekiwań, jakie ona musi spełniać w zakresie jakości jej działania.

1. Aplikacja ma działać na przeglądarkach internetowych np. Google Chrome, Opera, Safari, Mozilla.

2. Aplikacja ma umożliwić wprowadzanie dowolnej ilości produktów, posiłków, dni dietetycznych oraz list zakupowych.

3. Rozszerzalność systemu.

a. Możliwość zaimplementowania nowych funkcjonalności.

## 2. Koncepcja

Rozdział poświęcony jest przedstawieniu używanych języków oprogramowania, bibliotek, „frameworków”, narzędzi programistycznych oraz wzorców projektowych, z których korzystano podczas tworzenia aplikacji.

### 2.1. Języki programowania

C# jest obiektowym językiem programowania stworzonym przez Microsoft o bardzo szerokim zakresie zastosowania [8].

HTML (ang. *HyperText Markup Language*) to hipertekstowy język znaczników[1].

Kaskadowe arkusze stylów (ang. *Cascading Style Sheets*) to język służący do opisu formy prezentacji (wyświetlania) stron WWW.

JavaScript (w skrócie JS) jest skryptowym językiem programowania, który jest wykorzystywany do interakcji (z HTML) [3].

SQL to strukturalny język zapytań używany do tworzenia, modyfikowania baz danych.

### 2.2. Biblioteki, frameworki i protokoły komunikacyjne

Entity Framework jest to narzędzie służące do mapowania obiektowo-relacyjnego ORM (Object-Relational Mapping). Generuje obiekty oraz encje zgodnie z tabelami baz danych [9]. Wykorzystano bibliotekę JQuery (język JavaScript) oraz Bootstrap (HTML, JS,CSS).

HTTP (ang. *Hypertext Transfer Protocol*) jest to protokół transportowy, który służy do przesyłania hipertekstowych dokumentów sieci WWW (ang. *World Wide Web*), wzajemnego komunikowania się komputerów. Protokół precyzuje formę żądań klienta i odpowiedzi na nie.

### 2.3. Wzorce projektowe

Model-Widok-kontroler (ang. *Model-View-Controller*) to wzorec architektoniczny, który odzwierciedla układ warstw aplikacji. MVC zakłada podział na trzy główne warstwy [7].

Model jest pewną reprezentacją problemu bądź logiki aplikacji. Definiuje elementy, ich atrybuty i powiązania między nimi.

Widok opisuje, jak wyświetlić pewną część modelu w ramach interfejsu użytkownika.

Kontroler przyjmuje dane wejściowe od użytkownika, , zarządza aktualizacjami modelu oraz odświeżeniem widoków, pośredniczy między przesyłaniem danych pomiędzy warstwami warstwami.

### 2.4. Narzędzia programistyczne

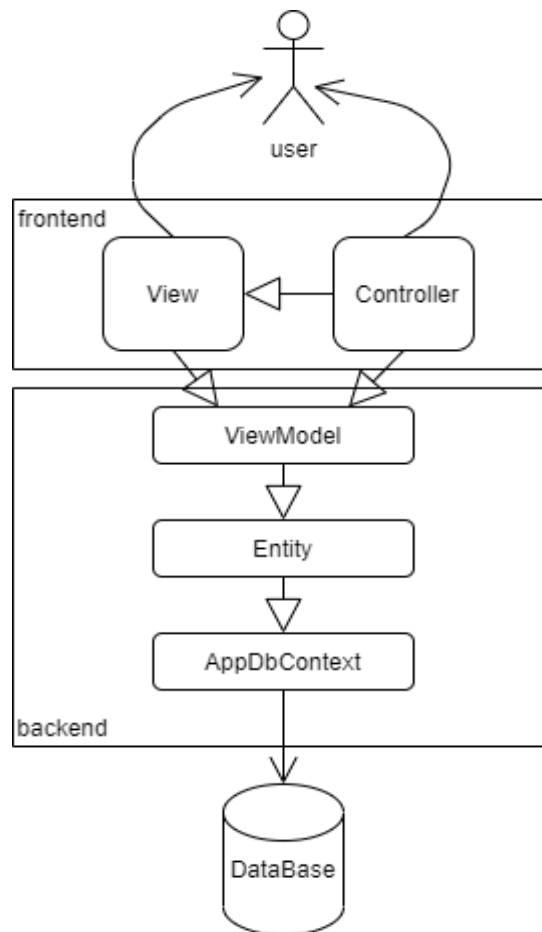
Microsoft Visual Studio jest to zintegrowane środowisko programistyczne utworzone przez firmę Microsoft, którego celem jest umożliwienie tworzenia między innymi: programów komputerowych, aplikacji i stron internetowych, aplikacji mobilnych [10]. Posiadana licencja Community (dla studentów).

ASP.NET MVC jest to platforma firmy Microsoft, wykorzystywana do tworzenia aplikacji internetowych. Bazuje na wykorzystywaniu wzorca projektowego MVC [7]. Posiadana licencja Apache License (open source).



## 2.5. Główne elementy architektury aplikacji

Na rysunku 1. zaprezentowano główne elementy architektury aplikacji.



Rysunek 1. Główne elementy architektury aplikacji.

Aplikacja wykorzystuje bazę danych - Azure SQL Database. Jest to relacyjna baza danych pracująca jako usługa w chmurze publicznej. Wykorzystuje silnik bazodanowy Microsoft SQL Server.

Kontroler jest to element, który odpowiada za akcje, jakie wykonuje użytkownik. Odbierane żądania odwzorowuje w postaci metod modelu. W kolejnym kroku przekazuje wyniki wykorzystanych metod do widoku. Celem jest również dbanie o walidację danych i bezpieczeństwo aplikacji.

Widok służy do prezentowania danych oraz reprezentacji wyjścia aplikacji. Wykorzystany w niniejszej aplikacji format wyjściowy to HTML. Widok odczytuje dane z modelu i wyświetla je na ekranie użytkownika. Służy również do tworzenia i inicjalizowania kontrolera, wyciągania danych z modelu i implementowania operacji aktualizacji.

Model zawiera dane i operacje, które nie są związane z obsługą interfejsu użytkownika. Model nie jest zależny od widoku. Odpowiedzialny jest za podstawowe funkcje aplikacji; zależności między widokami i kontrolerami są rejestrowane w widokach i przekazywane są informacje o zmianie danych.

### 3. Implementacja

W rozdziale implementacja została opisana struktura aplikacji internetowej (rysunek 1.), sposób komunikacji warstwy frontend z backend oraz sposób tworzenia bazy danych. Zostanie również zaprezentowany wygląd graficzny aplikacji.

#### 3.1. Struktura aplikacji

Do używania aplikacji niezbędna jest przeglądarka internetowa. Aplikacja jest zaprojektowana zgodnie z wzorcem projektowym Model-Widok-Kontroler (MVC), co w przyszłości pozwoli na łatwy rozwój aplikacji (zasada podstawienia Liskov). Aplikacja została podzielona na trzy oddzielne solucje: DietaApp, DietaApp.Core, DietaApp.Database.

W solucji DietaApp umieszczone są: kontrolery, modele, mapper, widoki i klasa Startup.

Kontrolery, które są odpowiedzialne za komunikację z widokami przy użyciu protokołu HTTP. Istnieje 6 kontrolerów, każdy z nich odpowiada za poszczególną funkcję aplikacji:

- DayController,
- DishController,
- HomeController,
- MealController,
- ProductController,
- ShoppingListController.

Modele zawierają dane, które nie są związane z obsługą interfejsu użytkownika. Są to struktury danych, które są wykorzystywane do przekazywania danych pomiędzy frontendem a backendem aplikacji:

- ConfirmationViewModel,
- DayViewModel,
- DaysInShoppingListViewModel,
- DaysListViewModel,
- DictionaryViewModel,
- DishListVm,
- DishViewModel,
- DishesInMealViewModel,
- MealListVm,
- MealViewModel,
- MealsInDayDto,
- ProductListVM,
- ProductViewModel,
- ProductsInDishViewModel,
- ShoppingListViewModel.

```

1.     public class ViewModelMapper
2.     {
3.         private IMapper IMapper;
4.         public ViewModelMapper()
5.         {
6.             IMapper = new MapperConfiguration(config =>
7.             {
8.                 config.CreateMap<DishDto, DishViewModel>()
9.                 .ReverseMap();
10.                config.CreateMap<DayDto, DayViewModel>()
11.                .ReverseMap();
12.                config.CreateMap<MealDto, MealViewModel>()
13.                .ReverseMap();
14.                config.CreateMap<ProductDto, ProductViewModel>()
15.                .ReverseMap();
16.                config.CreateMap<ProductsInDishDto, ProductsInDishViewModel>()
17.                .ReverseMap();
18.                config.CreateMap<DishesInMealDto, DishesInMealViewModel>()
19.                .ReverseMap();
20.                config.CreateMap<ShoppingListDto, ShoppingListViewModel>()
21.                .ReverseMap();
22.                config.CreateMap<DaysInShoppingListDto, DaysInShoppingListViewModel>()
23.                .ReverseMap();
24.                config.CreateMap<MealsInDayDto, MealsInDayViewModel>()
25.                .ReverseMap();
26.            }
27.                ).CreateMapper();
28.        }
29.        #region Product Maps
30.        public ProductViewModel Map(ProductDto product) =>
31.        IMapper.Map<ProductViewModel>(product);
32.        public List<ProductViewModel> Map(List<ProductDto> products) =>
33.        IMapper.Map<List<ProductViewModel>>(products);
34.
35.        public ProductDto Map(ProductViewModel product) =>
36.        IMapper.Map<ProductDto>(product);
37.        public List<ProductDto> Map(List<ProductViewModel> products) =>
38.        IMapper.Map<List<ProductDto>>(products);
39.        #endregion

```

*Przykład 1. Fragment kodu klasy ViewModelMapper.*

Mapper służy do zmniejszenia złożoności, którą napotykamy podczas wiązania modelu i komunikacji z encjami innych warstw. Na przykładzie 1. przedstawiono fragment kodu z mapowaniem encji warstwy DietaApp i DietaApp.Core.

Widoki służą do wyświetlania danych przy użyciu obiektu modelu. Poniżej przedstawiono listę widoków na przykładzie posiłków:

- AddMeal,
- ListOfMeals,

- ListOfProductsInMeal,
- NavigationOfMeals.

Klasa *Startup* odpowiada między innymi za połączenie z bazą danych, potok żądań aplikacji. Na przykładzie 2. przedstawiono fragment kodu przedstawiający między innymi dodanie interfejsów z innych warstw, adres serwera SQL.

```

1. public class Startup
2. {
3.     public Startup(IConfiguration configuration)
4.     {
5.         Configuration = configuration;
6.     }
7.     public IConfiguration Configuration { get; }
8.     public void ConfigureServices(IServiceCollection services)
9.     {
10.         services.AddDbContext<DietaAppDbContext>(options => options.
11. UseSqlServer("Server=tcp:dietaappserv.database.windows.net,1433;Initial" +
12. "Catalog=DietaAppDB;Persist Security Info=False;User" +
13. ID=dietaappserv;Password=u9jN6K9DgBwIBK;MultipleActiveResultSets=False;
14. Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;"));
15.         services.AddControllersWithViews();
16.         services.AddTransient<IDishRepository, DishRepository>();
17.         services.AddTransient<IProductRepository, ProductRepository>();
18.         services.AddTransient<IProductsInDishRepository,
19. ProductsInDishRepository>();
20.         services.AddTransient<IMealRepository, MealRepository>();
21.         services.AddTransient<IDishesInMealRepository, DishesInMealRepository>();
22.         services.AddTransient<IMealsInDayRepository, MealsInDayRepository>();
23.         services.AddTransient<IDayRepository, DayRepository>();
24.         services.AddTransient<IShoppingListRepository, ShoppingListRepository>();
25.         services.AddTransient<IDaysInShoppingListRepository,
26. DaysInShoppingListRepository>();
27.         services.AddTransient<DtoMapper>();
28.         services.AddTransient<ViewModelMapper>();
29.         services.AddTransient<IManager, Manager>() }

```

*Przykład 2. Fragment kodu klasy Startup.*

W solucji DietaApp.Core umieszczone są: Dto (ang. Data transfer object), klasa *Manager*, *Mapper*.

Dto (ang. Data transfer object) to struktury danych, które są wykorzystywane, aby przekazywać dane pomiędzy warstwami aplikacji. Do utworzonych struktury danych należą:

- BaseEntityDto,
- DayDto,
- DaysInShoppingListDto,
- DishesInMealDto,
- Meal2Dto,

- MealDto,
- MealsInDayDto,
- ProductsDto,
- ProductsInMealDto,
- ShoppingListDto.

Na przykładzie 3. przedstawiono strukturę *ProductsDto*.

```

1. using System;
2. using System.Collections.Generic;
3. using System.ComponentModel.DataAnnotations.Schema;
4.
5. namespace DietaApp.Core
6. {
7.     public class ProductDto : BaseEntityDto
8.     {
9.         public string Name { get; set; }
10.
11.         public int Kcal { get; set; }
12.
13.         public string Unit { get; set; }
14.
15.         public string Category { get; set; }
16.
17.         public List<ProductsInDishDto> ProductsInDish { get; set; }
18.     }
19. }

```

*Przykład 3. Struktura ProductsDto.*

Klasa *Manager* zawiera implementacje metod wykorzystywanych przez kontrolery poprzez Interfejsy (klasa *IManager*). Na przykładzie 4. przedstawiono fragment kodu klasy *Manager*.

```

1. using System;
2. using System.Collections;
3. using System.Collections.Generic;
4. using System.Linq;
5. using DietaApp.Core.Interfaces;
6. using DietaApp.Core.Mapper;
7. using DietaApp.Database;
8. using DietaApp.Database.Entities;
9.
10. namespace DietaApp.Core
11. {
12.     public class Manager : IManager
13.     {
14.         private readonly IDishRepository mDishRepository;
15.         private readonly IDayRepository mDayRepository;

```

```

16.         private readonly IProductRepository mProductRepository;
17.         private readonly IProductsInDishRepository
18. mProductsInDishRepository;
19.         private readonly IMealRepository mMealRepository;
20.         private readonly IDishesInMealRepository
21. mDishesInMealRepository;
22.         private readonly IShoppingListRepository
23. mShoppingListRepository;
24.         private readonly IMealsInDayRepository mMealsInDayRepository;
25.         private readonly IDaysInShoppingListRepository
26. mDaysInShoppingListRepository;
27.         private readonly DtoMapper mDtoMapper;
28.
29.         public Manager(
30.             IDishRepository dishRepository,
31.             IDayRepository dayRepository,
32.             IProductRepository productRepository,
33.             IProductsInDishRepository productsInDishRepository,
34.             IMealRepository mealRepository,
35.             IDishesInMealRepository dishesInMealRepository,
36.             IMealsInDayRepository mealsInDayRepository,
37.             IShoppingListRepository shoppingListRepository,
38.             IDaysInShoppingListRepository
39. daysInShoppingListRepository,
40.             DtoMapper mapper)
41.         {
42.
43.             mDtoMapper = mapper;
44.             mDishRepository = dishRepository;
45.             mDayRepository = dayRepository;
46.             mProductRepository = productRepository;
47.             mProductsInDishRepository = productsInDishRepository;
48.             mMealRepository = mealRepository;
49.             mDishesInMealRepository = dishesInMealRepository;
50.             mMealsInDayRepository = mealsInDayRepository;
51.             mDaysInShoppingListRepository =
52. daysInShoppingListRepository;
53.             mShoppingListRepository = shoppingListRepository;
54.         }
55.         public List<ProductDto> GetAllProducts(string filterString)
56.         {
57.             var productEntities =
58. mProductRepository.GetAllProducts().ToList();
59.
60.             if (!string.IsNullOrEmpty(filterString))
61.             {
62.                 productEntities = productEntities.Where(x =>
63. x.Name.Contains(filterString)).ToList();

```

```

64.         }
65.         return mDtoMapper.Map(productEntities);
66.     }

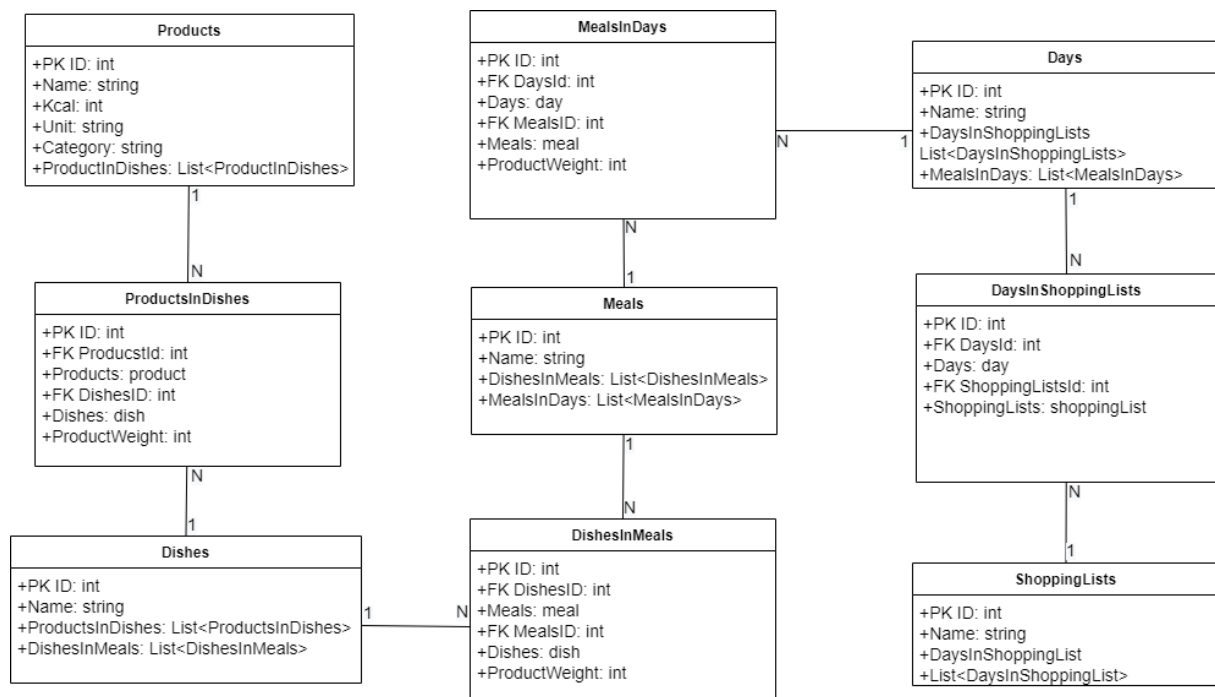
```

*Przykład 4. Fragment kodu klasy Manager.*

Mapper ułatwia komunikację pomiędzy warstwami aplikacji, a w tym konkretnym przypadku służy do komunikacji z encjami w solucji DietaApp.Database odpowiadającymi za strukturę bazy danych.

W solucji DietaApp.Database znajdują się: encje, repozytoria.

Encje przedstawione na rysunku 2. reprezentują tabele bazy danych aplikacji. Mapowanie encji do/z aplikacji odbywa się za pomocą klasy DietaAppDbContext.



*Rysunek 2. Diagram encji (bazy danych).*

Repozytoria zawierają interfejsy oraz klasy odzwierciedlające encje. W każdej z klas są zaimplementowane metody pozwalające pobrać wartości z danej encji bazy danych. Do listy utworzonych repozytoriów należą:

- DayRepository,
- DaysInShoppingListRepository,
- DishesInMealRepository,
- DishRepository,
- MealRepository,
- MealsInDayRepository,
- ProductRepository,
- ProductsInDishRepository,
- ShoppingListRepository.

Najważniejszym aspektem podczas tworzenia struktury aplikacji było wyszczególnienie pięciu elementów (produkty, dania, posiłki, dni dietetyczne, listy produktów) składających się na kompletną aplikację.

### 3.2. Tworzenie bazy danych

Baza danych użyta w aplikacji składa się z dziewięciu tabel przedstawionych na diagramie encji na rysunku 2. Przy wykorzystaniu Entity Framework tworzone są encje w bazie danych, następnie zaimplementowane w warstwie `DietApp.Database`. Aplikacja łączy się z bazą danych (Azure SQL Database) przy wykorzystaniu klasy `Startup` znajdującej się w solucji `DietApp`. Do utworzonych tabel należą:

- Days,
- DaysInShoppingLists,
- Dishes,
- DishesInMeals,
- Meals,
- MealsInDays,
- Products,
- ProductsInDishes,
- ShoppingLists.

### 3.3. Funkcje umożliwiające komunikacje (fragmenty kodu)

W solucji `DietApp` znajdują się kontrolery, które umożliwiają komunikację między warstwą frontendu i backendu. W większości kontrolerów znajdują się metody odpowiedzialne za pobieranie danych z bazy danych za pomocą [HTTP GET]. Dodawanie danych do tabel odbywa się za pomocą [HTTP POST]. Usuwanie danych z bazy danych odbywa się za pomocą metody [HTTP DELETE].

Proces odpowiedzialny za pobranie danych z bazy danych przy użyciu [HTTP GET] jest używany do wyświetlenia dodanych wcześniej wartości:

- produktów (nazwa, kcal, jednostka, kategoria),
- dań (nazwa, kcal),
- składników znajdujących się w wybranym daniu (nazwa, kcal, jednostka, kategoria),
- posiłków (nazwa, kcal),
- dań znajdujących się w wybranym posiłku (nazwa, kcal),
- dni dietetycznych (nazwa),
- posiłków znajdujących się w wybranym dniu (nazwa),
- list zakupów (nazwa),
- dni znajdujących się w wybranej liście zakupów (nazwa),
- każdej listy zakupów z osobna.

Na przykładzie 5. przedstawiono metodę `NavigationOfProducts()`, która wyświetla listę produktów. Metoda pobiera wartości wszystkich produktów z bazy danych. Z klasy `Manager` zostały wykorzystane metody dzięki „wstrzyknięciu” do konstruktora interfejsu przez DI (ang. `Dependency Injection`). Produkty zostały pobrane za pomocą metody `GetAllProduct()` z klasy `Manager`, która następnie po przypisaniu została przekazana do `ViewModelMapera`, żeby zmapować dane na `ViewModel` (w tym przypadku `ProductViewModel`).



```

1. public IActionResult NavigationOfProducts(string filterString)
2. {
3.     var productsDtos = mManager.GetAllProducts(filterString);
4.     var productViewModel = mViewModelMapper.Map(productsDtos);
5.
6.     return View(productViewModel);
7. }

```

*Przykład 5. Metoda NavigationOfProducts.*

Na przykładzie 6. przedstawiono fragment kodu widoku, który odpowiada za wyświetlanie wszystkich produktów, które zostały przekazane w ViewModelu.

```

1. <div class="text-center">
2.     <table class="table">
3.         <thead>
4.             <tr>
5.                 <th scope="col">
6.                     Id
7.                 </th>
8.                 <th scope="col">
9.                     Nazwa
10.                </th>
11.                <th scope="col">
12.                    Kcal
13.                </th>
14.                <th scope="col">
15.                    Jednostka
16.                </th>
17.                <th scope="col">
18.                    Kategoria
19.                </th>
20.                <th scope="col">
21.
22.                </th>
23.            </tr>
24.        </thead>
25.        <tbody>
26.            @foreach (var product in Model.OrderBy(i => i.Name))
27.            {
28.                <tr>
29.                    <td scope="row">@i</td>
30.                    <td>@product.Name</td>
31.                    <td>@product.Kcal</td>
32.                    <td>@product.Unit</td>
33.                    <td>@product.Category</td>
34.                    <td>
35.

```

```

36.         <a csp-area="" class="btn btn-danger" asp-controller="Product"
37. asp-action="Delete" asp-route-productId="@product.Id">Usuń</a>
38.             </td>
39.         </tr>
40.         i++;
41.     }
42. </tbody>
43. </table> </div>

```

*Przykład 6. Fragment kodu widoku NavigationOfProducts.*

Na przykładzie 7. przedstawiono metodę `ListOfProductsInDish()` służącą do wyświetlania produktów znajdujących się w wybranym daniu. Metoda używając metod zaimplementowanych w klasie `Manager`, pobiera wszystkie produkty związane z daniem. Dania są identyfikowane przez aplikację za pomocą `asp-route-dishId` (ID posiłku). Metoda wykorzystując ID dania pobiera z bazy danych odpowiednie danie oraz produkty składające się na danie korzystając z tabeli powiązań `ProductsInDish`.

```

1.  [HttpGet]
2.      public async Task<IActionResult> ListOfProductsInDish(int
3. mealId)
4.      {
5.          DishViewModel currentMeal;
6.          try
7.          {
8.              if (mealId == null)
9.              {
10.                 return NotFound();
11.             }
12.
13.             else
14.             {
15.                 var CurrentList = await _dietaAppDbContext
16.                     .Dishes
17.                     .Where(d=>d.Id==mealId)
18.                     .Include(pID => pID.ProductsInDish)
19.                     .ThenInclude(p => p.Product)
20.                     .ToListAsync();
21.
22.                 List<string> getAllCategory = new List<string>();
23.                 List<string> getAllName = new List<string>();
24.                 List<string> getAllKcal = new List<string>();
25.                 List<string> getAllUnit = new List<string>();
26.
27.                 foreach (var item in CurrentList)
28.                 {
29.                     var productsInDish = item.ProductsInDish;
30.                     foreach (var product in productsInDish)

```

```

31.         {
32.             getAllCategory.Add(product.Product.Category
33. );
34.             getAllName.Add(product.Product.Name);
35.             getAllKcal.Add(product.Product.Kcal);
36.             getAllUnit.Add(product.Product.Unit);
37.
38.         }
39.     }
40.
41.
42.     currentMeal = new DishViewModel
43.     {
44.         getCategory = getAllCategory,
45.         getName = getAllName,
46.         getKcal = getAllKcal,
47.         getUnit = getAllUnit
48.     };
49.     return View(currentMeal);
50. };
51. return View();
52. }
53. catch (NullReferenceException ex)
54. {
55.     return NotFound(ex);
56. }}

```

*Przykład 7. Metoda ListOfProductsInDish.*

Na przykładzie 8. przedstawiono fragment kodu widoku, na którym przedstawiono, jak wywołana jest metoda `NavigationOfDishes()` i przekazywane jest ID danego dania (`dishId`) jako argument.

```

1. <tbody>
2.     @for (int i = 0; i < Model.sumKcalInDish.Count(); i++)
3.     {
4.         var item = @Model.sumKcalInDish.ElementAt(i);
5.         var j = 0;
6.         <tr>
7.             <td scope="row">@counter</td>
8.             <td>@item.Key</td>
9.             <th scope="row">@item.Value</th>
10.            <td>
11.                @if (j < 1 && @Model.IdOfDish.ElementAt(i).Key != null)
12.                {
13.                    @for (int g = 0; g < j + 1; g++)
14.                    {
15.                        var valueItem = @Model.IdOfDish.ElementAt(i);

```

```

16. <a csp-area="" class="btn btn-primary" asp-controller="Dish" asp-
17. action="ListOfProductsInDish" asp-route-dishId="
18. @valueItem.Value">Szczegóły</a>
19.     @dishId = @valueItem.Value;
20. <a csp-area="" class="btn btn-danger" asp-controller="Dish" asp-
21. action="DeleteDish" asp-route-dishId=@dishId>Usuń</a>
22.         g++;
23.     }
24. }
25. </td>
26. </tr>
27.     counter++;
28. } </tbody>

```

*Przykład 8. Fragment kodu widoku NavigationOfDishes().*

Metodę wyświetlania każdej listy zakupowej z osobna przedstawiono na przykładzie 9.

