



POLITECHNIKA
LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI
I INFORMATYKI

Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Inżynieria Oprogramowania

Aplikacja internetowa do zbalansowanej diety niskotłuszczowej

Online application for balanced low-fat diet

Jakub Arciszewski

92829

Promotor: dr hab. inż. Dariusz Czerwiński, prof. uczelni

Lublin rok 2024

Spis Treści

Streszczenie	4
Abstract.....	4
1. Wstęp.....	5
2. Cel i zakres pracy	6
3. Analiza rynku	7
3.1. Low fat diet.....	7
3.2. Fitatu	8
3.3. MyNetDiary	9
3.4. Podsumowanie	11
4. Wykorzystane technologie	13
5. Projekt aplikacji.....	15
5.1. Wymagania funkcjonalne	15
5.2. Wymagania нефункционалне	15
5.3. Scenariusze	15
5.3.1 Rejestracji	15
5.3.2 Logowania	16
5.4. Diagramy UML.....	17
5.5. Baza danych.....	18
6. Implementacja	21
6.1. Back-end	21
6.2. Generowanie diet	27
6.3. Front-end.....	29
7. Testy	39
8. Podsumowanie.....	40
Bibliografia:	41

Streszczenie

Celem aplikacji webowej o dietach niskotłuszczowych jest pomóc użytkownikom, którzy chcieliby wybrać taką dietę. Potrzeby stosowania takiej diety mogą być różne, np. schudnięcie, uregulowanie HDL i LDL (tzw. dobry i zły cholesterol) czy obniżenie ryzyka chorób układu krążenia.

Z tego powodu aplikacja pomaga użytkownikom, którzy chcieliby skorzystać z diety niskotłuszczowej, ale np. nie stać ich na dietetyka lub nie posiadają odpowiedniej wiedzy, żeby ułożyć taką dietę samemu. W systemie znajdują się różne diety, z podziałem na wegetariańską i zwykłą oraz z różnymi kalorycznościami, do wyboru użytkownika.

Słowa kluczowe: dieta niskotłuszczowa, tłuszcz, cukrzyca

Abstract

The purpose of the web application about low-fat diets is to help users who would like to choose such a diet. The needs for following such a diet can vary, such as losing weight, regulating HDL and LDL (so-called good and bad cholesterol), or lowering the risk of cardiovascular disease.

For this reason, the app helps users who would like to follow a low-fat diet, but, for example, can't afford a nutritionist or don't have the knowledge to put together such a diet on their own. There are various diets in the system, divided into vegetarian and regular diets, and with different calorie counts for the user to choose from.

Keywords: low-fat diet, fat, diabetes

1. Wstęp

Problemy zdrowotne związane z odżywianiem stanowią duży problem w dzisiejszych czasach. Jak wynika ze statystyk Eurostatu [3], 52,7% dorosłych na terenie Unii Europejskiej ma nadwagę i ok. 30 mln zmagają się z cukrzycą [4]. Od lat zauważalny jest trend wzrostowy, a przewidywania nie są optymistyczne, trend ten utrzyma się w najbliższych latach. W związku z tym coraz więcej osób szuka rozwiązania wśród różnych aplikacji mobilnych i internetowych, przy pomocy których będą mogli odchudzić się i utrzymać swoją wagę w normie lub kontrolować swoją dietę w celach walki z cukrzycą.

Dieta niskotłuszczowa jest jednym ze sposobów na rozwiązanie wielu problemów zdrowotnych. Żeby dieta było niskotłuszczowa, tłuszcz nie powinien stanowić więcej niż 20% całości kalorii [8]. Jest to uzasadnione tym, że najbardziej szkodliwe dla zdrowia są tłuszcze nasycone oraz trans. W dzisiejszych czasach i przy obecnym stopniu przetworzenia żywności unikanie ich jest praktycznie niemożliwe [6]. W związku z tym proponowane jest ograniczenie konsumpcji tłuszczu do ok. 15%-20% całości spożywanych kalorii [7]. Dlatego też w tej pracy dieta będzie oscylowała właśnie ok. tych 20%.

Pomimo tego, że istnieje wiele aplikacji dietetycznych, zdecydowana większość z nich nie posiada żadnych diet niskotłuszczowych, więc aplikacja opracowana na rzecz tej pracy wypełnia pewną niszę.

2. Cel i zakres pracy

Celem pracy inżynierskiej jest zaprojektowanie i implementacja aplikacji internetowej w obsługującej diety niskotłuszczowe. Użytkownik będzie mógł wybrać jedną z diet oraz śledzić przebieg swojej diety na przestrzeni dni z ostatnich miesięcy. Pozwala to na zaoszczędzeniu czasu i pieniędzy potrzebnych na układanie takiej diety samemu lub przy pomocy dietetyka.

Zakres pracy obejmuje:

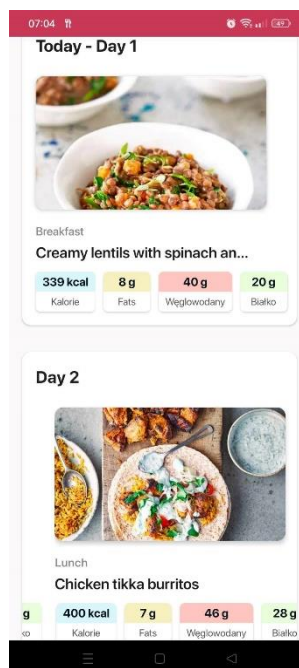
- Zbadanie i analiza aplikacji związanych z tematem diet
- Zrobienie projektu aplikacji
- Wybranie technologii, w której zostanie wykonana aplikacja
- Implementacja aplikacji
- Testowanie aplikacji i ew. poprawki

3. Analiza rynku

Pomimo dużym zainteresowaniem dietami i różnymi, cieszącymi się popularnością aplikacjami do diet, bardzo ciężko znaleźć aplikację poświęconą dietom niskotłuszczowym czy w przypadku aplikacji z wieloma dietami, z taką opcją.

3.1. Low fat diet

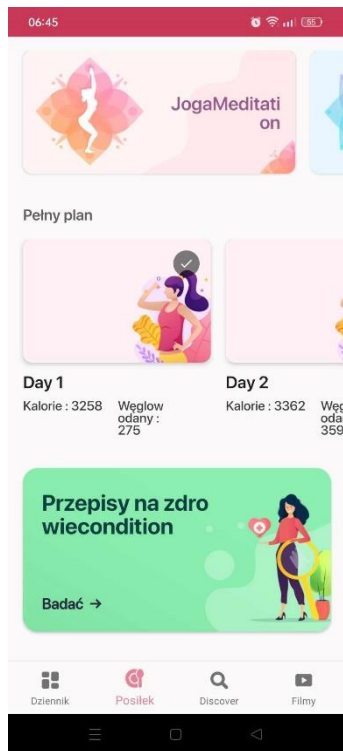
Jedyna aplikacja z dietą niskotłuszczową, którą udało się znaleźć autorowi, to aplikacja mobilna na Androida z Google Play o nazwie „Low fat diet”. Jest to aplikacja bardzo mało popularna (10 000+ pobrań). Generowane diety zawierają dania z tabelą wartości oraz składniki i przepis jak takie danie zrobić (rys. 3.1.).



Rys. 3.1. Wygenerowane tygodniowy plan na podstawie danych użytkownika

Pomimo zadowalającego, autora zdaniem, generowania diet, aplikacja jest kiepsko przetłumaczona, gdzie przykład tego można zauważyć na rysunku 3.2. Aplikacja jest przetłumaczona tylko częściowa z języka angielskiego na polski, w dodatku spora część polskich słów jest niepoprawnie lub w ogóle nieodmieniona. Nagminnie wyskakujące reklamy przy przechodzeniu do innych widoków również nie zachęca do korzystania z aplikacji.

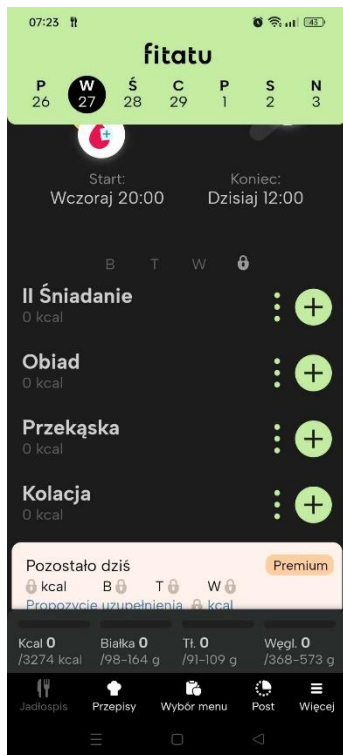
Bez zakupu premium, aplikacja generuje diety bez kolacji, w związku z tym, nie można powiedzieć, że aplikacja jest darmowa, gdyż nie można dostać pełnej diety w tej wersji.



Rys. 3.2. Przykład tłumaczenia

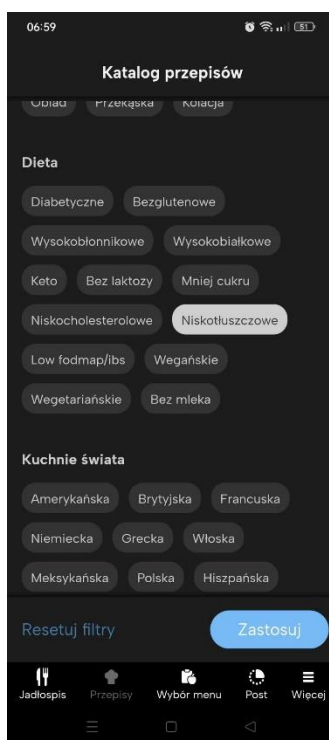
3.2. Fitatu

Fitatu jest jedną z najpopularniejszych aplikacji dietetycznych (5 mln+ pobrań).



Rys. 3.3. Widok główny w Fitatu

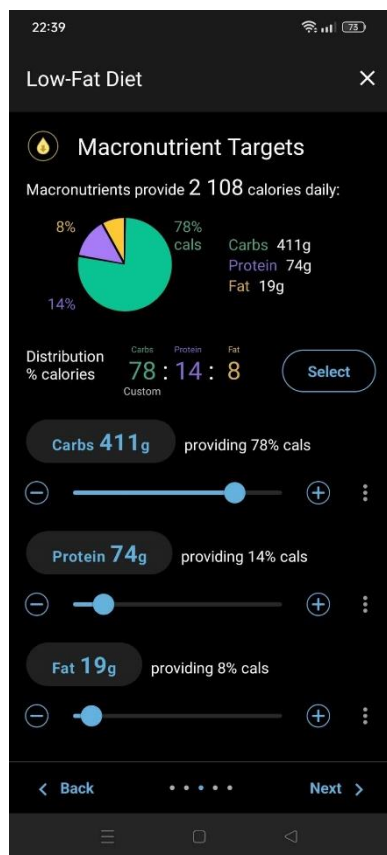
Jak widać na rysunku 3.3. aplikacja posiada bardzo przejrzysty i intuicyjny interfejs, który w dodatku jest bardzo estetyczny. Aplikacja niestety nie zawiera diety niskotłuszczowej w swoim repertuarze, aczkolwiek posiada katalog przepisów (rys. 3.4.), w którym można filtrować dania niskotłuszczowe. Jest tych dań bardzo dużo, w dodatku jest opcja wyszukiwania dań społeczności, dzięki czemu nie trzeba się przejmować zakupem premium, który w przeciwieństwie do poprzednika, jest jedynie luksusem, który dodaje użyteczne, lecz wcale niekonieczne funkcjonalności.



