Dietetyk+

System zarządzania salonem dietetycznym

DOKUMENTACJA TECHNICZNA

Projekt Semestralny
Przedmiot: Bazy Danych

Prowadzący: mgr inż. Aleksander Wojtowicz

Autor: Mateusz Serafin
Nr indeksu: 134969

React | .NET | TypeScript | PostgreSQL

Rzeszów, 21 stycznia 2026

Uniwersytet Rzeszowski

Streszczenie

Niniejsza dokumentacja opisuje aplikację webową dedykowaną do kompleksowego zarządzania gabinetem dietetycznym. System usprawnia codzienną pracę dietetyka poprzez automatyzację kluczowych procesów: od prowadzenia harmonogramu i rejestracji wizyt, przez archiwizację pomiarów, aż po wystawianie zaleceń medycznych. Zintegrowany moduł powiadomień e-mail pozwala na stały kontakt z pacjentem i efektywną organizację czasu pracy.

Projekt został zrealizowany w architekturze klient-serwer z wykorzystaniem technologii .NET 9 na backendzie oraz React 19 z TypeScript na frontendzie. Aplikacja korzysta z bazy danych PostgreSQL 17.

Słowa kluczowe: zarządzanie, zdrowie, dieta, React, .NET, TypeScript, PostgreSQL

Spis treści

Streszczenie **1**

Spis rysunków **4**

Spis tabel **5**

Spis listingów **6**

1 Wprowadzenie **7**

1.1 Kontekst i motywacja 7
1.2 Cel projektu 7
1.3 Zakres projektu 7
1.4 Użyte technologie 8

2 Architektura systemu **9**

2.1 Architektura wysokopoziomowa 9
2.2 Wzorce architektoniczne 9

3 Model danych **10**

3.1 Diagram ERD 10
3.2 Relacje między encjami 10
3.3 Opis tabel 11
3.4 Indexy 12

4 Implementacja kluczowych funkcjonalności **13**

4.1 System autentykacji JWT 13
4.2 Zarządzanie pacjentami 14
4.3 Zarządzanie dietami 15
4.4 Zarządzanie wynikami medycznymi 17
4.5 Zarządzanie wizytami 18
4.6 Zarządzanie zaleceniami medycznymi 20
4.7 Wysyłanie wiadomości mailowych 22

5 Frontend – Architektura React **25**

5.1 Komponenty UI (React-Bootstrap) 25
5.2 Customowe ikonki (React-Icons) 25
5.3 Wizualizacja danych (Recharts) 25
5.4 Responsywność 25

6 Frontend - Kluczowe funkcjonalności **26**

6.1 Wyszukiwarki 26
6.1.1 Wyświetlanie wizyt w danym dniu 26
6.1.2 Wyświetlanie zawartości diety 27
6.1.3 Kalendarz filtrowanie wizyt 28
6.1.4 Filtrowanie pomiarów medycznych 29
6.1.5 Zabepieczenie dostępu do aplikacji 29

7 Zrzuty ekranu **31**
7.1 Ekran logowania 31
7.2 Dashboard 31
7.3 Podstrony 31
7.3.1 /klienci 32
7.3.2 /klient/:id (informacje o kliencie) 32
7.3.3 /klient/:id (wykres bmi) 33
7.3.4 /kalendarz 33
7.3.5 /diety 34
7.3.6 /wizyty 34
7.3.7 /admin 35
7.4 Okna Modalne 35
7.4.1 Rejestracja pacjenta 35
7.4.2 Rejestracja pacjenta na wizytę 36
7.4.3 Zmiana statusu wizyty 36
7.4.4 Dodawanie nowej diety 37
7.4.5 Dodawanie pomiaru medycznego 37
7.4.6 Przypisywanie zaleceń medycznych 38
7.4.7 Tworzenie konta nowego pracownika 39
7.4.8 Edycja ustawień 39
7.4.9 Lista Pracowników 40
7.5 Wygląd wiadomości mailowej 41
7.6 Widok mobilny 42
7.6.1 /dashboard 42
7.6.2 /klienci 43
7.6.3 /kalendarz 44
7.6.4 /diety 45
7.6.5 /wizyty 46

8 Wnioski i dalszy rozwój **47**
8.1 Osiągnięte wyniki 47
8.2 Podsumowanie 47

Spis rysunków

1 Architektura wysokopoziomowa systemu 9
2 Diagram ERD 10
3 Prosty formularz logowania 31
4 Główny ekran aplikacji, zawierający Menu 31
5 Podstrona zawierająca listę klientów 32
6 Panel informacji z danymi klienta 32
7 Panel informacji z danymi klienta 33
8 Kalendarz dietetyka 33
9 Lista diet 34
10 Wizyty 34
11 Panel Administratora 35
12 Rejestrowanie nowego pacjenta 35
13 Rejestracja pacjenta na wizytę 36
14 Zmiana statusu wizyty 36
15 Dodawanie nowej diety do bazy danych 37
16 Dodawanie nowego pomiaru medycznego danego pacjenta do bazy danych . 37
17 Przypisanie zalecenia medycznego do danego pacjenta 38
18 Panel Administratora - tworzenie konta nowego pracownika 39
19 Panel Administratora - ustawienia aplikacji 39
20 Panel Administratora - lista pracowników 40
21 Panel Administratora - edycja konta pracownika 40
22 Zalecenia żywieniowe dostarczone w wiadomości mailowej do pacjenta . . . 41
23 Wygląd '/home' na urządzeniu mobilnym 42
24 Wygląd '/klienci' na urządzeniu mobilnym 43
25 Wygląd '/kalendarz' na urządzeniu mobilnym 44
26 Wygląd '/diety' na urządzeniu mobilnym 45
27 Wygląd '/wizyty' na urządzeniu mobilnym 46

Spis tabel

1 Stack technologiczny projektu 8

Listings

1 Generowanie tokenu JWT w .NET Core 13
2 Pobieranie danych pacjenta o danym peselu 14
3 Rejestracja pacjenta 14
4 Pobieranie listy wszystkich pacjentów 15
5 Pobieranie listy wszystkich diet 15
6 Pobieranie pliku PDF diety o danym ID 16
7 Dodanie nowej diety do bazy danych 16
8 Dodawanie nowego pomiaru medycznego do bazy danych 17
9 Rejestracja pacjenta na wizytę 18
10 Pobranie wszystkich wizyt, które mają odbyć się danego dnia, które poprowadzić ma dany dietetyk 19
11 Edycja statusu wizyty, o podanym identyfikatorze 20
12 Przypisanie zaleceń medycznych, uwzględniając opis zalecenia medycznego oraz zalecaną przez dietetyka dietę. 20
13 Struktura wiadomości mailowej 23
14 Filtrowanie listy klientów 26
15 Filtrowanie listy diet 26
16 Wizyty do przeprowadzenia w danym dniu 26
17 Wyświetlenie zawartości diety w postaci PDF 27
18 Filtrowanie wizyt w kalendarzu dietetyka 28
19 Filtrowanie wizyt używanych w tabeli, oraz wykresie pomiarów 29
20 Filtrowanie wizyt używanych w tabeli, oraz wykresie pomiarów 29

1 Wprowadzenie

1.1. Kontekst i motywacja

Systemy zarządzania placówkami medycznymi są w tych czasach standardem. Pozwala on w sporym stopniu usprawnić pracę ich pracownikom, a nawet często odciążyć ich z wielu formalności, dzięki czemu mogą skupić się na najważniejszym czyli zdrowiu pacjentów.

1.2. Cel projektu

Głównym założeniem było opracowanie intuicyjnej aplikacji webowej, która zdejmuje z dietetyka ciężar biurokracji i przyspiesza przyjmowanie pacjentów. Dzięki wykorzystaniu zasad Responsive Web Design, system działa niezawodnie na każdym urządzeniu, oferując nowoczesny, czysty interfejs, który nie wymaga długiego wdrożenia.

1.3. Zakres projektu

Projekt obejmuje pełen cykl wytwarzania oprogramowania, od analizy wymagań po wdrożenie na serwerze produkcyjnym.

- **Backend:** REST API w .NET 9 .
- **Frontend:** React 18 z TypeScript
- **Baza danych:** Relacyjna baza PostgreSQL 17

1.4. Użyte technologie

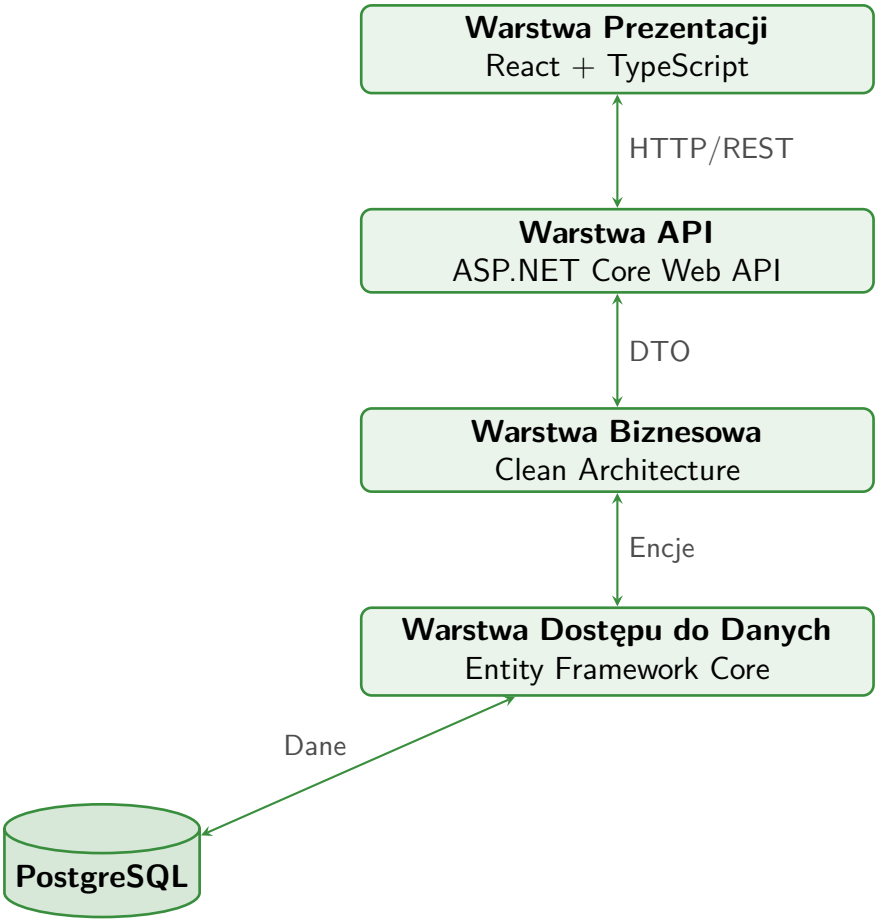
Tabela 1: Stack technologiczny projektu

Kategoria	Technologia	Szczegóły / Uzasadnienie
Backend & Dane		
Platforma	**.NET 10.0**	Najnowsza wersja platformy programistycznej.
REST API	**ASP.NET Core 10.0.1**	Framework do budowy wydajnych usług REST API.
ORM	**EF Core 10.0**	Podejście Code First, wysoka wydajność mapowania.
Baza danych	**PostgreSQL 17**	Zaawansowany, obiektowo-relacyjny silnik SQL.
Szyfrowanie	**BCrypt 4.0.3**	Biblioteka do bezpiecznego hashowania danych wrażliwych.
Komunikacja	**MailKit 4.14.1**	Obsługa i wysyłanie wiadomości e-mail w standardzie SMTP.
Frontend & UI		
Framework	**React 19.0**	Najpopularniejsza biblioteka komponentowa.
Język	**TypeScript 5.3**	Statyczne typowanie zapewniające bezpieczeństwo kodu.
Wykresy	**Recharts**	Responsywne i deklaratywne wizualizacje danych.

2 Architektura systemu

2.1. Architektura wysokopoziomowa

System został zaprojektowany w architekturze trójwarstwowej z wyraźnym podziałem odpowiedzialności, co ułatwia jego rozwój, testowanie i utrzymanie . Komunikacja między klientem a serwerem odbywa się bezstanowo poprzez protokół HTTPS.