Metoda ListOfDaysInShoppingList() służy do przetwarzania informacji pobieranych z bazy danych, żeby uzyskać jako dane wyjściowe zliczenie wag per produkt w obrębie listy zakupowej i selekcje po poszczególnych kategoriach. Danymi wejściowymi jest ID listy zakupów (shoppingListId). Następnie do zmiennej CurrentList mapowana jest cała struktura powiązań aż do produktu. Następnie przeszukiwane jest całe drzewo i mapowane agregując dane do słowników.

```

1. [HttpGet]
2.     public async Task<IActionResult>
3. ListOfDaysInShoppingList(string shoppingListId)
4.     {
5.         int IdShoppingList = Int32.Parse(shoppingListId);
6.
7.         var CurrentList= _dietaAppDbContext.ShoppingLists
8.             .Where(sl=>sl.Id==IdShoppingList)
9.             .Include(disl=>disl.DaysInShoppingList)
10.            .ThenInclude(d=>d.Day)
11.            .ThenInclude(mid=>mid.MealInDays)
12.            .ThenInclude(m=>m.Meal)
13.            .ThenInclude(dim=>dim.DishesInMeal)
14.            .ThenInclude(d=>d.Dish)
15.            .ThenInclude(pid => pid.ProductsInDish)
16.            .ThenInclude(p=>p.Product)
17.            .FirstOrDefault();
18.         int temp = 0;
19.
20.         Dictionary<string, int> sumDistinctProduct = new
21. Dictionary<string, int>();
22.         Dictionary<string, int> sumWeightProduct = new Dictionary<string,
23. int>();

```

```

24. Dictionary<string, string> categoryProduct = new Dictionary<string,
25. string>();
26.
27.
28.         foreach (var day in CurrentList.DaysInShoppingList)
29.         {
30.             var meals = day.Day.MealInDays;
31.             foreach (var item in meals)
32.             {
33.                 var prod = item.Meal.DishesInMeal;
34.                 foreach (var meal in prod)
35.                 {
36.                     var productsInDish = meal.Dish.ProductsInDish;
37.                     foreach (var product in productsInDish)
38.                     {
39.                         if (sumDistinctProduct.ContainsKey(product.Product.Name))
40.                         {
41.                             sumDistinctProduct[product.Product.Name]++;
42.                         }
43.                     else
44.                     {
45.                         sumDistinctProduct.Add(product.Product.Name, 1);
46.                     }
47.                     if (sumWeightProduct.ContainsKey(product.Product.Name)) {
48.                         int actualyValueInDictionary =
49. sumWeightProduct[product.Product.Name];
50.                         temp = actualyValueInDictionary + product.ProductWeight;
51.                         sumWeightProduct[product.Product.Name] = temp;
52.                     }
53.                     else
54.                     {
55.                         sumWeightProduct.Add(product.Product.Name, product.ProductWeight);
56.                     }
57.                     if (!categoryProduct.ContainsKey(product.Product.Name)) {
58.                         categoryProduct.Add(product.Product.Name, product.Product.Category);
59.                     }
60.                 }
61.             }
62.         }
63. ShoppingListViewModel dictionaryViewModel = new ShoppingListViewModel
64.         {
65.             sumDistinctProduct = sumDistinctProduct,
66.             sumWeightProduct = sumWeightProduct,
67.             categoryProduct= categoryProduct
68.         }
69.         return View(dictionaryViewModel); }

```

*Przykład 9. Metoda ListOfDaysInShoppingList.*

Wartość zwracana metody *ListOfDaysInShoppingList()* przekazywana jest do widoku, którego fragmentu kodu przedstawiono na przykładzie 10.

```
1. <div class="text-center">
2.     @*@catValue*@
3.     @if (catFruits.Count() > 0)
4.     {
5.         <h1>Owoce i Warzywa</h1>
6.         <table class="table">
7.             <thead>
8.                 <tr>
9.                     <th scope="col">
10.                    </th>
11.                    <th scope="col">
12.                        Nazwa
13.                    </th>
14.                    <th scope="col">
15.                        Waga/Pojemność
16.                    </th>
17.                </tr>
18.            </thead>
19.            <tbody>
20.                @foreach (var cat in catFruits.OrderBy(i=>i.Value))
21.                {
22.                    @foreach (var i in
23. Model.sumWeightProduct.OrderBy(i=>i.Key))
24.                    {
25.                        if (cat.Key == i.Key)
26.                        {
27.                            <tr>
28.
29.                                <td>
30.                                    <input type="checkbox"
31. onclick="myFunction()" value="CrossOut" id="kupione">
32.                                </td>
33.                                <td>@i.Key</td>
34.                                <td>@i.Value</td>
35.                            </tr>
36.                        }
37.                    }
38.                }
39.            </tbody>
40.        </table>
```

Przykład 10. Fragment kodu widoku *ListOfDaysInShoppingList*.

Komunikacja odpowiedzialna za dodanie danych do bazy danych przy użyciu [HTTP POST] jest używana do dodania do różnych tabel:

- produktu wraz z jego atrybutami,
- dania wraz z jego nazwą oraz dodanie powiązań z produktami do tabeli ProductsInDish,
- posiłku wraz z jego nazwą oraz dodanie powiązań z daniami do tabeli DishesInMeal,
- dnia wraz z jego nazwą oraz dodanie powiązań z posiłkami do tabeli MealsInDay,
- listy zakupowej wraz z nazwą oraz dodanie powiązań z dniami do tabeli DaysInShoppingList.

Metoda dodawania produktów, posiłków, dni, list zakupowych wraz z ich atrybutami i tabelami powiązań została przedstawiona na przykładzie 11. (dodawania posiłków).

Metoda *AddMeall()* na wejściu otrzymuje nazwę posiłku (*mealName*) oraz listę (*Meal3Dto*), która zawiera pola waga: *int* oraz nazwaProduktu: *string*.

Metoda zapisuje do bazy nowy posiłek i pobiera jego ID, żeby dodać do tabeli *ProductsInMeal* wszystkie powiązania między posiłkiem a produktami. Na wyjściu metody *AddMeall()* zwracany jest widok posiłków.

```

1.      [HttpPost]
2.      public async Task<IActionResult> AddMeall(string mealName,
3.          List<Meal2Dto> dishes)
4.      {
5.          var meal = new Meal
6.          {
7.              Name = mealName
8.          };
9.          _dietaAppDbContext.Add<Meal>(meal);
10.         _dietaAppDbContext.SaveChanges();
11.
12.         int mealId = meal.Id;
13.         List<int> IdDishesList = new List<int>();
14.         foreach (var dish in dishes)
15.         {
16.             IdDishesList.Add(Int32.Parse(dish.dishName)); ;
17.         }
18.         for (int i = 0; i < dishes.Count(); i++)
19.         {
20.             var bindingsTable = new DishesInMeal
21.             {
22.                 MealId = mealId,
23.                 DishId = IdDishesList[i]
24.             };
25.             _dietaAppDbContext.AddAsync<DishesInMeal>(bindingsTable);
26.             _dietaAppDbContext.SaveChanges();
27.             //return RedirectToAction("NavigationOfProducts");
28.         }
29.         return Ok();

```

*Przykład 11. Metoda AddMeall.*

Na przykładzie 12. przedstawiono fragment kodu z widoku AddMeal. Najważniejszą funkcją jest wywołanie metody (napisana w języku JavaScript) AddMeal().

```
1.     <div>
2.         Nazwa posiłku: <br />
3.         <input id="mealName" />
4.     </div>
5.     <div>
6.         <a href="#" style="margin-top:29px;"
7.             class="btn btn-info add-row" role="button"> <i class="fa
8. fa-plus"></i>&nbsp;Dodaj danie</a>
9.     </div>
10.    <br />
11.
12.    <div id="firstDriver_initialForm">
13.        <div class="container">
14.            <div class="dynamic-rows">
15.                <div class="row" id="template" data-order="0">
16.
17.                    <div class="row filter">
18.                        <br />
19.                        <div class="col-md-8">
20. <select class="dish form-control" style="border-color:gray;font-
21. size:13px" asp-items="@Model.Dishes">
22.     <option selected value="">Wszystkie</option>
23.     </select>
24.     </div>
25.     <div class="col-md-3">
26.     <button type="submit" class="btn btn-danger delete-row" data-
27. val="0"><i class="fa fa-trash"></i>&nbsp;Usuń </button>
28.     </div>
29.     </div>
30.     </div>
31.     </div>
32.     </div>
33.     <br />
34.     <button type="submit" class="btn btn-success"
35. onclick="AddMeal()">Dodaj posiłek</button> </div>
```

*Przykład 12. Widok AddMeal.*

Metoda *AddMeal()* znajdująca się na przykładzie 12. przyjmuje jako dane wejściowe nazwę posiłku oraz listę produktów w posiłku; dane te są przekazane przez aplikację do metody *AddMeal()*, którą przedstawiono na przykładzie 13.

```
1.     $.ajax({
2.         url: '@Url.Action("AddMeal", "Meal")',
3.         type: "POST",
4.         data: {
```



```

5.         mealName: mealName, dishes: dishes
6.     },
7.
8.     success: function (result) {
9.         //alert("Posiłek dodany");
10.        //$(".alert").show();
11.        //    var param = JSON.stringify(result);
12.
13.        var url = '@Url.Action("Confirmation", "Home", new {name
14. = "posiłek.", table = "listy", controller = "Meal", method =
15. "NavigationOfMeals" })';
16.        window.location.href = url;
17.
18.    },
19.    error: function (xhr) {
20.    },
21.    });
22.    $(".closebtn").click(function () {
23.        var url = '@Url.Action("NavigationOfMeals", "Meal")';
24.        window.location.href = url;
25.    });
26.    }

```

*Przykład 13. Metoda AddMeal.*

Komunikacja odpowiedzialna za usuwanie danych z bazy danych przy użyciu [HttpDelete] jest używana do usuwania danych znajdujących się w tabelach:

- Products,
- Dishes,
- Meals,
- Days,
- ShoppingLists,

Metodę usuwania produktów, dań, posiłków, dni i list zakupowych przestawiono na przykładzie 14. (usuwanie produktu).

Metoda *Delete()* jako dane wejściowe przyjmuje ID produktu (*productId*), następnie korzystając z klasy *Manager* wywołuje metodę na usunięcie danego produktu oraz na pobranie wszystkich produktów. Finalnie zwraca widok z listą wszystkich produktów.

```

1. public IActionResult Delete(int productId)
2. {
3.
4.     mManager.DeleteProduct(new ProductDto { Id = productId });
5.     var productDtos = mManager.GetAllProducts(null);
6.     var productViewModels = mViewModelMapper.Map(productDtos);
7.     return View("NavigationOfProducts", productViewModels);

```

*Przykład 14. Przykład metody Delete.*

Wartość zwracana metody *Delete()* przekazywana jest do widoku (w tym przypadku produktu), którego fragment kodu przedstawiono na przykładzie 15.

```
1. <div class="text-center">
2.     <table class="table">
3.         <thead>
4.             <tr>
5.                 <th scope="col">
6.                     Id
7.                 </th>
8.                 <th scope="col">
9.                     Nazwa
10.                </th>
11.                <th scope="col">
12.                    Kcal na 100g/ml
13.                </th>
14.                <th scope="col">
15.                    Jednostka
16.                </th>
17.                <th scope="col">
18.                    Kategoria
19.                </th>
20.                <th scope="col">
21.
22.                </th>
23.            </tr>
24.        </thead>
25.        <tbody>
26.            @foreach (var product in Model.OrderBy(i => i.Name))
27.            {
28.                <tr>
29.                    @*<th scope="row">@product.Id</th>*&
30.                    <td scope="row">@i</td>
31.                    <td>@product.Name</td>
32.                    <td>@product.Kcal</td>
33.                    <td>@product.Unit</td>
34.                    <td>@product.Category</td>
35.                    <td>
36.                        <a class="btn btn-danger" asp-
37. controller="Product" asp-action="Delete" asp-route-
38. productId="@product.Id">Usuń</a>
39.                    </td>
40.                </tr>
41.                i++;}
42.            </tbody>
43.        </table></div>
```

Przykład 15. Fragment kodu widoku *NavigationOfProducts*.

W rozdziale dotyczącym implementacji przedstawiono strukturę hierarchiczną plików składających się na aplikację oraz zaprezentowano fragmenty kodu metod wykorzystywanych między innymi do pobierania, wysyłania danych do bazy danych. Przedstawiono również komunikację poszczególnych metod, warstw aplikacji (frontend < -- > backend).

## 4. Testy funkcjonalne

Testy funkcjonalne to rodzaj testów oprogramowania, który weryfikuje aplikację pod względem wymagań funkcjonalnych. Celem jest przetestowanie każdej funkcji aplikacji poprzez zapewnienie odpowiednich danych wejściowych i weryfikację danych wyjściowych pod względem wymagań funkcjonalnych [14].

Poniżej przedstawiono schemat testowania:

1. Dane wejściowe – wybranie odpowiednich elementów na ekranie, wartości wpisanych danych oraz zrzut ekranu z aplikacji.
2. Oczekiwany rezultat – rezultat działania danej funkcjonalności, który użytkownik oczekuje po wprowadzeniu danych.
3. Dane wyjściowe – zgodność wyników wraz ze zrzutem ekranu z aplikacji potwierdzającym rezultat.

Poniżej przedstawiono listę testów funkcjonalnych:

1. Wyświetlenie listy produktów.
2. Dodawanie produktu.
3. Usuwanie produktu.
4. Wyświetlenie listy produktów, które będą wykorzystane w dalszych testach.
5. Wyświetlenie listy dań.
6. Dodawanie dania.
7. Szczegóły dania.
8. Usuwanie dania.
9. Wyświetlenie listy dań, które będą wykorzystane w dalszych testach.
10. Wyświetlenie listy posiłków.
11. Dodawanie posiłku.
12. Szczegóły posiłku.
13. Usuwanie posiłku.
14. Wyświetlenie listy posiłków, które będą wykorzystane w dalszych testach.
15. Wyświetlenie listy dni.
16. Dodawanie dnia.
17. Szczegóły dnia.
18. Usuwanie dnia.
19. Wyświetlenie listy dni, które będą wykorzystane w dalszych testach.
20. Wyświetlenie list zakupów.
21. Dodawanie listy zakupów.
22. Szczegóły listy zakupów.
23. Wyświetlenie dodanej listy zakupowej.
24. Usuwanie listy zakupowej.

Lista testów odwzorowuje czynności niezbędne do prawidłowego i kompletnego użytkowania wszystkich możliwości jakie oferuje aplikacja.

Wykonane testy funkcjonalne:

1. Wyświetlenie listy produktów.
  - 1.1. Oczekiwany rezultat: Wyświetlenie pustej listy produktów (brak dodanych produktów).

1.2. Dane wyjściowe zgodne z oczekiwaniami: rezultat przedstawiono na rysunku 3.

The screenshot shows the 'DietaApp' interface. At the top, there is a header with the app name 'DietaApp' and a button '<- Powrót do głównego Menu'. Below the header, the title 'Produkty' is displayed. Under the title, there is a green button 'Dodaj Produkt' and a search bar with a green 'Filtr' button. Below these elements is a table with five columns: 'Id', 'Nazwa', 'Kcal na 100g/ml', 'Jednostka', and 'Kategoria'. The table is currently empty. At the bottom of the screen, there is a footer with the text '© 2020 - DietaApp - Privacy'.

Rysunek 3. Pusta lista produktów.

2. Dodawanie produktu.

2.1. Dane wejściowe:

Nazwa: truskawka,

kcal: 31,

jednostka: gram,

kategoria: Owoce.

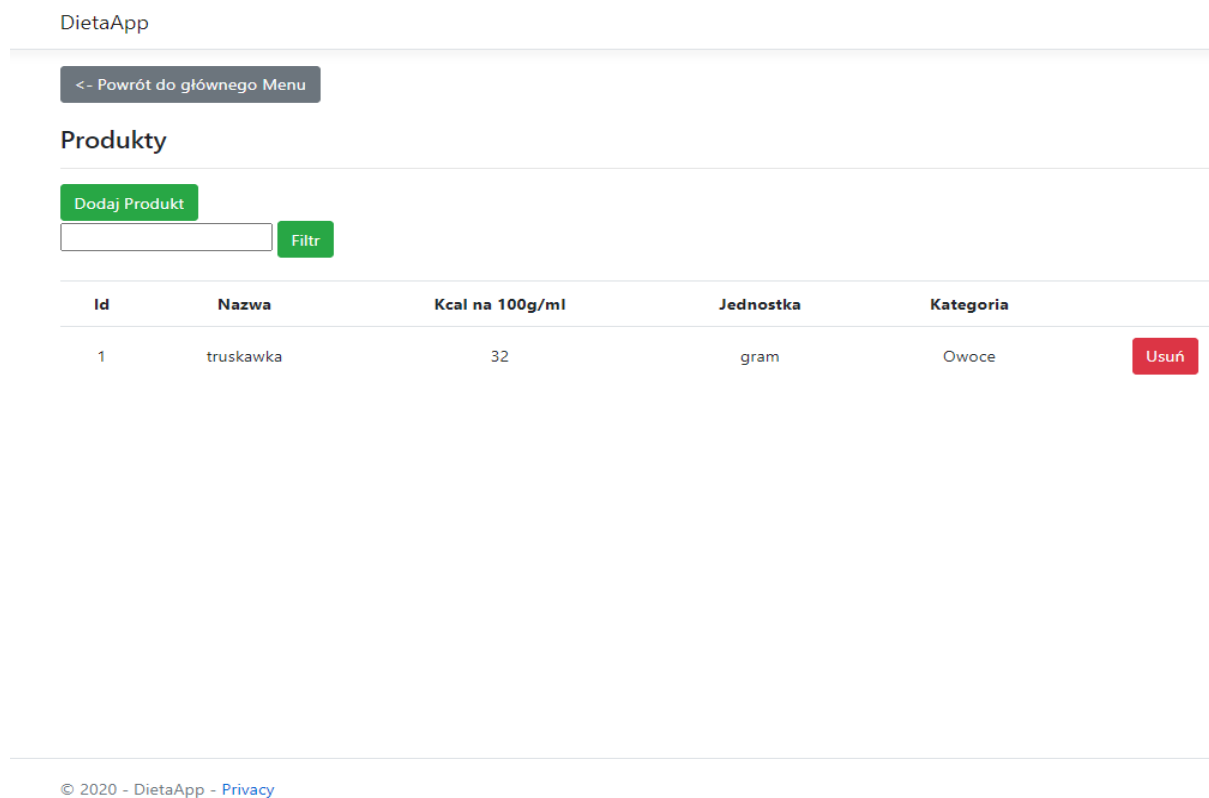
The screenshot shows the 'DietaApp' interface for adding a product. At the top, there is a header with the app name 'DietaApp' and a button '<- Powrót do listy produktów'. Below the header, the title 'Produkt' is displayed. Under the title, there are several input fields: 'Nazwa produktu:' with the value 'truskawka', 'Kcal (100):' with the value '32', 'Jednostka:' with a dropdown menu showing 'gram', and 'Kategoria:' with a dropdown menu showing 'Owoce'. Below these fields is a green button 'Dodaj Produkt'. At the bottom of the screen, there is a footer with the text '© 2020 - DietaApp - Privacy'.

Rysunek 4. Dane wejściowe testu dodawania produktu.

- 2.2. Oczekiwany rezultat: Potwierdzenie dodania produktu oraz widoczność dodanego produktu „truskawka” podczas wyświetlenia listy produktów.
- 2.3. Dane wyjściowe zgodne z oczekiwaniami: potwierdzenie dodania produktu przedstawione na rysunku 5. oraz sprawdzenie widoczności dodanego produktu „truskawka” na liście produktów przedstawione na rysunku 6.



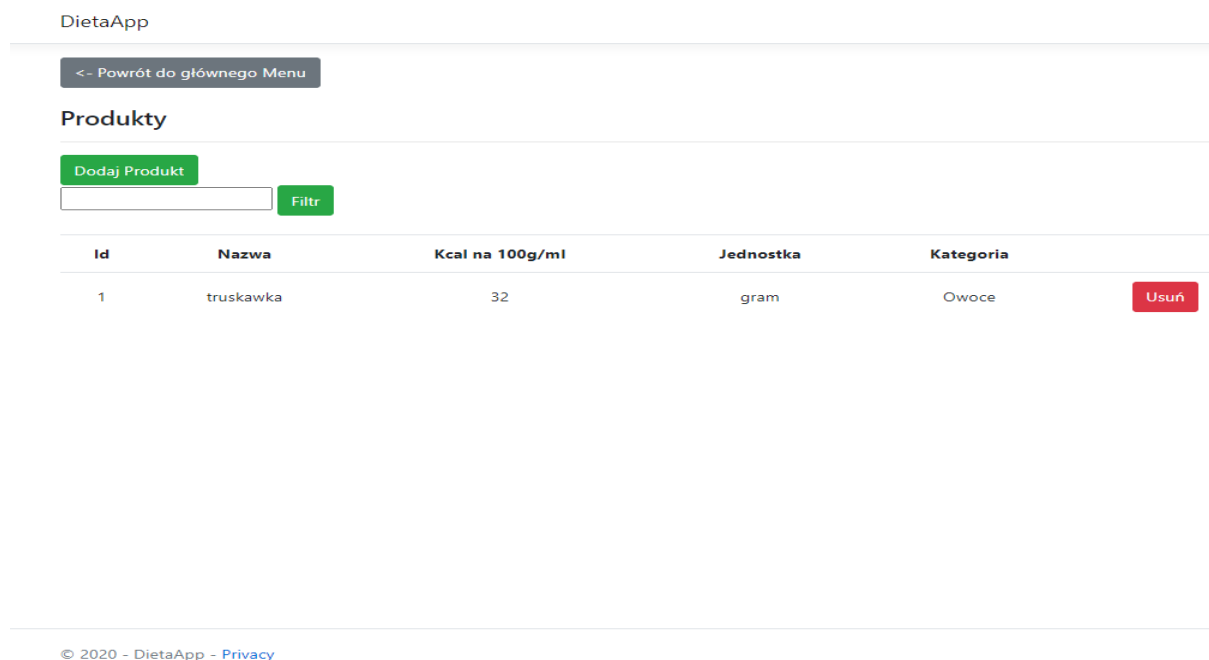
*Rysunek 5. Potwierdzenie dodania produktu.*



*Rysunek 6. Lista produktów po dodaniu produktu „truskawka”.*

### 3. Usuwanie produktu.

#### 3.1. Wyświetlenie listy produktów przed usunięciem produktu (rysunek 7.).



Rysunek 7. Lista produktów.