Rys. 3.4. Katalog przepisów Fitatu

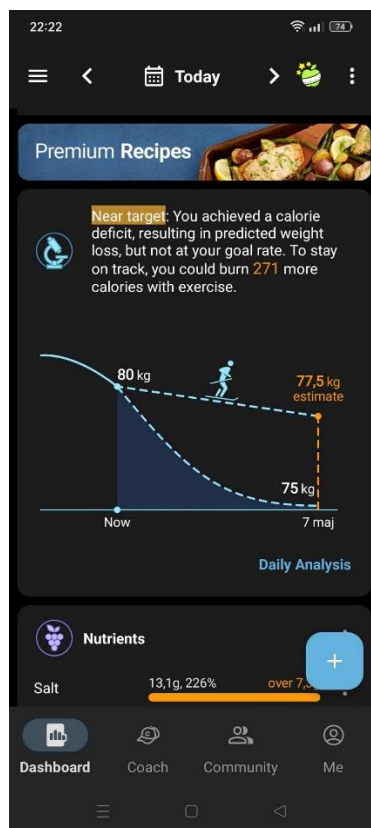
3.3. MyNetDiary

MyNetDiary jest aplikacją dietetyczną, która nie posiada żadnych darmowych diet, jednak w dietach premium znajduje się dieta niskotłuszczowa. W tej diecie domyślnie 25% spożywanych kalorii jest w postaci tłuszczu, jednak można to zmienić przy pomocy suwaków (rys. 3.5.) do takich wartości, jakich byśmy chcieli. W dodatku jest opcja wyłączenia z diety wybranych rodzajów tłuszczów, co pomaga unikać najbardziej szkodliwych, takich jak tłuszcze trans czy nasycone.



Rys. 3.5. Suwaki z makroskładnikami

Aplikacja posiada mnogą ilość funkcjonalności, np. takich jak przewidywanie naszej wagi (rys. 3.6.) na końcu miesiąca na podstawie naszych dotychczasowych nawyków żywieniowych. Pobiera od użytkownika dane na temat jego masy ciała, aktywności fizycznej, spożytych dań, wypitej wody oraz szczegółowe dane odnośnie makroskładników i to wszystko w darmowej wersji. Z tego względu nie dziwi duża popularność aplikacji, ponad 5 milionów pobrań w Google Play.



Rys. 3.6. Przewidywana waga

3.4. Podsumowanie

Na podstawie analizy rynku, autor doszedł do wniosku, iż brakuje aplikacji poświęconym dietom niskołuszczowym, a istniejące często wymagają zakupu premium. Dlatego chciał on stworzyć aplikację, która byłaby zupełnie darmowa. Rezygnacja z opcji jakiegokolwiek premium daje do zrozumienia użytkownikowi, że od początku korzysta w pełni z możliwości aplikacji tworzonej w tej pracy.

Ze wszystkich aplikacji wybrane zostały 3, które najlepiej przedstawiają obecną pozycję rynku.

Pierwsza przedstawia obraz większej części aplikacji z dietą niskołuszczową, tj. większość funkcji schowana za subskrypcją, kiepsko wykonany interfejs z wybrakowanym tłumaczeniem.

Druga jest aplikacją do wielu diet oraz jest bardzo popularna, jednak pomimo tego, że nie zawiera samej diety niskołuszczowej, posiada pewne opcje w postaci filtrowania dań oraz sama w sobie jest bardzo użyteczna jako aplikacja do poprawienia swojego zdrowia czy sylwetki.

MyNetDiary posiada równie dobry interfejs co Fitatu, cieszy się również podobną

popularnością, posiada jednak dietę niskotłuszczową, w dodatki w pełni konfigurowalną, nawet można wykluczyć konkretne rodzaje tłuszczu, co jest rzadkością w aplikacjach dietetycznych. Jedynym minusem jest fakt, że sama dieta oraz funkcjonalności z nią związane są dostępne jedynie za płatną subskrypcją. Pomimo tego, aplikacja ta została uznana przez autor za złoty standard tego jak powinna wyglądać aplikacja o dietach, nie tylko niskotłuszczowych, ale również innych.

4. Wykorzystane technologie

Aplikacja będzie robiona zgodnie z założeniem webstacka SERN, tj. MySQL, Express.js, React, Node.js.

MySQL

MySQL jest system zarządzania relacyjnymi bazami danych (RDBMS) i jest rozwiązaniem darmowym oraz open-source. Został napisany w C oraz C++ i działa na większości popularnych systemach operacyjnych, takich jak Windows, macOS czy Linux oraz wielu innych. Od lat jest drugim najpopularniejszym RDBMS i najpopularniejszym darmowym [5].

Express.js

Express.js jest frameworkiem do aplikacji internetowych, służącym do tworzenia RESTowego api w Node.js. Jest rozwiązaniem darmowym oraz open-source. Express jest najpopularniejszym frameworkiem do Node.js, dzięki czemu jego nauka jest bardzo łatwa, ze względu na dużą liczbę poradników do niego. To, że jest lekki i minimalistyczny, powoduje, że jest najczęściej wybieranym frameworkiem do Node.js.

React

React [10] jest biblioteką do języka programowania JavaScript. Służy do tworzenia front-endu w aplikacjach internetowych. Przoduje wśród bibliotek i frameworków do JavaScriptu [1].

Node.js

Node.js jest wieloplatformowym, back-endowym środowiskiem uruchomieniowym do pisania aplikacji w JavaScript. Działa asynchronicznie, co znacząco wpływa na prędkość działania aplikacji i pisania kodu. Działa na Google V8 JavaScript, co umożliwia kompilację do kodu maszynowego i wykonanie kodu bezpośrednio, z pominięciem interpretacji. Serwer tej aplikacji działa właśnie na serwerze Node'owym.

Sequelize

Sequelize jest narzędziem do mapowania obiektowo-relacyjnie dla wielu popularnych RDBMS, takich jak Microsoft MySQL Server, MySQL, PostgreSQL, SQLite czy Oracle. Jest to najpopularniejszy ORM do Node.js [11]. Ze względu na popularność, a w związku z tym liczbę poradników, Sequelize jest bardzo przystępny do nauki i użytkowania.

Postman

Postman jest oprogramowaniem służącym do testowania API. Przy jego pomocy można wysyłać zapytania do przygotowanych endpointów i wyświetlać status odpowiedzi oraz odpowiedź. Dzięki temu można szybko i sprawnie sprawdzić dowolne API, zanim zostanie ono zaimplementowane na stronie, bądź gdy jest już zaimplementowane, przetestować czy działanie naszego API jest zgodnie z oczekiwanym działaniem.

Git

Git jest systemem rozproszonej kontroli wersji. Służy do śledzenia zmian w kodzie źródłowym projektu podczas jego rozwoju oraz kontroli owego projektu poprzez, m.in. możliwość wysyłania plików do repozytorium, tworzenia i scalania gałęzi oraz wyróżnia się od innych systemów kontroli wersji tzw. staging area (pl. obszar przejściowy), w którym pliki czekają na zatwierdzenie, zanim zostaną wysłane do repozytorium.

5. Projekt aplikacji

W tym rozdziale zostanie przedstawiony projekt aplikacji.

5.1. Wymagania funkcjonalne

- Użytkownik ma mieć możliwość rejestracji, logowania i wylogowania
- Użytkownik powinien mieć możliwość usunięcia konta
- Użytkownik powinien mieć możliwość zmiany języka oraz rozmiaru czcionki

5.2. Wymagania niefunkcjonalne

- Ciągły dostęp do połączenia internetowego
- Czytelny, minimalistyczny design (kolor czcionki w kontraście do tła, odpowiedni rozmiar czcionki, stonowana paleta kolorów)
- Strona dostępna cały czas (24/7)
- System powinien być zrobiony zgodnie z webstackiem SERN, tj. baza danych MySQL, serwer Node obsługiwany przy pomocy Expressa oraz strona zrobiona w ReactJS.
- Strona powinna działać na przeglądarkach Mozilla Firefox v102, Chrome v109 i Safari v12.1 lub nowszych oraz na wszystkich systemach operacyjnych
- Strona powinna być dostępna w językach polskim i angielskim

5.3. Scenariusze

Ten podrozdział jest poświęcony scenariuszom związanym z działaniem aplikacji. Na podstawie tych scenariuszy zostały przygotowane diagramy.

5.3.1 Rejestracji

Opis: Scenariusz opisujący rejestrację do serwisu

Aktorzy: Użytkownik

Warunki początkowe: Użytkownik znajduje się na stronie

Warunki końcowe: Użytkownik zakłada konto w aplikacji

Przebieg główny:

1. Użytkownik wchodzi na stronę webową aplikacji
2. Użytkownik wciska przycisk *Sign Up* w celu założenia konta
3. System przekierowuje użytkownika do podstrony z formularzem rejestracyjnym
4. Użytkownik wypełnia formularz
5. Użytkownik wciska przycisk *Choose diet*
6. System sprawdza poprawność wypełnienia formularza

7. System wyświetla użytkownikowi dostępne diety
8. Użytkownik wybiera dietę
9. Użytkownik wciska przycisk *Choose Calories*
10. System wyświetla użytkownikowi dostępne opcje kaloryczności
11. Użytkownik wybiera jedną z opcji
12. Użytkownik wciska przycisk *Create Account*
13. System sprawdza poprawność wszystkich danych
14. System przekierowuje użytkownika na stronę logowania

Przebieg alternatywny:

13. Użytkownik podaje niepełne dane
 - System wyświetla komunikat, żeby uzupełnić odpowiednie dane
13. Użytkownik podaje niepoprawne dane
 - System wyświetla komunikat o niepoprawnych danych

5.3.2 Logowania

Opis: Scenariusz opisujący logowanie do serwisu

Aktorzy: Użytkownik

Warunki początkowe: Użytkownik znajduje się na stronie

Warunki końcowe: Użytkownik zostaje zalogowany do aplikacji

Przebieg główny:

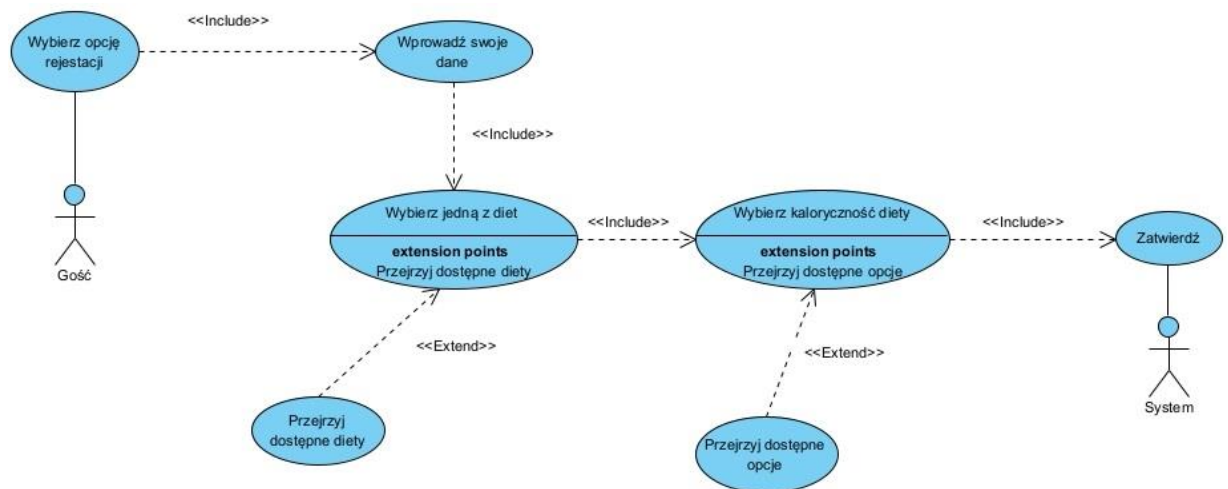
1. Użytkownik wchodzi na stronę webową aplikacji
2. Użytkownik wypełnia formularz logowania
3. Użytkownik klika w przycisk *Login*
4. System sprawdza poprawność wypełnienia formularza oraz zgodność danych
5. System tworzy token i wysyła go użytkownikowi
6. System przekierowuje użytkownika na stronę główną

Przebieg alternatywny:

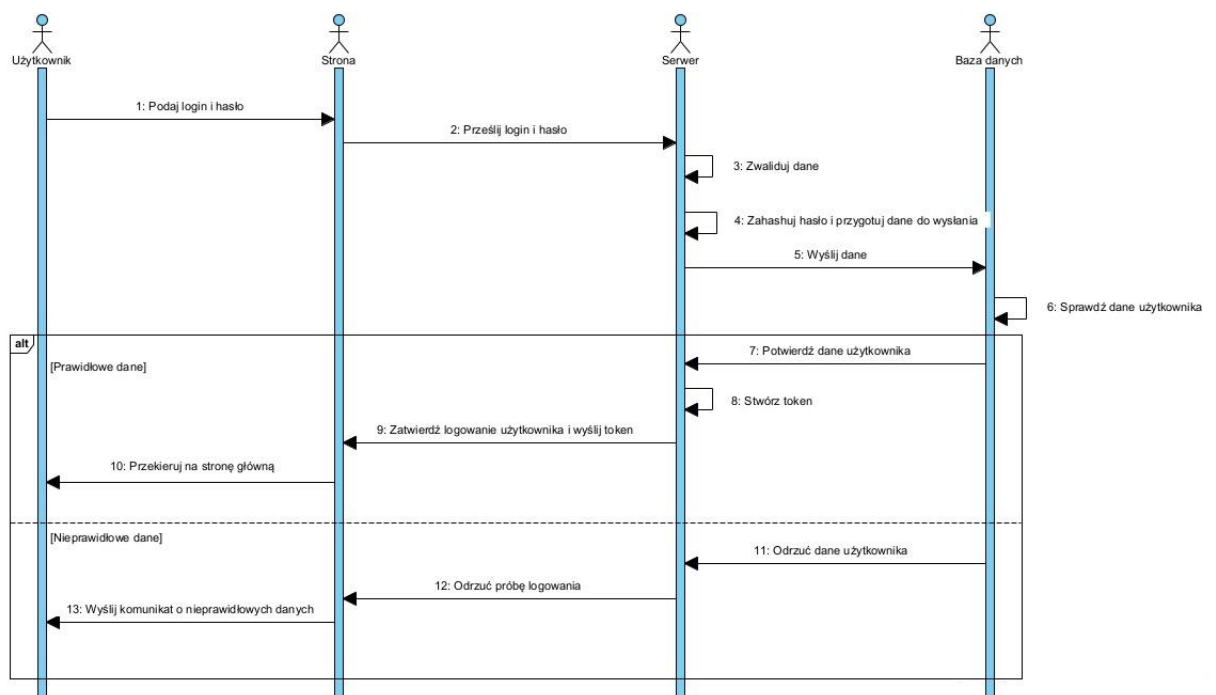
4. Użytkownik podaje niepełne dane
 - System wyświetla komunikat, żeby uzupełnić odpowiednie dane
4. Użytkownik podaje nieprawidłowe dane
 - System wyświetla komunikat o niepoprawnych danych

5.4. Diagramy UML

Na podstawie scenariuszy zostały stworzone diagramy UML [9]. Jest to język modelowania, który służy do tworzenia i prezentowania systemów w formie graficznej. Ma to na celu ułatwić pracę programistom i jednocześnie być czytelne dla i zrozumiałe dla klienta, który może być nietechniczny. W postaci diagramu przypadku użycia oraz sekwencyjnego zostały przedstawione scenariusze rejestracji (rys. 5.1.) i logowania (rys. 5.2.).



Rys. 5.1. Diagram przypadków użycia - rejestracja



Rys. 5.2. Diagram sekwencyjny - logowanie

5.5. Baza danych

Baza danych jest zrealizowana w MySQL przy pomocy ORM Sequelize. Na rysunku 5.3. widoczne jest 8 tabel.

Tabele *Diets* i *Users* są połączone relacją 1:N, ponieważ do jednego użytkownika może być przypisana tylko jedna dieta, ale sama dieta może być używana przez wielu użytkowników.

Tabele *Diets* i *Dishes_histories* są połączone relacją 1:N, ponieważ każda dieta może mieć tylko jedną historię dań, jednak jest wiele historii dań, po jednej przypisanej do każdej diety.

Tabele *Dishes* i *Diets_has_dishes* są połączone relacją 1:N, ponieważ jedno danie może wystąpić w danej diecie tylko raz, ale różne diety mogą mieć te same danie.

Tabele *Dishes* i *Macronutrients* są połączone relacją 1:N, ponieważ danie może mieć tylko jedną tabelę wartości odżywczych, ale jedna tabela wartości może być przypisana do wielu dań.

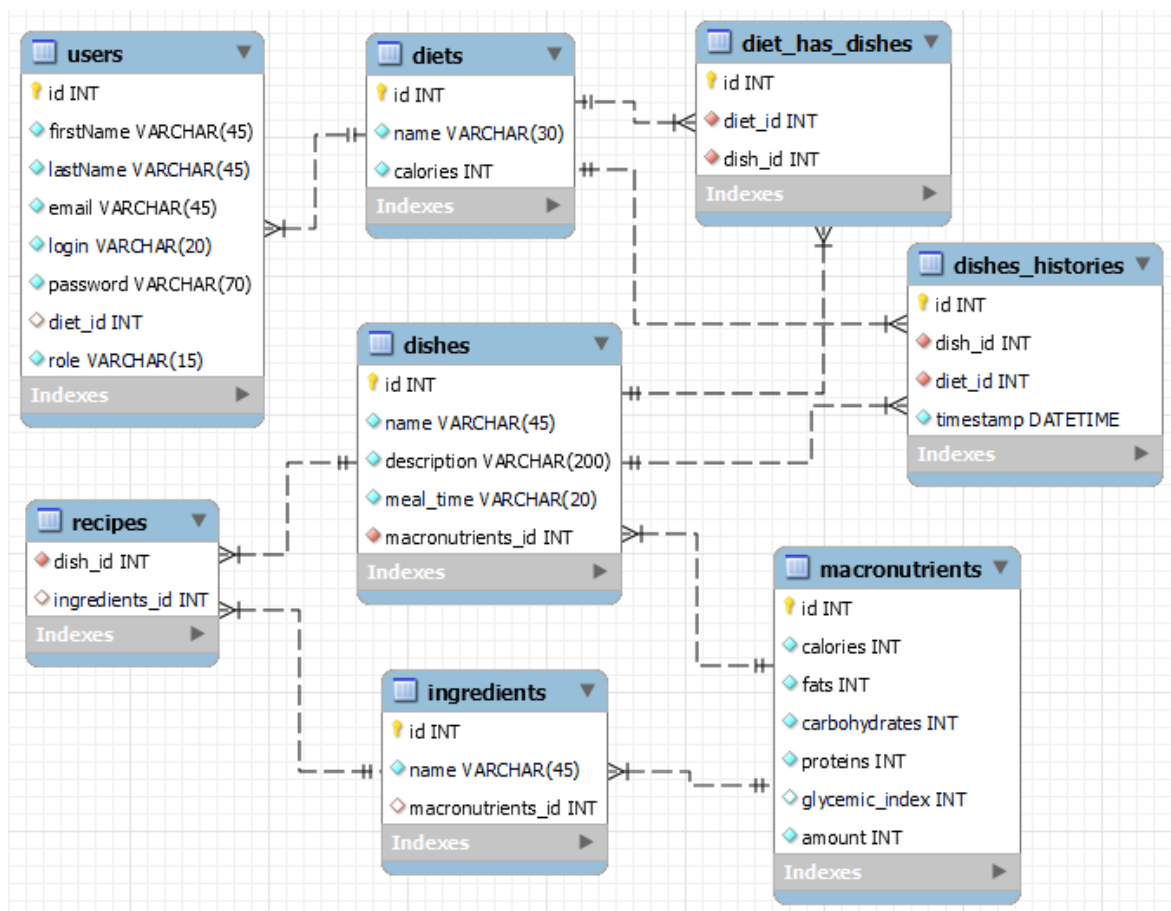
Tabele *Ingredients* i *Macronutrients* są połączone relacją 1:N, ponieważ składnik może mieć tylko jedną tabelę wartości odżywczych, ale jedna tabela wartości może być przypisana do wielu składników.

Tabela *Dishes_histories* jest tabelą, w której są przechowywane wylosowane dania z tego tygodnia, dieta oraz timestamp. Przy porównaniu id diety z tej tabeli oraz tabeli *Users* dostajemy informację, z której diety mają być wyświetlane dania użytkownikowi, a dzięki timestamp uzyskujemy informacje, które dania wyświetlać aktualnego dnia.

Tabela *Diets_has_dishes* jest tabelą pośrednią pomiędzy tabelami *Dishes* i *Diets*, w celu połączenia tych tabel. Wpływa to na wydajność i szybkość zapytań, gdy jest potrzeba wyświetlenia danych z obu tabel na raz.