Rysunek 1: Architektura wysokopoziomowa systemu

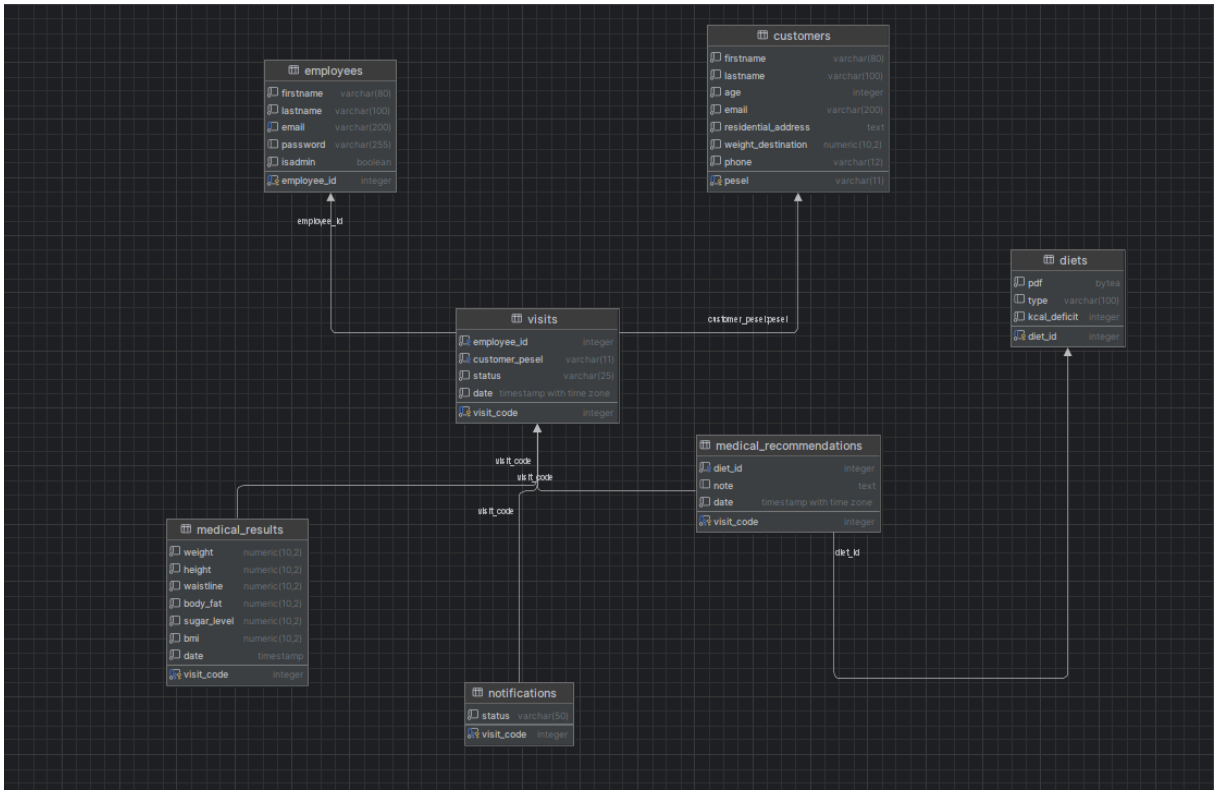
2.2. Wzorce architektoniczne

Projekt backendu opiera się na zasadach **Clean Architecture** zaproponowanych przez Roberta C. Martina , co zapewnia separację logiki biznesowej od szczegółów implementacyjnych (takich jak baza danych czy framework UI). Dodatkowo w projekcie zastosowano następujące wzorce projektowe:

- **JWT (JSON Web Tokens):** Otwarty standard (RFC 7519) wykorzystywany do bezpiecznej, bezstanowej autentykacji użytkowników i wymiany informacji .

3 Model danych

3.1. Diagram ERD



Rysunek 2: Diagram ERD

3.2. Relacje między encjami

1. **employees (1 → N) visits**
   - FK: visits.employee_id → employees.employee_id
2. **customers (1 → N) visits**
   - FK: visits.customer_pesel → customers.pesel
3. **visits (1 → 1) medical_results**
   - FK: medical_results.visit_code → visits.visit_code
4. **visits (1 → 1) medical_recommendations**
   - FK: medical_recommendations.visit_code → visits.visit_code
5. **diets (1 → 1) medical_recommendations**

- FK: medical_recommendations.diet_id → diets.diet_id

6. **visits (1 → 1) notifications**
   - FK: notifications.visit_code → visits.visit_code

3.3. Opis tabel

1. **employees** - dane pracowników
   - 🔑 **employee_id Integer** → identyfikator
   - **firstname Varchar(80)** → imie
   - **lastname Varchar(100)** → nazwisko
   - **email Varchar(200)** → email (unikalne)
   - **password Varchar(255)** → hasło (hashowane)
   - **isAdmin boolean** → czy ma uprawnienia Admnistratora (domyślnie = false)
2. **customers** - dane klientów
   - 🔑 **pesel Varchar(11)** - identyfikator
   - **firstname Varchar(80)** → imie
   - **lastname Varchar(100)** → nazwisko
   - **email Varchar(200)** → email (unikalne)
   - **age Integer** → wiek
   - **residential_address text** → adres zamiekszania
   - **phone Varchar(12)** → numer telefonu, uwzględniając numer kierunkowy
3. **visits**
   - 🔑 **visit_code integer** → identyfikator wizyty (automatycznie inkrementowany)
   - 🔗 **employee_id Integer** → identyfikator pracownika prowadzęcego wizytę
   - 🔗 **customer_pesel Varchar(11)** → pesel pacjenta, który ma sie stawić na wizytę
   - **status Varchar(25)** → status wizyty (domyślnie = "active")
   - **date timestamptz** → data rozpoczęcia wizyty (automatycznie ustawiana)
4. **diets**
   - 🔑 **diet_id Integer** → identyfikator diety (automatycznie inkrementowany)

- **type Varchar(100)** → rodzaj diety
- **kcal_deficit Integer** → deficyt kaloryczny
- **pdf bytea** → plik pdf z dietą

5. **medical_results**
   - 🔑 **visit_code Integer** → id pomiaru jest takie samo jak wizyty na której zostało przypisane
   - **weight Numeric(10,2)** → waga
   - **height Numeric(10,2)** → wzrost
   - **waistline Numeric(10,2)** → obwód w talii
   - **body_fat Numeric(10,2)** → tkanka tłuszczowa
   - **sugar_level Numeric(10,2)** → poziom cukru
   - **bmi Numeric(10,2)** → bmi liczone automatycznie przez backend
   - **date timestamp** → data wykonania pomiaru
6. **medical_recommendations**
   - 🔑 **visit_code Integer** → id zalecenia jest takie samo jak wizyty na której zostało przypisane
   - 🔗 **diet_id** → przypisana dieta do zalecenia
   - **note text** → opis zalecenia/notatka
   - **date timestamptz** → data wystawienia zalecenia (automatycznie ustawiana)

3.4. Indexy

- **visits**.ix_visits_employeeid_date - indeks na id pracownika oraz datę wizyty w tabeli Wizyt
- **employees**.employees_email_uindex - indeks na email pracownika

4 Implementacja kluczowych funkcjonalności

4.1. System autentykacji JWT

Baza danych wykorzystana stworzona na potrzeby projektu składa się z 7 tabel. Została znormalizowana do postaci 3NF.

```
public class AuthService
{
    private readonly string _jwtKey;

    public AuthService(IConfiguration configuration)
    {
        _jwtKey = configuration["ApiSettings:MyApiKey"];
    }

    public string GenerateJwt(Employee employee)
    {
        var key = Encoding.UTF8.GetBytes(_jwtKey);

        var tokenHandler = new JwtSecurityTokenHandler();
        var tokenDescriptor = new SecurityTokenDescriptor
        {
            Subject = new ClaimsIdentity(new[]
            {
                new Claim("id", employee.EmployeeId.ToString()),
                new Claim("firstName", employee.firstName),
                new Claim("lastName", employee.lastName),
                new Claim("isadmin", employee.isadmin ? "1" : "0")
            }),
            Expires = DateTime.UtcNow.AddHours(10),
            SigningCredentials = new SigningCredentials(new
    SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha512Signature)
        };

        var token = tokenHandler.CreateToken(tokenDescriptor);
        return tokenHandler.WriteToken(token);
    }
}
```

Listing 1: Generowanie tokenu JWT w .NET Core

4.2. Zarządzanie pacjentami

Zaimplementowana została kompleksowa obsługa pacjentów po stronie API, w którym zrealizowany jest w pełni CRUD (Create, Read, Update, Delete).

```
[HttpGet("{pesel}")]
public async Task<ActionResult<Customer>> GetCustomer(string pesel)
{
    var customer = await _context.Customers
        .Include(c => c.Visits)
        .ThenInclude(v => v.Recomendation)
        .FirstOrDefaultAsync(c => c.pesel == pesel);

    if (customer == null)
    {
        return NotFound();
    }

    return customer;
}
```

Listing 2: Pobieranie danych pacjenta o danym peselu

```
[HttpPost]
public async Task<ActionResult<Customer>> PostCustomer(Customer customer)
{
    if (_context.Customers.Any(c => c.pesel == customer.pesel))
    {
        return Conflict("Klient o tym numerze PESEL już istnieje.");
    }

    if (_context.Customers.Any(c => c.email == customer.email))
    {
        return Conflict("Klient o tym adresie email już istnieje.");
    }

    if (_context.Customers.Any(c => c.phone == customer.phone))
    {
        return Conflict("Klient o tym numerze telefonu już istnieje.");
    }

    _context.Customers.Add(customer);
```

```
    await _context.SaveChangesAsync();

    return CreatedAtAction(nameof(GetCustomer), new { pesel = customer.pesel },
     customer);
}
```

Listing 3: Rejestracja pacjenta

```
[HttpGet]
public async Task<ActionResult<IEnumerable<Customer>>> GetCustomers()
{
    return await _context.Customers
        .Select(c => new Customer (
            c.pesel,
            c.firstName,
            c.lastName,
            c.age,
            c.email,
            c.residentialAddress,
            c.weightDestination,
            c.phone
        ))
        .ToListAsync();
}
```

Listing 4: Pobieranie listy wszystkich pacjentów

4.3. Zarządzanie dietami

Dietetyk ma możliwość w szybki sposób pozyskać z bazy danych wszystkie zapisane w niej diety, a także w szybki sposób dodać, nową dietę do bazy, w postaci pliku PDF.

```
[HttpGet]
public async Task<ActionResult<IEnumerable<DietRecord>>> GetDiets()
{
    return await _context.Diets.Select(e => new DietRecord(
        e.dietId,
        e.type,
        e.kcalDeficit
    )).ToListAsync();
}
```

Listing 5: Pobieranie listy wszystkich diet

```
[HttpGet("{id}/pdf")]
public async Task<IActionResult> GetDietPdf(int id)
{
    var diet = await _context.Diets
                            .FirstOrDefaultAsync(e => e.dietId == id);
    if (diet == null || diet.pdf == null)
    {
        return NotFound();
    }
    return File(diet.pdf, "application/pdf", $"diet_{id}.pdf");
}
```

Listing 6: Pobieranie pliku PDF diety o danym ID

```
[HttpPost]
public async Task<ActionResult<Diet>> PostDiet([FromForm] string type, [
    FromForm] int kcalDeficit, [FromForm] IFormFile pdf)
{
    byte[] pdfData = Array.Empty<byte>();

    if (pdf?.Length > 0)
    {
        using var ms = new MemoryStream();
        await pdf.CopyToAsync(ms);
        pdfData = ms.ToArray();
    }

    var diet = new Diet
    {
        type = type,
        kcalDeficit = kcalDeficit,
        pdf = pdfData
    };

    _context.Diets.Add(diet);
    await _context.SaveChangesAsync();

    return CreatedAtAction(nameof(GetDiet), new { id = diet.dietId }, diet);
```

```
}
```

Listing 7: Dodanie nowej diety do bazy danych

4.4. Zarządzanie wynikami medycznymi

Aplikacja umożliwa dietetykowi przechowywanie wyników medycznych każdego pacjenta. Przyjęte jest, że na jedną wizytę przypada jeden pomiar medyczny. Na pomiar składa się między innymi: (**wzrost, waga, obwód talii, tkanka tłuszczowa, cukier**) oraz bmi, automatycznie liczone przez aplikację według wzoru:

$$\text{BMI} = \frac{waga}{wzrost^2}$$

```
[HttpPost]
public async Task<ActionResult<MedicalResultRecord>> PostMedicalResult(
    MedicalResultRecord medicalResult)
{
    if (medicalResult.height <= 0 || medicalResult.weight <= 0)
    {
        return BadRequest("Zle dane pomiarowe");
    }


    var existing = await _context.MedicalResults
    .AnyAsync(m => m.MedicalResultId == medicalResult.MedicalResultId);

    if (existing)
        return Conflict("Dla tej wizyty istnieje już pomiar");

    var entity = new MedicalResult
    {
        MedicalResultId = medicalResult.MedicalResultId,
        weight = medicalResult.weight,
        height = medicalResult.height,
        waistLine = medicalResult.waistLine,
        bodyFat = medicalResult.bodyFat,
        sugarLevel = medicalResult.sugarLevel,
        bmi = Math.Round(medicalResult.weight / Math.Pow(medicalResult.height
    / 100.0, 2), 2),
        date = DateTime.UtcNow
```

```
    };

    _context.MedicalResults.Add(entity);
    await _context.SaveChangesAsync();

    var resultRecord = new MedicalResultRecord(
        entity.MedicalResultId,
        entity.weight,
        entity.height,
        entity.bodyFat,
        entity.waistLine,
        entity.sugarLevel,
        entity.bmi,
        entity.date
    );

    return CreatedAtAction(nameof(GetMedicalResult), new { id = entity.
    MedicalResultId }, resultRecord);
}
```