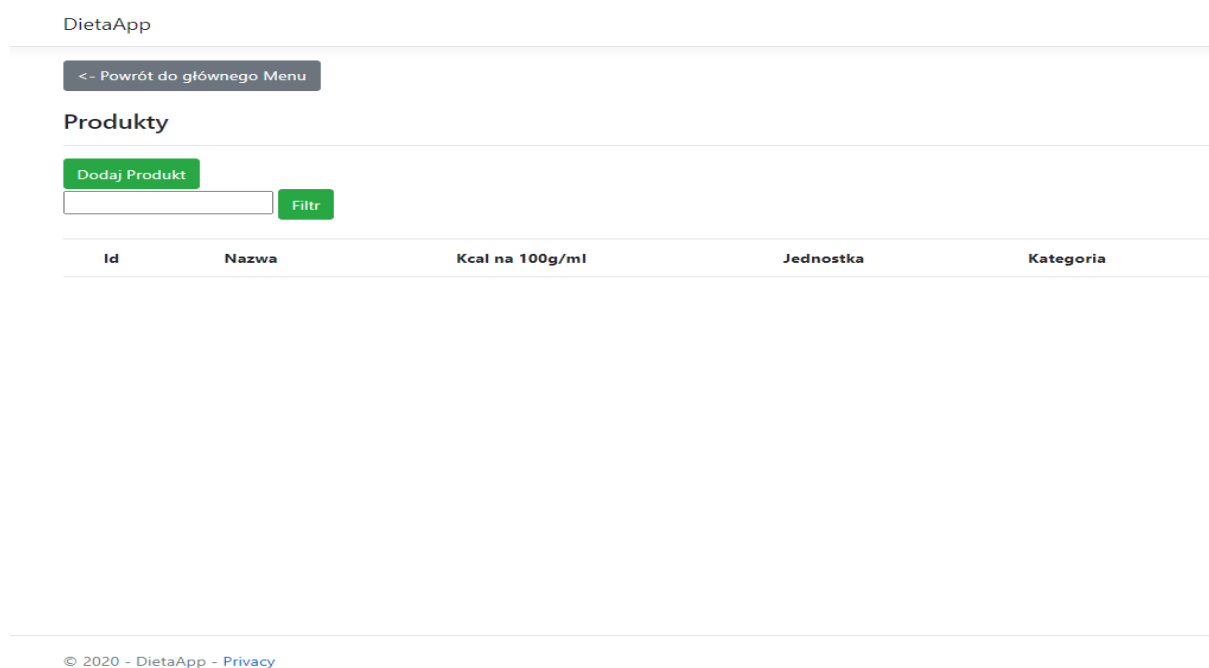
#### 3.2. Dane wejściowe:

Id produktu: 1.

Nazwa produktu: truskawka,

3.3. Oczekiwany rezultat: Produkt o nazwie „truskawka” i Id = 1, usunięty z listy produktów.

3.4. Dane wyjściowe zgodne z oczekiwaniami: Rezultat przedstawiony na rysunku 8.



Rysunek 8. Rezultat testu usuwania produktu.

4. Wyświetlenie listy produktów, które będą wykorzystane w dalszych testach.
- 4.1. Dane wejściowe: Wprowadzenie kilku produktów (filet z kurczaka, jogurt naturalny, mleko, truskawka, warzywa na patelnie firmy Hortex).
- 4.2. Oczekiwany rezultat: Pojawienie się produktów zgodnych z atrybutami na liście wszystkich produktów.
- 4.3. Dane wyjściowe zgodne z oczekiwaniami: aktualna lista z dodanymi produktami przedstawiona na rysunku 9.

DietaApp

[<- Powrót do głównego Menu](#)

**Produkty**

[Dodaj Produkt](#)

[Filtr](#)

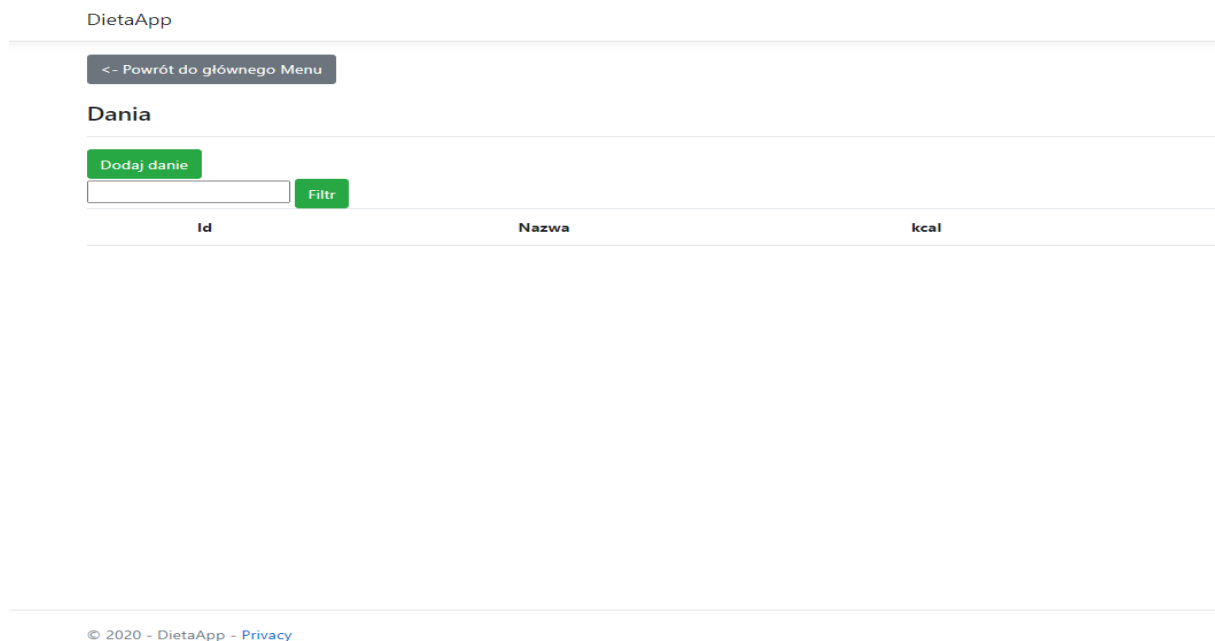
Id	Nazwa	Kcal na 100g/ml	Jednostka	Kategoria	
1	filet z kurczaka	165	gram	Mięso	<a href="#">Usuń</a>
2	jogurt naturalny	60	gram	Nabiał	<a href="#">Usuń</a>
3	mleko	42	ml	Nabiał	<a href="#">Usuń</a>
4	truskawka	32	gram	Owoce	<a href="#">Usuń</a>
5	warzywa na patelnie firmy Hortex	60	gram	Mrożonki	<a href="#">Usuń</a>

© 2020 - DietaApp - [Privacy](#)

*Rysunek 9. Zaktualizowana lista produktów.*

5. Wyświetlenie listy dań.
- 5.1. Oczekiwany rezultat: Wyświetlenie pustej listy dań (brak dodanych dań).
- 5.2. Dane wyjściowe zgodne z oczekiwaniami: rezultat przedstawiono na rysunku 10.





Rysunek 10. Pusta lista dań.

## 6. Dodawanie dania.

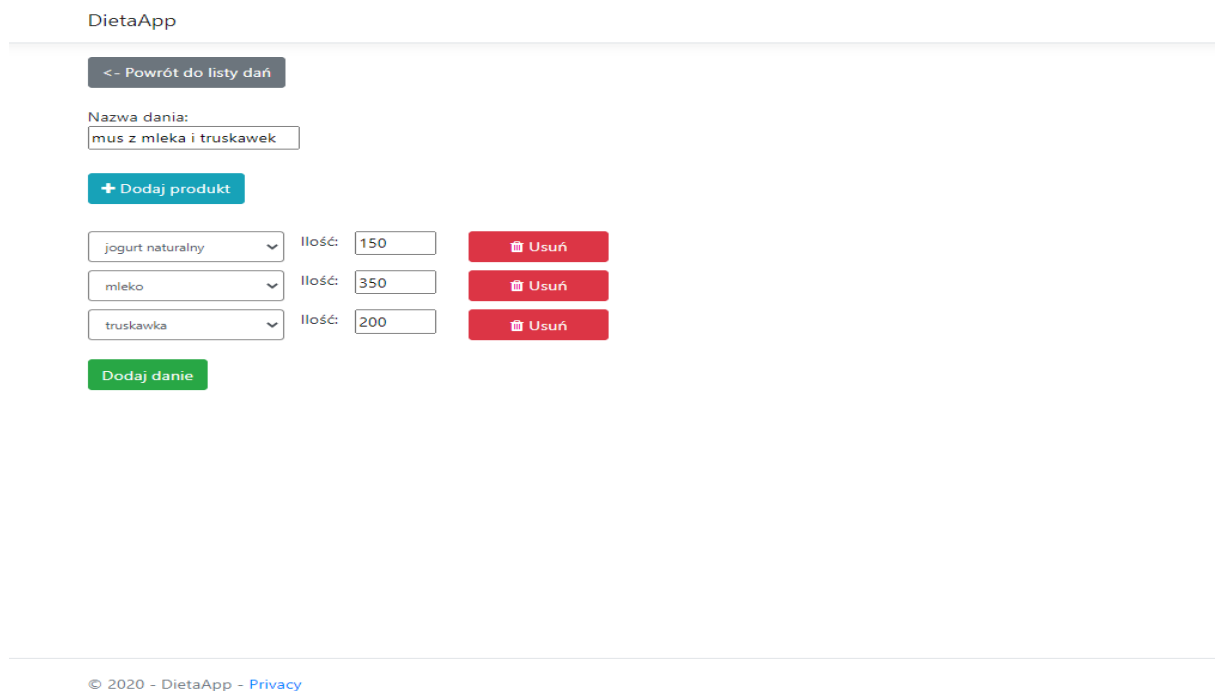
### 6.1. Dane wejściowe:

Nazwa dania: mus z mleka i truskawek,

Nazwa produktu: jogurt naturalny, ilość: 150g,

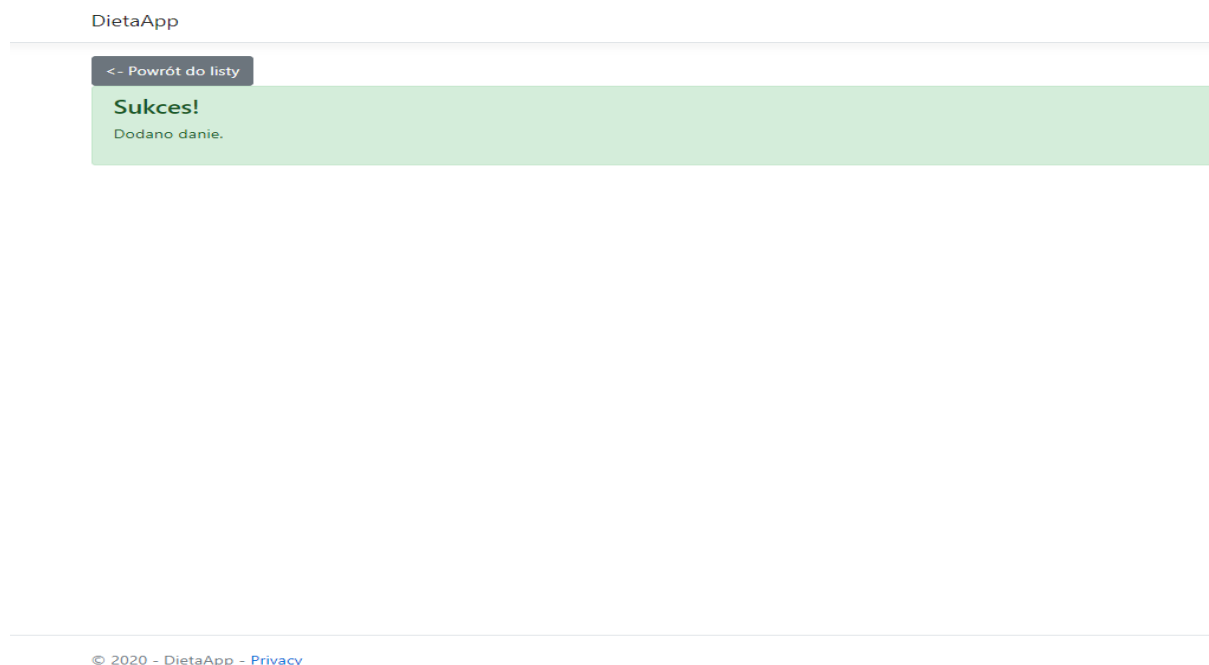
Nazwa produktu: mleko, ilość: 350g,

Nazwa produktu: truskawka, ilość: 200g.

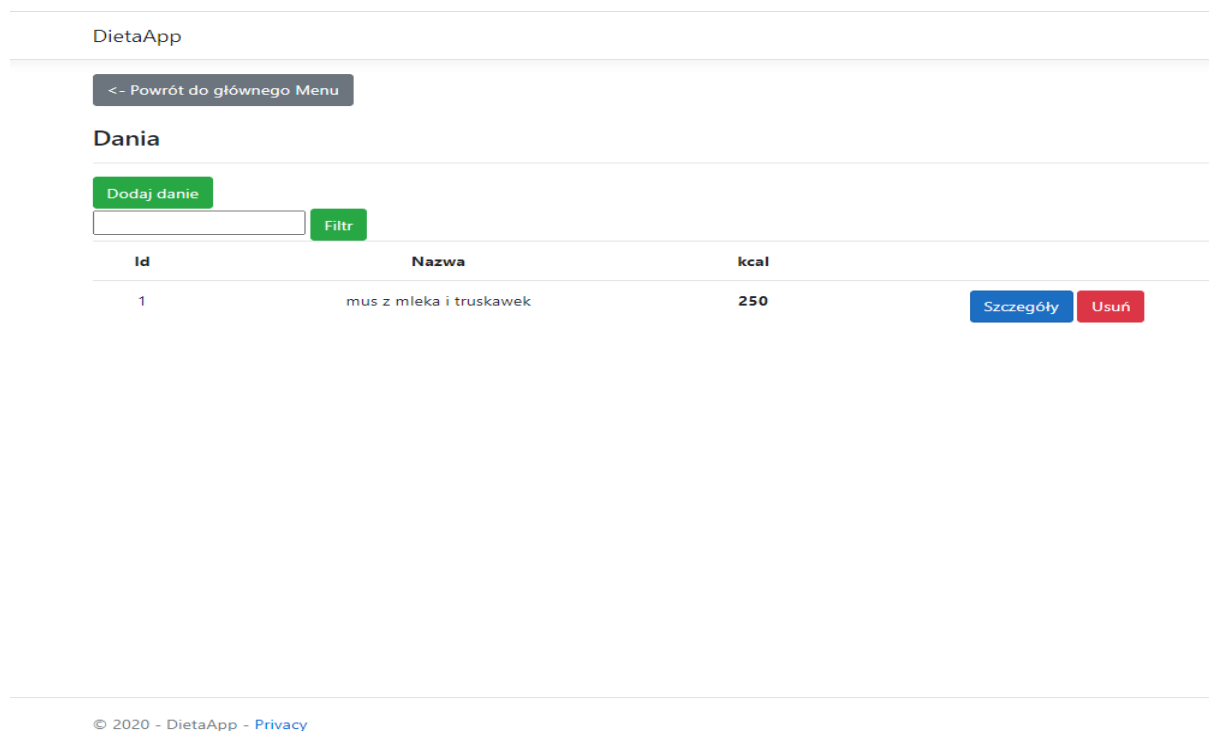


Rysunek 11. Dane wejściowe testu dodawania dania.

- 6.2. Oczekiwany rezultat: Potwierdzenie dodania dania oraz widoczność dodanego dania (wraz z sumą 250 kcal) „mus z mleka i truskawek” podczas wyświetlenia listy dań.
- 6.3. Dane wyjściowe zgodne z oczekiwaniami: potwierdzenie dodania dania przedstawione na rysunku 12 oraz sprawdzenie widoczności dodanego dania „mus z mleka i truskawek” na liście dań przedstawione na rysunku 13.



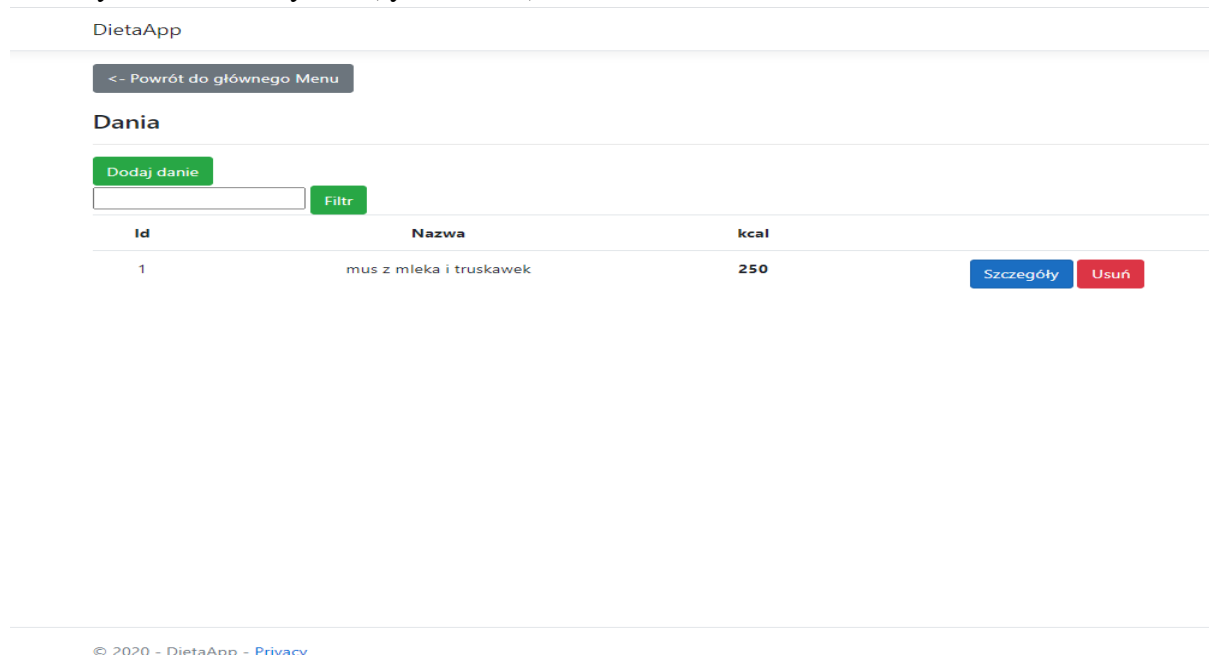
*Rysunek 12. Potwierdzenie dodania dania.*



*Rysunek 13. Aktualna lista dań po dodaniu dania „mus z mleka i truskawek”.*

## 7. Szczegóły dania.

### 7.1. Wyświetlenie listy dań (rysunek 14.).



Rysunek 14. Lista dań.

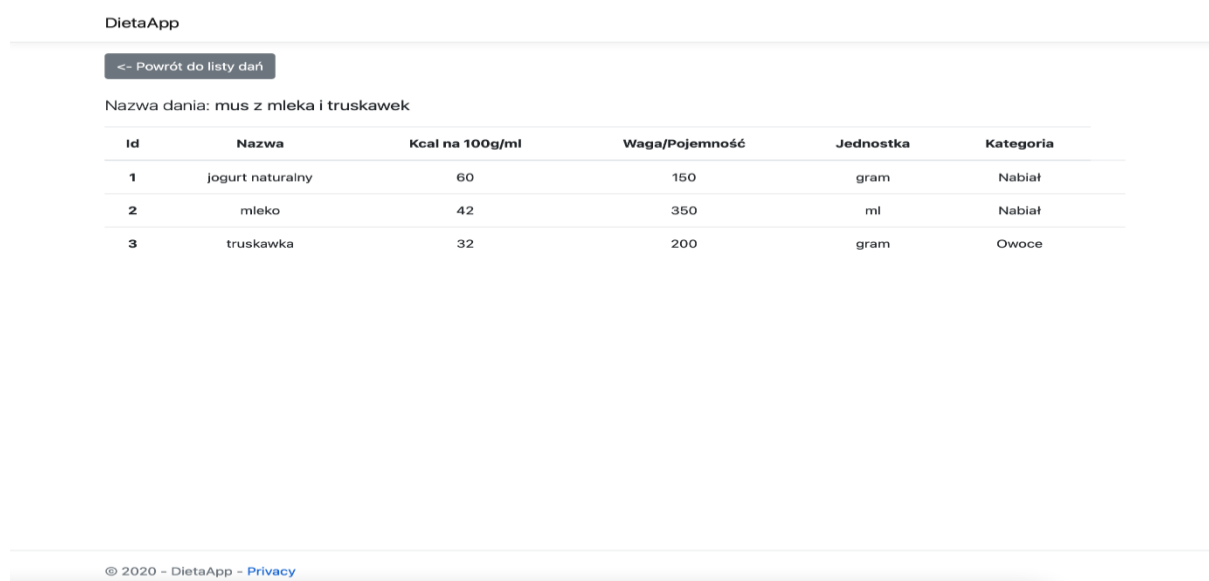
### 7.2. Dane wejściowe:

Id dania: 1.

Nazwa dania: mus z mleka i truskawek,

7.3. Oczekiwany rezultat: Wyświetlenie szczegółów (produktów) dania o nazwie „mus z mleka i truskawek” i Id = 1

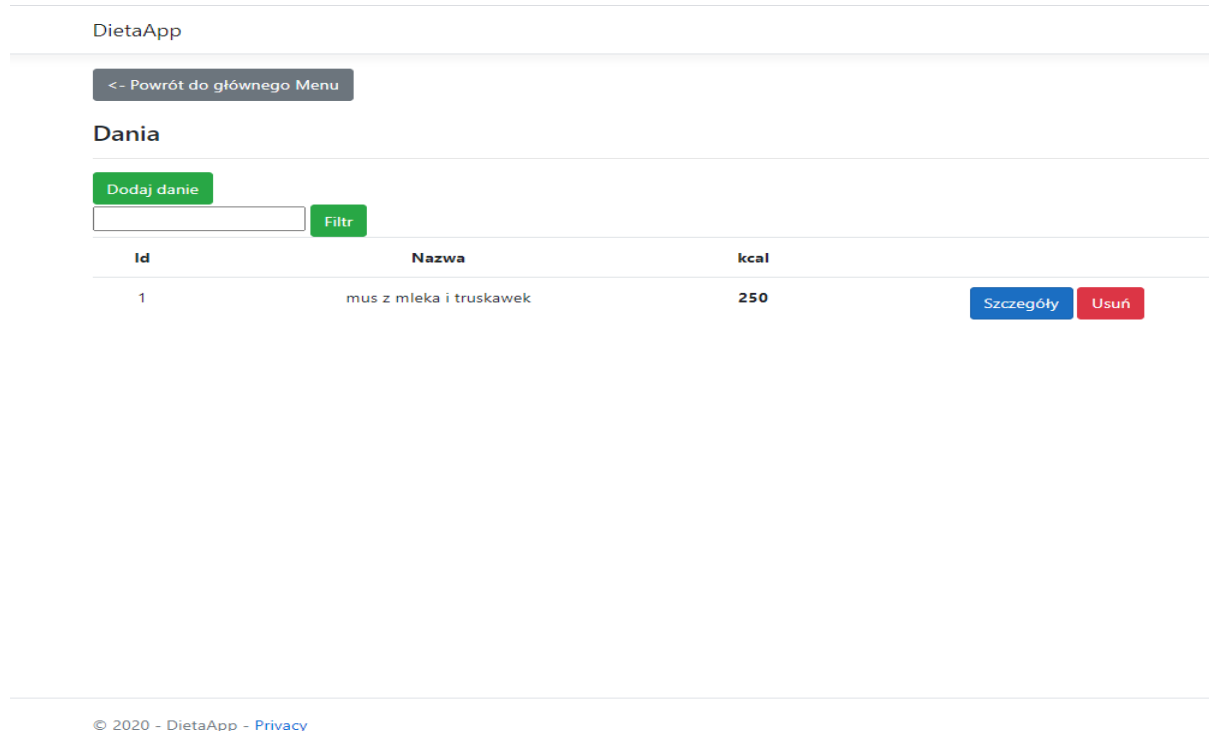
7.4. Dane wyjściowe zgodne z oczekiwaniami: Wyświetlenie nazwy dania „mus z mleka i truskawek” oraz produktów wchodzących w jego skład - jogurt naturalny (150g), mleko(350g), truskawka (200g). Rezultat przedstawiony na rysunku 15.



Rysunek 15. Szczegóły dania „mus z mleka i truskawek”.

## 8. Usuwanie dania.

### 8.1. Wyświetlenie listy dań przed usunięciem produktu (rysunek 16.).



Rysunek 16. Aktualna lista dań przed usunięciem.

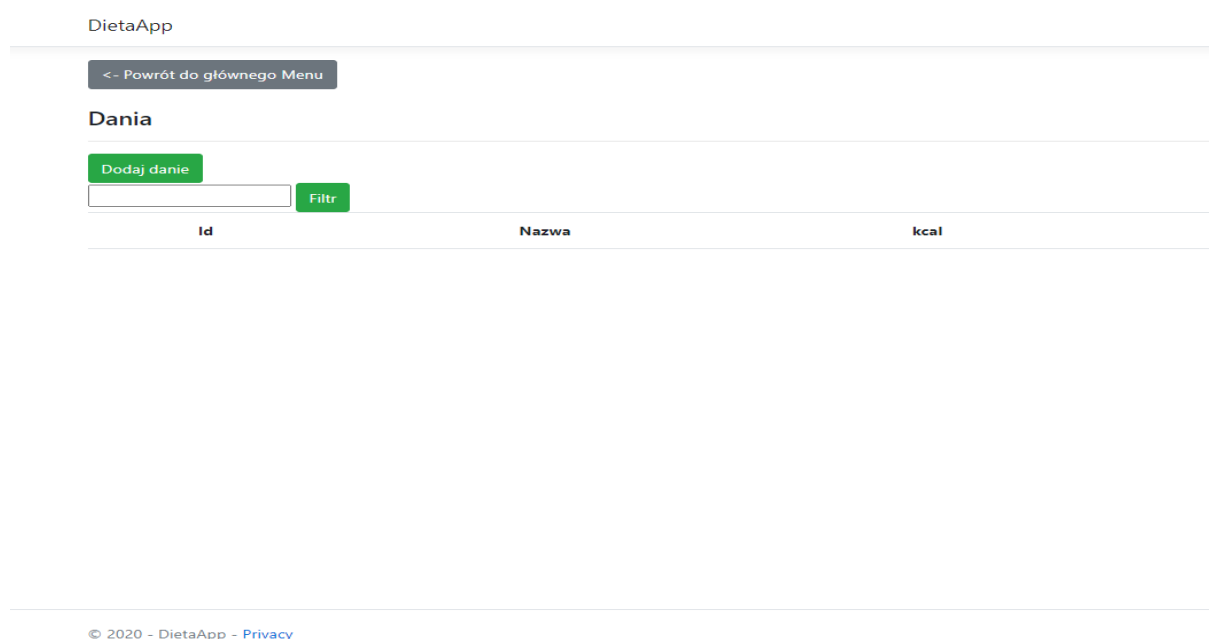
### 8.2. Dane wejściowe:

Id dania: 1.