Tabela *Recipes* jest tabelą pośrednią pomiędzy tabelami *Dishes* i *Ingredients*.

Opis poszczególnych pól tabel jest przedstawiony w tabeli 5.1.



Rys. 5.3 Schemat bazy danych

Cała baza opiera się o tabelę *dishes*, z którą są powiązane, pośrednio lub bezpośrednio, wszystkie tabele. Zawiera ona informacje o daniu, a przyłączone są do niej informacje o wartościach odżywczych w postaci tabeli *Macronutrients* oraz składnikach, poprzez tabelę *Recipes*. Jest to zrobione w ten sposób w celu uniknięcia szukania danych po całych tabelach z rekordami, gdy prościej i wydajniej jest odnosić się do rekordów poprzez identyfikator. Tabela *Dishes* jest połączona z tabelą *Diets* poprzez tabelę *Diets_has_dishes* z tego samego względu.

Baza jest w postaci normalnej, na przykładzie składników, są one w oddzielnej tabeli, ze względu na to, iż jedno danie składa się z wielu składników, czyli powinny być wydzielone, żeby uniknąć redundancji.

Tab. 5.1. Opis pól tabel

Tabela	Pole	Opis
Users	diet_id	Dieta wybrana i przypisana do użytkownika
	role	Rola użytkownika
Diets	calories	Ilość dziennych kalorii
Dishes	name	Nazwa dania
	description	Opis dania
	meal_time	Pora dnia dania, potrzebna do generowania diet
Ingredients	name	Nazwa składnika dania
Macronutrients	calories	Kalorie składnika bądź dania
	fats	Tłuszcze składnika bądź dania
	carbohydrates	Węglowodany składnika bądź dania
	proteins	Białko składnika bądź dania
	glycemic_index	Indeks glikemiczny składnika bądź dania
	amount	Ilość składnika bądź dania, podana w gramach
Dishes_histories	timestamp	Pora dodania dania przez algorytm

6. Implementacja

Po analizie rynku, zaprojektowaniu wszystkiego i dobraniu technologii w jakiej będzie wykonana aplikacja pozostała jej implementacja. Zgodnie z założeniami projektu, aplikacja generuje dania, na podstawie typu diety oraz jej kaloryczności.

6.1. Back-end

Back-end jest podstawą każdej aplikacji, stanowi połączenie pomiędzy bazą danych i klientem aplikacji. W przypadku tej aplikacji, serwer oprócz obsługi bazy przy pomocy ORMa Sequelize i modeli tabel, zajmuje się generowaniem diet, walidacją danych oraz w przypadku pierwszego uruchomienia, tworzy bazę za pomocą plików migracyjnych.

Jak widać na listingu 6.1., istnieją foldery: config, controllers, migrations, models, node_modules. Folder config zawiera plik konfiguracyjny wykorzystywanymi przez Sequelize do tworzenia oraz łączenia się z bazą. W bazie jest tworzony użytkownik z ograniczonymi możliwościami (rys. 6.1.). Serwer łączy się z bazą jako ten użytkownik. Ma to na celu zabezpieczyć bazę przed potencjalną utratą kontroli nad bazą.

	Grants for diet_user@localhost
▶	GRANT USAGE ON *.* TO `diet_user`@`localhost`
	GRANT SELECT, INSERT, UPDATE, DELETE ON `inzynierka`.* TO `diet_user`@`localhost`

Rys. 6.1. Uprawnienia użytkownika

Katalog node_modules zawiera biblioteki dla Node.js. Folder migrations zawiera pliki migracyjne, które przy pomocy wiersza poleceń Sequelize tworzą tabele, zawarte w nich atrybuty oraz relacje. Folder models, podobnie jak migrations zawiera strukturę bazy, jednak w tym przypadku jest to wykorzystywane przez Sequelize przy wysyłaniu zapytań do bazy danych.

Następnie, gdy już jest wszystko przygotowane, kontrolery oraz endpointy zawarte kolejno w controllers i routes zawierają API oraz algorytmy potrzebne do obsługi i działania bazy oraz klienta.

Listing 6.1. Struktura plików serwera

```

  server
  |
  +-- config
  |   +-- config.json
  |
  +-- controllers
  |   +-- diet_has_dishesController.js
  |   +-- dietsController.js
  |   +-- dishes_historiesController.js
  |   +-- dishesController.js
  |   +-- ingredientsController.js
  |   +-- macronutrientsController.js
  |   +-- recipesController.js
  |   +-- userController.js
  |
  +-- migrations
  |   +-- 1-diets.js
  |   +-- 1-macronutrients.js
  |   +-- 2-dishes.js
  |   +-- 2-ingredients.js
  |   +-- 2-users.js
  |   +-- diet_has_dishes.js
  |   +-- dishes_histories.js
  |   +-- recipes.js
  |
  +-- models
  |   +-- diet_has_dishes.js
  |   +-- diets.js
  |   +-- dishes_histories.js
  |   +-- dishes.js
  |   +-- index.js
  |   +-- ingredients.js
  |   +-- macronutrients.js
  |   +-- recipes.js
  |   +-- users.js
  |
  +-- node_modules
  |
  +-- routes
  |   +-- auth.js
  |   +-- diet_has_dishes.js
  |   +-- diets.js
  |   +-- dishes_histories.js
  |   +-- dishes.js
  |   +-- ingredients.js
  |   +-- macronutrients.js
  |   +-- recipes.js
  |   +-- users.js
  |
  +-- .dockerignore
  |
  +-- .env
  |
  +-- db.js
  |
  +-- Dockerfile
  |
  +-- index.js
  |
  +-- package-lock.json
  |
  +-- package.json

```

Listing 6.2. Funkcja odpowiadająca za rejestrację użytkownika

```
const postUser = asyncHandler (async (req, res) => {
  try {
    const { error } = validate(req.body);
    if (error)
      return res.status(400).send({ message: error.details[0].message });

    const userExists = await userSchema.findOne({ where: { email: req.body.email } });
    if (userExists)
      return res.status(409).send({ message: "User with given email already exists!" });

    const salt = await bcrypt.genSalt(Number(process.env.SALT));
    const hashPassword = await bcrypt.hash(req.body.password, salt);

    const newUser = await userSchema.create({
      firstName: req.body.firstName,
      lastName: req.body.lastName,
      email: req.body.email,
      login: req.body.login,
      password: hashPassword,
      diet_id: req.body.diet_id
    });

    res.status(201).send({ message: "User created successfully", data: newUser });
  } catch (error) {
    console.error("Error creating user:", error);
    res.status(500).send({ message: "Internal Server Error" });
  }
});
```

Listing 6.3. Funkcja odpowiadająca za walidację danych rejestracji

```
const validate = (data) => {
  const schema = Joi.object({
    firstName: Joi.string().required().label("First Name"),
    lastName: Joi.string().required().label("Last Name"),
    email: Joi.string().email().required().label("Email"),
    login: Joi.string().required().label("Login"),
    password: passwordComplexity().required().label("Password"),
    repeat_password: Joi.string().valid(Joi.ref('password')).required().label("Repeat Password").messages({
      'any.only': 'Passwords must be the same',
    }),
    diet_id: Joi.number().integer()
  })
  return schema.validate(data)
}
```

Na listingu 6.2. znajduje się kod funkcji, która odpowiada za tworzenie użytkownika. Najpierw waliduje dane podane przez użytkownika (listing 6.3.), później sprawdza czy użytkownik z podanym emailem istnieje, jeżeli nie, to haszuje hasło i wysyła dane razem z zahaszowanym hasłem do bazy. Jeżeli nie nastąpi jakiś problem z bazą, użytkownik zostanie pomyślnie zarejestrowany.

Walidacja danych jest robiona przy pomocy Joi, biblioteki specjalnie stworzonej do walidacji danych w języku JavaScript. Sprawdzane jest, czy zostały wypełnione

wszystkie pola oraz w przypadku hasła, ze względów bezpieczeństwa, wymagane jest min. 8 znaków, z czego co najmniej jeden specjalny, jedna cyfra, jedna mała i jedna duża litera. Wymagane jest również powtórne podanie hasła.

Hasło jest haszowane przy pomocy algorytmu bcrypt, z biblioteki o takiej samej nazwie. Jest to algorytm, tzw. slow hashing algorithm, czyli algorytm, który jest wolniejszy niż większość algorytmów haszujących, dzięki czemu jego złamanie również jest wolniejsze, dlatego jest on popularny i zalecany do haszowania haseł, zamiast, np. algorytmów z rodziny SHA [10].

Listing 6.4. Funkcja odpowiadająca za logowanie użytkownika

```
router.post("/", async (req, res) => {
  const { error } = validate(req.body);
  if (error)
    return res.status(400).send({ message: error.details[0].message });
  const user = await userSchema.findOne({ where: { login: req.body.login } })

  if (!user)
    return res.status(401).send({ message: "Invalid login or password" })
  const validPassword = await bcrypt.compare(
    req.body.password,
    user.password
  )
  if (!validPassword)
    return res.status(401).send({ message: "Invalid login or password" })
  const token = jwt.sign({ userId: user.id, role: user.role }, process.env.JWTPRIVATEKEY, { expiresIn: '7d' })
  res.status(200).send({ data: { token }, message: "logged in successfully" })
})
```

Listing 6.5. Funkcja odpowiadająca za walidację danych logowania

```
const validate = (data) => {
  const schema = Joi.object({
    login: Joi.string().required().label("Login"),
    password: Joi.string().required().label("Password"),
  })
  return schema.validate(data)
}
```

Logowanie (listing 6.4.) przebiega podobnie jak rejestracja, z tym, że z mniejszą liczbą danych. Najpierw dane są sprawdzane pod kątem poprawności (listing 6.5.), później czy użytkownik istnieje, a następnie hasło jest haszowane algorytmem bcrypt i wysyłane do bazy w celu porównania. Jedyna istotna różnica, to tworzenie tokenu JWT, przy pomocy biblioteki jsonwebtoken. JWT służy do bezpiecznej komunikacji pomiędzy klientem i serwerem. W tym przypadku JWT zawiera id użytkownika oraz jego rolę. Jest to bezpieczne, ponieważ wygenerowany token zawiera prywatny klucz, który służy jako

‘podpis’ serwera, co powoduje, że takiego tokenu nie można łatwo podrobić i podszyć się pod administratora.

Listing 6.6. Walidacja tokenu i roli administratora oraz związane z nimi endpointy

```
const verifyToken = (req, res, next) => {
  const token = req.header('Authorization')?.replace('Bearer ', '');
  if (!token) {
    return res.status(401).json({ message: 'Access denied. No token provided.' });
  }
  try {
    const decoded = jwt.verify(token, process.env.JWTPRIVATEKEY);
    req.user = decoded;
    next();
  } catch (error) {
    res.status(400).json({ message: 'Invalid token.' });
  }
};

const isAdmin = (req, res, next) => {
  if (req.user && req.user.role === 'admin') {
    next();
  } else {
    res.status(403).json({ message: 'Access denied. Not an admin.' });
  }
};

router.get("/verify", verifyToken, (req, res) => {
  res.send(req.user)
});

router.get("/admin", verifyToken, isAdmin, (req, res) => {
  res.send("Admin");
});
```

Token jest walidowany i dekodowany przy pomocy funkcji z biblioteki jsonwebtoken (listing 6.6.). Następnie wysyłana jest informacja zwrotna do klienta, żeby wiedział jakie informacje może wyświetlić użytkownikowi (w przypadku roli) lub jakie dane mu wyświetlić (w przypadku id użytkownika).