Listing 8: Dodawanie nowego pomiaru medycznego do bazy danych

4.5. Zarządzanie wizytami

Na zarządzanie wizytami składają się między innymi: rejestracja pacjenta na wizytę, pobranie wszystkich wizyt przypisanych do danego dietetyka, edycja statusu wizyty, pobranie wizyt które mają być zrealizowane danego dnia, przez danego dietetyka

```
public async Task<ActionResult<RegisterVisitRecord>> RegisterClient(
   RegisterVisitRecord visit)
{
    if (await _context.Visits.AnyAsync(v => v.Date == visit.Date && v.
   CustomerPesel == visit.CustomerPesel))
    {
        return Conflict("Klient już jest zarejestrowany na tą wizytę");
    }
    if (await _context.Visits.AnyAsync(v =>
        v.EmployeeId == visit.EmployeeId &&
        visit.Date >= v.Date.AddMinutes(-30) &&
        visit.Date <= v.Date.AddMinutes(30)))
    {
```

```
12          return Conflict("Ten termin jest zajęty (wizyty muszą by'c w odstępie
        min. 30 min)");
13      }
14
15      var visitEntity = new Visit
16      {
17          EmployeeId = visit.EmployeeId,
18          CustomerPesel = visit.CustomerPesel,
19          status = "active",
20          Date = visit.Date.UtcDateTime
21      };
22
23      _context.Visits.Add(visitEntity);
24      await _context.SaveChangesAsync();
25
26      return Ok(visit);
27  }
```

Listing 9: Rejestracja pacjenta na wizytę

```
1  [HttpGet("date={date}/employeeId={id}")]
2  public async Task<ActionResult<IEnumerable<VisitRecord>>> GetVisit(DateOnly
       date, int id)
3  {
4
5      var startOfDay = new DateTimeOffset(date.ToDateTime(TimeOnly.MinValue),
       TimeSpan.Zero);
6
7      var endOfDay = startOfDay.AddDays(1).AddTicks(-1);
8
9      return await _context.Visits
10         .Where(v => v.EmployeeId == id && v.Date >= startOfDay && v.Date <=
       endOfDay)
11         .OrderBy(v => v.Date)
12         .Select(v => new VisitRecord(
13             v.VisitId,
14             v.Date.DateTime,
15             v.EmployeeId,
16             v.CustomerPesel,
17             v.status,
18             v.Customer,
```

```
            v.Recomendation
        ))
        .ToListAsync();
}
```

Listing 10: Pobranie wszystkich wizyt, które mają odbyć się danego dnia, które poprowadzić ma dany dietetyk

```
[HttpPut("{id}")]
public IActionResult UpdateVisitStatus(int id, [FromBody] VisitStatus
    visitUpdated)
{
    var visit = _context.Visits.FirstOrDefault(v => v.VisitId == id);
    if (visit == null)
        return NotFound("Nie znaleziono wizyty");

    visit.status = visitUpdated.status;
    _context.SaveChanges();

    return Ok(visit);
}
```

Listing 11: Edycja statusu wizyty, o podanym identyfikatorze

4.6. Zarządzanie zaleceniami medycznymi

Zalecenia medyczne składają się głównie z: opisu zalecenia medycznego, w formie krótkiej wiadomości tekstowej oraz dołączonego pliku PDF, który jest przechowywany w formie binarnej w bazie danych. Zalecenia medyczne są wysyłane do pacjetna drogą mailową, automatycznie po przypisaniu zalecenia przez dietetyka.

```
[HttpPost]
public async Task<ActionResult<MedicalRecommendationRecord>>
    PostMedicalRecommendation(MedicalRecommendationRecord
    medicalRecommendation)
{
    if (string.IsNullOrWhiteSpace(medicalRecommendation.note))
        return BadRequest("Notatka nie może by'c pusta.");


    var entity = new MedicalRecomendations
```

```
    {
        dietId = medicalRecommendation.dietId,
        note = medicalRecommendation.note,
        MedicalRecomendationsId = medicalRecommendation.
    MedicalRecomendationsId,
        date = DateTimeOffset.UtcNow
    };

    _context.MedicalRecommendations.Add(entity);
    await _context.SaveChangesAsync();

    var resultRecord = new MedicalRecommendationRecord(
        entity.MedicalRecomendationsId,
        entity.dietId,
        entity.note,
        entity.date
    );

    //wielowątkowosc
    _ = Task.Run(async () =>
    {
        try
        {

            using var scope = HttpContext.RequestServices.CreateScope();
            var db = scope.ServiceProvider.GetRequiredService<AppDbContext>();

            var diet = await db.Diets.FirstOrDefaultAsync(d => d.dietId ==
    medicalRecommendation.dietId)
                        ?? throw new Exception("Nie znaleziono diety.");
            var visit = await db.Visits.FirstOrDefaultAsync(v => v.VisitId ==
    medicalRecommendation.MedicalRecomendationsId)
                         ?? throw new Exception("Nie znaleziono wizyty.");
            var customer = await db.Customers.FirstOrDefaultAsync(c => c.pesel
     == visit.CustomerPesel)
                            ?? throw new Exception("Nie znaleziono klienta.");
            var employee = await db.Employees.FirstOrDefaultAsync(e => e.
    EmployeeId == visit.EmployeeId)
                            ?? throw new Exception("Nie znaleziono pracownika."
    );

```

```
            try
            {
                SendMedicalRecommendation.sendMedicalRecommendation(
                    medicalRecommendation.note,
                    diet.pdf,
                    customer.firstName,
                    customer.email,
                    employee.firstName + " " + employee.lastName
                );
            }
            catch (Exception mailEx)
            {
                Console.WriteLine($"Błąd wysyłki maila w tle: {mailEx}");
            }
        }
        catch (Exception dbEx)
        {
            Console.WriteLine($"Błąd pobierania danych w tle: {dbEx}");
        }
    });
    return CreatedAtAction(nameof(GetMedicalRecommendation), new { id = entity.
    MedicalRecomendationsId }, resultRecord);
}
```

Listing 12: Przypisanie zaleceń medycznych, uwzględniając opis zalecenia medycznego oraz zalecaną przez dietetyka dietę.

4.7. Wysyłanie wiadomości mailowych

Do wysyłania wiadomości drogą mailowa, aplikacja korzysta z bibliotek: MailKit.Net.Smtp, oraz MimeKit. W celach projektowych zostało utworzone specjalne konto mailowe

"dietetykplus2025@gmail.com", które jest zintegrowane z aplikacją.

```
        public static async Task sendMedicalRecommendation(string note, byte[] pdf, string firstName, string email, string dietetyk)
    {
        var message = new MimeMessage();
        message.From.Add(new MailboxAddress("Dietetyk+", "dietetykplus2025@gmail.com"));
        message.To.Add(new MailboxAddress(firstName, email));
        message.Subject = "Zalecenia żywieniowe " + DateTime.Now.ToString("yyyy-MM-dd");

        var body = new TextPart("html")
        {
            Text = $@"
    <p><strong>Przesyłamy Panu/Pani zalecenia żywieniowe z wizyty z dnia {DateTime.Now.ToLongDateString()}</strong></p>
    <p><strong>Tre’s’c Zalecenia:</strong> {note}</p>
    <p>Dietetyk: {dietetyk}</p><br><br>
    <p><i>Prosimy pobra’c załącznik z Dietą, kt’ory znajduje się poniżej</i></p>
"
        };

        var attachment = new MimePart("application", "pdf")
        {
            Content = new MimeContent(new System.IO.MemoryStream(pdf)),
            ContentDisposition = new ContentDisposition(ContentDisposition.Attachment),
            ContentTransferEncoding = ContentEncoding.Base64,
            FileName = "Zalecenia" + DateTime.Now.ToString("yyyy-MM-dd") + ".pdf"
        };
        var multipart = new Multipart("mixed");
        multipart.Add(body);
        multipart.Add(attachment);
        message.Body = multipart;
        using (var client = new SmtpClient())
        {
            await client.ConnectAsync("smtp.gmail.com", 587, MailKit.Security.SecureSocketOptions.StartTls);
            await client.AuthenticateAsync("dietetykplus2025@gmail.com", "
```

```
    fgxbaauuftpemenu");
33              await client.SendAsync(message);
34              await client.DisconnectAsync(true);
35          }
36      }
37  }
```

Listing 13: Struktura wiadomości mailowej

5 Frontend – Architektura React

5.1. Komponenty UI (React-Bootstrap)

Do zbudowania przejrzystego interfejsu użytkownika została wykorzystana biblioteka **React-Bootstrap**. Została ona wykorzystana głównie do implementacji **okien modalnych** oraz **formularzy**.

5.2. Customowe ikonki (React-Icons)

Do uzyskania lepiej przejrzystości interfejsu zostały wykorzystane niestandardowe ikonki pochodzące z biblioteki **React-Icons**.

5.3. Wizualizacja danych (Recharts)

Do prezentacji danych analitycznych wykorzystano bibliotekę **Recharts** [1]. Jest to rozwiązanie zbudowane natywnie dla Reacta, oparte na komponentach SVG, co gwarantuje wysoką responsywność i łatwość konfiguracji wykresów kołowych oraz słupkowych.

5.4. Responsywność

Aplikacja jest w pełni responsywna. Użytkownik w wygodny, i przejrzysty sposób może korzystać z aplikacji, dzięki zawsze może mieć swoje narzędzie pracy pod ręką.

6 Frontend - Kluczowe funkcjonalności

6.1. Wyszukiwarki

```
#wyszukiwanie klientów po nazwie, nazwisku, peselu, miejscu zamieszkania
let searchedCustomers = customers.filter(customer =>
    customer.pesel?.includes(searchedValue) ||
    customer.firstName?.toLowerCase().includes(searchedValue.toLowerCase()) ||
    customer.lastName?.toLowerCase().includes(searchedValue.toLowerCase()) ||
    (customer.firstName?.toLowerCase() + " "
    + customer.lastName?.toLowerCase() === searchedValue.toLowerCase()) ||
    customer.residentialAddress?.toLowerCase().includes(searchedValue.
    toLowerCase())
);
```

Listing 14: Filtrowanie listy klientów

```
#wyszukiwanie diet po rodzaju, deficycie kalorycznym
let searchedDiets = diets.filter(diet =>
    (diet.type?.toLowerCase().includes(searchedValue?.toLowerCase() || "")) ||
        (diet.kcalDeficit != null && diet.kcalDeficit.toString().includes(
    searchedValue || ""))
    ).sort((a, b) => a.kcalDeficit - b.kcalDeficit);
```

Listing 15: Filtrowanie listy diet

6.1.1. Wyświetlanie wizyt w danym dniu

```
return (
    <div className={"visits-container"}>
        <div className={"visit-text"}>
            Wizyty w dniu {new Date().toLocaleDateString()}
        </div>
        {visits?.map(visit => {
            if (visit?.status === "active" || visit?.status === "postponed") {
                return (
                    <VisitCard
                        goTo={() => {
                            navigate(`../visits/${visit.customer.pesel}/${
    visit.visitId}`)
```

```
                        }}
                        now={visitCount++ == 0}
                        clientData={[visit.customer.firstName, visit.customer.
    lastName]}
                        date={new Date(visit.date).toLocaleTimeString('pl-PL',
     {
                            hour: '2-digit',
                            minute: '2-digit'
                        })}
                    />
                );
            }
        })}
    </div>
);
```

Listing 16: Wizyty do przeprowadzenia w danym dniu

6.1.2. Wyświetlanie zawartości diety

```
async function getPDF(dietId: number) {
    const response = await fetch(`https://localhost:7081/api/Diets/${dietId}/
    pdf`);
    if (!response.ok) {
        alert("Nie znaleziono pliku PDF!");
        return;
    }

    const blob = await response.blob();
    const url = URL.createObjectURL(blob);
    window.open(url);
}

{
    searchedDiets.map(diet => {
        const Icon = dietIcons.get(diet.type);
        return (
            <div key={diet?.dietId} onClick={() => getPDF(diet?.dietId)}>
                <DietCard
                    icon={Icon ? <Icon /> : null}
                    name={diet.type}
```

```
                    kcal={diet.kcalDeficit}
                />
            </div>
        );
    })
}
```

Listing 17: Wyświetlenie zawartości diety w postaci PDF

6.1.3. Kalendarz filtrowanie wizyt

```
const events = useMemo(() => {
    return visits
        .filter(visit => employee ? Number(visit.employeeId) === employee.id :
     true)
        .map(visit => {
            const startDate = new Date(visit.date);
            let endDate = addMinutes(startDate, Number(config?.visit_duration)
    );

            if (startDate.getDate() !== endDate.getDate()) {
                endDate = new Date(
                    startDate.getFullYear(),
                    startDate.getMonth(),
                    startDate.getDate(),
                    23, 59, 59
                );
            }

            const mycustomer = customer?.find(c => c.pesel == visit?.
    customerPesel);

            return {
                id: visit.visitId,
                customer: mycustomer,
                title: mycustomer?.firstName + " " + mycustomer?.lastName,
                start: startDate,
                end: endDate,
                allDay: false,
                status: visit.status
            };
```

```
        });
}, [visits, employee]);
```