Nazwa dania: mus z mleka i truskawek,

8.3. Oczekiwany rezultat: Danie o nazwie „mus z mleka i truskawek” i Id = 1, usunięty z listy dań.

8.4. Dane wyjściowe zgodne z oczekiwaniami: Rezultat przedstawiony na rysunku 17.



Rysunek 17. Lista dań po usunięciu.

9. Wyświetlenie listy dań, które będą wykorzystane w dalszych testach.
- 9.1. Dane wejściowe: Wprowadzenie dodatkowego dania „filet z warzywami”.
- 9.2. Oczekiwany rezultat: Pojawienie się dania zgodnego z atrybutami na liście wszystkich dań.
- 9.3. Dane wyjściowe zgodne z oczekiwaniami: aktualna lista z dodanymi daniami przedstawiona na rysunku 18.

DietaApp

[<- Powrót do głównego Menu](#)

### Dania

[Dodaj danie](#)

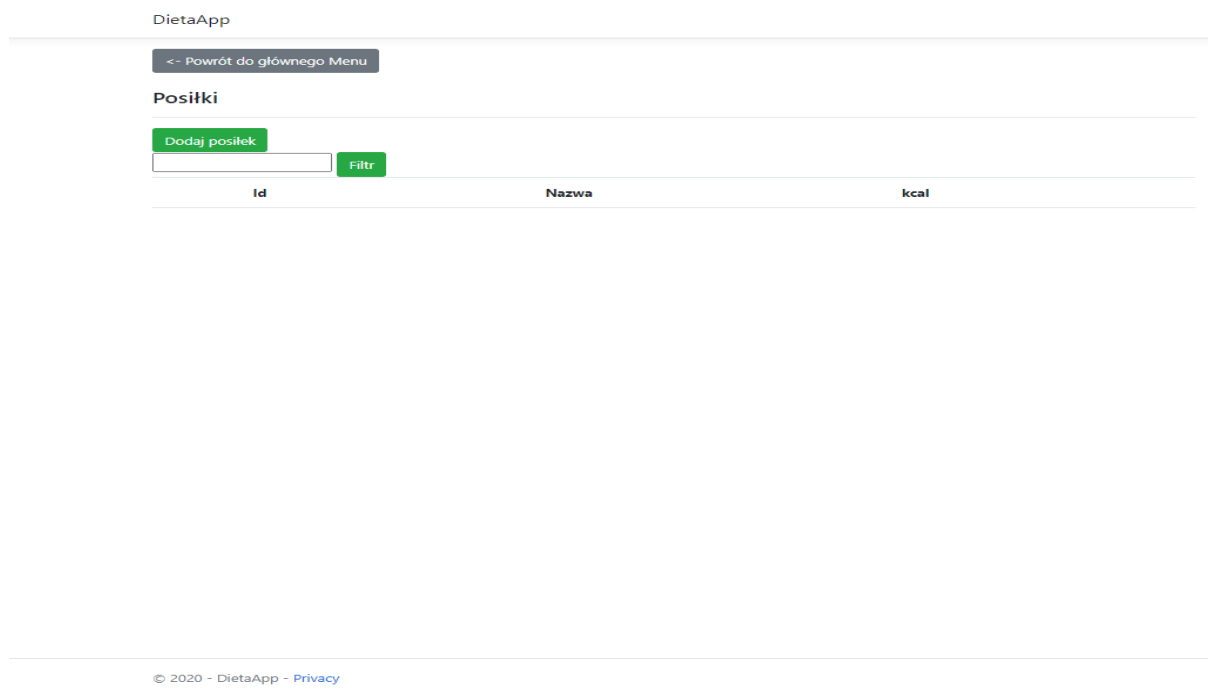
[Filtr](#)

Id	Nazwa	kcal	
1	mus z mleka i truskawek	250	<a href="#">Szczegóły</a> <a href="#">Usuń</a>
2	filet z warzywami	510	<a href="#">Szczegóły</a> <a href="#">Usuń</a>

© 2020 - DietaApp - [Privacy](#)

*Rysunek 18. Zaktualizowana lista dań.*

10. Wyświetlenie listy posiłków.
- 10.1. Oczekiwany rezultat: Wyświetlenie pustej listy posiłków (brak dodanych posiłków).
- 10.2. Dane wyjściowe zgodne z oczekiwaniami: rezultat przedstawiono na rysunku 19.



Rysunek 19. Pusta lista posiłków.

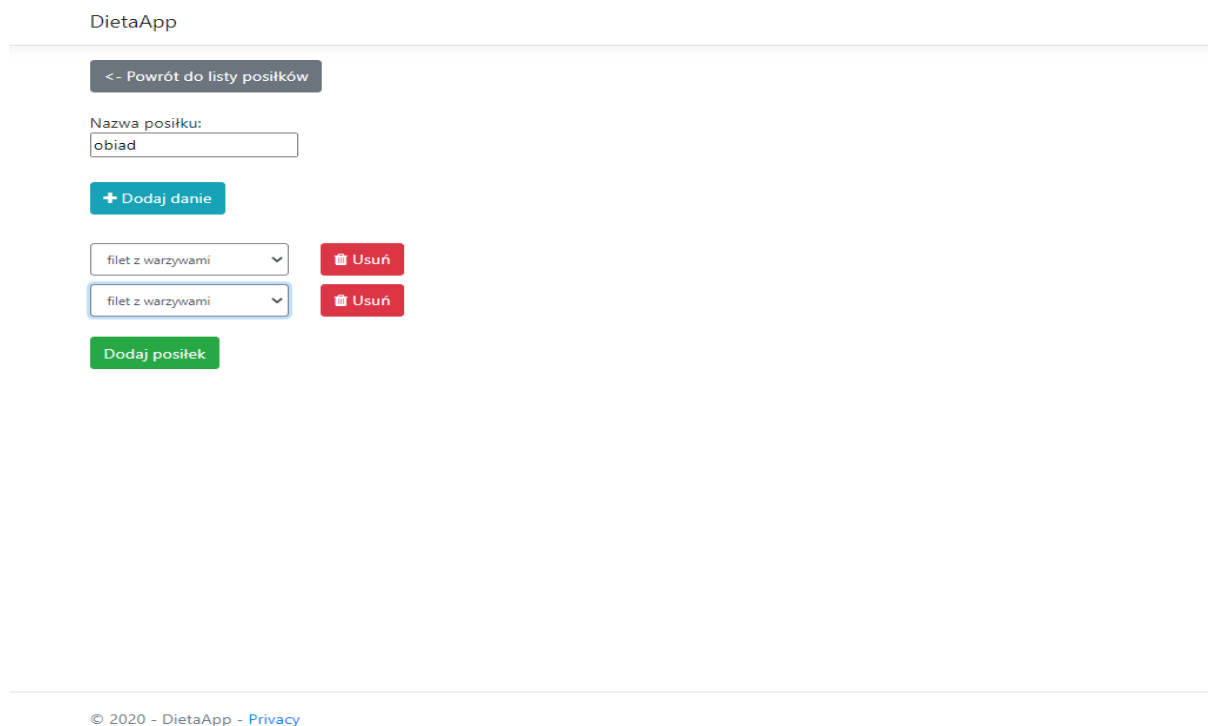
## 11. Dodawanie posiłku.

### 11.1. Dane wejściowe:

Nazwa posiłku: obiad,

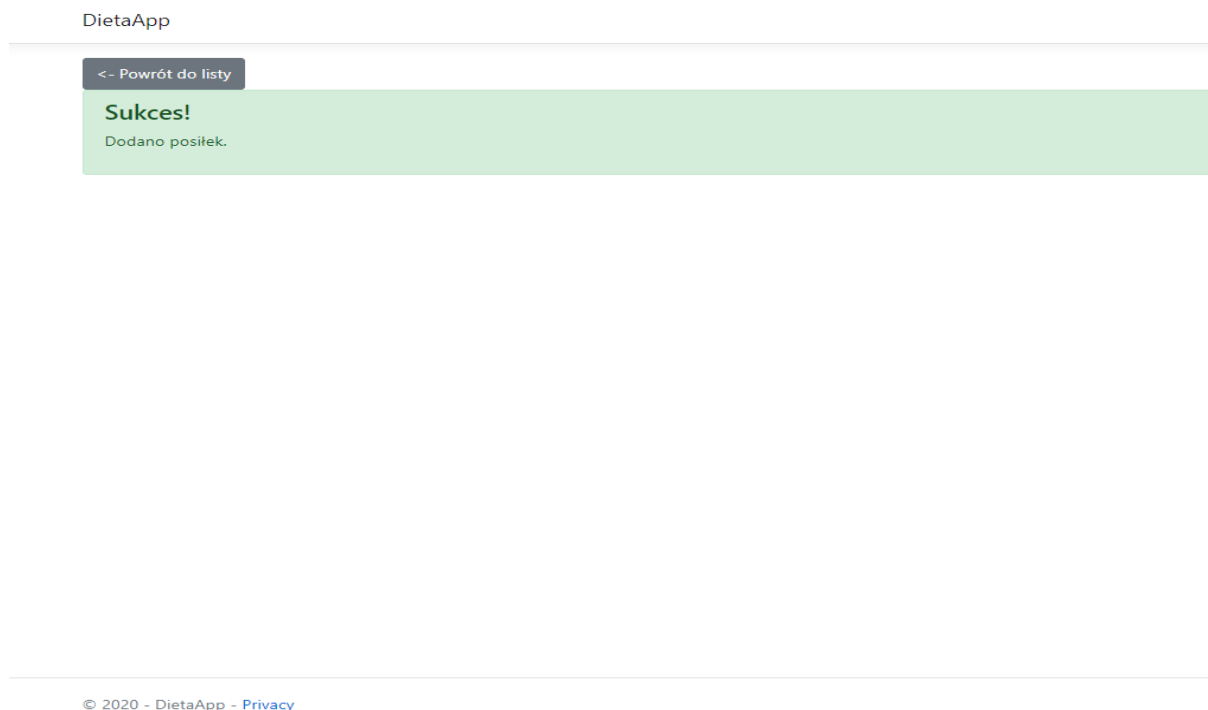
Nazwa dania: filet z warzywami,

Nazwa dania: filet z warzywami,

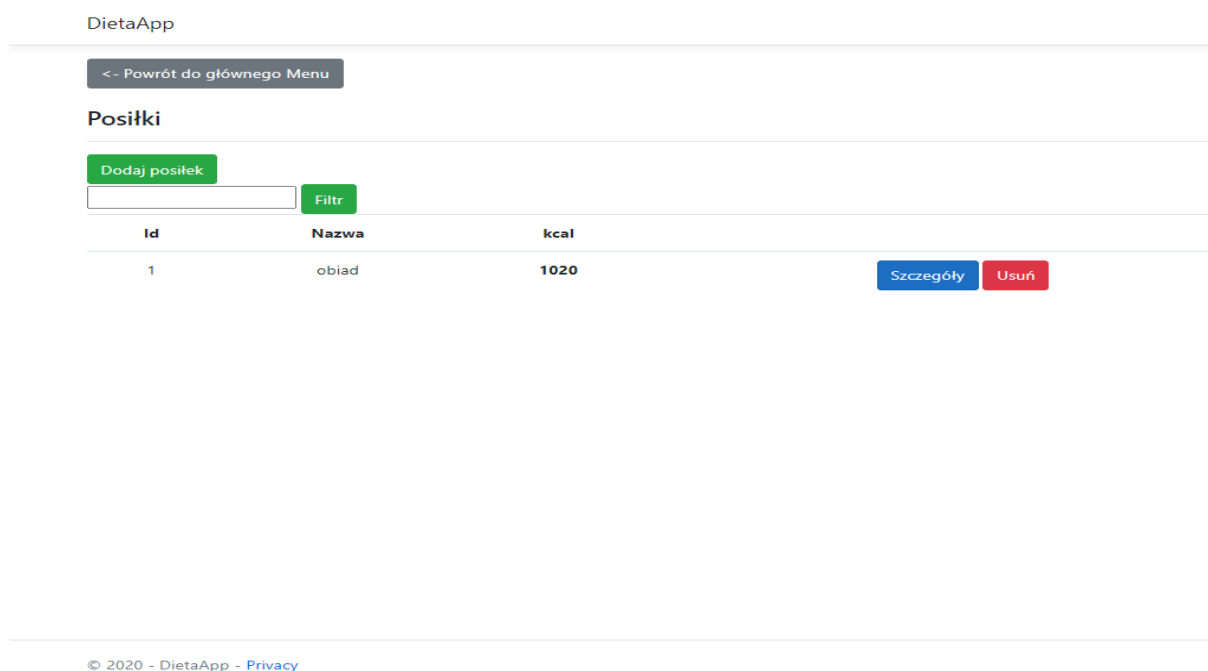


Rysunek 20. Dane wejściowe testu dodawania posiłku.

- 11.2. Oczekiwany rezultat: Potwierdzenie dodania posiłku oraz widoczność dodanego posiłku (wraz z sumą 1020 kcal) „obiad” podczas wyświetlenia listy posiłków.
- 11.3. Dane wyjściowe zgodne z oczekiwaniami: potwierdzenie dodania posiłku przedstawione na rysunku 21. oraz sprawdzenie widoczności dodanego posiłku „obiad” na liście posiłków przedstawione na rysunku 22.



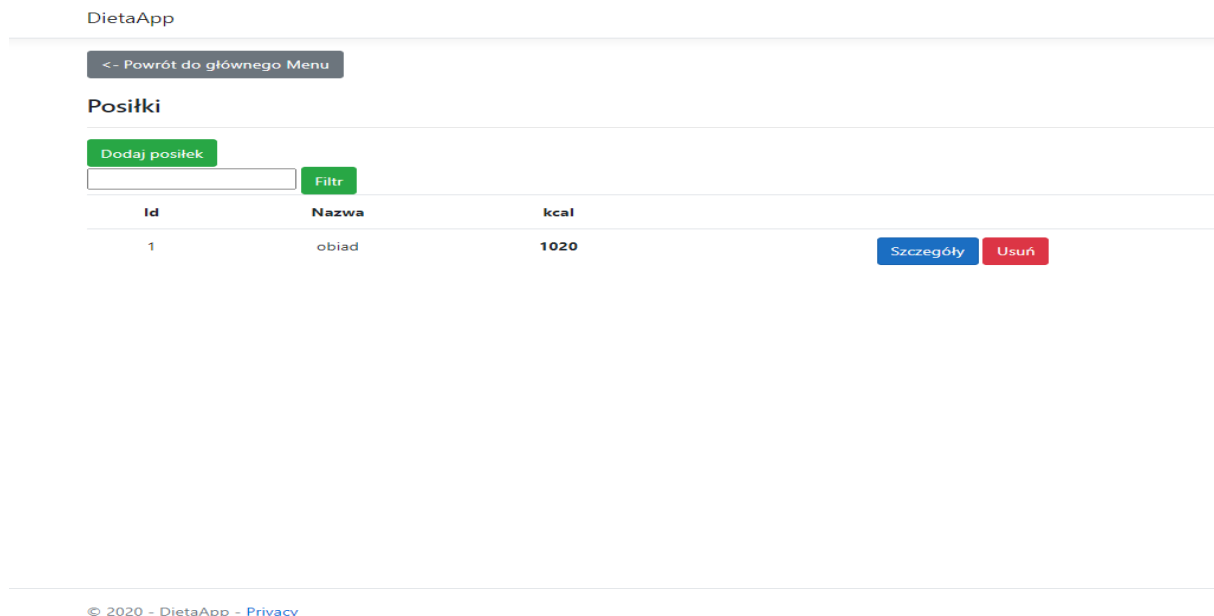
Rysunek 21. Potwierdzenie dodania posiłku.



Rysunek 22. Aktualna lista posiłków po dodaniu posiłku „obiad”.

## 12. Szczegóły posiłku.

### 12.1. Wyświetlenie listy posiłków (rysunek 23.).



Rysunek 23. Lista posiłków.

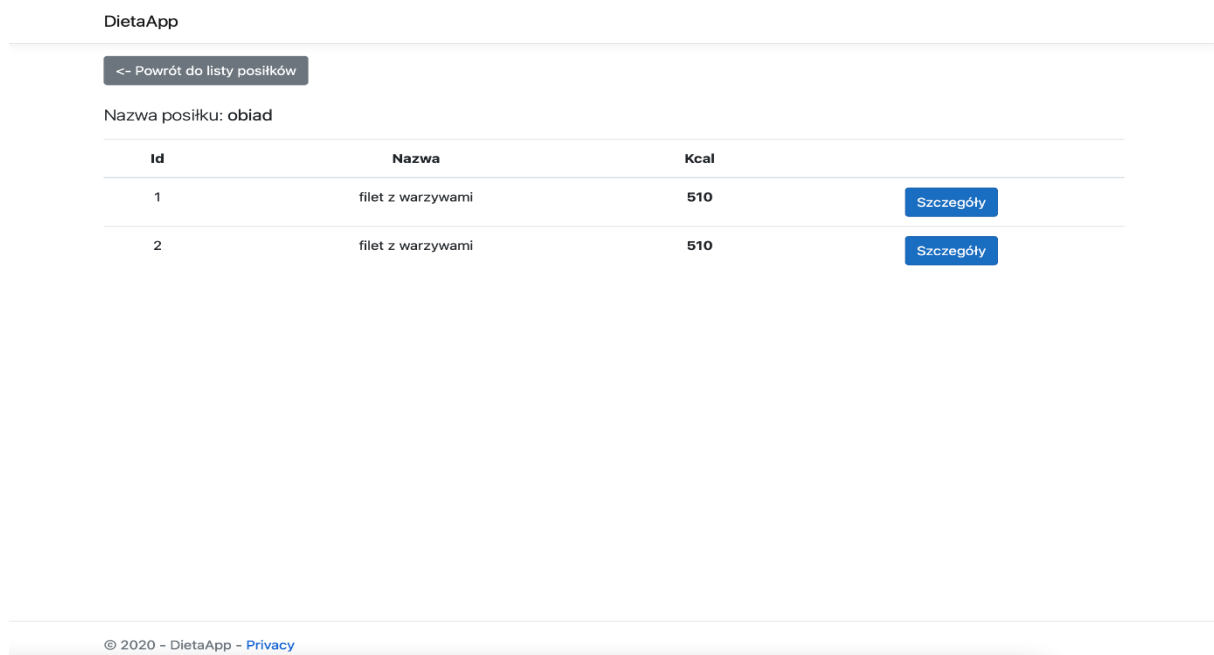
### 12.2. Dane wejściowe:

Id posiłku: 1.

Nazwa posiłku: obiad,

12.3. Oczekiwany rezultat: Wyświetlenie szczegółów (dań) posiłku o nazwie „obiad” i Id = 1.

12.4. Dane wyjściowe zgodne z oczekiwaniami: Wyświetlenie nazwy posiłku „obiad” oraz dań wchodzących w jego skład – 2 x filet z warzywami (510 kcal). Rezultat przedstawiony na rysunku 24.

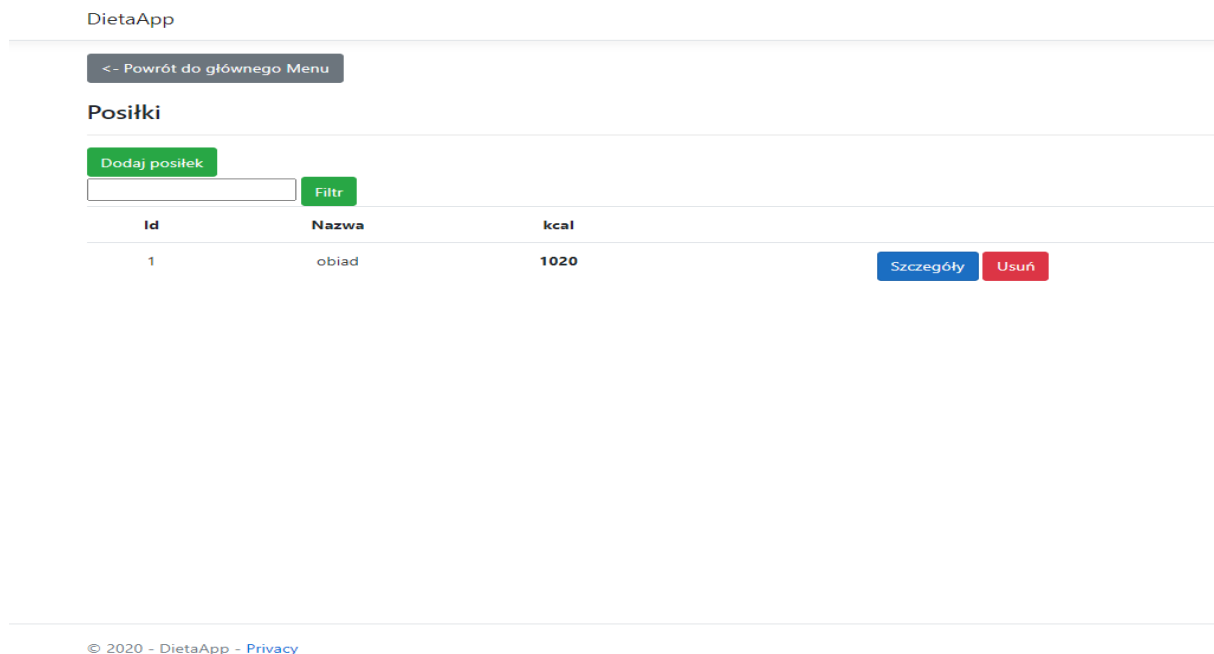


Rysunek 24. Szczegóły posiłku „obiad”.



### 13. Usuwanie posiłku.

#### 13.1. Wyświetlenie listy posiłków przed usunięciem produktu (rysunek 25.).



Rysunek 25. Lista posiłków przed usunięciem.

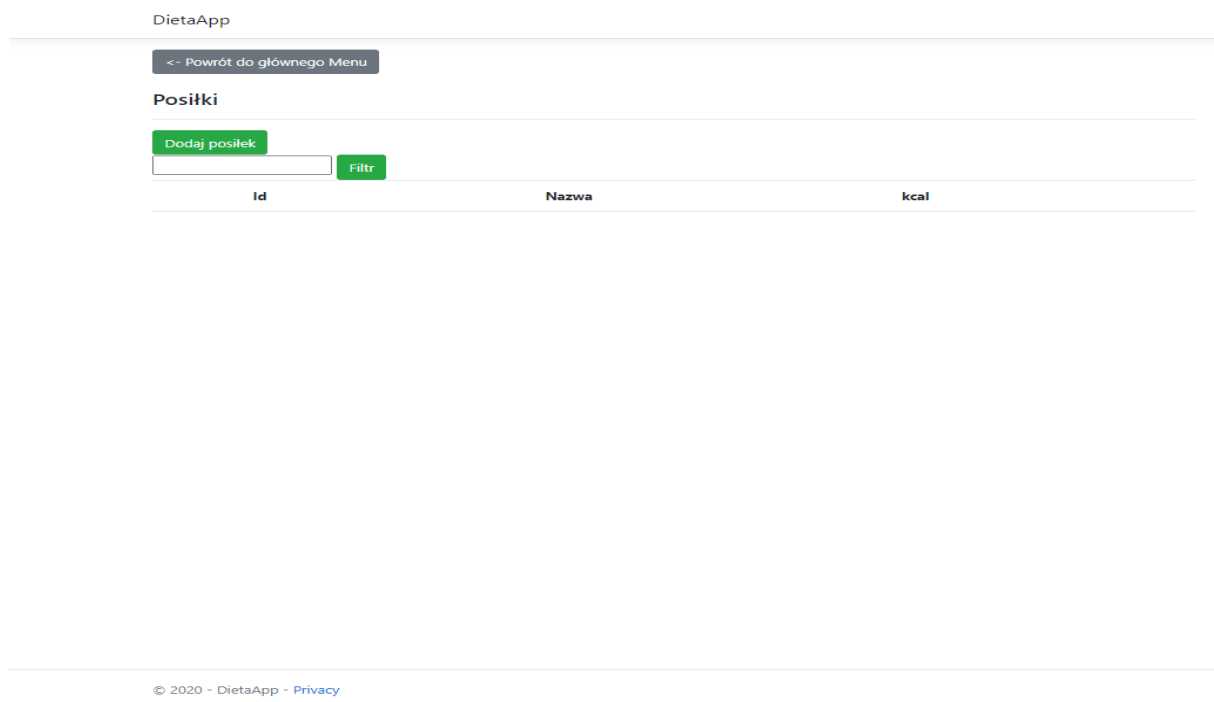
#### 13.2. Dane wejściowe:

Id dania: 1.

Nazwa dania: obiad,

13.3. Oczekiwany rezultat: Posiłek o nazwie „obiad” i Id = 1, usunięty z listy posiłków.

13.4. Dane wyjściowe zgodne z oczekiwaniami: Rezultat przedstawiony na rysunku 26.



Rysunek 26. Rezultat testu usuwania posiłku.