Listing 6.7. Funkcja odpowiadająca za wyświetlanie dań

```
const getDishesH = asyncHandler(async (req, res) => {
  try {
    const id=req.query.UID;
    const query = await sequelize.query("SELECT dishes_histories.dish_id, dishes_histories.timestamp, recipes.ingredients_id FROM dishes_histories "
    + "LEFT JOIN recipes ON recipes.dish_id = dishes_histories.dish_id "
    + "RIGHT JOIN users ON users.diet_id=dishes_histories.diet_id WHERE users.id=${id};",
    { type: Sequelize.QueryTypes.SELECT });

    const detailedDataMap = new Map();
    for (const dishes of query) {
      const dishId = dishes.dish_id;
      if (!detailedDataMap.has(dishId)) {
        detailedDataMap.set(dishId, { dish: null, ingredients: [] });
      }
      const dishData = detailedDataMap.get(dishId);
      if (!dishData.dish) {
        dishData.dish = await getDishById(dishId);
      }
      if (dishes.ingredients_id !== null) {
        const ingredient = await getIngredientById(dishes.ingredients_id);
        dishData.ingredients.push(ingredient);
      }
    }
    const dishesHistory = Array.from(detailedDataMap.values()).map(({ dish, ingredients }) => ({ dish, ingredients }));

    res.status(200).json({ data: [...dishesHistory] });
  } catch (error) {
    console.error("Error retrieving dishes with recipe data:", error);
    throw error;
  }
});
```

Funkcja widoczna na listingu 6.7. pobiera id użytkownika (UID) z parametrów żądania i wysyła zapytanie, w którym na podstawie UID wyszukuje dietę danego użytkownika, szuka takiej samej diety w historii dań i z tabeli *Recipes* pobiera szczegółowe dane odnośnie tych dań. Funkcja grupuje wszystkie powtarzające się pary danie-składnik jako jedno danie – wiele składników i zwraca tablicę dań ze szczegółowymi danymi i wszystkimi składnikami.

6.2. Generowanie diet

Najważniejszym algorytmem w całym tym projekcie jest algorytm służący do generowania diety. Na listingu 6.8. widnieje funkcja, która m.in. za to odpowiada. Funkcja ta najpierw sprawdza jaki jest dzień tygodnia, w przypadku każdego poniedziałku, następuje reset diet, tzn. tabela *Dishes_histories* zostaje opróżniona, następnie dla każdej diety po kolei są wybierane dania i dzielone na podstawie pory dnia. Dania muszą spełnić pewne warunki, żeby zostały w ogóle brane pod uwagę przy generowaniu diety. Te warunki to:

- Śniadanie musi stanowić 30%-35% kalorii z całego dnia
- Obiad musi stanowić 45%-50% kalorii z całego dnia
- Kolacja musi stanowić 15%-20% kalorii z całego dnia
- Żadne danie nie może zawierać więcej niż równowartość 20% kalorii w postaci tłuszczu

Kiedy dania już zostaną zebrane w tabelach, z podziałem na pory dnia, pozostaje wybrać losowe 3, po jednym dla każdej pory i dodać do tabeli *Dishes_histories*.

Sumarycznie, wylosowane dania będą miały od 90%-105% kalorii wybranej przez użytkownika diety, średnio 97,5%, co zapewnia powolny, lecz stały, ujemny bilans kaloryczny. Spadek masy ciała będzie powolny, lecz użytkownik nie będzie miał problemów ze zmęczeniem, wynikającym z dużego obciążenia kalorii i ciągłego braku energii. Jest to częsty problem, w wyniku którego wiele osób nie trzyma się danej diety odpowiednio rygorystycznie lub całkiem jej zaprzestaje.

Dane są losowane dla każdej diety, w rezultacie otrzymujemy liczbę rekordów $3n$, gdzie n to liczba diet znajdujących się w bazie danych, maksymalnie $21n$, dla 7 dni tygodnia.

Algorytm jest uruchamiany przy pomocy funkcji z biblioteki *cron*, automatycznie, każdego dnia o 2:00 (zapas ok. 2h, gdyby ktoś chciał spojrzeć na to jakiej diety miał się trzymać).

Listing 6.8. Algorytm generujący dania dla każdej diety

```
const generateDiet = asyncHandler(async (req, res) => {
  try {
    const currentDay = new Date().getDay();
    if (currentDay === 1) {
      await dishes_historiesSchema.destroy({ where: {} });
    }
    const diets = await dietSchema.findAll({});

    for (const diet of diets) {
      const dietId = diet.id;
      const dietCalories = diet.calories;

      const maxCalories = [Math.floor(dietCalories * 0.35), Math.floor(dietCalories * 0.5), Math.floor(dietCalories * 0.2)]
      const minCalories = [Math.floor(dietCalories * 0.3), Math.floor(dietCalories * 0.45), Math.floor(dietCalories * 0.15)]

      const maxFats = [Math.floor(maxCalories[0] / 8 * 0.2), Math.floor(maxCalories[1] / 8 * 0.2), Math.floor(maxCalories[2] / 8 * 0.2)]

      const getDishesByMealTimeAndDiet = async (mealTime) => {
        const dietData = await getDietHasDishById(dietId);
        const allDishes = dietData.dishes;
        const filteredDishes = allDishes.filter(dish => dish.meal_time === mealTime);
        return filteredDishes;
      };

      const getRandomDishByMealTime = async (dishes, minCal, maxCal, fats) => {
        let selectedDish;
        while (!selectedDish && dishes.length > 0) {
          const randomIndex = Math.floor(Math.random() * dishes.length);
          const randomDish = dishes[randomIndex];
          const macronutrientData = await getMacronutrientById(randomDish.macronutrientData.id);
          if (macronutrientData && macronutrientData.calories >= minCal && macronutrientData.calories <= maxCal && macronutrientData.fats <= fats) {
            selectedDish = randomDish;
          } else {
            dishes.splice(randomIndex, 1);
          }
        }
        return selectedDish;
      };

      const timestamp = new Date();

      try {
        let existingDishIds = await getDishesHBByDiet(dietId);

        const breakfastDishes = await getDishesByMealTimeAndDiet("Breakfast", dietId);
        const dinnerDishes = await getDishesByMealTimeAndDiet("Dinner", dietId);
        const supperDishes = await getDishesByMealTimeAndDiet("Supper", dietId);

        const suitableBreakfastDishes = breakfastDishes.filter(dish => !existingDishIds.includes(dish.id));
        const suitableDinnerDishes = dinnerDishes.filter(dish => !existingDishIds.includes(dish.id));
        const suitableSupperDishes = supperDishes.filter(dish => !existingDishIds.includes(dish.id));

        const breakfastDish = await getRandomDishByMealTime(suitableBreakfastDishes, minCalories[0], maxCalories[0], maxFats[0]);
        const dinnerDish = await getRandomDishByMealTime(suitableDinnerDishes, minCalories[1], maxCalories[1], maxFats[1]);
        const supperDish = await getRandomDishByMealTime(suitableSupperDishes, minCalories[2], maxCalories[2], maxFats[2]);

        if (breakfastDish && dinnerDish && supperDish) {
          await Promise.all([
            dishes_historiesSchema.create({ diet_id: dietId, dish_id: breakfastDish.id, timestamp }),
            dishes_historiesSchema.create({ diet_id: dietId, dish_id: dinnerDish.id, timestamp }),
            dishes_historiesSchema.create({ diet_id: dietId, dish_id: supperDish.id, timestamp })
          ]);
          console.log("Algorithm executed successfully for diet ", dietId);
        } else {
          console.log("Couldn't find suitable dishes for diet ", dietId);
        }
      } catch (error) {
        console.error("Error processing diet: ", dietId, error.message);
        continue;
      }
    }
    res.status(200).send("Diet generation completed successfully");
  } catch (error) {
    console.error("Error in algorithm:", error);
    res.status(500).send("Internal server error");
  }
});

cron.schedule("00 02 * * *", generateDiet);
```

6.3. Front-end

Front-end jest istotną częścią każdej aplikacji. To z nią użytkownik ma najwięcej styczności i jest kluczowa dla jego doświadczenia. Warstwa wizualna jest pierwszą z jaką użytkownik ma styczność i to od niej zależy czy użytkownik po wejściu na stronę zechce na niej pozostać czy też nie.

Pomimo, że większa część aplikacji to część back-endowa, to autor postarał się, żeby front-end nie pozostał zaniedbany. Interfejsy, umieszczone w podrozdziale 6.4, są zrobione w miarę nowocześnie, tzn. minimalistyczny design, prosty, lecz czytelny. Przy tworzeniu warstwy wizualnej zostały wzięte pod uwagę osoby z niepełnosprawnościami wzrokowymi – czcionka jest w odpowiednim kontraście do tła, a w opcjach aplikacji jest możliwość zmiany rozmiaru czcionki na większy. Szata graficzna jest w ciemniejszych tonach ze względu na to, iż odpowiada to preferencjom większości ludzi [2].

Listing 6.9. Struktura plików klienta

```

client
├── node_modules
├── public
│   ├── locales
│   │   ├── en
│   │   │   └── translation.json
│   │   └── pl
│   │       └── translation.json
│   ├── favicon.ico
│   ├── index.html
│   ├── logo192.png
│   ├── logo512.png
│   ├── manifest.json
│   └── robots.txt
├── src
│   ├── components
│   │   ├── MainButton.jsx
│   │   ├── Popup.js
│   │   ├── styles.modules.css
│   │   └── useFontSize.js
│   ├── pages
│   │   ├── Login
│   │   │   ├── index.jsx
│   │   │   └── styles.module.css
│   │   ├── Main
│   │   │   ├── DishDisplay.js
│   │   │   ├── index.jsx
│   │   │   └── styles.module.css
│   │   ├── Options
│   │   │   ├── index.jsx
│   │   │   └── styles.module.css
│   │   ├── Signup
│   │   │   ├── DietCalories.js
│   │   │   ├── DietCarouselDisplay.js
│   │   │   ├── FormDisplay.js
│   │   │   ├── index.jsx
│   │   │   └── Signup.module.css
│   ├── utils
│   │   ├── fonts.css
│   │   ├── i18n.js
│   │   ├── isAdmin.js
│   │   ├── verifyToken.js
│   │   ├── App.css
│   │   ├── App.js
│   │   ├── App.test.js
│   │   ├── index.css
│   │   ├── index.js
│   │   ├── normal.png
│   │   ├── reportWebVitals.js
│   │   ├── setupTests.js
│   │   ├── vege.png
│   │   ├── .dockerignore
│   │   ├── .gitignore
│   │   ├── Dockerfile
│   │   ├── package-lock.json
│   │   └── package.json

```

Na listingu 6.9. widoczne są foldery: locales, components, pages, utils oraz node_modules. Folder locales zawiera pliki z wersjami językowymi, polską (listing 6.10.) i angielską (listing 6.11.). W folderze components znajdują się komponenty, z których korzystają strony. Są to komponenty ogólnego przeznaczenia, wykorzystywane przez wiele stron i związane z UI. W folderze pages znajdują się strony oraz komponenty, które są tylko przez nie używane. Oprócz tego jest jeszcze folder utils oraz pliki takie jak App.js czy index.js. W utils znajdują się komponenty niezwiązane z UI, które wspomagają działanie stron, np. isAdmin, który sprawdza na podstawie tokenu rolę użytkownika. App.js odpowiada za adresy, tzn. pod jakim adresem znajduje się dana strona oraz jakie warunki trzeba spełnić, żeby się na nią dostać (np. bycie zalogowanym).