Listing 18: Filtrowanie wizyt w kalendarzu dietetyka

6.1.4. Filtrowanie pomiarów medycznych

```
const customerMeasurements = useMemo(() => {
        if (!customer?.visits) return [];

        return customer.visits
            .map(visit => {
                const visitResult = result.find(
                    r => r.medicalResultId === visit.visitId
                );
                if (!visitResult) return null;

                return {
                    visitId: visit.visitId,
                    date: visitResult.date.substring(0, 10),
                    bmi: visitResult.bmi,
                    employeeId: visit.employeeId
                };
            })
            .filter(Boolean);
    }, [customer, result]);
```

Listing 19: Filtrowanie wizyt używanych w tabeli, oraz wykresie pomiarów

6.1.5. Zabepieczenie dostępu do aplikacji

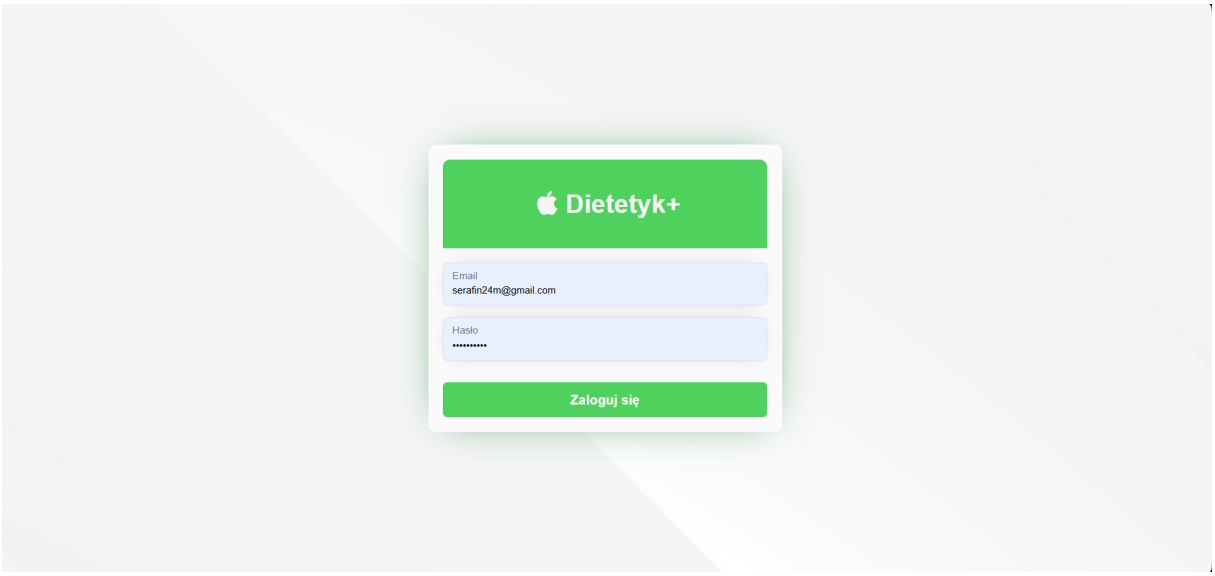
```

export const AppRoutes = () => {
    return (
        <BrowserRouter>
            <Routes>
                <Route path="/login" element={<Login />} />

                <Route element={<ProtectedRoute />}>
                    <Route path="/home" element={<Home />}>
```

```
                        <Route index element={<Menu />} />
                        <Route path="menu" element={<Menu />} />
                        <Route path="clients" element={<Clients />} />
                        <Route path="clients/:pesel" element={<ClientInfoPanel
    />} />
                        <Route path="calendar" element={<Calendar />} />
                        <Route path="diets" element={<Diets />} />
                        <Route path="visits" element={<Visits />} />
                        <Route path="visits/:pesel/:visit" element={<
   CurrentVisit />} />
                        <Route path="admin" element={<AdminPanel />} />
                    </Route>
                </Route>

                <Route path="*" element={<Navigate to="/login" replace />} />
            </Routes>
        </BrowserRouter>
    );
};
import { Navigate, Outlet } from "react-router-dom";
#nie zalowanego użytkownika automatycznie przenosi do strony logowania
export const ProtectedRoute = () => {
    const token = localStorage.getItem("token");

    if (!token) {
        return <Navigate to="/login" replace />;
    }

    return <Outlet />;
};
```

Listing 20: Filtrowanie wizyt używanych w tabeli, oraz wykresie pomiarów

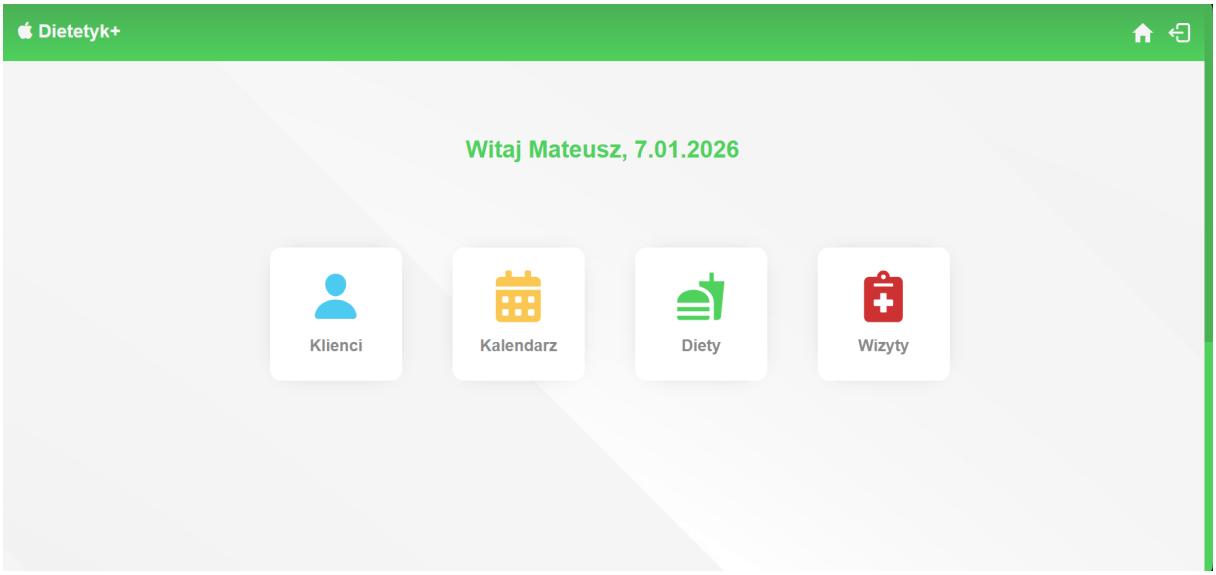
7 Zrzuty ekranu

7.1. Ekran logowania



Rysunek 3: Prosty formularz logowania

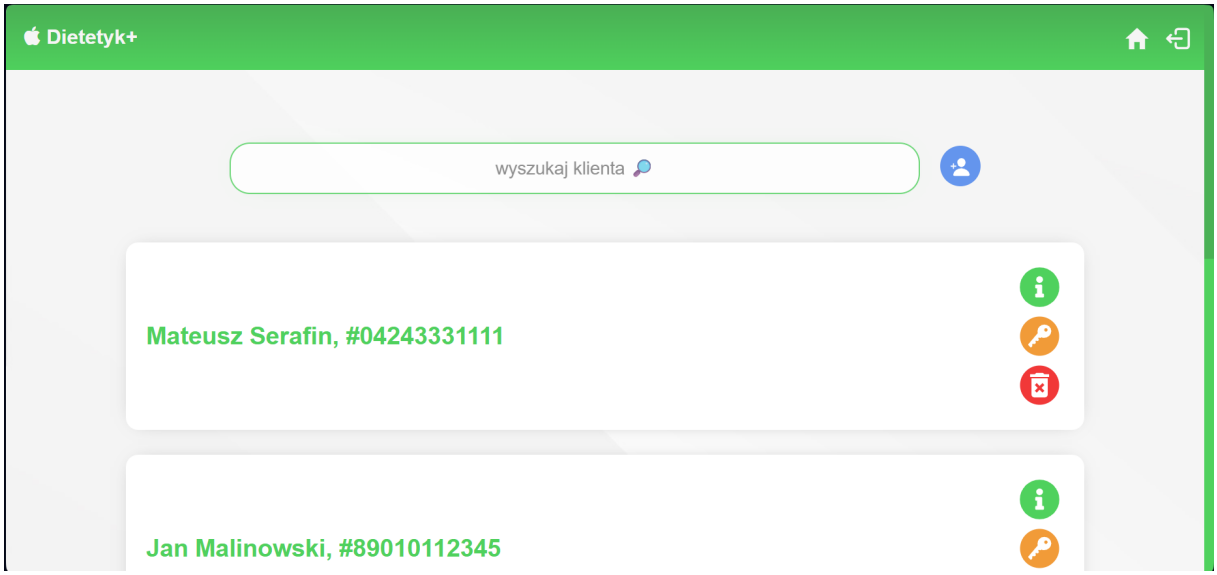
7.2. Dashboard



Rysunek 4: Główny ekran aplikacji, zawierający Menu

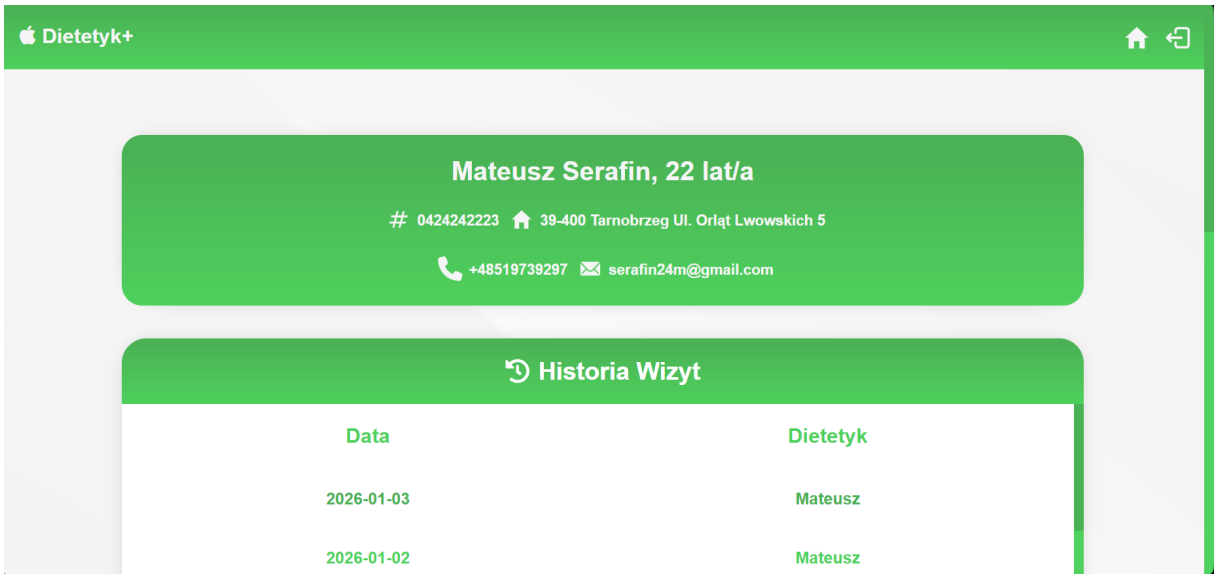
7.3. Podstrony

7.3.1. /klienci



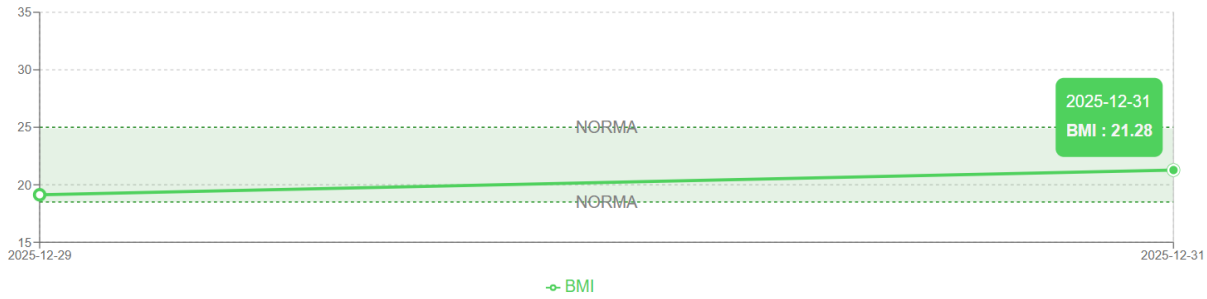
Rysunek 5: Podstrona zawierająca listę klientów