14. Wyświetlenie listy posiłków, które będą wykorzystane w dalszych testach.

14.1. Dane wejściowe: Wprowadzenie dodatkowego posiłku „obiad+deser”.

14.2. Oczekiwany rezultat: Pojawienie się posiłku zgodnego z atrybutami na liście wszystkich posiłków.

14.3. Dane wyjściowe zgodne z oczekiwaniami: aktualna lista z dodanymi posiłkami przedstawiona na rysunku 27.

Id	Nazwa	kcal		
1	obiad	1020	Szczegóły	Usuń
2	obiad + deser	760	Szczegóły	Usuń

*Rysunek 27. Zaktualizowana lista posiłków.*

15. Wyświetlenie listy dni.

15.1. Oczekiwany rezultat: Wyświetlenie pustej listy dni (brak dodanych dni).

15.2. Dane wyjściowe zgodne z oczekiwaniami: rezultat przedstawiono na rysunku 28.

Id	Nazwa
----	-------

*Rysunek 28. Pusta lista dni.*

16. Dodawanie dnia.

16.1. Dane wejściowe:

Nazwa dnia: dwa obiady,

Nazwa posiłku: obiad,

Nazwa posiłku: obiad.

DietaApp

<- Powrót do listy dni

Nazwa dnia diety:  
dwa obiady

+ Dodaj posiłek

obiad

obiad

Usuń

Usuń

Dodaj dzień diety

© 2020 - DietaApp - Privacy

*Rysunek 29. Dane wejściowe testu dodawania dnia.*

16.2. Oczekiwany rezultat: Potwierdzenie dodania dnia oraz widoczność dodanego dnia „dwa obiady” podczas wyświetlenia listy dni.

16.3. Dane wyjściowe zgodne z oczekiwaniami: potwierdzenie dodania dnia przedstawione na rysunku 30 oraz sprawdzenie widoczności dodanego dnia „dwa obiady” na liście dni przedstawione na rysunku 31.

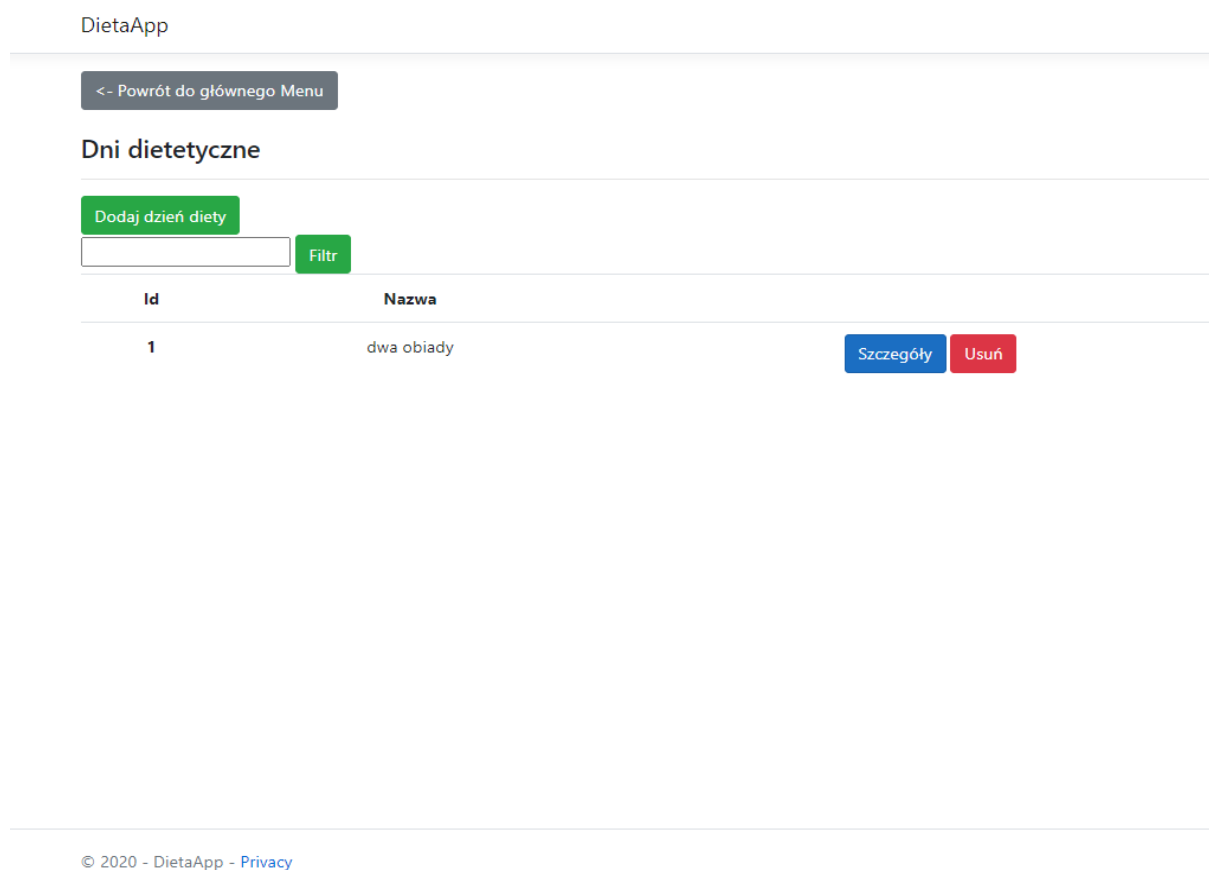
DietaApp

<- Powrót do listy

**Sukces!**  
Dodano dzień dietetyczny.

© 2020 - DietaApp - Privacy

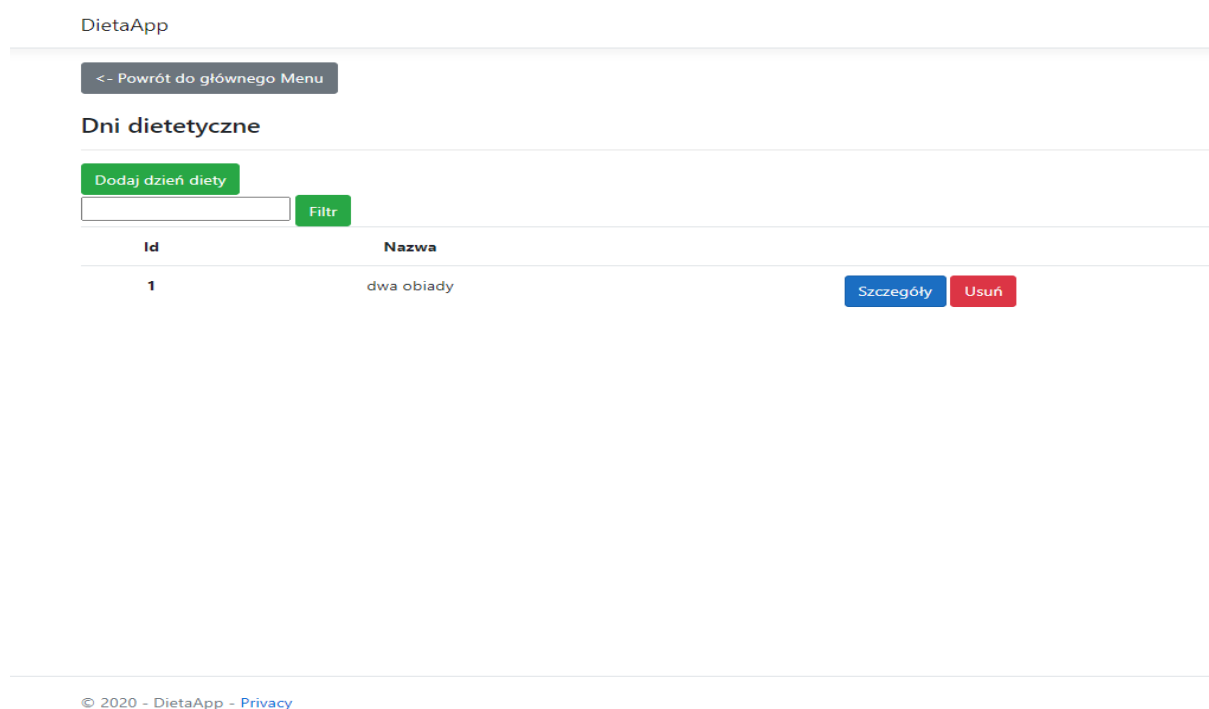
*Rysunek 30. Potwierdzenie dodania dnia.*



*Rysunek 31. Aktualna lista dni po dodaniu dnia „dwa obiady”.*

17. Szczegóły dnia.

17.1. Wyświetlenie listy dni (rysunek 32.).



*Rysunek 32. Lista dni.*

17.2. Dane wejściowe:

Id dnia: 1.

Nazwa dnia: dwa obiady,

17.3. Oczekiwany rezultat: Wyświetlenie szczegółów (posiłków) dnia o nazwie „dwa obiady” i Id = 1.

17.4. Dane wyjściowe zgodne z oczekiwaniami: Wyświetlenie nazwy dnia „obiad” oraz posiłków składających się na wybrany dzień – 2 x obiad. Rezultat przedstawiony na rysunku 33.

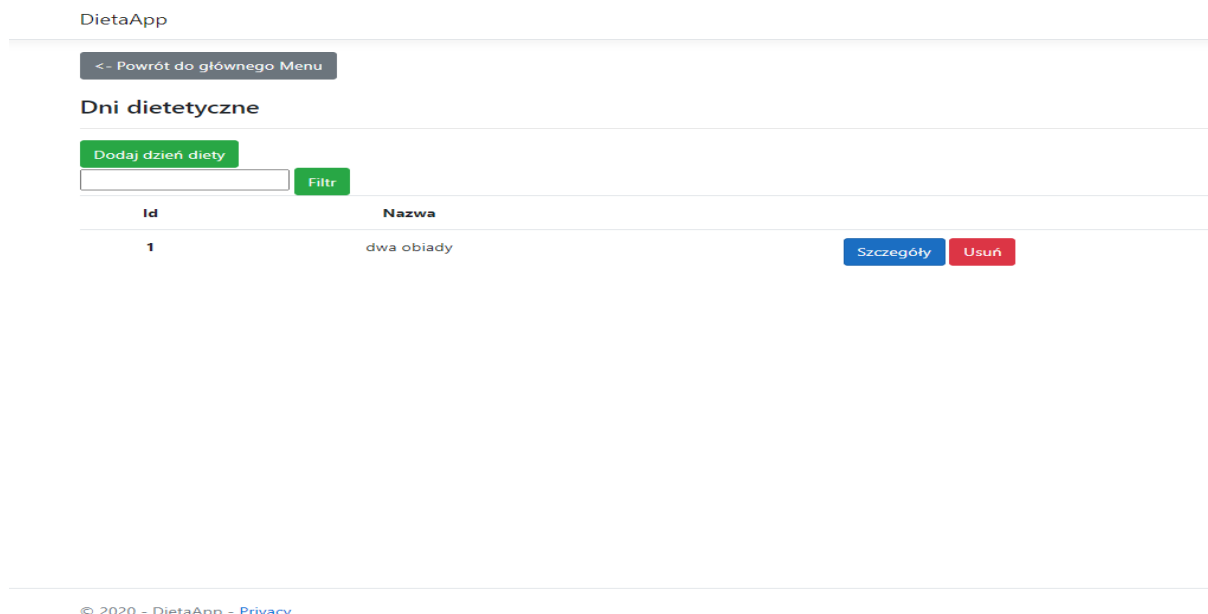
The screenshot shows the DietaApp interface. At the top, there is a header with the text "DietaApp". Below the header, there is a button labeled "<- Powrót do listy dni". Underneath the button, the text "Nazwa dnia: dwa obiady" is displayed. Below this text is a table with two columns: "Id" and "Nazwa". The table contains two rows of data. The first row has "1" in the "Id" column and "obiad" in the "Nazwa" column. The second row has "2" in the "Id" column and "obiad" in the "Nazwa" column. To the right of each row, there is a blue button labeled "Szczegóły". At the bottom of the screen, there is a footer with the text "© 2020 - DietaApp - Privacy".

Id	Nazwa
1	obiad
2	obiad

Rysunek 33. Szczegóły dnia „dwa obiady”.

18. Usuwanie dnia.

18.1. Wyświetlenie listy dni przed usunięciem dnia (rysunek 34.).



Rysunek 34. Lista dni przed usunięciem.

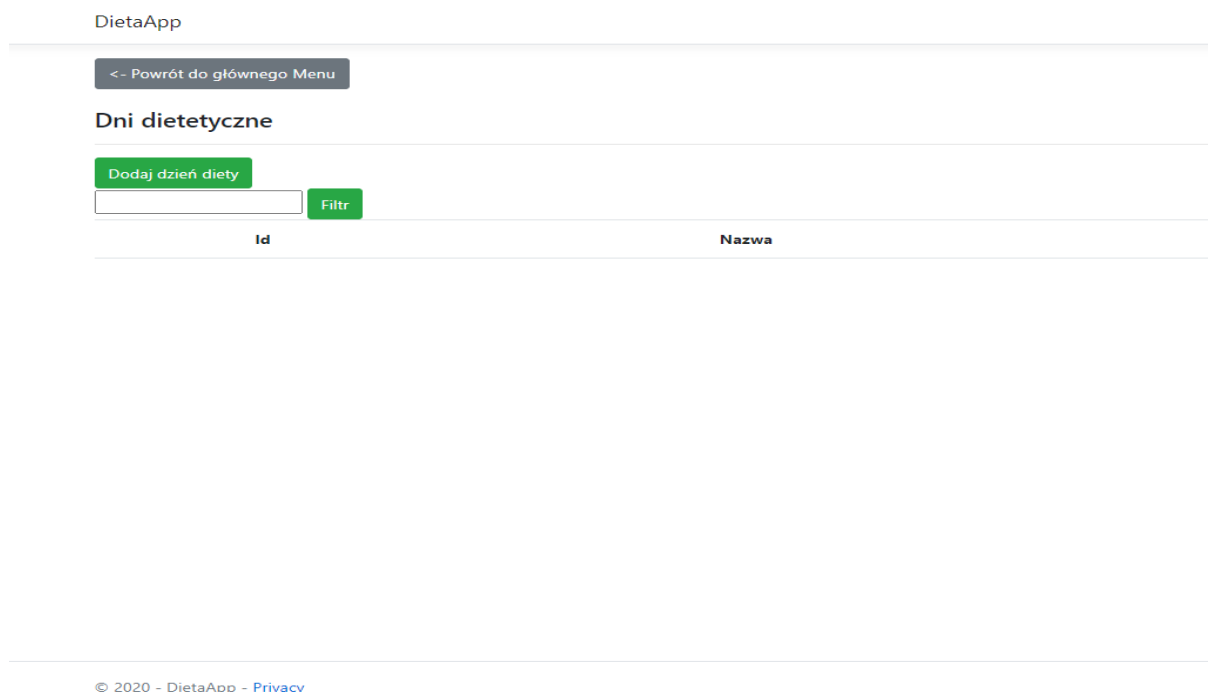
18.2. Dane wejściowe:

Id dnia: 1.

Nazwa dania: dwa obiady,

18.3. Oczekiwany rezultat: dzień o nazwie „dwa obiady” i Id = 1, usunięty z listy dni.

18.4. Dane wyjściowe zgodne z oczekiwaniami: Rezultat przedstawiony na rysunku 35.



Rysunek 35. Rezultat testu usuwania dnia.

19. Wyświetlenie listy dni, które będą wykorzystane w dalszych testach.

19.1. Dane wejściowe: Wprowadzenie dodatkowego dnia „2x obiad + deser”.

- 19.2. Oczekiwany rezultat: Pojawienie się dnia zgodnego z atrybutami na liście wszystkich dni.
- 19.3. Dane wyjściowe zgodne z oczekiwaniami: aktualna lista z dodanymi dniami przedstawiona na rysunku 36.

DietaApp

<- Powrót do głównego Menu

### Dni dietetyczne

**Dodaj dzień diety**

**Filtr**

Id	Nazwa	
1	dwa obiady	<b>Szczegóły</b> <b>Usuń</b>
2	2x obiad+deser	<b>Szczegóły</b> <b>Usuń</b>

© 2020 - DietaApp - [Privacy](#)

*Rysunek 36. Zaktualizowana lista dni.*

20. Wyświetlenie listy dostępnych list zakupów.
- 20.1. Oczekiwany rezultat: Wyświetlenie pustej listy list zakupów (brak dodanych list zakupowych).
- 20.2. Dane wyjściowe zgodne z oczekiwaniami: rezultat przedstawiono na rysunku 37.

DietaApp

<- Powrót do głównego Menu

### Listy Zakupów

**Dodaj listę zakupów**

**Filtr**

Id	Nazwa
----	-------

© 2020 - DietaApp - [Privacy](#)

*Rysunek 37. Pusta lista dostępnych list zakupów.*

## 21. Dodawanie listy zakupów.

### 21.1. Dane wejściowe:

Nazwa listy: Lista numer 1,

Nazwa dnia: 2x obiad+deser,

Nazwa dnia: dwa obiady.

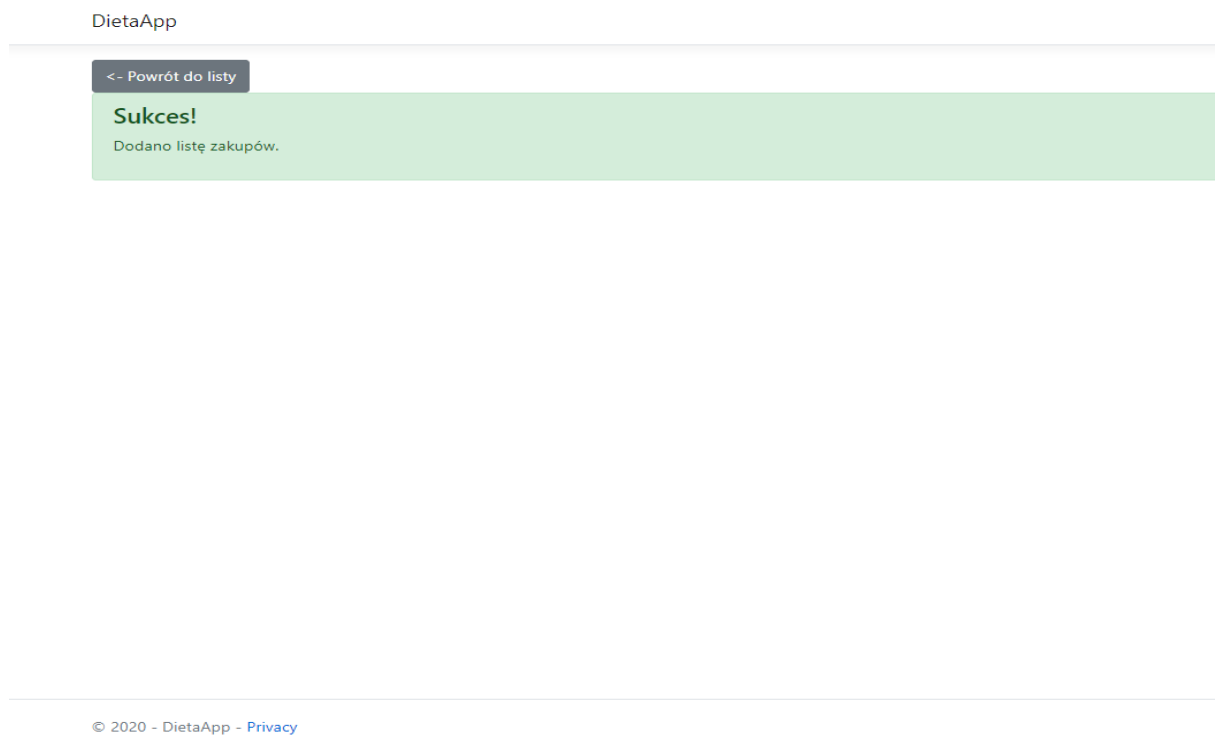
The screenshot shows the DietaApp interface for adding a shopping list. At the top, there is a header with the text "DietaApp". Below the header, there is a button labeled "<- Powrót do list zakupowych". Underneath, there is a label "Nazwa listy zakupów:" followed by a text input field containing "Lista numer 1". Below the input field, there is a blue button labeled "+ Dodaj dzień diety". Underneath, there are two rows of items. The first row has a dropdown menu showing "2x obiad+deser" and a red button labeled "Usuń". The second row has a dropdown menu showing "dwa obiady" and a red button labeled "Usuń". At the bottom, there is a green button labeled "Dodaj listę zakupów". At the very bottom of the screen, there is a footer with the text "© 2020 - DietaApp - [Privacy](#)".

Rysunek 38. Dane wejściowe testu dodawania listy zakupów.

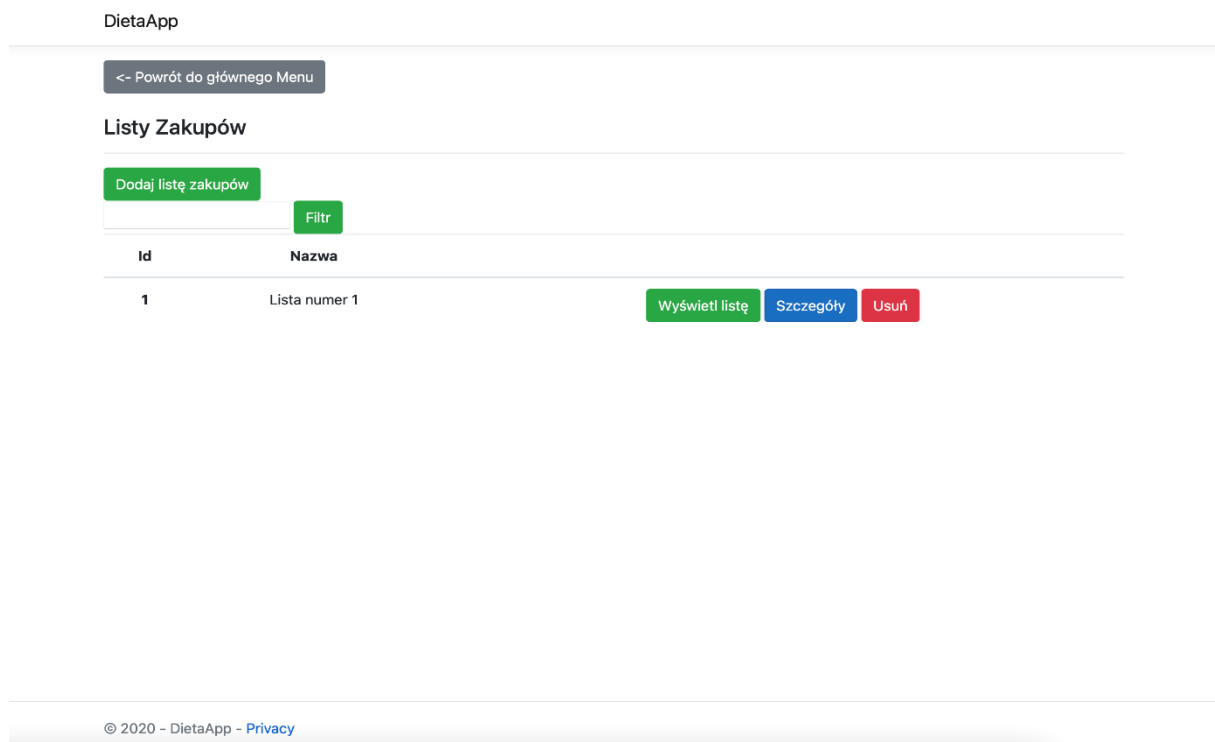
21.2. Oczekiwany rezultat: Potwierdzenie dodania listy zakupów oraz widoczność dodanej listy zakupów „Lista numer 1” podczas wyświetlenia list zakupów.

21.3. Dane wyjściowe zgodne z oczekiwaniami: potwierdzenie dodania listy zakupów przedstawione na rysunku 39 oraz sprawdzenie widoczności dodanej listy zakupów „Lista Numer 1” na liście list zakupów przedstawione na rysunku 40.