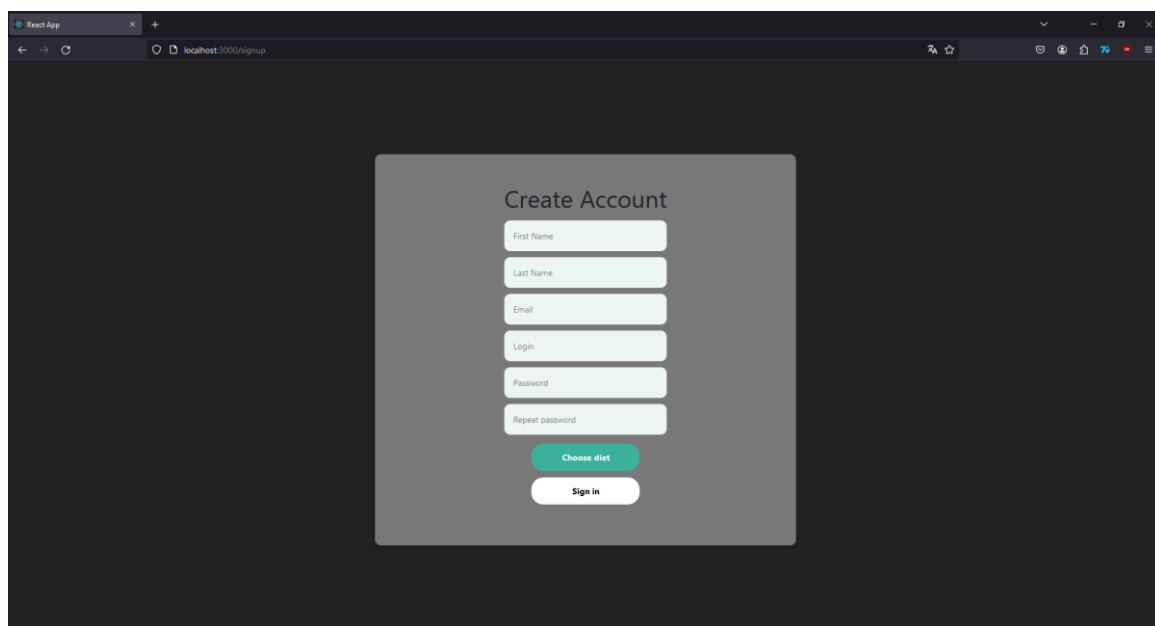
Listing 6.10. Tłumaczenia w języku polskim

```
{
  "Main Button": "Apka dietetyczna",
  "Breakfast": "Śniadanie: ",
  "Lunch": "II Śniadanie: ",
  "Dinner": "Obiad: ",
  "Supper": "Kolacja: ",
  "Options": "Opcje",
  "Logout": "Wyloguj",
  "Login": "Zaloguj",
  "Delete User": "Usuń swoje konto",
  "Language": "Język:",
  "Font size": "Rozmiar czcionki:",
  "AddDish": "Dodaj danie",
  "Dish Name": "Nazwa dania",
  "Description": "Opis",
  "Fats": "Tł:",
  "Proteins": "B:",
  "Carbohydrates": "W:",
  "Amount": "Ilość:",
  "Calories": "Kalorie",
  "Close": "Zamknij",
  "Submit": "Dodaj",
  "Ingredients": "Składniki"
}
```

Listing 6.11. Tłumaczenia w języku angielskim

```
"Main Button": "Diet app",  
"Breakfast": "Breakfast: ",  
"Lunch": "Lunch: ",  
"Dinner": "Dinner: ",  
"Supper": "Supper: ",  
"Options": "Options",  
"Logout": "Logout",  
"Login": "Login",  
"Delete User": "Delete your account",  
"Language": "Language:",  
"Font size": "Font size:",  
"AddDish": "Add dish",  
"Dish Name": "Dish name",  
"Description": "Description",  
"Fats": "F:",  
"Proteins": "P:",  
"Carbohydrates": "C:",  
"Amount": "Amount:",  
"Calories": "Calories",  
"Close": "Close",  
"Submit": "Submit",  
"Ingredients": "Ingredients"
```

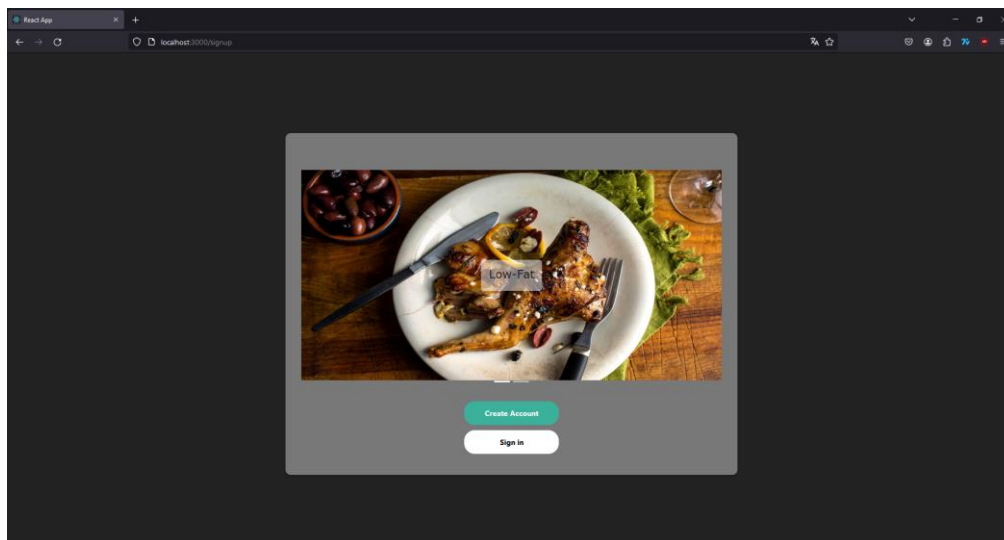
Strona rejestracji (listing 6.12.) składa się z trzech komponentów. Pierwszy z nich to formularz rejestracji (Rys. 6.2.), w którym znajdują się pola z podstawowymi danymi użytkownika, które należy uzupełnić.



The screenshot displays a web browser window with the address bar showing 'localhost:3000/signup'. The main content is a 'Create Account' form. The form has a title 'Create Account' and six input fields: 'First Name', 'Last Name', 'Email', 'Login', 'Password', and 'Repeat password'. Below these fields are two buttons: a green 'Choose diet' button and a white 'Sign in' button. The form is set against a dark gray background.

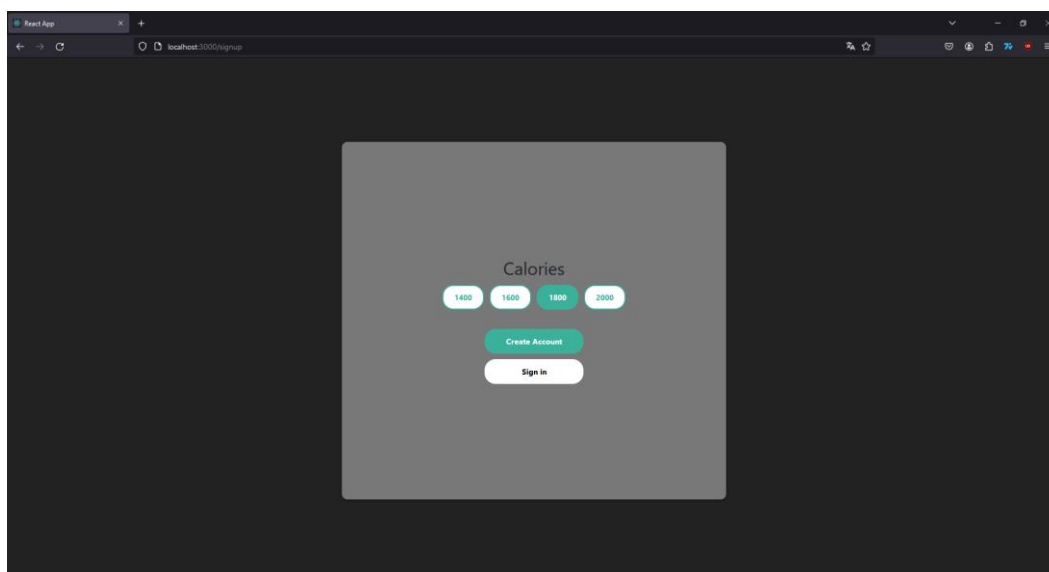
Rys. 6.2. Widok rejestracji

Następnie po wciśnięciu przycisku *Choose Diet* zostaje wyświetlony nowy komponent zamiast formularza, komponent z wyborem diety (Rys. 6.3.). Znajduje się w nim tzw. ‘karuzela’, z której można wybrać dietę, a następnie przejść do kolejnego komponentu, poprzez wciśnięcie *Choose Calories*.



Rys. 6.3. Widok wyboru diety

Wtedy wyświetla się ostatni komponent, czyli wybór kalorii (rys. 6.4.). Po wybraniu jednej z opcji poprzez wciśnięcie przycisku danej opcji, zostaje podświetlony przycisk. Jeżeli użytkownik jest już pewny swojego wyboru, pozostaje zatwierdzić swój wybór, co spowoduje wysłanie danych do serwera, gdzie zostaną sprawdzone, jeśli wszystko jest poprawnie, użytkownik właśnie założył nowe konto. W innym wypadku zostanie wyświetlony odpowiedni komunikat z prośbą o uzupełnienie lub poprawienie danych.