7.3.2. /klient/:id (informacje o kliencie)



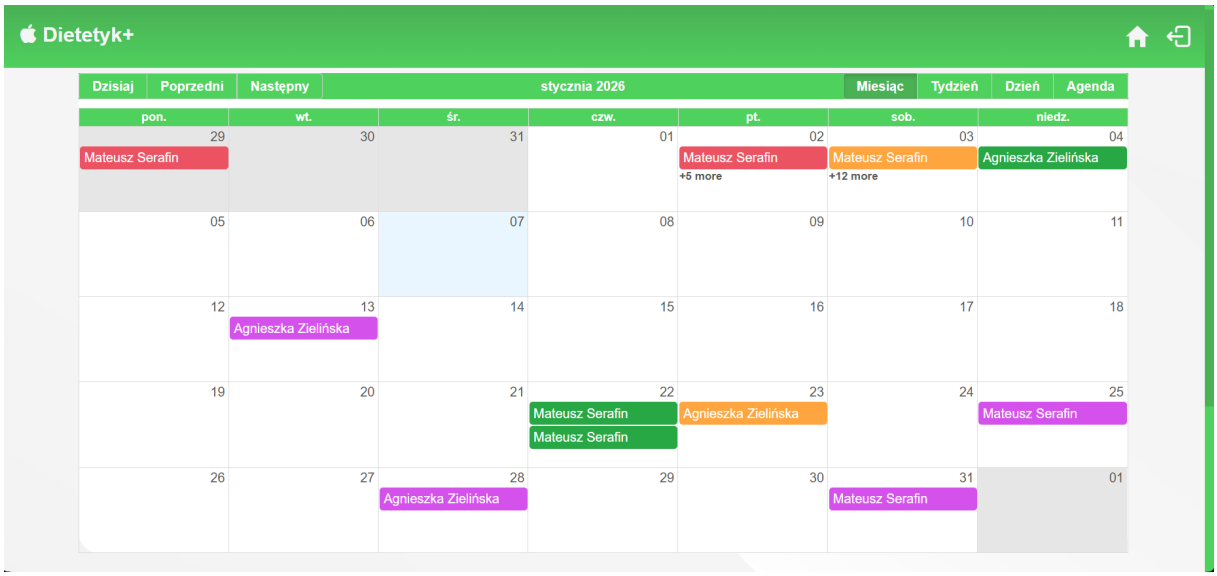
Rysunek 6: Panel informacji z danymi klienta

7.3.3. /klient/:id (wykres bmi)



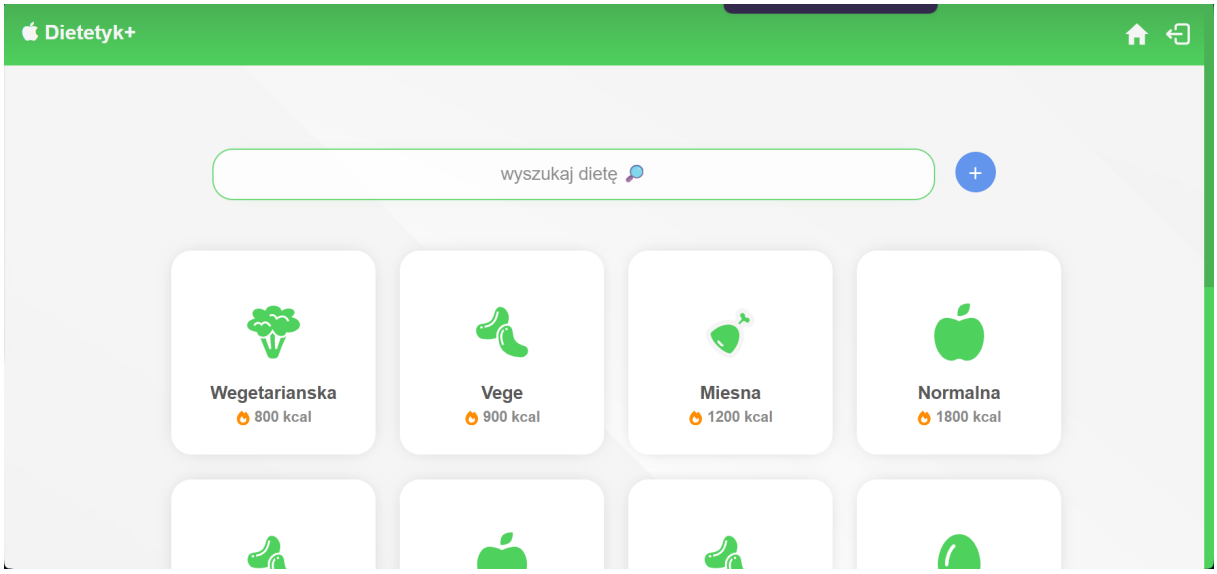
Rysunek 7: Panel informacji z danymi klienta

