AKADEMIA TARNOWSKA
Wydział Politechniczny

Kierunek: Informatyka

2024/2025

Bartosz Lasko

PRACA INŻYNIERSKA

Aplikacja prozdrowotna

Promotor pracy:

mgr inż. Tomasz Gądek

Tarnów, *2025*

Spis treści

1. Wstęp

We współczesnym świecie społeczeństwo, coraz bardziej jest świadome, jak bardzo spożywane jedzenie ma wpływ na pracę całego organizmu. Obecnie ludzie przez dostęp do mediów, wiedzą coraz więcej na tematy związane ze zdrowiem, a przez to szukają ułatwień w postaci technologii.

1.1 Motywacja

Obecnie w społeczeństwie pojawiają się trendy, których głównym hasłem jest dbanie o zdrowie poprzez odpowiednie nawyki żywieniowe. Ludzie niestety mają problem z regularnością w swoich postanowieniach przez brak wystarczającej ilości czasu.

Dostęp do technologii znacząco przyspiesza pewne procesy związane ze stylem życia, bo pomaga w łatwy i szybki sposób kontrolować pewne nawyki. Dla większości ludzi, wszechobecny dostęp do internetu otwiera wiele możliwości, takich jak:

- zapisywanie i odczyt ogromnych ilości danych z internetu,
- możliwość komunikowania się przez internet z innymi ludźmi pomimo ogromnych odległości,
- zadowalająca szybkość łączy internetowych,
- stosunkowo niski koszt korzystania z internetu.

Stworzenie aplikacji pozwoliłoby wykorzystać te możliwości przez szeroką grupę użytkowników.

1.2 Cel pracy

Celem pracy dyplomowej jest zaprojektowanie i implementacja aplikacji internetowej, pozwalającej bardziej świadomie dbać o zdrowie przez kontrolowanie składu spożywanych posiłków. Dodatkowo użytkownik ma możliwość kontrolować masę ciała i nawodnienie organizmu. Udostępnione zostanie forum społecznościowe, na którym użytkownicy mogą między sobą wymieniać się przepisami i wiadomościami.

1.3 Zakres pracy

System aplikacji internetowej został zaimplementowany w oparciu o trzy moduły:

- aplikacja serwerowa,
- aplikacja kliencka,
- baza danych.

2. Analiza biznesowa

W tym rozdziale zostały wymienione istotne funkcjonalności dla aplikacji. Także zostaną przedstawione istotne cechy programu, który będzie zawierał w sobie przyjęte standardy projektowania aplikacji webowych.

2.1 Wymagania funkcjonalne

Najważniejszym zadaniem aplikacji będzie ułatwienie użytkownikowi w skuteczny sposób zadbać o zdrowie w oparciu o zdrowy styl życia: odpowiednie odżywianie, nawadnianie organizmu oraz zapewnienie wystarczającej ilości ruchu.

Dla użytkownika będzie istotne, aby aplikacja pozwalała na:

- obliczanie zapotrzebowania kalorycznego według aktualnych norm, opierając się o dane użytkownika: wzrost, waga, aktywność fizyczna;
- wyszukiwanie produktów z bazy danych aplikacji, wraz z informacją o kaloryczności i makroskładnikach;
- monitorowanie postępów poprzez generowanie codziennych wizualizacji zawierających informację o spożytych makroskładnikach wraz z udziałem kalorii;
- monitorowanie nawodnienia organizmu;
- przypominanie o ruchu;
- kontrolę masy ciała;
- korzystanie z forum, gdzie użytkownik będzie mógł dzielić się osiągnięciami oraz przepisami.

Aplikacja będzie miała możliwość na utworzenie konta administratora. Funkcjonalności dostępne dla administratora:

- Nakładanie kar dla użytkownika, w postaci ograniczenia dla niego funkcjonalności na forum.
- Blokowanie konta użytkownika.
- Usuwanie konta.
- Manipulowanie bazą danych z produktami spożywczymi.

2.2 Wymagania niefunkcjonalne

2.2.1 Responsywność

W obecnych czasach, tworzenie aplikacji webowych wymaga odpowiedniego dopasowania rozmiaru treści do okna przeglądarki użytkownika.
Stało się to standardem, ponieważ obecnie korzysta się z szerokiej gamy ekranów o różnych rozmiarach. W raporcie z końca 2023 roku wynika, że w 60% ludzie korzystają z urządzeń mobilnych, a tylko w 40% z urządzeń desktopowych.

Aplikację zaprojektowano tak, aby mogła w czasie rzeczywistym dopasować wyświetlanie treści do zmiany wielkości okna przeglądarki.
Rozmiary elementów do wyświetlenia przez przeglądarkę będą zdefiniowane w procentach względem wymiarów okna. Rozwiązanie to umożliwia zachowanie spójności
w wyświetlaniu elementów treści, w sposób możliwie jak najbardziej intuicyjny
dla użytkownika, zachowując przy tym estetykę.

2.2.2 Bezpieczeństwo

Zabezpieczanie aplikacji jest priorytetem, gdyż w obecnych czasach dane są niezwykle cenne, a użytkownik powinien czuć się bezpiecznie. Inaczej nie będzie chętny do korzystania z aplikacji.

Zastosowano w aplikacji najnowszy standard autoryzacji (*OAuth 2.0*), wykorzystujący system tokenów JWT (ang. *JSON Web Token*). Zapewnia to bezpieczny dostęp do sesji użytkownika, chroniąc określone zasoby strony w przeglądarce.