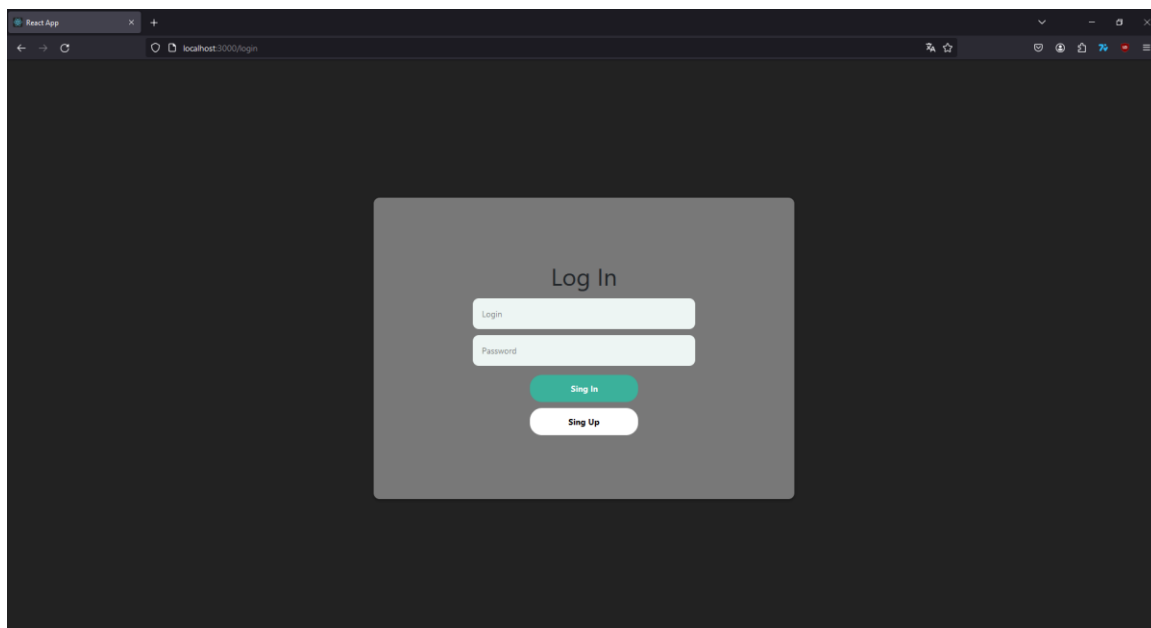


Rys. 6.4. Widok wyboru kalorii

Listing 6.12. Fragment kodu strony rejestracji

```
return (
  <div className={styles.signup_container}>
    <div className={styles.signup_form_container}>
      <form className={styles.form} onSubmit={handleSubmit}>
        {!dataFilled && (
          <FormDisplay
            handleFormChange={handleFormChange}
            firstName={data.firstName}
            lastName={data.lastName}
            email={data.email}
            login={data.login}
            password={data.diet_id.password}
            repeat_password={data.repeat_password}
            handleFillData={handleFillData}
          />
        )}
        {dataFilled && !dietChosen && (
          <DietCarouselDisplay
            handleSlideChange={handleSlideChange}
            handleChooseDiet={handleChooseDiet}
          />
        )}
        {dietChosen && (
          <DietCalories
            handleCaloriesChange={handleCaloriesChange}
            handleSubmit={handleSubmit}
          />
        )}
        {error && <div className={styles.error_msg}>{error}</div>}
        <Link to="/login">
          <button name="login_btn" type="button" className={styles.white_btn}>
            Sign in
          </button>
        </Link>
      </form>
    </div>
  </div>
);
};
```

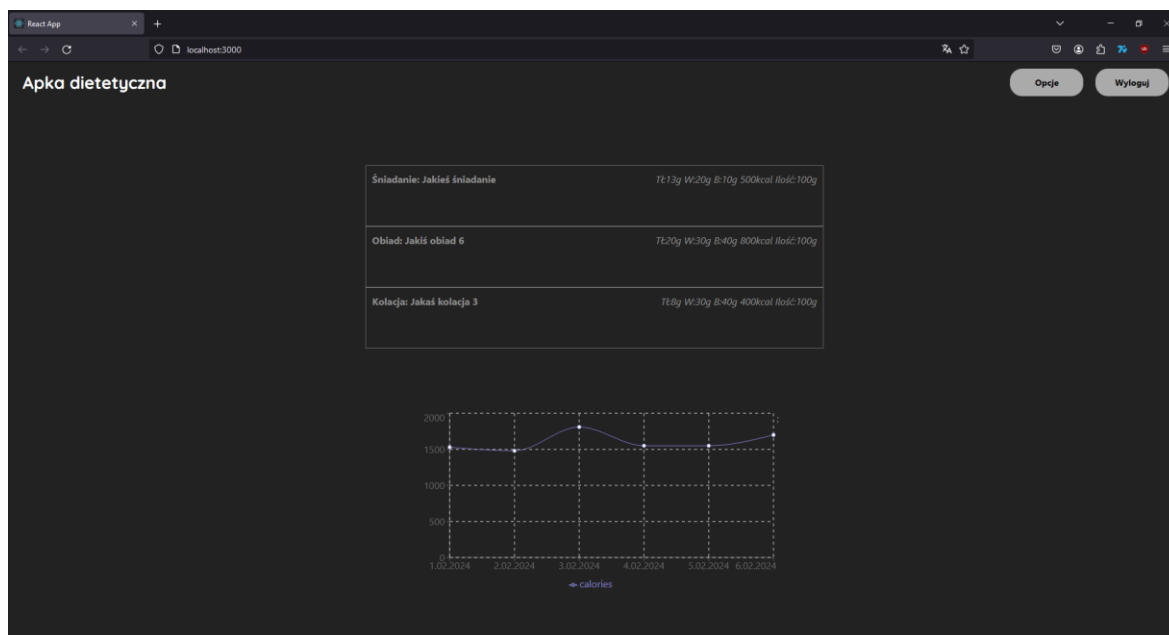
W przypadku, gdy w naszej pamięci lokalnej nie znajduje się ważny token, po wejściu na stronę lub po pomyślnej rejestracji, użytkownika wita widok logowania (rys. 6.5.). Podobnie jak w formularzu rejestracyjnym, tutaj należy podać swoje dane logowania, które zostaną wysłane do serwera w celach walidacji i w przypadku niepowodzenia zostanie wyświetlony komunikat o niepoprawnych danych. Jeżeli dane są poprawne, to do pamięci lokalnej zostaje zapisany token, który ma ważność 7 dni, a użytkownik zostaje przekierowany na stronę główną (rys. 6.6.).



Rys. 6.5. Widok logowania

W momencie ładowania się strony głównej zostają wysłane zapytania o weryfikację tokenu w celu uzyskania informacji na temat diety oraz roli użytkownika. Dane o diecie są potrzebne do wyświetlenia odpowiednich dań, a rola w celach weryfikacji jakie dane bądź podstrony powinny zostać udostępnione użytkownikowi. Na stronie głównej wyświetlają się dania, przyciski wylogowania i opcji oraz wykres dań z obecnego tygodnia (listing 6.14.).

W przypadku wylogowania, zostaje usunięty token, a użytkownik przekierowany z powrotem na stronę logowania.

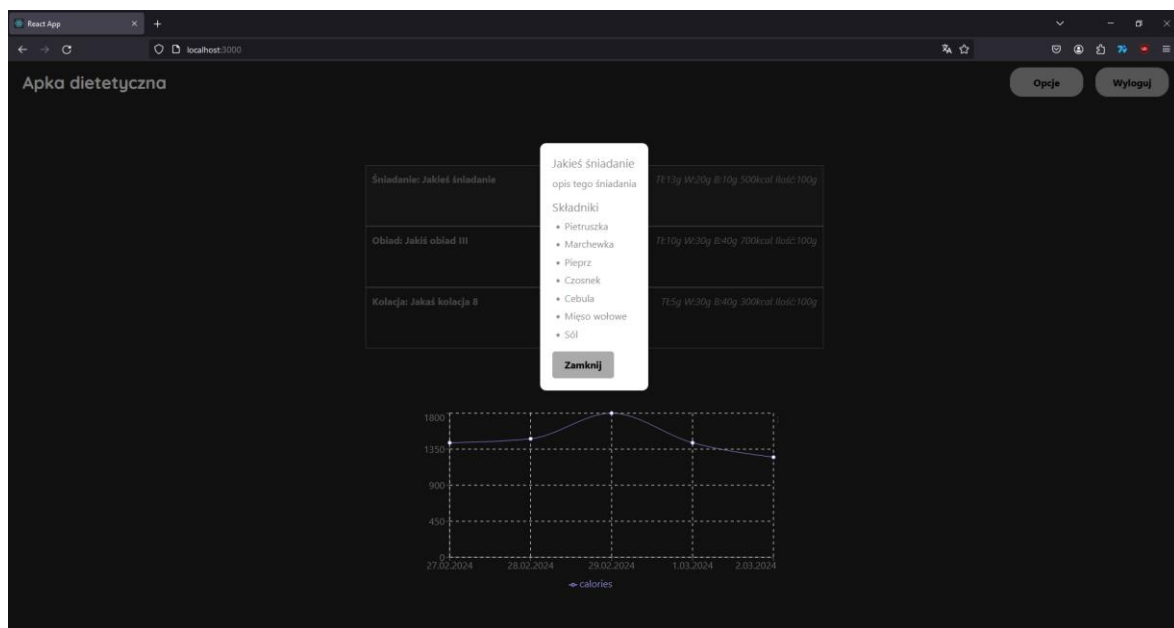


Rys. 6.6. Widok główny

Za wyświetlanie dań odpowiada oddzielny komponent, DishDisplay.js (listing 6.13.). Dane są przekazywane do niego, a następnie wyświetlane. W przypadku naciśnięcia na danie, zostaną wyświetlone szczegółowe dane odnośnie danego dania (rys. 6.7.).

Listing. 6.13. Fragment kodu komponentu wyświetlania dania

```
return (
  <div
    className={styles.dishes}
    style={{ borderTop: "1px solid rgb(107, 107, 107)", fontSize: fontSize }}
    onClick={toggleModal}
  >
    <div className={styles.dishInfo}>
      <div className={styles.dishName}>
        {t(meal_time)} {dish_name}
      </div>
      <div className={styles.macronutrientData}>
        {macronutrientData && (
          <>
            {t("Fats")}{fats}g {t("Carbohydrates")}{carbohydrates}g
            {t("Proteins")}{proteins}g {calories}kcal {t("Amount")}{amount}g
          </>
        )}
      </div>
    </div>
    {showModal && (
      <div className={styles.modalBackdrop}>
        <div className={styles.modal}>
          <h2 style={{fontSize: fontSize*1.2}}>{dish_name}</h2>
          <p>{description}</p>
          <h3 style={{fontSize: fontSize*1.2}}>{t("Ingredients")}</h3>
          <ul>
            {ingredients.map((ingredient, index) => (
              <li key={index}>{ingredient.name}</li>
            ))}
          </ul>
          <button onClick={toggleModal}>{t("Close")}</button>
        </div>
      </div>
    )}
  </div>
);
```



Rys. 6.7. Szczegóły danego dania

. Wykres jest tworzony na podstawie sumy kalorii dań z danego dnia, następnie te sumy są rozpisane według daty. Po najechaniu na wykres podświetla się pozycja i wyświetlają się informacje, w tym przypadku liczba kalorii z całego oraz data.