7.3.4. /kalendarz



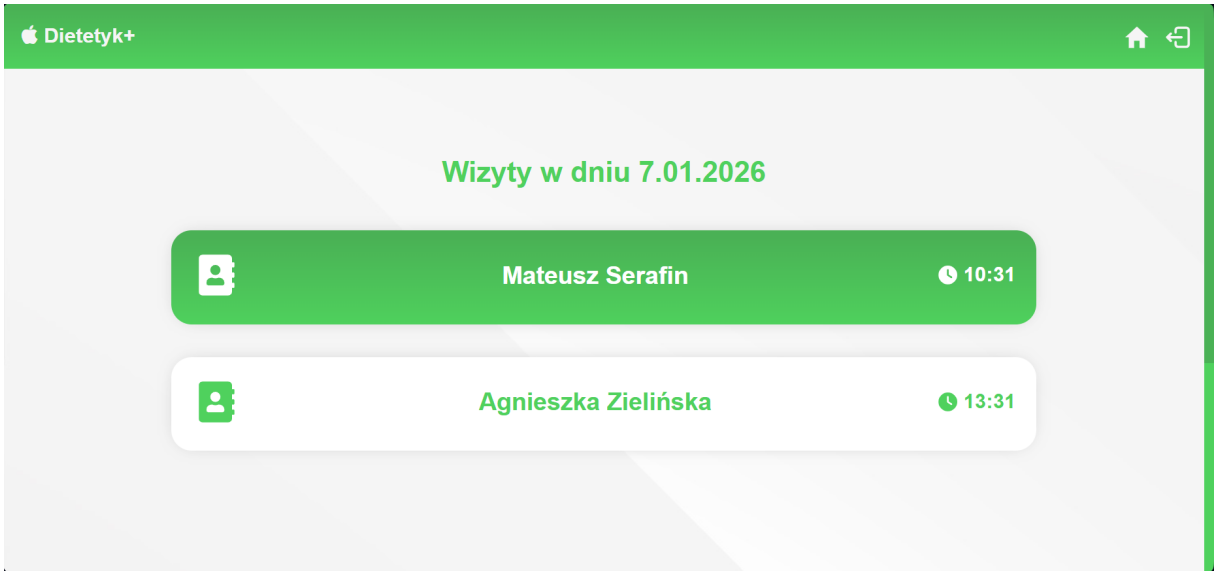
Rysunek 8: Kalendarz dietetyka

7.3.5. /diety



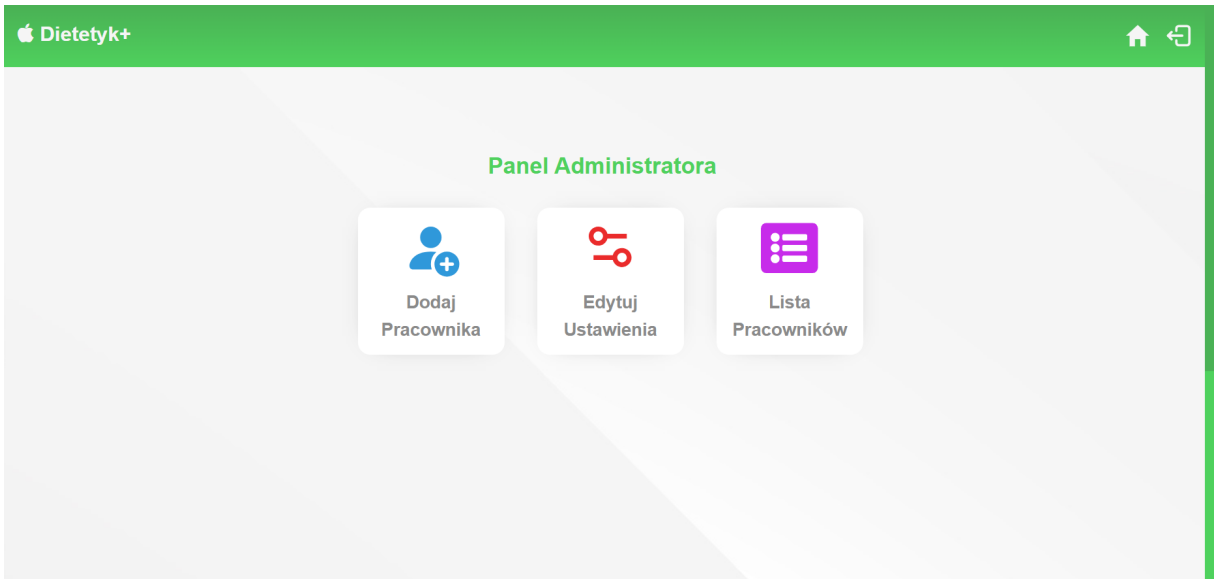
Rysunek 9: Lista diet

7.3.6. /wizyty



Rysunek 10: Wizyty

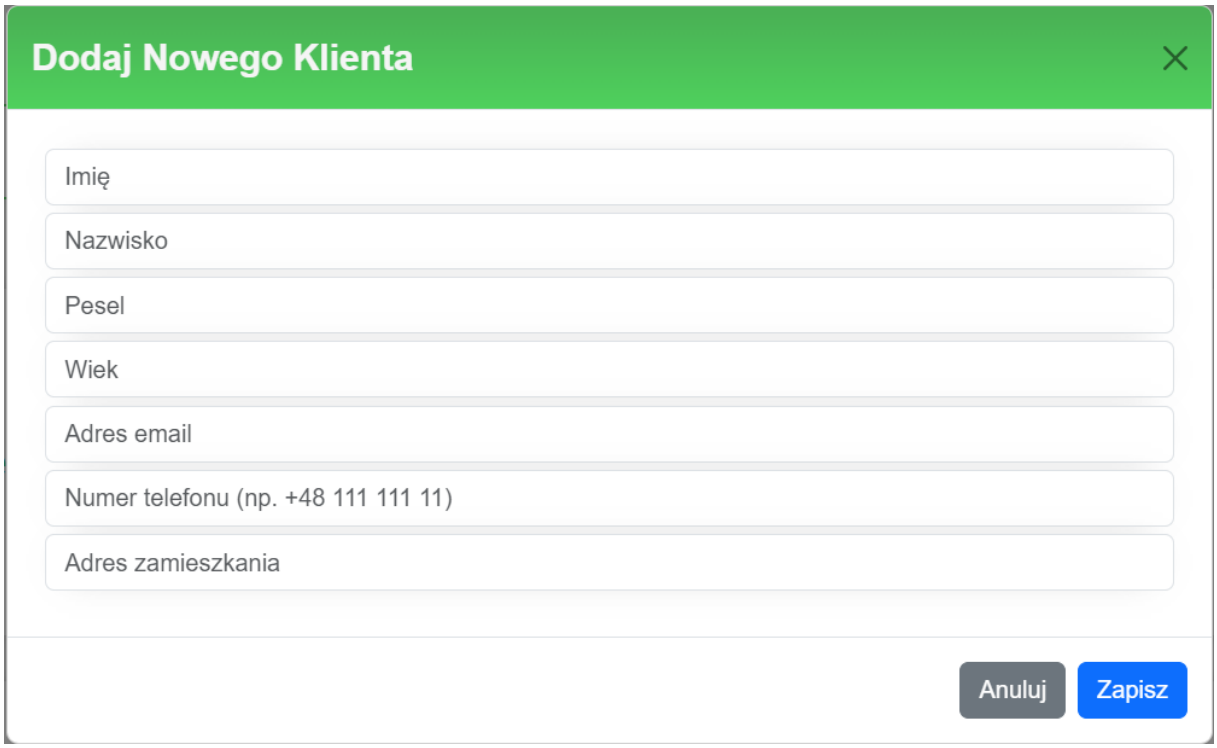
7.3.7. /admin



Rysunek 11: Panel Administratora

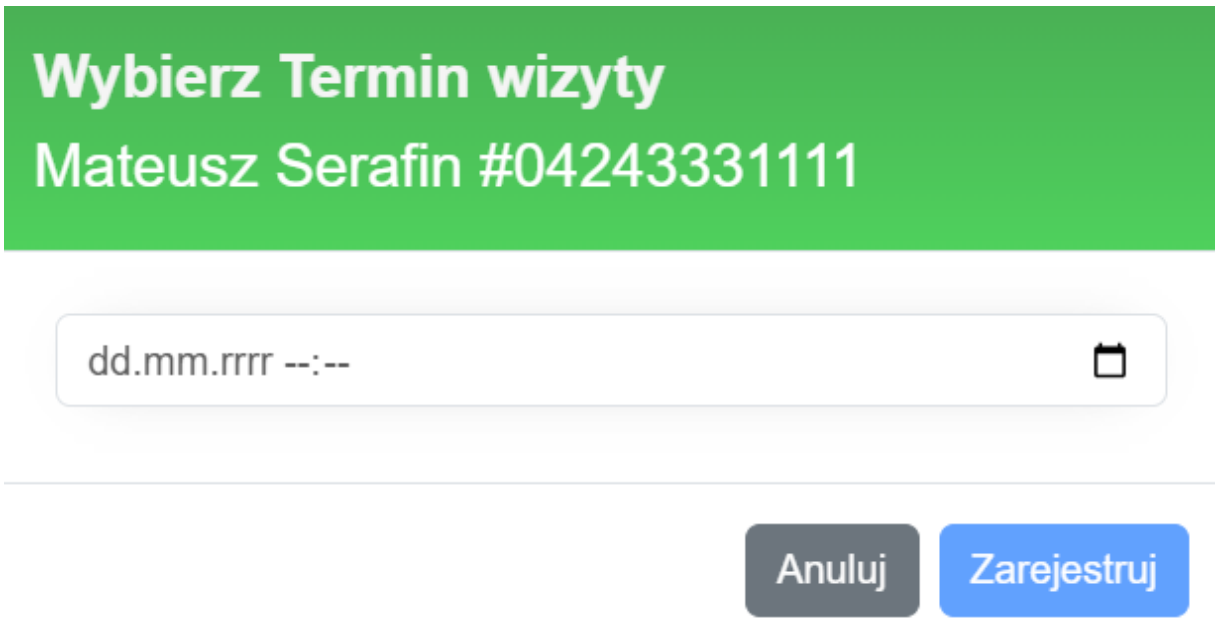
7.4. Okna Modalne

7.4.1. Rejestracja pacjenta



Rysunek 12: Rejestrowanie nowego pacjenta

7.4.2. Rejestracja pacjenta na wizytę



Rysunek 13: Rejestracja pacjenta na wizytę

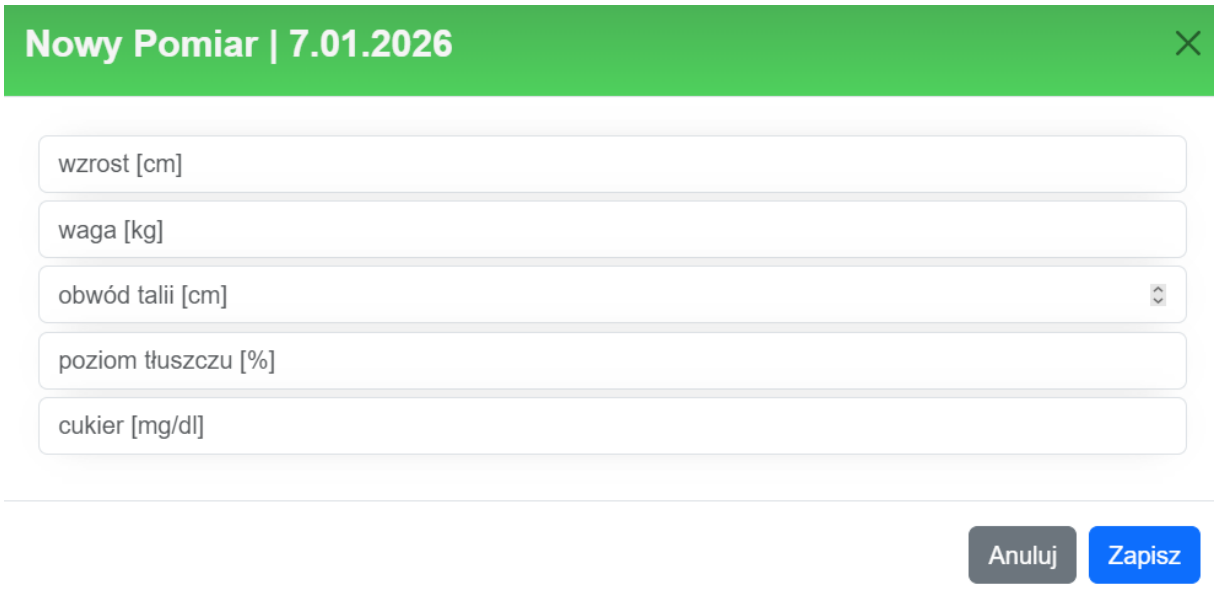
7.4.3. Zmiana statusu wizyty

Rysunek 14: Zmiana statusu wizyty

7.4.4. Dodawanie nowej diety

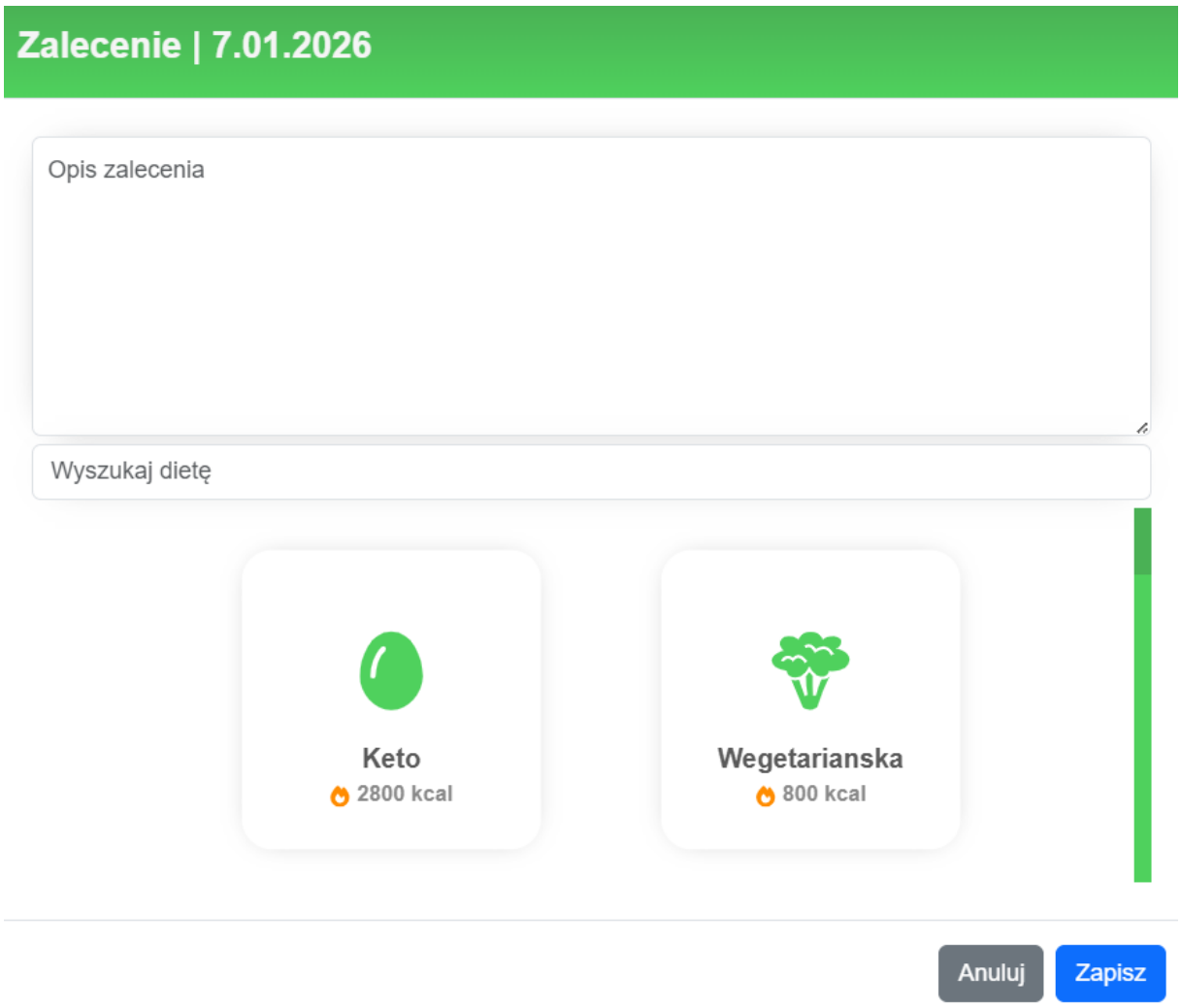
Rysunek 15: Dodawanie nowej diety do bazy danych

7.4.5. Dodawanie pomiaru medycznego



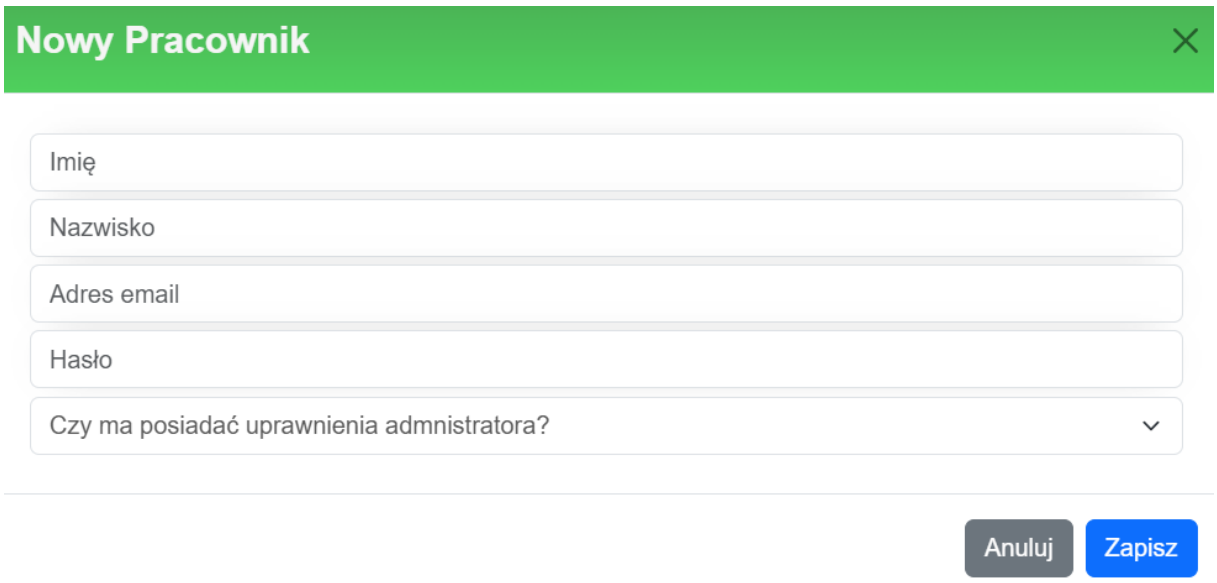
Rysunek 16: Dodawanie nowego pomiaru medycznego danego pacjenta do bazy danych

7.4.6. Przypisywanie zaleceń medycznych



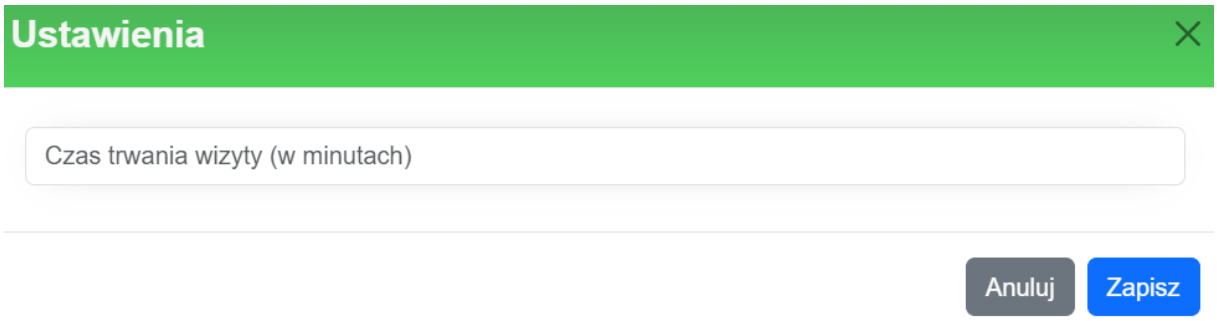
Rysunek 17: Przypisanie zalecenia medycznego do danego pacjenta