*Rysunek 39. Potwierdzenie dodania listy zakupów.*



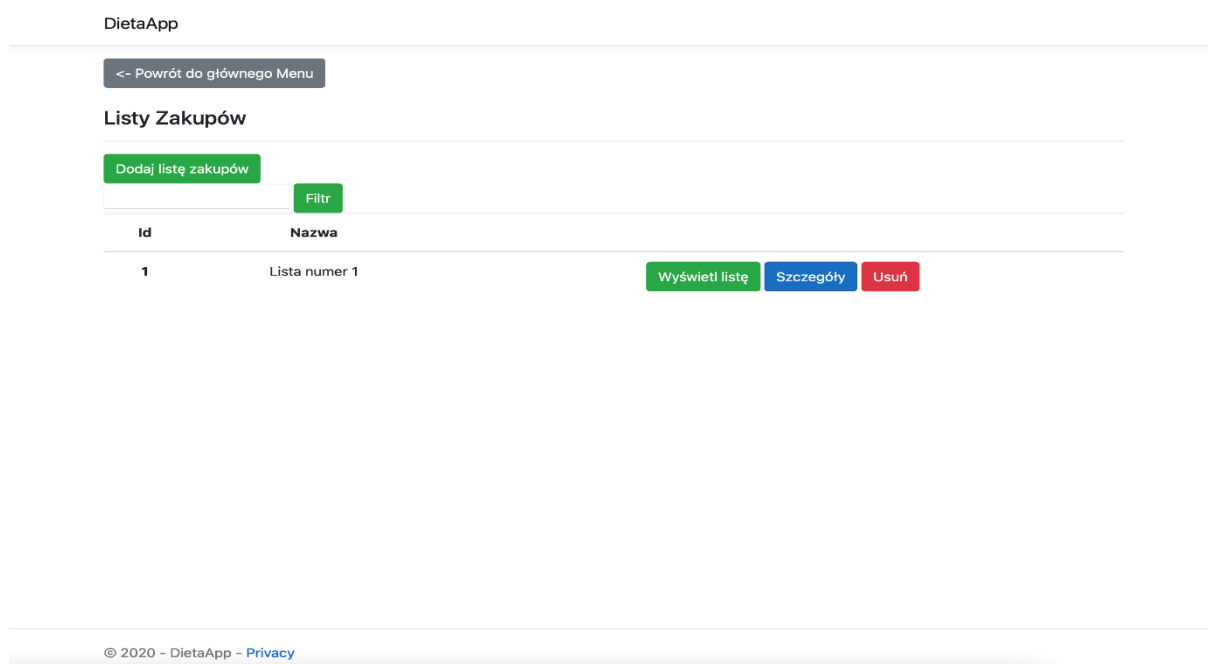
*Rysunek 40. Lista dostępnych list zakupów po dodaniu listy zakupów „Lista numer 1”.*

22. Wyświetlenie listy zakupów.

22.1. Dane wejściowe:

Id listy zakupów: 1,

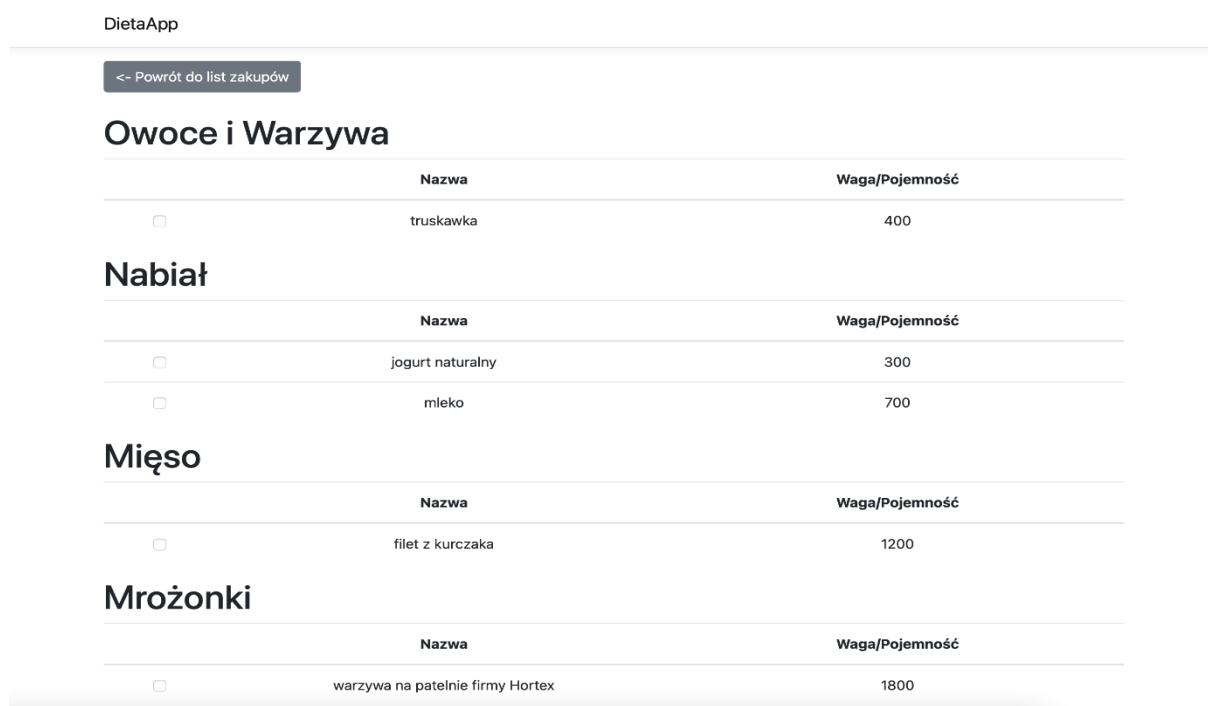
Nazwa listy zakupów: Lista numer 1.



Rysunek 41. Lista dostępnych list zakupów.

22.2. Oczekiwany rezultat: Wyświetlenie dodanej listy zakupów z podziałem na kategorie w których zawarte są produkty z odpowiednio zsumowanymi wagami (w tym przypadku truskawka 400g, jogurt naturalny 300g, mleko 700ml, filet z kurczaka 1200g, warzywa na patelnie firmy Hortex 1800g).

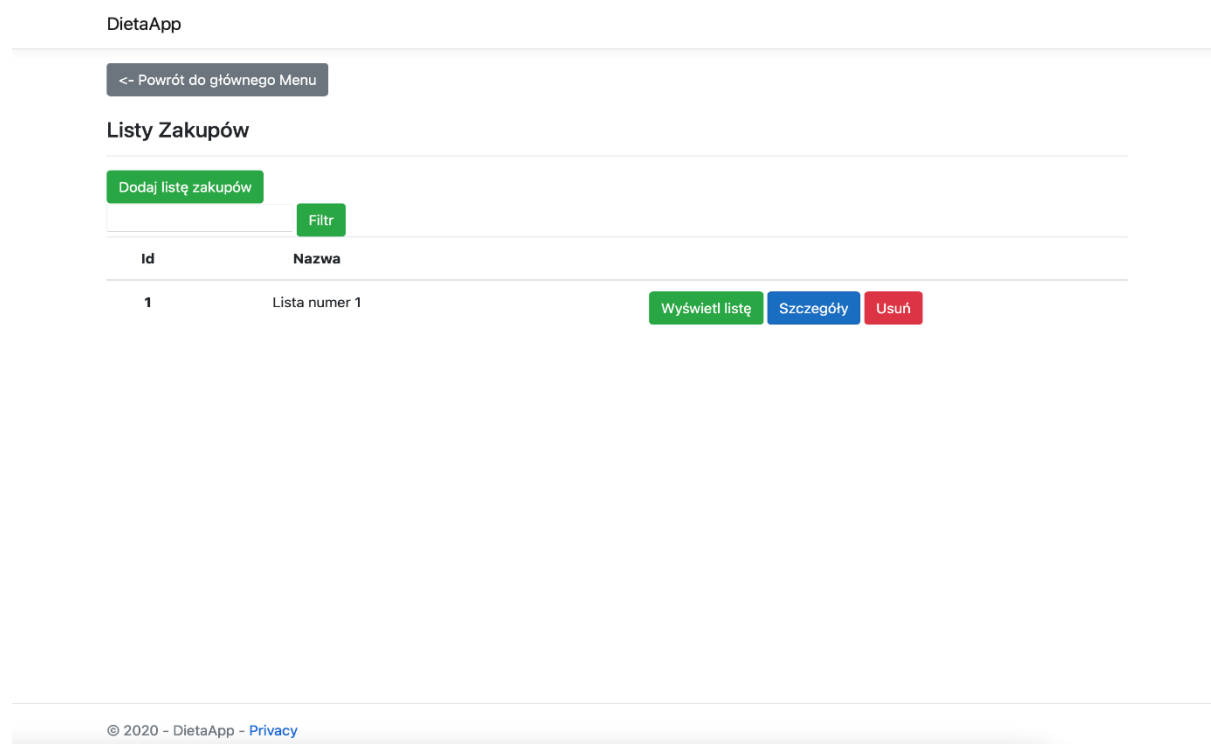
22.3. Dane wyjściowe zgodne z oczekiwaniami: Rezultat przedstawiony na rysunku 42.



Rysunek 42. Wyświetlenie listy zakupów „Lista numer 1”.

## 23. Szczegóły listy zakupów.

### 23.1. Wyświetlenie listy dostępnych list zakupów (rysunek 43.).



*Rysunek 43. Lista dostępnych list zakupów.*

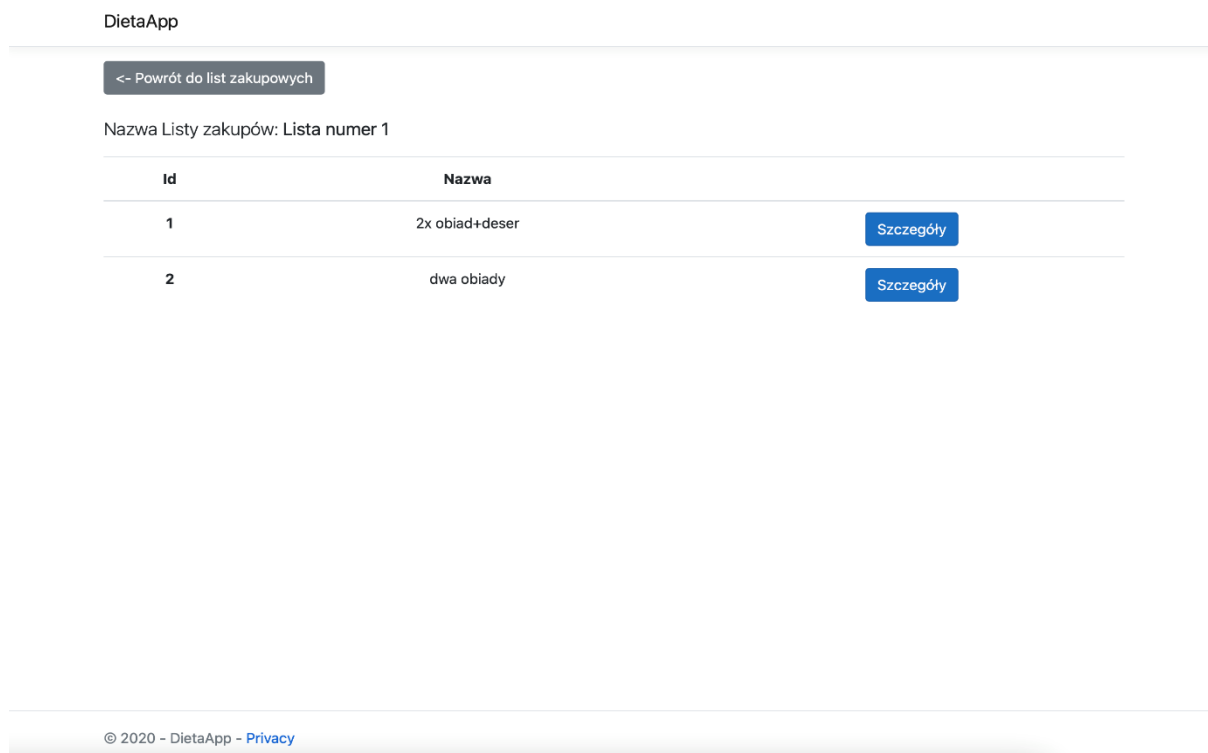
### 23.2. Dane wejściowe:

Id listy zakupów: 1.

Nazwa listy zakupów: Lista numer 1,

23.3. Oczekiwany rezultat: Wyświetlenie szczegółów (dni) listy zakupów o nazwie „Lista numer 1” i Id = 1.

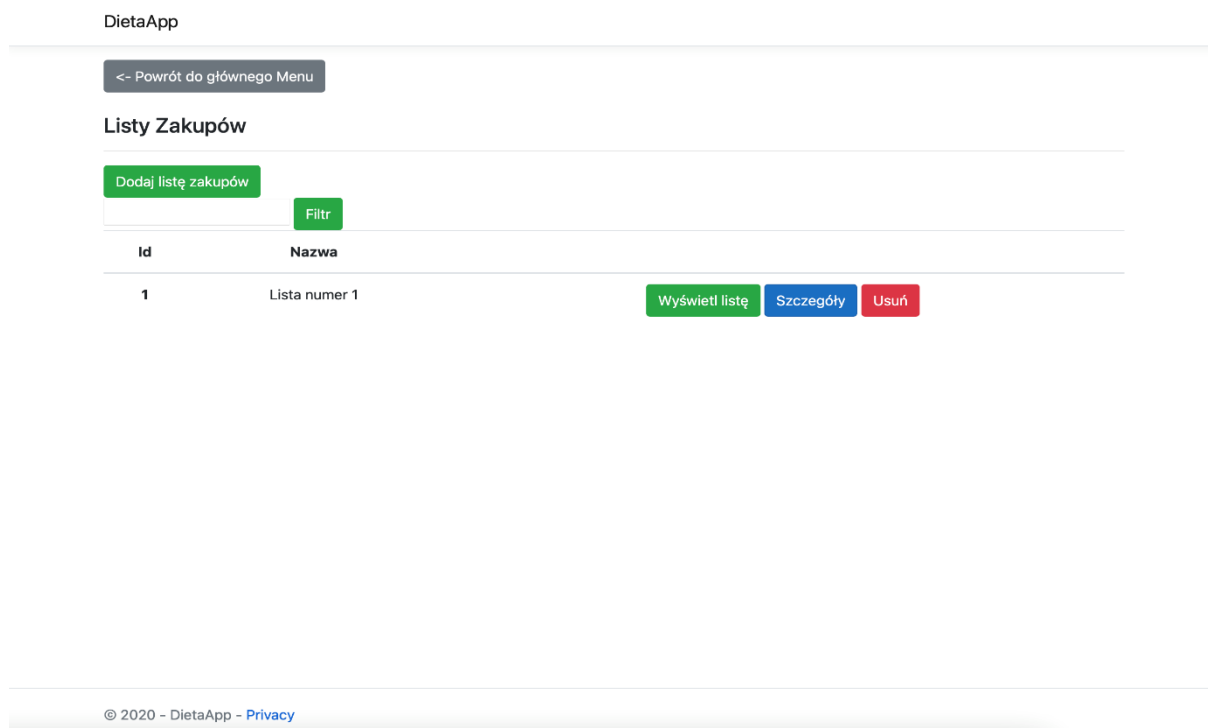
23.4. Dane wyjściowe zgodne z oczekiwaniami: Wyświetlenie nazwy dnia „Lista numer 1” oraz dni składających się na wybraną listę zakupów– 2x obiad+deser, dwa obiady . Rezultat przedstawiony na rysunku 44.



*Rysunek 44. Szczegóły dnia „dwa obiady”.*

## 24. Usuwanie listy zakupów.

24.1. Wyświetlenie listy dostępnych list zakupów przed usunięciem konkretnej listy (rysunek 45.).



*Rysunek 45. Lista dostępnych list zakupów przed usunięciem.*

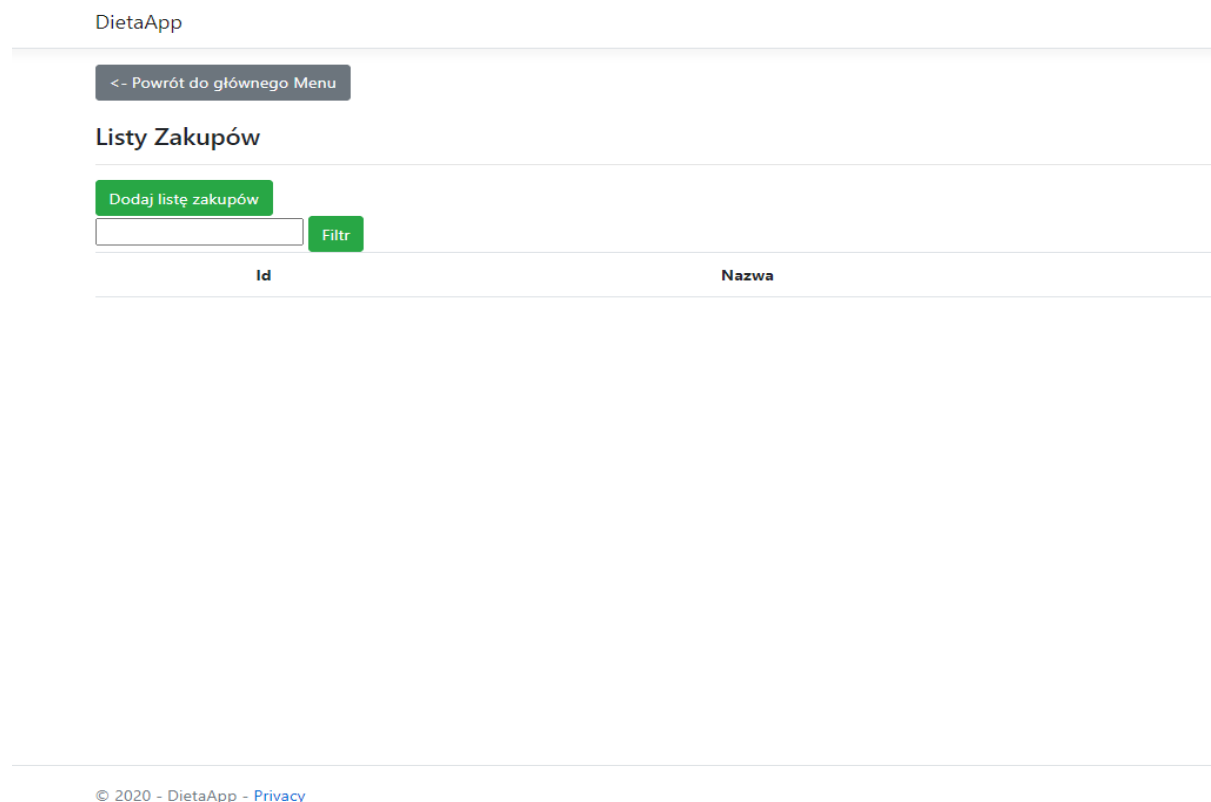
24.2. Dane wejściowe:

Id listy zakupów: 1,

Nazwa listy zakupów: Lista numer 1.

24.3. Oczekiwany rezultat: lista zakupów o nazwie „Lista numer 1” i Id = 1, usunięta z listy dostępnych list zakupów.

24.4. Dane wyjściowe zgodne z oczekiwaniami: Rezultat przedstawiony na rysunku 46.



*Rysunek 46. Rezultat testu usuwania listy zakupów.*

Wyniki testów świadczą o zgodności funkcji zaimplementowanych w aplikacji. Wybór danych niekompletnych jest ignorowany, np. przy tworzeniu posiłku niewpisanie ilości danego produktu lub brak wybranego produktu z listy skutkuje brakiem danego elementu w finalnej wersji posiłku.

## 5. Instrukcje użytkownika

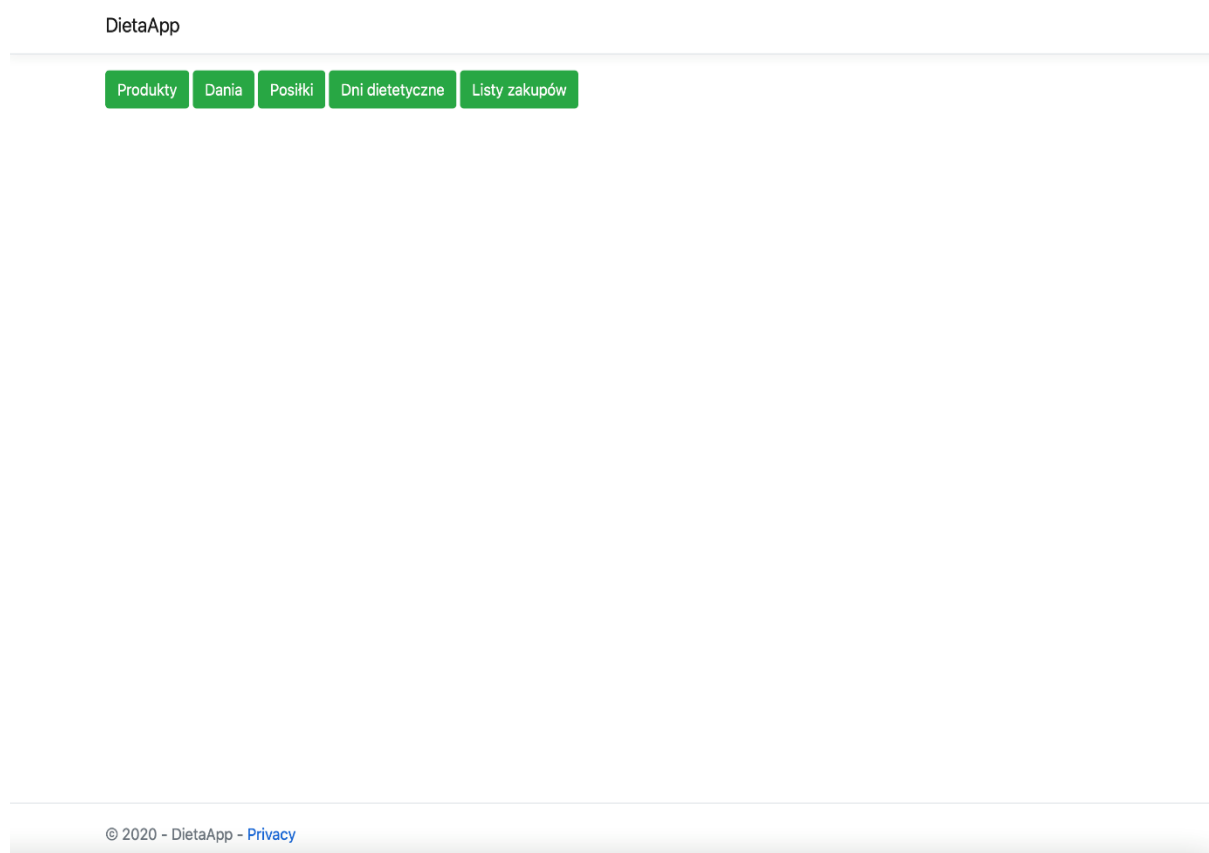
W tym rozdziale na podstawie prostych przykładów opisano możliwości oraz wszystkie kroki, jakie użytkownik musi wykonać, aby uzyskać najważniejszą funkcjonalność aplikacji, jakim jest posegregowana pod względem kategorii lista zakupów. Powielane funkcje np. filtrowanie, zostaną przedstawione tylko w jednej opcji.

Główne elementy aplikacji:

1. Strona startowa (menu główne) aplikacji.
2. Produkty.
2. Dania.
3. Posiłki.
4. Dni dietetyczne.
5. Listy zakupowe.

1. Strona startowa aplikacji.

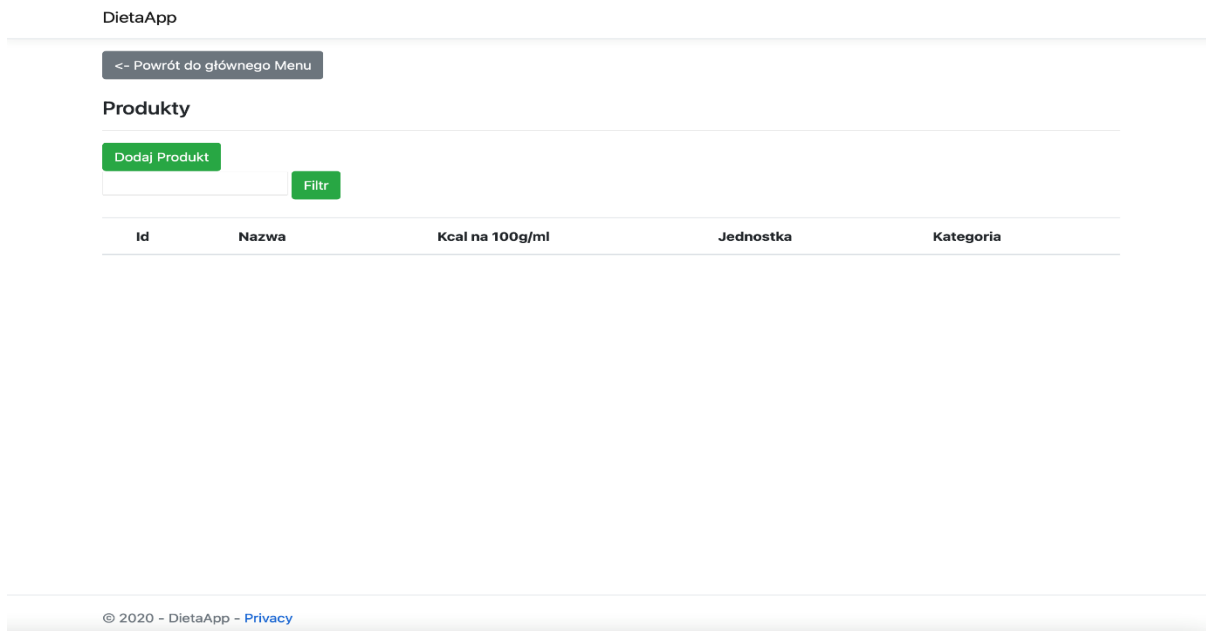
Na rysunku 47. przedstawiono stronę startową (menu główne aplikacji), na której wyświetlone są cztery główne funkcjonalności aplikacji – produkty, posiłki, dni dietetyczne i listy zakupów. Osiągnięcie celu, jakim jest lista zakupowa wymaga przejścia po wszystkich elementach po kolei od lewej strony zaczynając i wypełniając aplikację danymi.



Rysunek 47. Strona startowa (menu główne) aplikacji.