Listing 6.14. Fragment kodu komponentu rysowania wykresu

```
import React from "react";
import { useTranslation } from "react-i18next";
import { LineChart, Line, XAxis, YAxis, CartesianGrid, Tooltip, Legend } from 'recharts';
import "../styles/modules.css";

const Charts = ({ dates, calories }) => {
  const { t, i18n } = useTranslation();

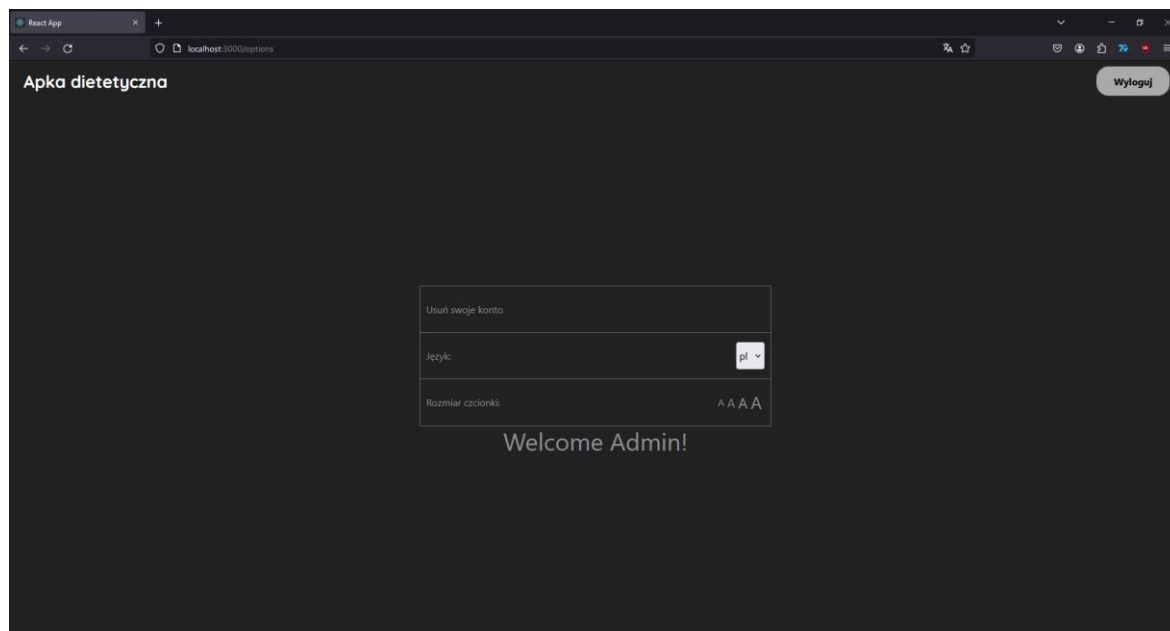
  if (!dates || !calories || dates.length === 0 || calories.length === 0) {
    return <p>No data available for chart</p>;
  }

  const chartData = dates.map((date, index) => ({ date, calories: calories[index] }));

  return (
    <div className="chart-container">
      <LineChart width={600} height={300} data={chartData}>
        <XAxis dataKey="date" />
        <YAxis />
        <CartesianGrid stroke="#eee" strokeDasharray="5 5" />
        <Tooltip contentStyle={{ backgroundColor: "#f5f5f5", border: "1px solid #d5d5d5" }} />
        <Legend />
        <Line type="monotone" dataKey="calories" stroke="#8884d8" />
      </LineChart>
    </div>
  );
};

export default Charts;
```

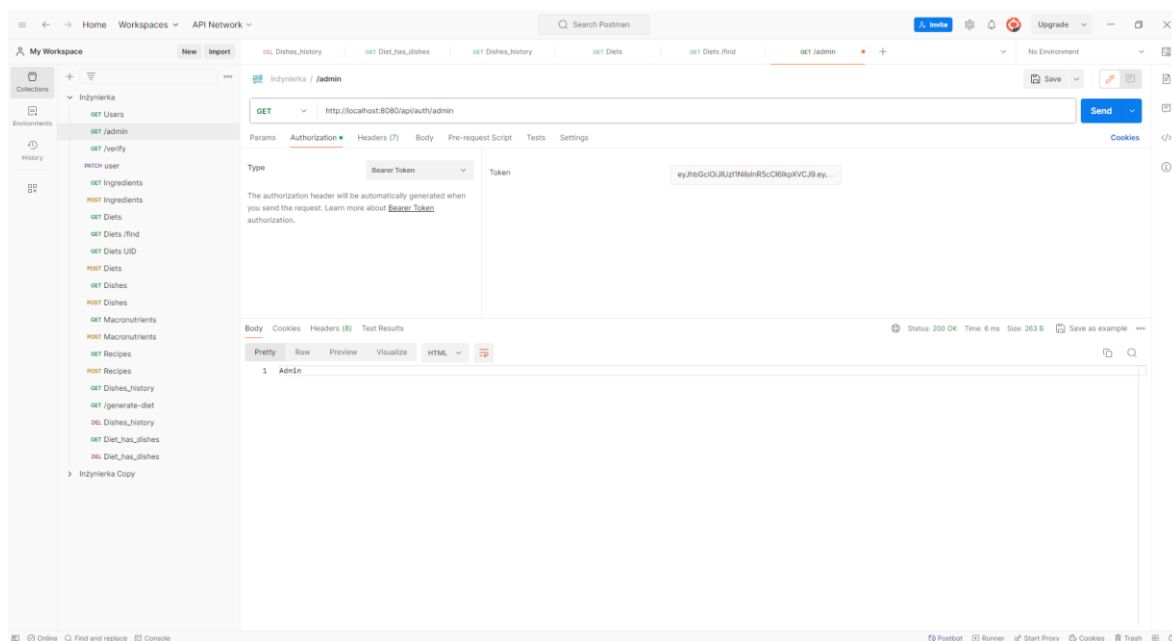
Na samym końcu pozostaje podstrona z opcjami (rys. 6.8.), w której użytkownik ma możliwość zmiany rozmiaru czcionki, języka lub usunięcia swojego konta.



Rys. 6.8. Widok opcji z konta administratora

7. Testy

Całość aplikacji została przetestowana przy pomocy Postmana (rys. 7.1.) oraz testów manualnych. Większość kodu to API, w związku z tym narzędzie takie jak Postman jest nieodłącznym elementem przez cały okres rozwoju kodu źródłowego aplikacji.



Rys. 7.1. Weryfikacja tokenu administratora

Jest zaawansowanym narzędziem, pozwala na stworzenie kolekcji, w której grupowane są zapytania. Same zapytania mają wiele opcji, oprócz najbardziej podstawowych, typu wysłanie zapytań z parametrami bądź JSONem w body zapytania, jest również możliwość wysłania Bearer Tokena w celu autentykacji administratora.

8. Podsumowanie

Praca nad projektem i implementacją aplikacji pozwoliła zdobyć autorowi ogromną wiedzę na tematy związane ze specjalnością jego kierunku, jakim jest Inżynieria Oprogramowania. Ze względu na to, iż jest to praca samodzielna, zgłębił wiedzę na temat wszystkich części wytwarzania oprogramowania, tj. front-end, back-end oraz bazy danych. Z faktu, że React.js oraz Express, który jest frameworkiem do Node.js, korzystają z języka JavaScript wynika, że autor będzie mógł szukać pracy w zawodzie czy to jako back-end czy front-end developer.

Wcześniej wspomniane algorytmy haszujące oraz informacje na temat ich bezpieczeństwa, informacje na temat solenia hasła czy użycia pieprzu, umiejętność zaprojektowania i znormalizowania bazy, pisanie własnego API to wiedza i umiejętności, które autor nabył, odświeżył, bądź zgłębił ponad to co wiedział.

Z powodu braków na etapie projektowania oraz na wczesnym etapie implementacji, praca w wersji końcowej jest dosyć skromna, aczkolwiek przejście przez cały proces jej tworzenia, łącznie z elementami, po których już nie ma śladu w tej wersji, był bardzo dobrą metodą nauczania się wielu rzeczy.

Z racji tego, że autor pracy chciałby w dalszym ciągu szlifować swoje umiejętności oraz naukę z zakresu tworzenia oprogramowania, zwłaszcza części związanej z back-endem oraz bezpieczeństwem, prace nad projektem będą kontynuowane.

Bibliografia:

- [1] DistantJob, <https://distantjob.com/blog/javascript-frameworks>, zasoby z dnia 01.11.2023
- [2] EarthWeb, <https://earthweb.com/how-many-people-use-dark-mode/>, zasoby z dnia 27.02.2024
- [3] Eurostat, Overweight and obesity BMI statistics, https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Overweight_and_obesity_-_BMI_statistics, zasoby z dnia 01.11.2023
- [4] Eurostat, <https://ec.europa.eu/eurostat/web/products-eurostat-news/-/EDN-20171113-1>, zasoby z dnia 01.11.2023
- [5] GeeksForGeeks, <https://www.geeksforgeeks.org/most-popular-databases/>, zasoby z dnia 01.11.2023
- [6] Gormley JJ, Juturu V (2010). "Partially Hydrogenated Fats in the US Diet and Their Role in Disease". In De Meester F, Zibadi S, Watson RR (eds.). Modern Dietary Fat Intakes in Disease Promotion. Nutrition and Health. Totowa, NJ: Humana Press. pp. 85–94. <https://doi.org/10.1111/dom.13164>
- [7] Jeannie T., Campbell H. T., etc., Effects of an energy-restricted low-carbohydrate, high unsaturated fat/low saturated fat diet versus a high-carbohydrate, low-fat diet in type 2 diabetes: A 2-year randomized clinical trial, Diabetes, Obesity and Metabolism, Volume 20, Wydanie 4, strony 858-871, <https://doi.org/10.1111/dom.13164>
- [8] Lichtenstein H. Alice, Van Horn L., Very Low Fat Diets, Circulation, September 1, 1998 Vol 98, Issue 9, <https://doi.org/10.1161/01.CIR.98.9.935>
- [9] Lucidchart, <https://www.lucidchart.com/pages/pl/czym-jest-uml-unified-modeling-language>, zasoby z dnia 30.12.2023
- [10] Medium, <https://medium.com/nerd-for-tech/sha-2-and-bcrypt-encryption-algorithms-e0c0599b0da>, zasoby z dnia 27.02.2024
- [11] Prisma, <https://www.prisma.io/dataguide/database-tools/top-nodejs-orms-query-builders-and-database-libraries#sequelize>, zasoby z dnia 01.11.2023
- [12] React: The library for web and native user interfaces, <https://react.dev>, zasoby z dnia 26.10.2023