7.4.7. Tworzenie konta nowego pracownika



Rysunek 18: Panel Administratora - tworzenie konta nowego pracownika

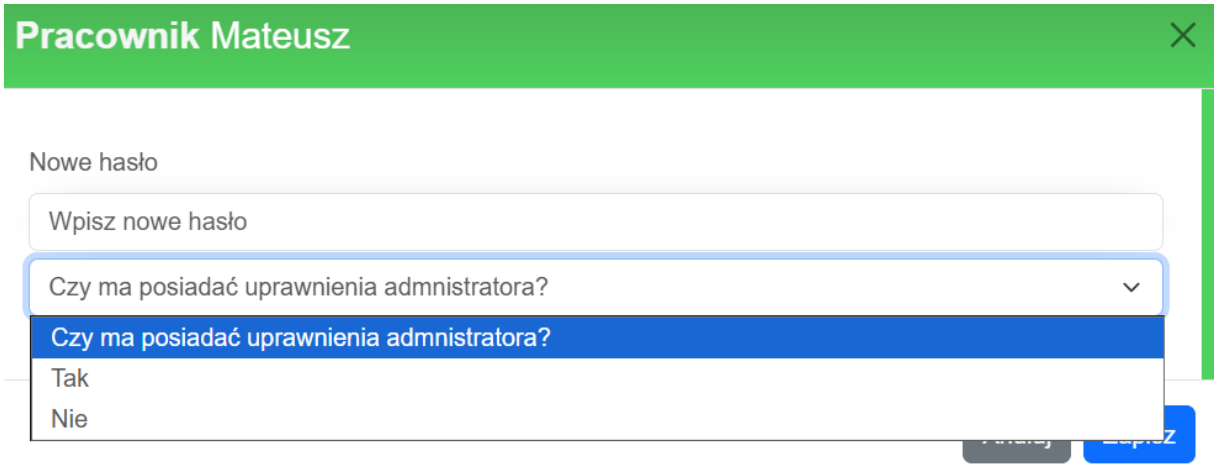
7.4.8. Edycja ustawień



Rysunek 19: Panel Administratora - ustawienia aplikacji

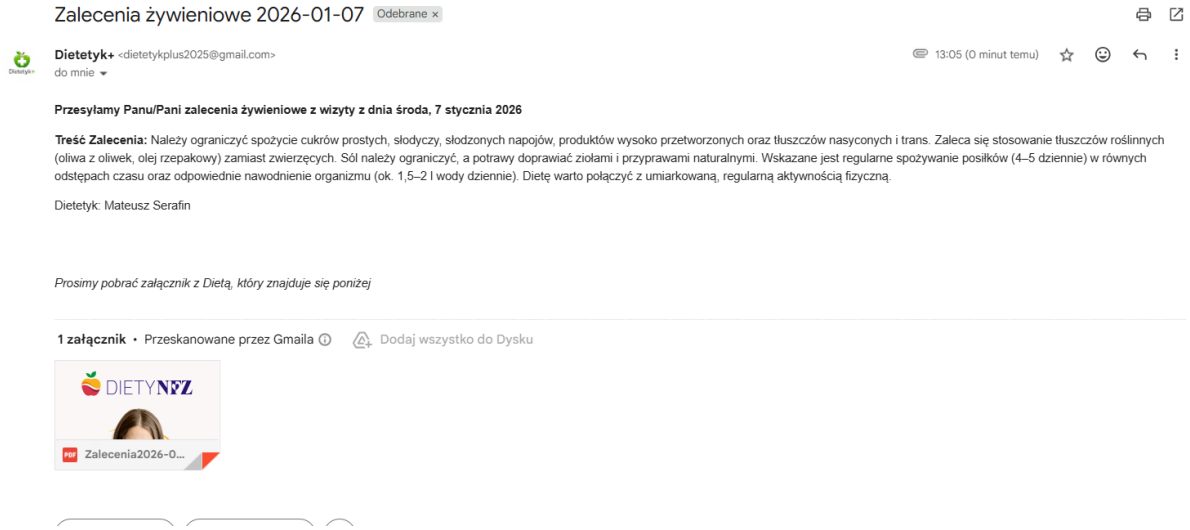
7.4.9. Lista Pracowników

Rysunek 20: Panel Administratora - lista pracowników



Rysunek 21: Panel Administratora - edycja konta pracownika

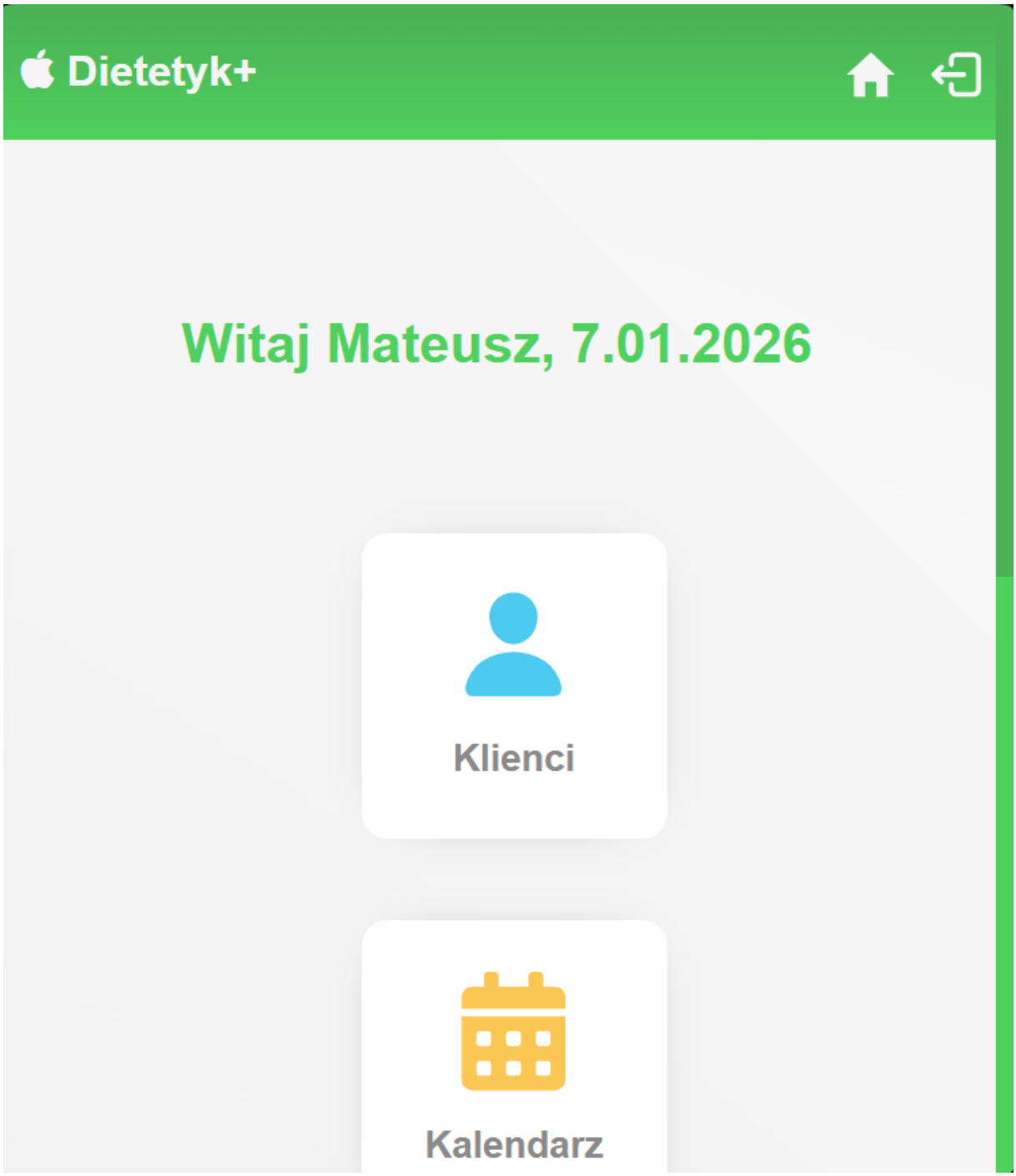
7.5. Wygląd wiadomości mailowej



Rysunek 22: Zalecenia żywieniowe dostarczone w wiadomości mailowej do pacjenta

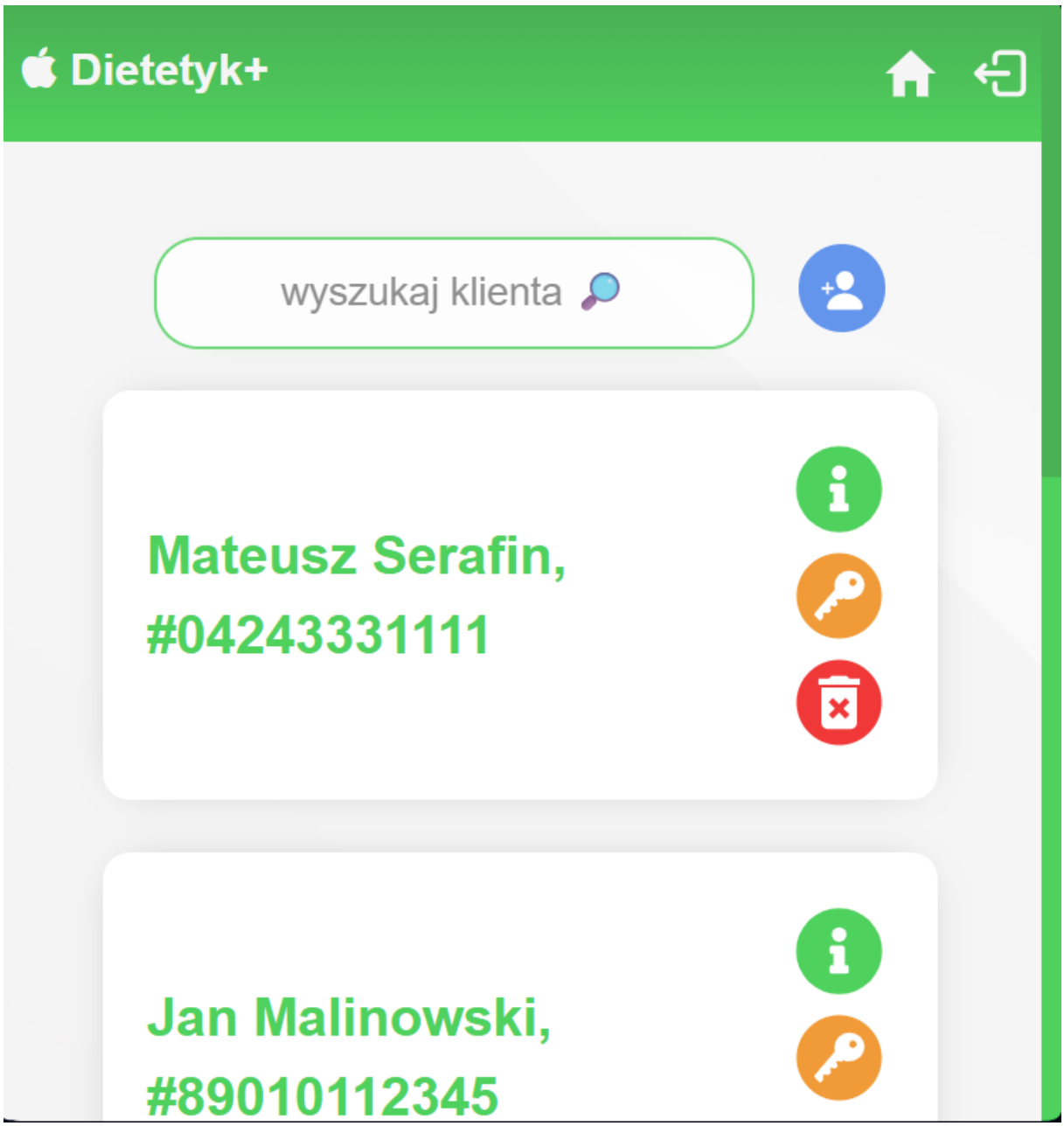
7.6. Widok mobilny

7.6.1. /dashboard



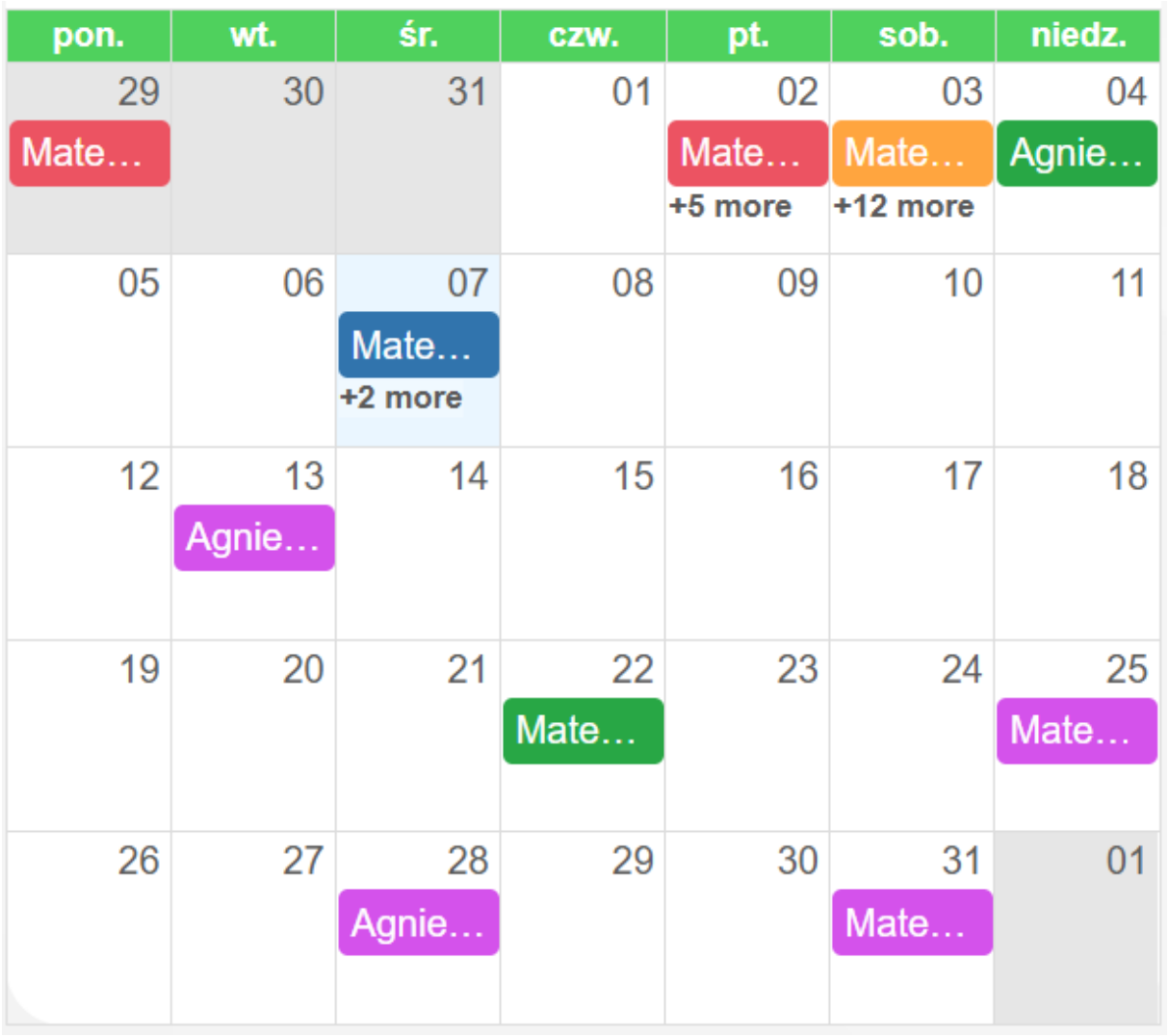
Rysunek 23: Wygląd '/home' na urządzeniu mobilnym

7.6.2. /klienci



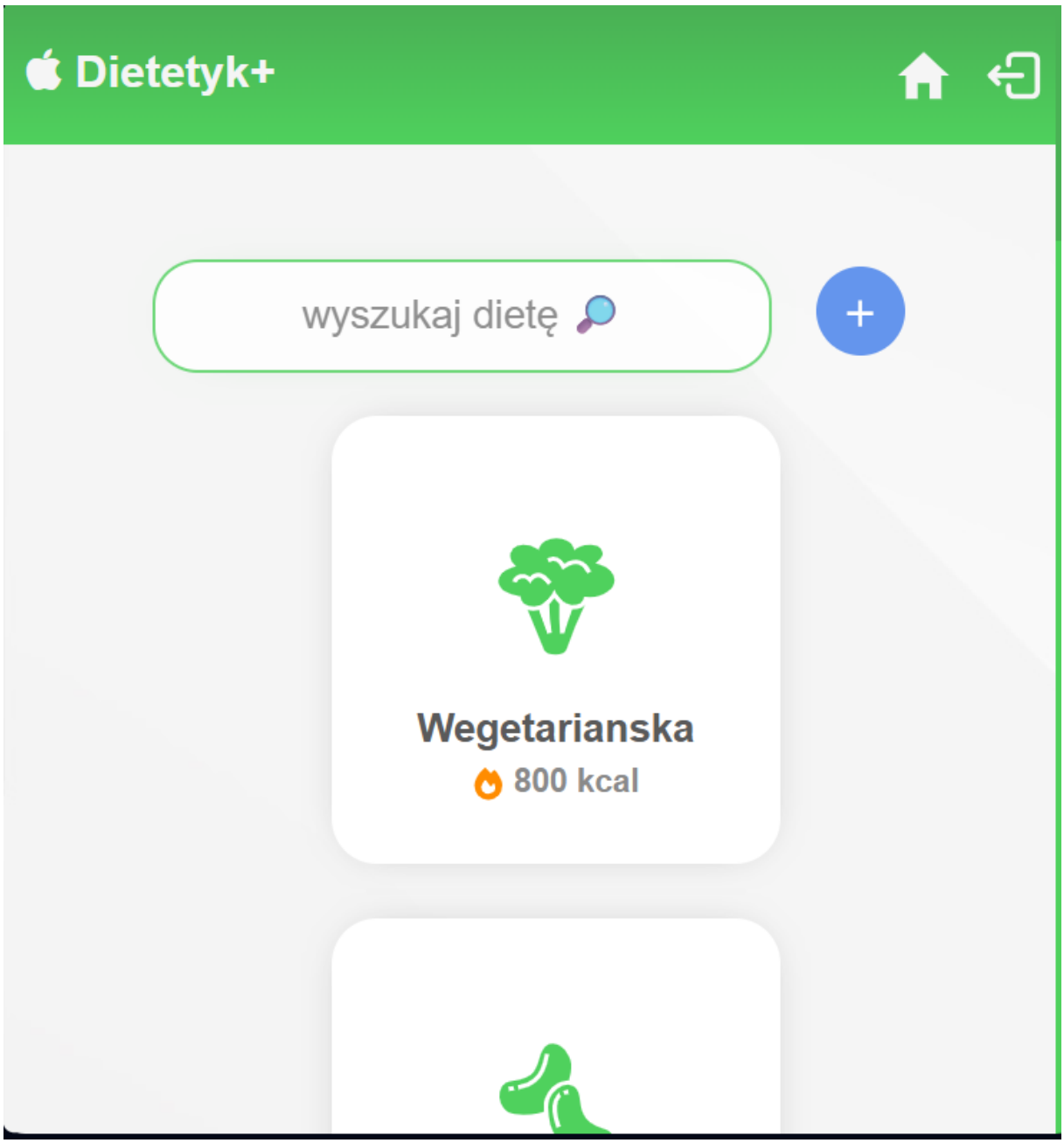
Rysunek 24: Wygląd '/klienci' na urządzeniu mobilnym

7.6.3. /kalendarz



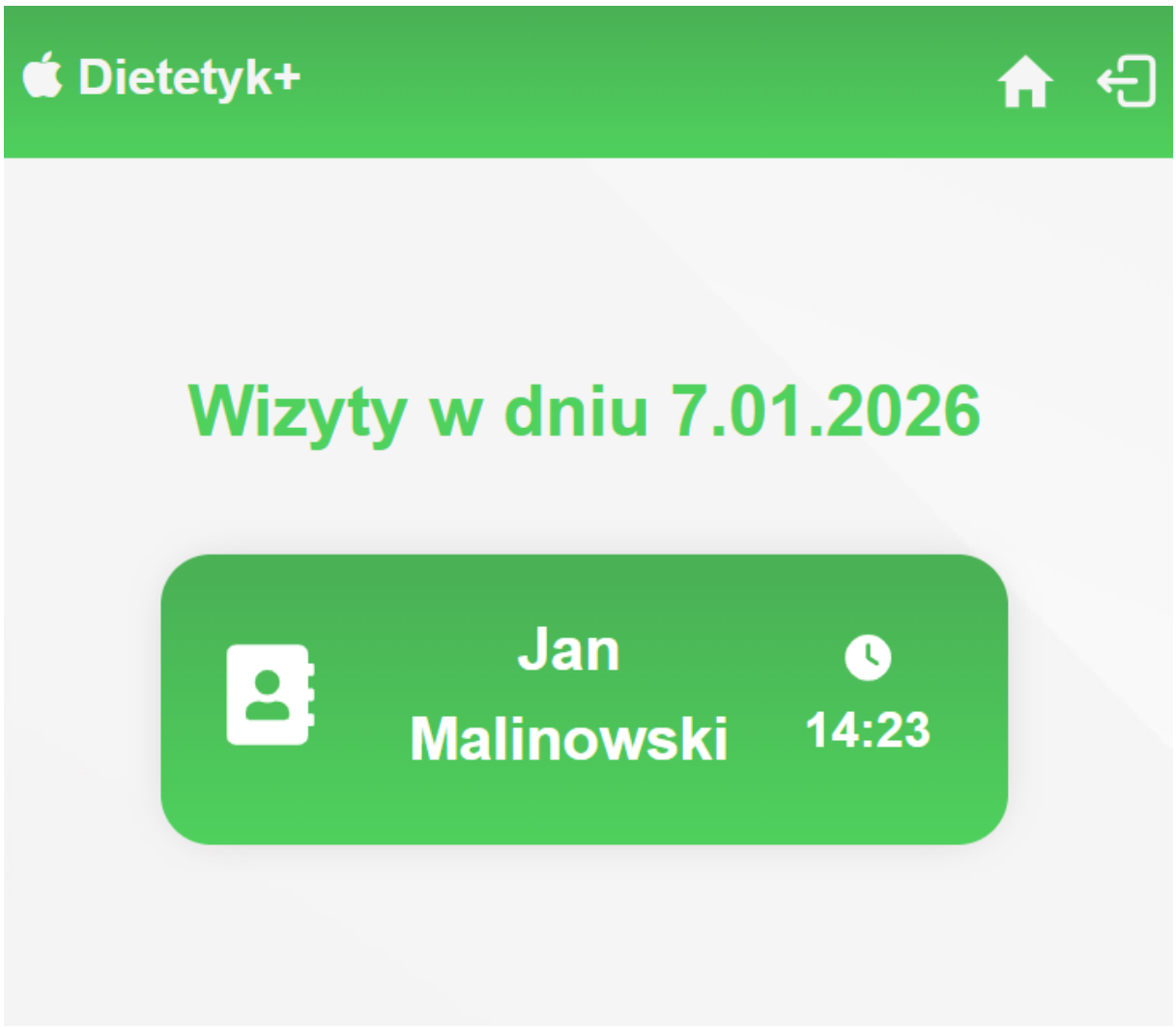
Rysunek 25: Wygląd '/kalendarz' na urządzeniu mobilnym

7.6.4. /diety



Rysunek 26: Wygląd '/diety' na urządzeniu mobilnym

7.6.5. /wizyty



Rysunek 27: Wygląd '/wizyty' na urządzeniu mobilnym

8 Wnioski i dalszy rozwój

Projekt został w pełni zrealizowany i znacznie przekroczył początkowe założenia.

8.1. Osiągnięte wyniki

- **Wydajność:** Krótki czas odpowiedzi z API.
- **Responsywność:** Pełna zgodność z podejściem mobile-first.