## 2. Produkty.

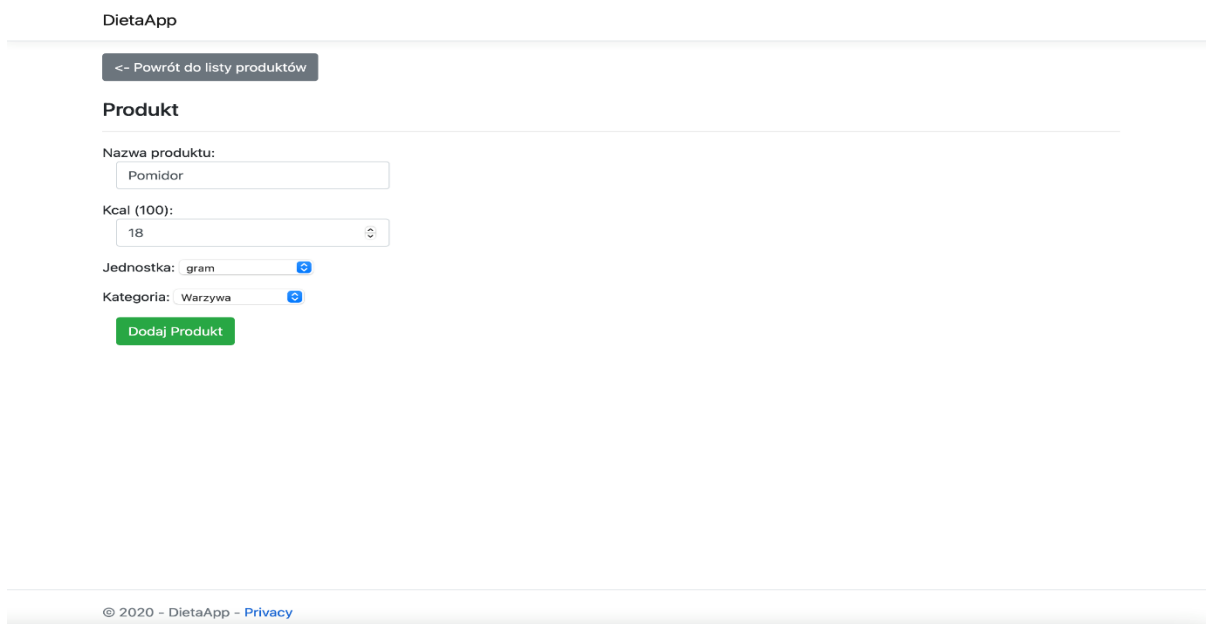
Pierwszą grupą, od której należy zacząć wypełnianie danych, są „Produkty”. Wchodząc w produkty wyświetlana jest lista wszystkich dodanych przez nas produktów. Listę przedstawiono na rysunku 48.



Rysunek 48. Ekran z listą produktów.

Na tym etapie istnieją trzy możliwości:

- „Powrót do głównego menu” powoduje powrót to strony startowej.
- „Dodaj produkt” – wybierając tę opcję ukazuje nam się ekran przedstawiony na rysunku 49, który umożliwia dodanie produktu.



Rysunek 49. Ekran dodawania produktu.

Należy wypełnić odpowiednio nazwę produktu (tekst), Kcal (liczba) oraz wybrać jednostkę i kategorię. Pominięcie wybrania którejś opcji zwróci informację o błędzie i produkt nie zostanie dodany. Przykład takiej walidacji przedstawiono na rysunku 50.

The screenshot shows the 'Produkt' form in the DietaApp. At the top, there is a button '<- Powrót do listy produktów'. Below it, the form fields are: 'Nazwa produktu:' with the value 'Pomidor', 'Kcal (100):' with the value '18', 'Jednostka:' with a dropdown menu showing 'Wybierz Jednostkę' and a red error message 'Wybierz Jednostkę', and 'Kategoria:' with a dropdown menu showing 'Wybierz Kategorię' and a red error message 'Wybierz Kategorię'. At the bottom of the form is a green button 'Dodaj Produkt'. The footer of the app shows '© 2020 - DietaApp - Privacy'.

*Rysunek 50. Ekran informujący o błędnym wprowadzeniu danych produktu.*

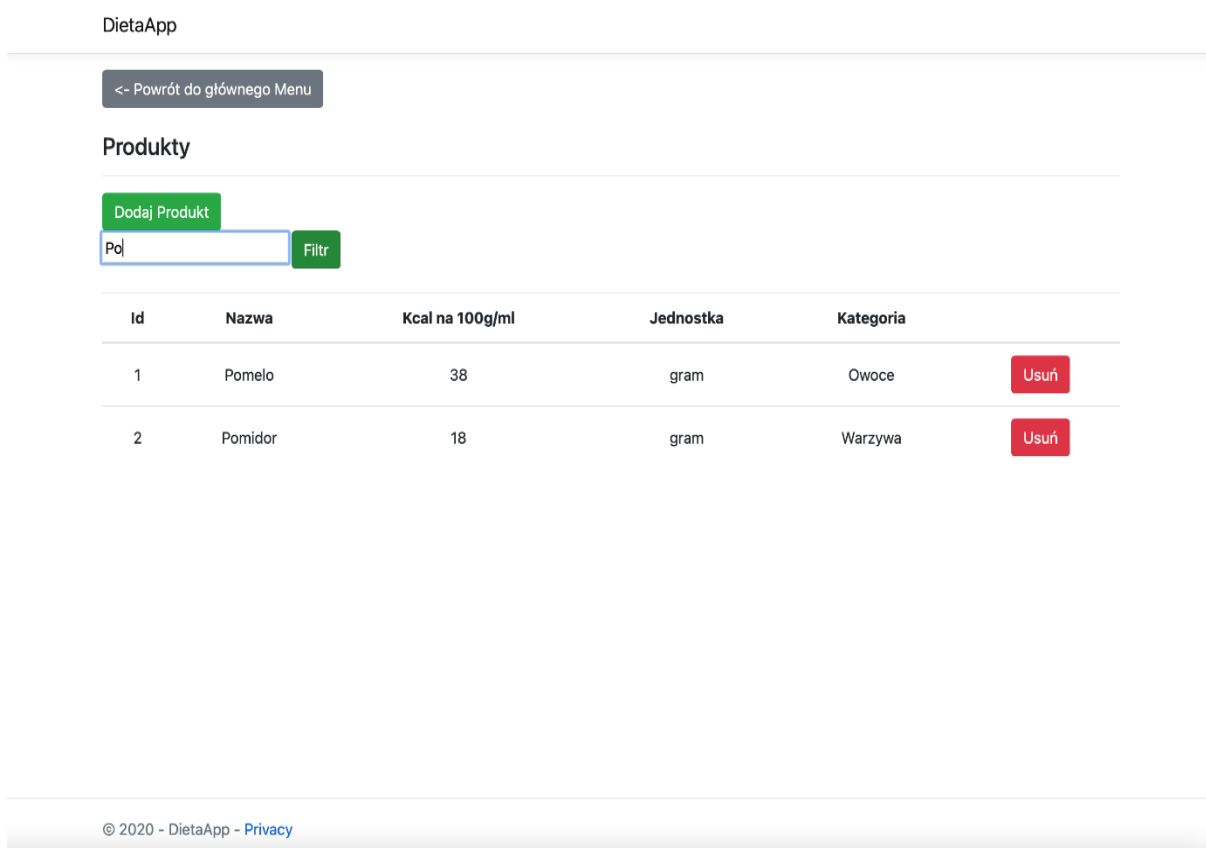
Jeżeli walidacja przeszła poprawnie, to produkt został dodany i widnieje na liście produktów. Potwierdzeniem prawidłowego dodania jest wyświetlenie komunikatu przedstawionego na rysunku 51.

The screenshot shows the DietaApp interface after a successful product addition. At the top, there is a button '<- Powrót do listy'. Below it, a green success message box displays 'Sukces!' and 'Dodano produkt.'. The footer of the app shows '© 2020 - DietaApp - Privacy'.

*Rysunek 51. Ekran potwierdzający dodanie produktu.*



Możliwe jest teraz usunięcie produktu przez naciśnięcie na przycisk „Usuń”.  
 - „Filtr” umożliwi wyselekcjonowanie na ekranie elementów bazując na wprowadzonej nazwie, przykład zaprezentowano na rysunku 52.

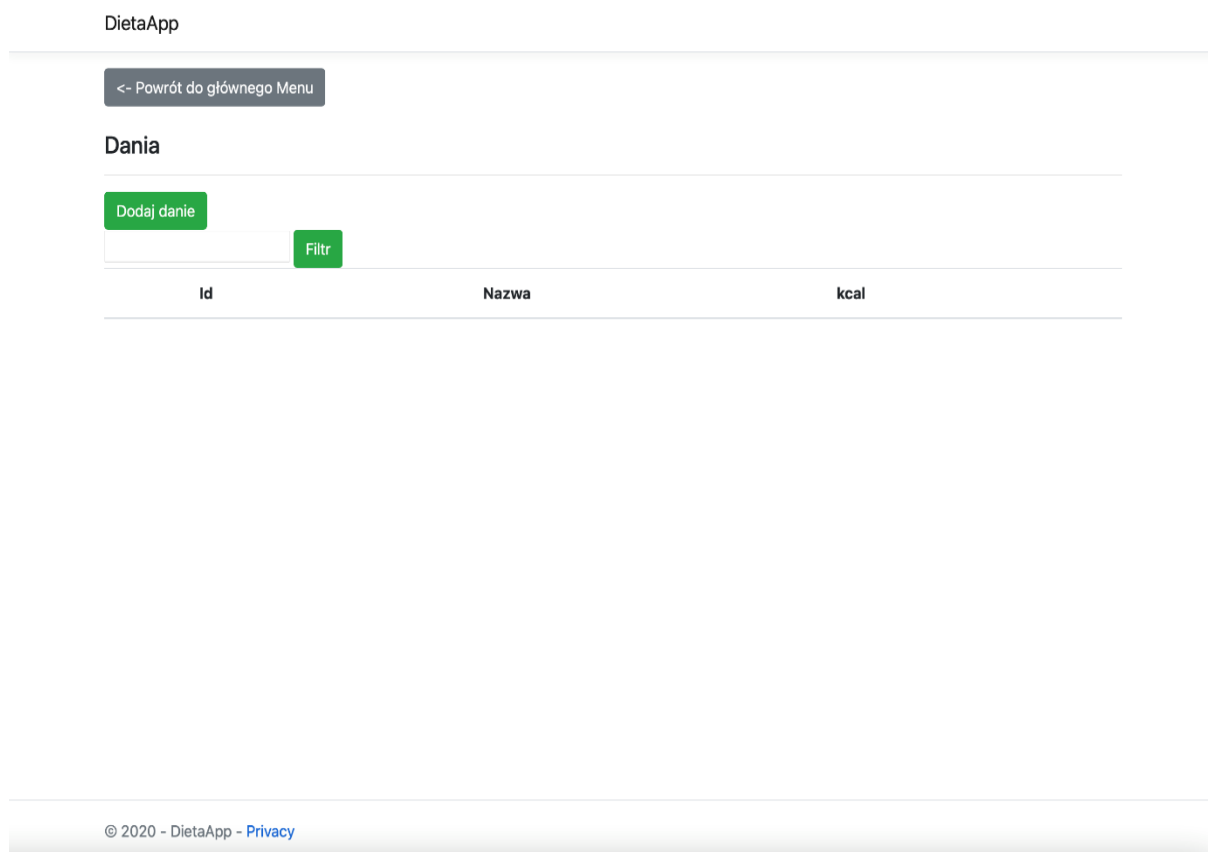


Rysunek 52. Ekran po zastosowaniu funkcji filtrowania „Po”.

- „Usuń” umożliwia usunięcie danego produktu na stałe. Funkcja ta powielana jest na pozostałych funkcjonalnościach (posiłki, dni dietetyczne i listy zakupowe).  
 Należy odpowiednio pododawać produkty według potrzeb. Po dodaniu produktów można przejść do ich wykorzystania podczas tworzenia dań, cofając się do głównego menu i wybierając „Dania”.

### 3. Dania.

Bazując na naszej liście produktów należy utworzyć dania. Z menu głównego należy wybrać opcję „Dania”, co przekieruje nas na listę utworzonych dań, którą przedstawiono na rysunku 53.



*Rysunek 53. Ekran z listą posiłków.*

Ekran z daniami zawiera więcej możliwości w porównaniu do ekranu z produktami. Różni się wygląd i metoda dodawania posiłku oraz przy wyświetlaniu listy dań jest widoczna ich kaloryczność i istnieje dodatkowa opcja „Szczegóły”, która umożliwia wyświetlenie produktów składających się na danie.

- „Dodaj danie” przedstawiono na rysunku 54. Opcje widniejące na ekranie dodawania dania:

- „+Dodaj produkt” powoduje dodanie kolejnego elementu umożliwiającego wybór danego produktu (z listy produktów) i możliwość wpisania jego ilości. Przy każdym elemencie dodanym przez „+Dodaj produkt” widnieje również możliwość usunięcia danego elementu – „Usuń”.
- „Usuń” usuwa poszczególny element.
- „Dodaj danie” dodaje danie do listy dań.

DietaApp

<- Powrót do listy dań

Nazwa dania:  
Sałatka

+ Dodaj produkt

Gruszka	Ilość: 300	Usuń
Pomelo	Ilość: 200	Usuń
Pomidor	Ilość: 100	Usuń

Dodaj danie

© 2020 - DietaApp - [Privacy](#)

Rysunek 54. Ekran dodawania dania.

- Lista dań którą przedstawiono na rysunku 55 zawiera dodatkową opcję dla każdego utworzonego dania – „Szczegóły”.

DietaApp

<- Powrót do głównego Menu

Dania

Dodaj danie

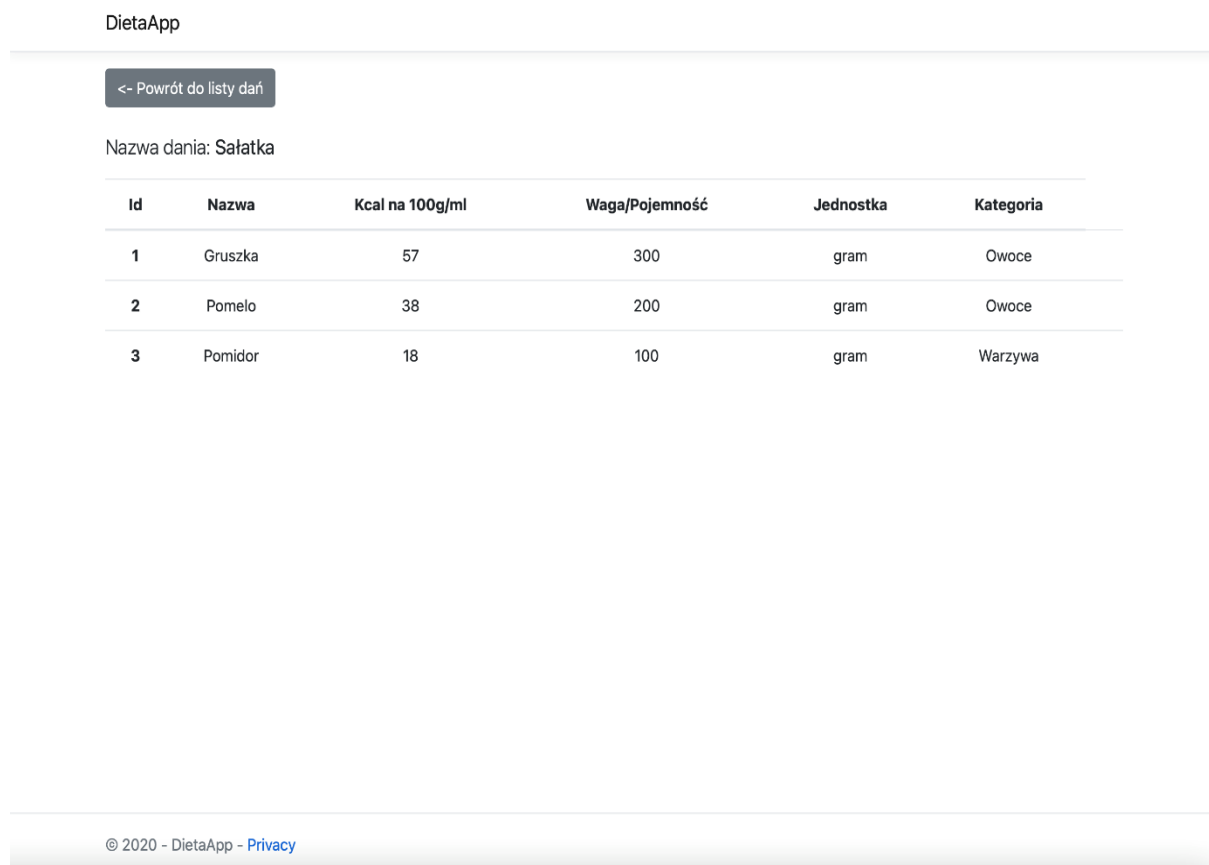
Filtr

Id	Nazwa	kcal	
1	Sałatka	265	<a href="#">Szczegóły</a> <a href="#">Usuń</a>

© 2020 - DietaApp - [Privacy](#)

Rysunek 55. Ekran z listą dań.

- „Szczegóły” umożliwiają wyświetlenie ekranu zawierającego nazwę dania i listę produktów składających się na danie. Efekt wyboru opcji „Szczegóły” przedstawiono na rysunku 56.



The screenshot shows the 'DietaApp' interface. At the top, there is a button labeled '<- Powrót do listy dań'. Below it, the text 'Nazwa dania: Salatka' is displayed. A table lists the ingredients of the dish. The table has six columns: 'Id', 'Nazwa', 'Kcal na 100g/ml', 'Waga/Pojemność', 'Jednostka', and 'Kategoria'. There are three rows of data: 1. Gruszka (57 kcal, 300g, gram, Owoce), 2. Pomelo (38 kcal, 200g, gram, Owoce), and 3. Pomidor (18 kcal, 100g, gram, Warzywa). At the bottom of the screen, there is a footer with the text '© 2020 - DietaApp - [Privacy](#)'.

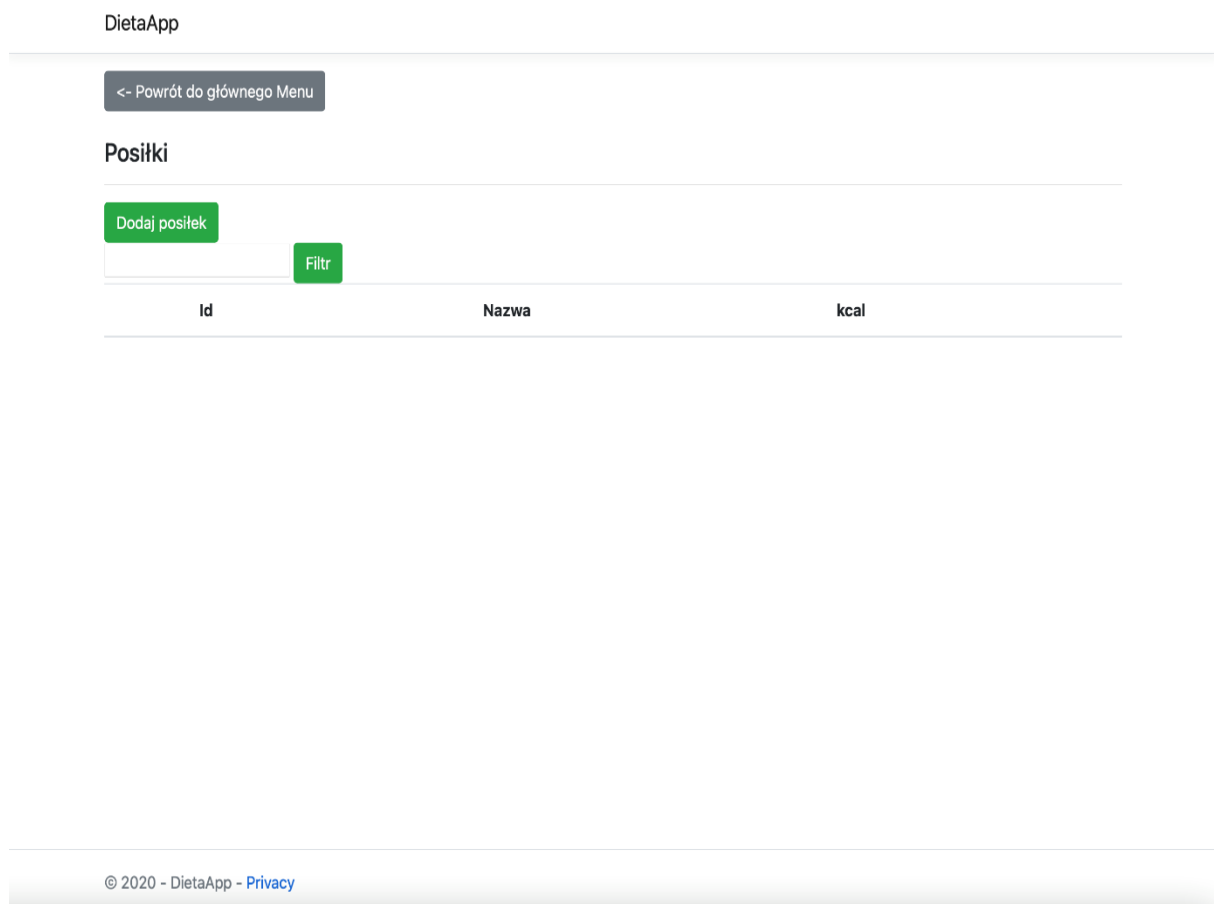
Id	Nazwa	Kcal na 100g/ml	Waga/Pojemność	Jednostka	Kategoria
1	Gruszka	57	300	gram	Owoce
2	Pomelo	38	200	gram	Owoce
3	Pomidor	18	100	gram	Warzywa

*Rysunek 56. Ekran z listą produktów w daniu „Salatka”.*

Po dodaniu dań, następnym krokiem jest utworzenie na ich podstawie posiłków. W tym celu należy powrócić do menu głównego i wybrać opcję „Posiłki”.

#### 4. Posiłki.

Na podstawie utworzonych wcześniej dań można utworzyć posiłki. Aby móc to zrobić, należy z głównego menu wybrać opcję „Posiłki”. Wybór ten przekierowuje użytkownika na listę utworzonych posiłków, którą przedstawiono na rysunku 57.



*Rysunek 57. Ekran z listą posiłków.*

Posiłki dodawane są w podobny sposób jak dania, jednak tutaj do wyboru są wcześniej utworzone dania.

- „Dodaj posiłek” przedstawiono na rysunku 58. Opcje widniejące na ekranie dodawania posiłku:

- „+Dodaj danie” powoduje dodanie kolejnego elementu umożliwiającego wybór danego dania (z listy dań) i możliwość wpisania jego ilości. Przy każdym elemencie dodanym przez „+Dodaj danie” widnieje również możliwość usunięcia danego elementu – „Usuń”.
- „Usuń” usuwa poszczególny element.
- „Dodaj posiłek” dodaje posiłek do listy posiłków.

DietaApp

<- Powrót do listy posiłków

Nazwa posiłku:  
Śniadanie (2x Rogal i sałatka)

+ Dodaj danie

Rogal z almette

Usun

Salatka

Usun

Rogal z almette

Usun

Dodaj posiłek

© 2020 - DietaApp - [Privacy](#)

Rysunek 58. Ekran dodawania posiłku.

- Listę posiłków przedstawiono na rysunku 59. Opcja „Szczegóły” wyświetla posiłki składające się na dany dzień.

DietaApp

<- Powrót do głównego Menu

Posiłki

Dodaj posiłek

Filtr

Id	Nazwa	kcal	
1	Śniadanie (2x Rogal i sałatka)	629	<a href="#">Szczegóły</a> <a href="#">Usun</a>

© 2020 - DietaApp - [Privacy](#)

Rysunek 59. Ekran z listą posiłków.

- „Szczegóły” umożliwiają wyświetlenie ekranu zawierającego nazwę wybranego posiłku i listę dań (możliwość wyboru „Szczegółów” danego posiłku co powoduje wyświetlenie listy produktów) składających się na dany posiłek. Efekt wyboru opcji „Szczegóły” przedstawiono na rysunku 60.

DietaApp

[<- Powrót do listy posiłków](#)

Nazwa posiłku: Śniadanie (2x Rogal i sałatka)

Id	Nazwa	Kcal	
1	Rogal z almette	182	<a href="#">Szczegóły</a>
2	Sałatka	265	<a href="#">Szczegóły</a>
3	Rogal z almette	182	<a href="#">Szczegóły</a>

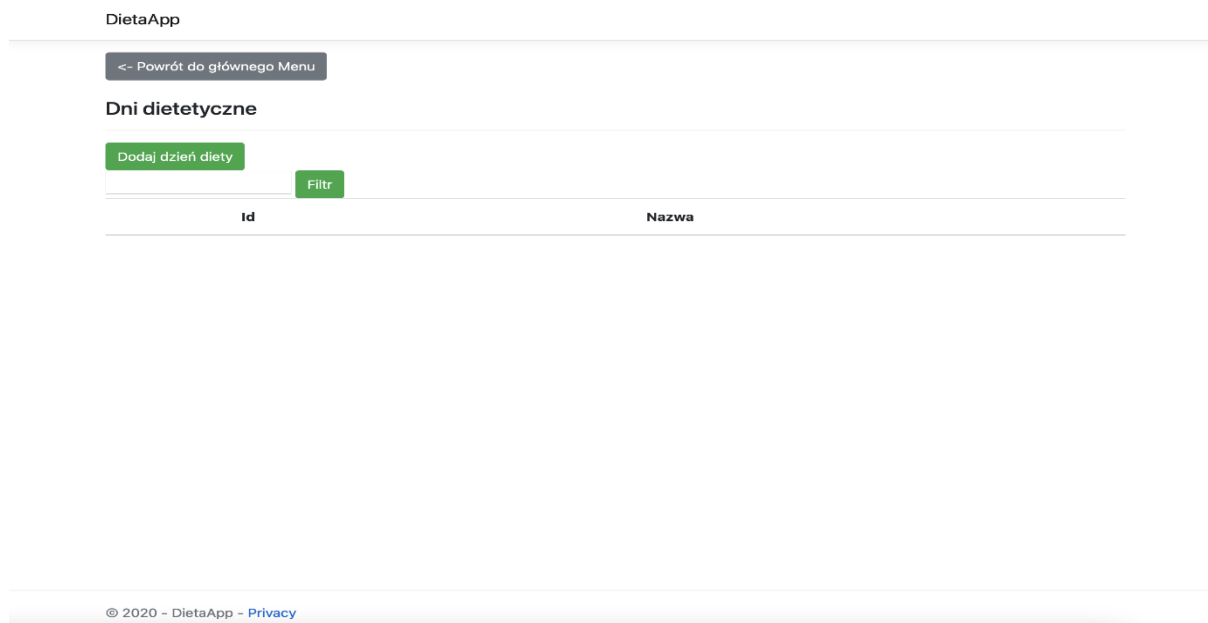
© 2020 - DietaApp - [Privacy](#)

*Rysunek 60. Ekran z listą dań posiłku.*

Po dodaniu posiłków można przejść do ich wykorzystania podczas tworzenia dni dietetycznych, cofając się do głównego menu i wybierając „Dni dietetyczne”.