- **Bezpieczeństwo:** Korzystanie z autoryzacji za pomocą tokenu JWT oraz szyfrowaniem danych za pomocą BCrypt

8.2. Podsumowanie

Projekt wykazał, że możliwe jest stworzenie aplikacji webowej na wysokim poziomie, która może być powszechnie używana na codzień w salonach dietetycznych.

Kod źródłowy projektu jest dostępny w publicznym repozytorium na GitHub:
https://github.com/xserafineq/dietetykplus
https://github.com/xserafineq/DietetykAPI

Bibliografia

[1] Recharts Group, “Recharts documentation,” https://recharts.org/en-US/, 2024, dostęp: 18.11.2025.

[2] Meta Platforms, “React documentation,” https://react.dev, 2025, dostęp: 15.10.2025.

[3] Microsoft, “Typescript documentation,” https://www.typescriptlang.org/docs/, 2025, dostęp: 18.11.2025.

[4] M. Jones, J. Bradley, and N. Sakimura, “Json web token (jwt),” https://datatracker.ietf.org/doc/html/rfc7519, IETF, RFC 7519, 2015.

[5] Microsoft, “.net 9 documentation,” https://learn.microsoft.com/pl-pl/dotnet/, 2025, dostęp: 15.10.2025.

[6] React-Icons Group, “React-icons documentation,” https://react-icons.github.io/react-icons/, 2024, dostęp: 18.11.2025.

[7] React-Bootstrap Contributors, “React-bootstrap documentation,” https://react-bootstrap.github.io/, 2024, dostęp: 18.11.2025.

[8] D. Mazieres and N. Provos, “Bcrypt.net-next - password-hashing function,” https://www.nuget.org/packages/BCrypt.Net-Next, 2024, dostęp: 10.01.2026.