## 5. Dni dietetyczne.

Na podstawie utworzonych wcześniej posiłków można utworzyć dni dietetyczne. Aby móc to zrobić, należy z głównego menu wybrać opcję „Dni dietetyczne”. Ten wybór przekierowuje użytkownika na listę utworzonych dni dietetycznych, którą przedstawiono na rysunku 61.

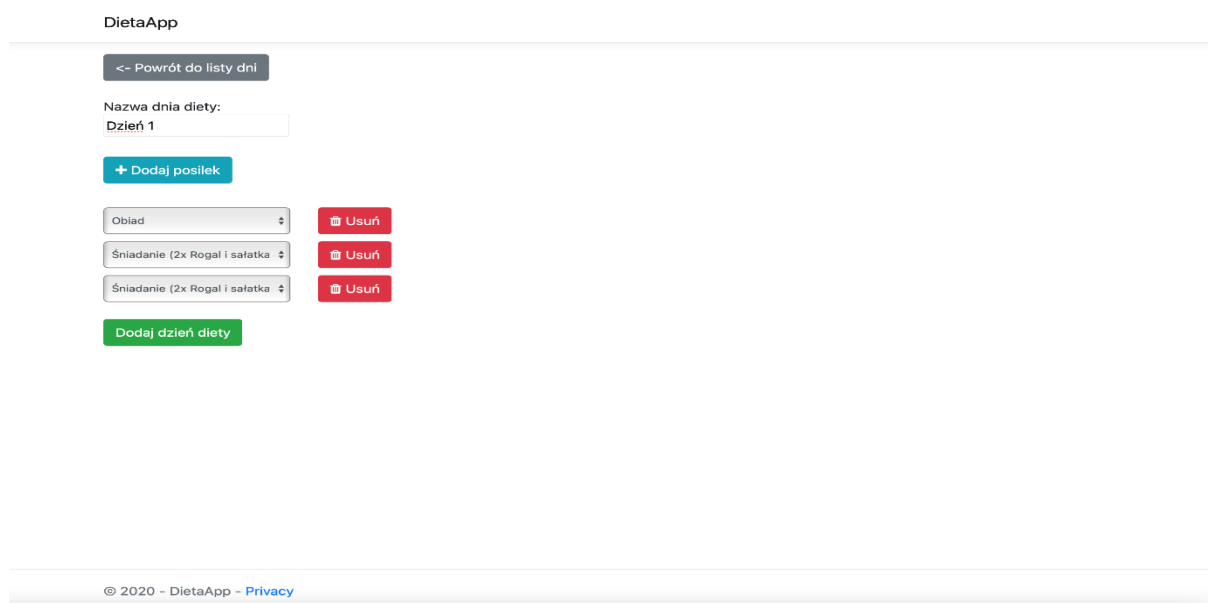


Rysunek 61. Ekran z listą dni dietetycznych.

Dni dietetyczne dodawane są w podobny sposób jak posiłki, jednak tutaj do wyboru są wcześniej utworzone posiłki.

- „Dodaj dzień diety” przedstawiono na rysunku 62. Opcje widniejące na ekranie dodawania dni:

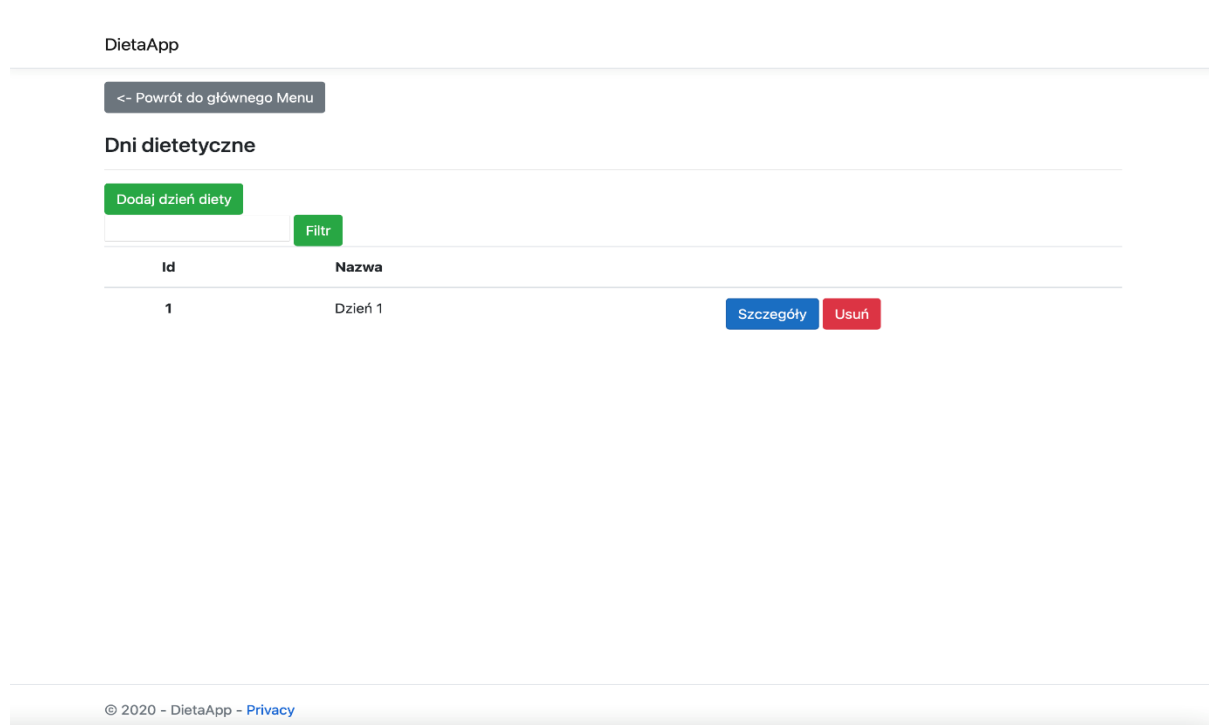
- „+Dodaj posiłek” powoduje dodanie kolejnego elementu umożliwiającego wybór danego posiłku (z listy posiłków). Przy każdym elemencie dodanym przez „+Dodaj posiłek” widnieje również możliwość usunięcia danego elementu – „Usuń”.
- „Usuń” usuwa poszczególny element.
- „Dodaj dzień diety” dodaje dzień do listy dni dietetycznych.



Rysunek 62. Ekran dodawania dnia dietetycznego.

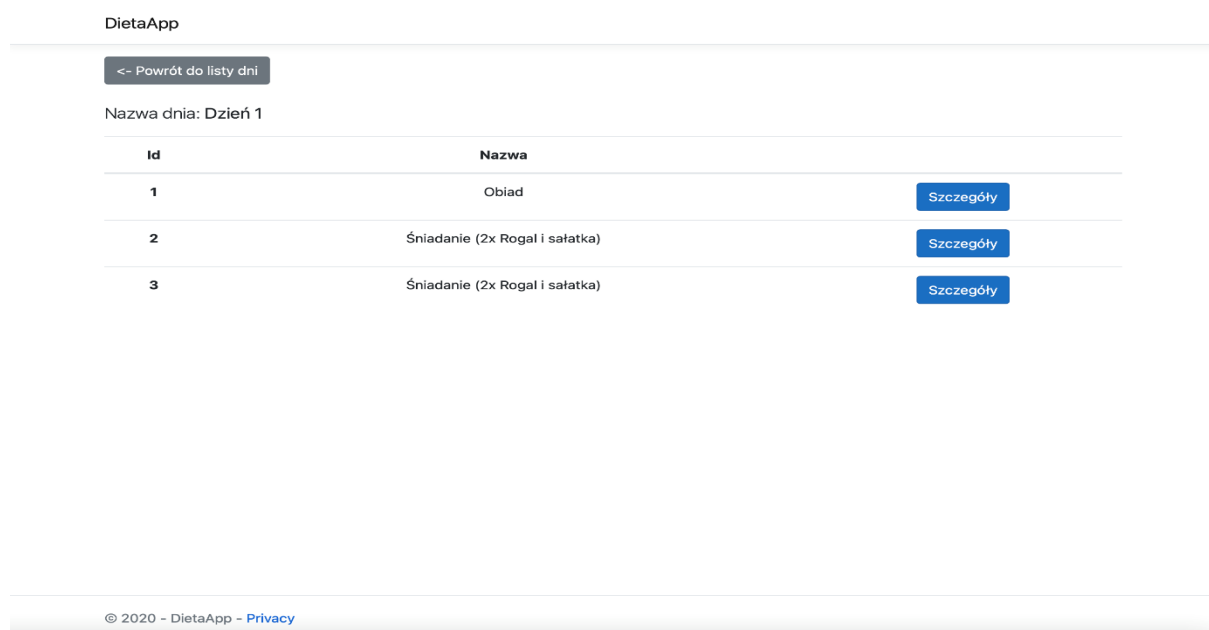


- Listę dni dietetycznych przedstawiono na rysunku 63. Opcja „Szczegóły” wyświetla posiłki składające się na dany dzień.



Rysunek 63. Ekran z listą dni dietetycznych.

- „Szczegóły” umożliwiają wyświetlenie ekranu zawierającego nazwę wybranego dnia dietetycznego i listę posiłków (możliwość wyboru „Szczegółów” danego posiłku co powoduje wyświetlenie listy produktów) składających się na dany dzień. Efekt wyboru opcji „Szczegóły” przedstawiono na rysunku 64.

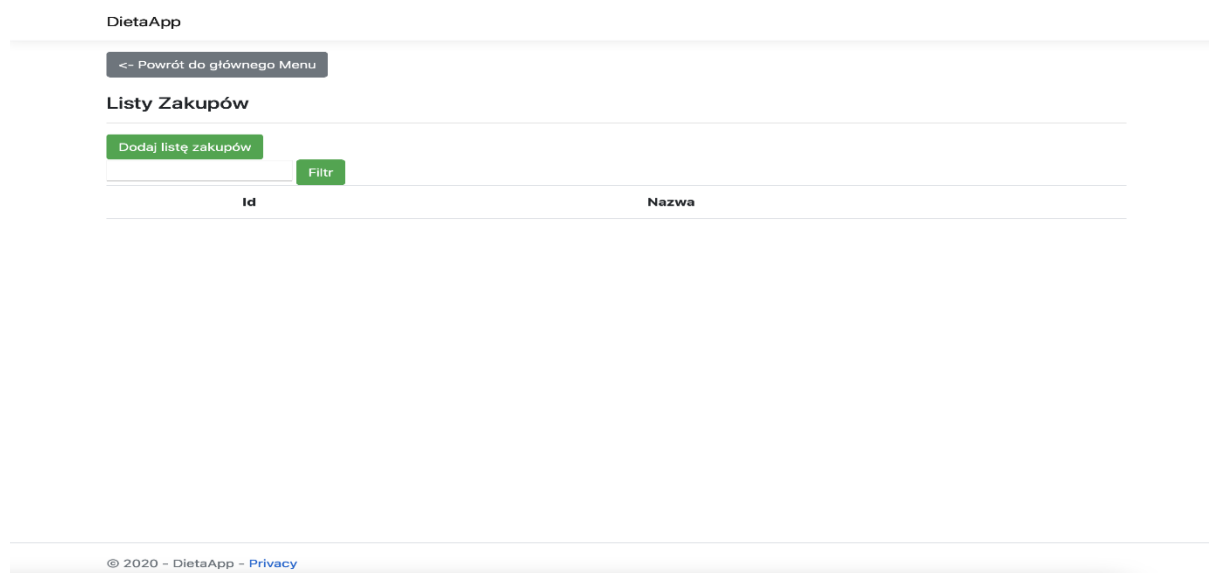


Rysunek 64. Ekran z listą posiłków dnia „Dzień 1”.

Po dodaniu dni dietetycznych, finalny krokiem jest utworzenie na ich podstawie listy zakupów. W tym celu należy powrócić do menu głównego i wybrać opcję „Listy zakupów”.

## 6. Listy zakupów.

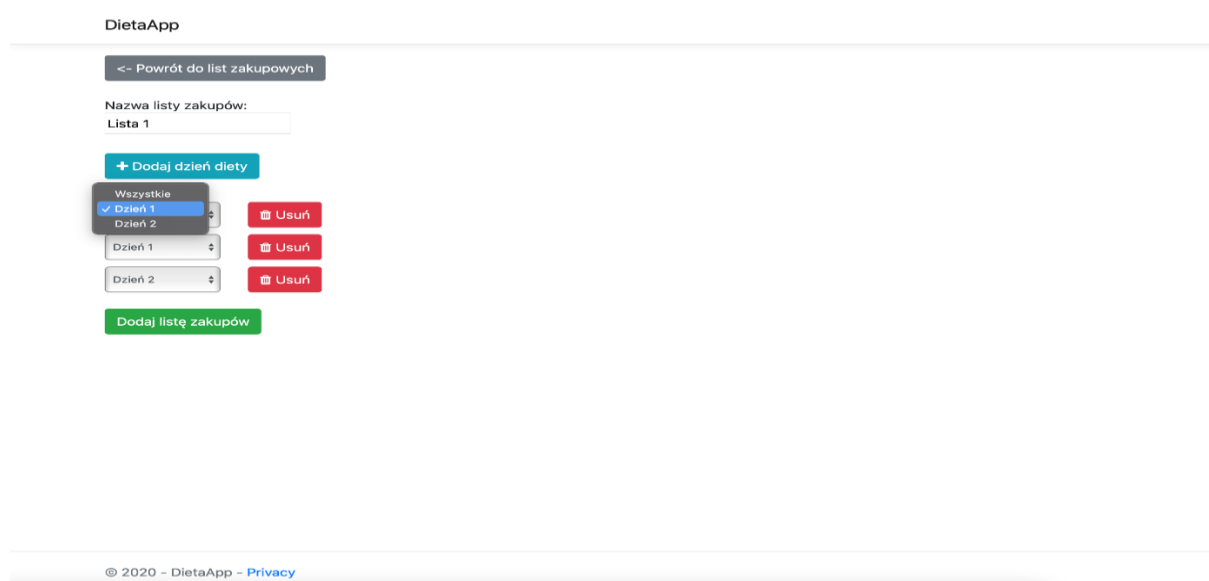
Na podstawie dni dietetycznych można utworzyć listę zakupów, aby móc to zrobić należy z głównego menu wybrać opcję „Listy zakupów”. Ten wybór przekierowuje na listy zakupowe. Ekran z listami zakupów przedstawiono na rysunku 65.



Rysunek 65. Ekran z listami zakupów.

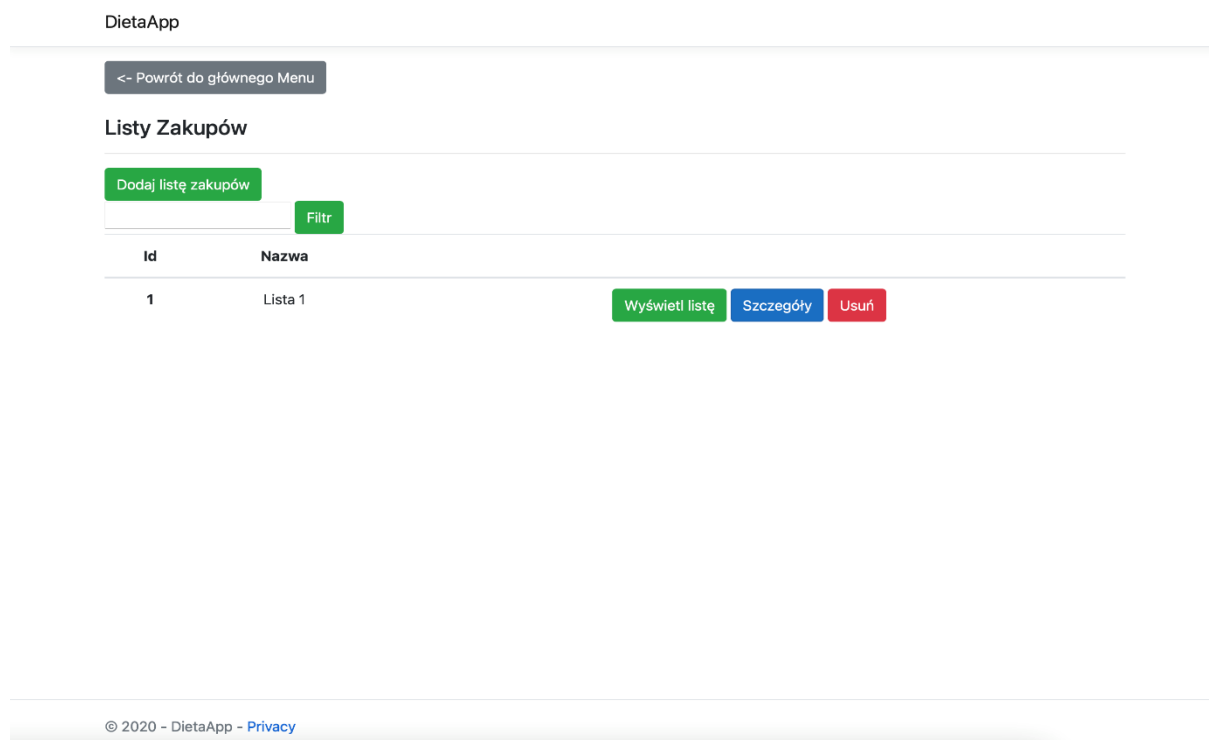
Lista zakupów bazuje na wcześniej utworzonych dniach dietetycznych. Ekran dodawania obsługiwany jest tak samo jak w przypadku dodawania dni dietetycznych z tym, że do wyboru są dni.

- „Dodaj listę zakupów” przedstawiono na rysunku 66.



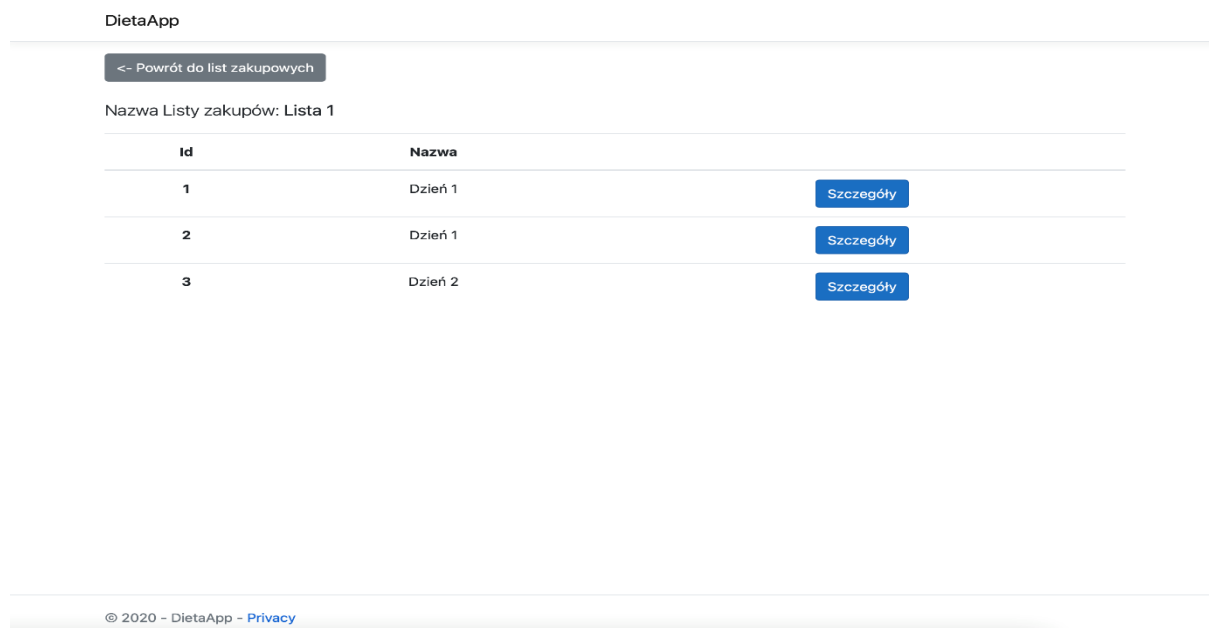
Rysunek 66. Ekran dodawania listy zakupów.

- Listę dni dietetycznych przedstawiono na rysunku 67. Opcja „Szczegóły” wyświetla dni składające się na listę zakupów (możliwość wejścia „głębiej” w szczegóły dnia), natomiast wybór „Lista Zakupów” wyświetli posegregowaną według kategorii listę zakupów.



Rysunek 67. Ekran z dniami składającymi się na listę zakupów.

- „Szczegóły” przedstawiono na rysunku 68, wyświetlone są dni dietetyczne składające się na listę zakupów.



Rysunek 68. Ekran z dniami składającymi się na listę zakupów „Lista 1”.

- „Lista zakupów” to opcja, która wyświetla posegregowaną listę produktów potrzebną do przygotowania wszystkich posiłków składających się na dni dietetyczne dodane do listy. Na rysunku 69. przedstawiono wyświetloną listę zakupów.

DietaApp

<- Powrót do list zakupów

### Owoce i Warzywa

	Nazwa	Waga/Pojemność
<input type="checkbox"/>	Gruszka	1500
<input type="checkbox"/>	Pomelo	1000
<input type="checkbox"/>	Pomidor	650

### Nabiał

	Nazwa	Waga/Pojemność
<input type="checkbox"/>	Almette śmietankowe	100

### Mięso

	Nazwa	Waga/Pojemność
<input type="checkbox"/>	Filet z kurczaka	600

### Pieczynwo

	Nazwa	Waga/Pojemność
<input type="checkbox"/>	Rogal maślany	500

Rysunek 69. Ekran z listą posiłków dnia „Dzień 1”.

Po wykonaniu opisanych w tym rozdziale kroków, użytkownik otrzymuje posegregowaną według kategorii listę zakupów, na której może zaznaczać „checkboxa” w celu oznaczenia produktów jako znalezionych.

## 6. Podsumowanie

Niniejszy rozdział stanowi podsumowanie realizacji internetowej – DietaApp. Omówiono wykonane prace oraz ewentualne kierunki rozwoju aplikacji.

Celem było zaprojektowanie oraz zaimplementowanie aplikacji umożliwiającej tworzenie posegregowanych według kategorii list zakupowych. Założenia projektowe zostały całkowicie zrealizowane.

Zrealizowany projekt podyktowany był chęcią zoptymalizowania czasu spędzonego w marketach.

Praca ta nie wyczerpuje w pełni tematu. Aplikacja ma perspektywy rozwoju w wielu kierunkach, np.:

- system rejestracji i logowania,
- zmiana wyglądu aplikacji,
- wdrożenie nowych funkcjonalności,
- zrobienie aplikacji mobilnej.

## Bibliografia

1. J. Duckett, HTML i Cs. Zaprojektuj i zbuduj witrynę WWW, Helion, Warszawa 2011.
2. J. Ejdys, U. Kobylińska, A. Lulewicz-Sas,, Zintegrowane systemy zarządzania jakością, środowiskiem i bezpieczeństwem pracy, Oficyna Wydawnicza Politechniki Białostockiej, Białystok 2012.
3. E. Freeman, E. Robson, Head First JavaScript Programming, Helion, Warszawa 2015.
4. M. Lis, Tworzenie stron www. Praktyczny kurs. Wydanie II, Helion, Gliwice 2012.
5. S. Wrycza, Informatyka ekonomiczna. Podręcznik akademicki, Polskie Wydawnictwo Ekonomiczne, Warszawa 2010.

## Źródła internetowe

6. <https://www.anylist.com/>. [Data uzyskania dostępu: 28.12.2020 r.]
7. <https://docs.microsoft.com/pl-pl/aspnet/core/mvc/overview?view=aspnetcore-5.0>. [Data uzyskania dostępu: 28.12.2020 r.]
8. <https://docs.microsoft.com/pl-pl/dotnet/csharp/getting-started/>. [Data uzyskania dostępu: 28.12.2020 r.]
9. <https://www.plukasiewicz.net/Artykuly/EntityFramework>. [Data uzyskania dostępu: 28.12.2020 r.]
10. <https://visualstudio.microsoft.com/pl/vs/community/>. [Data uzyskania dostępu: 28.12.2020 r.]
11. <http://wazniak.mimuw.edu.pl/index.php?title=Io-3-wyk-Slajd21>. [Data uzyskania dostępu: 28.12.2020 r.]
12. <http://wazniak.mimuw.edu.pl/index.php?title=Io-3-wyk-Slajd23>. [Data uzyskania dostępu: 28.12.2020 r.]
13. <https://www.modestprogrammer.pl/programowanie-zgodne-z-regulam-solid-poradnik-dla-poczatkujacych-programistow>. [Data uzyskania dostępu: 28.12.2020 r.]
14. <https://www.guru99.com/functional-testing.html>. [Data uzyskania dostępu: 28.12.2020 r.]