2.2.3 Intuicyjność interfejsu użytkownika

Interfejs powinien być zrozumiały dla użytkownika. Kolorystyka, rozmiary elementów i animacje - powinny być spójne i dopasowane tematycznie.
Wszystkie wymienione aspekty mają bardzo istotny wpływ na korzystanie z aplikacji przez potencjalnie zainteresowanych użytkowników.

3. Opis wykorzystanych technologii

3.1 Java

Język został zaprojektowany w 1991 r. przez firmę Sun Microsystems pod kierownictwem Jamesa Goslinga. Pierwotnie został stworzony dla ówczesnej telewizji interaktywnej. Jednak okazał się zbyt zaawansowany. W 1996 r. wydano pierwszą implementację języka w wersji publicznej jako Java 1.0. Język ten stale jest rozwijany i obecnie doczekał się dwudziestu trzech wersji. [1]

Twórcy Javy zrealizowali cel - opracowany język powinien działać na wszystkich systemach operacyjnych niezależnie od technologii sprzętowej oraz konfiguracji. Udało się to dzięki wykorzystaniu JVM (ang. *Java Virtual Machine*).

Jest to wirtualna maszyna, do której dostarcza się kod bajtowy po skompilowaniu programu napisanego w Javie.

Najistotniejsze cechy języka:

- silne typowanie,
- obiektowość,
- przenośność,
- wydajność,
- bezpieczeństwo,
- wielowątkowość,
- zarządzanie pamięcią.

3.2 Maven

Jest to narzędzie ułatwiające zarządzanie projektami, które tworzone są w Javie. Ułatwia dodawanie do projektu zależności (biblioteki lub frameworka).
Wszystkie ustawienia do projektu znajdują się w pliku o nazwie pom.xml (ang. *Project Object Model*). Plik ten zawiera: nazwę projektu, użyte pluginy, zależności oraz informację jaki będzie typ zbudowanego projektu (*JAR, WAR*). [10]

Maven wykorzystuje zasadę konwencja nad konfiguracją (ang. *convention over configuration*). Przez to większość domyślnych ustawień jest wystarczająca. Proces kompilacji oraz budowania projektu jest znacznie ułatwiony, jak również przeprowadzanie testów jednostkowych i integracyjnych.
Narzędzie to pozwala na generowanie raportów z testów.

3.3 Spring i Spring Boot

W 2002 r. pierwszą wersję Springa napisał Rod Johnson, a w 2003 r. została wydana na licencji Apache w wersji 2.0. Spring jest frameworkiem Javy, a Spring Boot jest rozszerzeniem tego frameworka.

Spring bazuje na funkcjach, takich jak:

- wstrzykiwanie zależności (ang. *Dependency Injection*),
- odwrócenie kontroli (ang. *Inversion of Control*),
- ekosystem modułów.

Wstrzykiwanie zależności - obiekty zarządzane przez kontekst Springa (ang. *Beans*) oraz komponenty mają zadeklarowane zależności, a framework je wstrzykuje podczas wykonywania programu.

Odwrócenie kontroli - framework odpowiedzialny jest nad przebiegiem procesu tworzenia komponentów i obiektów z kontekstu Springa oraz zarządzanie nimi uwzględniając zachodzące relacje między tymi obiektami w kontrolowanym cyklu życia.

Ekosystem modułów - całość ekosystemu składa się z wielu unikatowych modułów posiadających dedykowane dla siebie funkcje. Odpowiednio można wybierać dokładnie, z których modułów chce się korzystać bez ingerencji pozostałych części ekosystemu. Wspomaga to wydajność.

Wprowadzono dla programisty model pisania kodu polegający na używaniu adnotacji (ang. *Annotation*). To metadane umieszczane w kodzie nad klasą Javy lub nad jej metodami. Dostarczają dodatkowe informacje dla kompilatora i środowiska wykonawczego.
Mają za zadanie pomóc w konfiguracji i zarządzaniu zachowaniem aplikacji bez potrzeby pisania rozbudowanego kodu. [2]

Tabela 3.1 Przykłady adnotacji Springa [opracowanie własne]

Adnotacja	**Opis**
@Component	Klasa staje się komponentem Springa i będzie przez niego zarządzana
@GetMapping	Umożliwia metodzie odpowiadać na żądanie HTTP typu GET
@PostMapping	Umożliwia metodzie odpowiadać na żądanie HTTP typu POST
@DeleteMapping	Umożliwia metodzie odpowiadać na żądanie HTTP typu DELETE
@Controller	Klasa otrzymuje zdolność do obsługi zdarzeń HTTP
@RestController	Połączenie adnotacji @Controller i @ResponseBody, czyli tworzony jest komponent REST dla klasy zwracający JSON lub XML.

@Service	Informuje, że w klasie znajdzie się logika biznesowa
@Repository	Oznacza klasę jako repozytorium z dostępem do bazy danych

Najważniejsze cechy Spring Boot:

- automatyczna konfiguracja,
- szybkie uruchamianie aplikacji (wbudowany serwer),
- minimalizacja plików konfiguracyjnych,
- łatwiejsze zarządzanie zależnościami,
- dostęp do Spring Boot Starter i Actuator.

3.4 MySQL

Został wydany w 1995 r. przez szwedzką firmę MySQL AB. Od 2010 r. należy do firmy Oracle. Jest to system zarządzania relacyjnymi bazami danych o otwartym dostępie. Cieszy się wysoką popularnością.

Relacyjne bazy danych to takie, gdzie dane przechowywane są w tabelach oraz posiadają klucz główny. Tabele mogą mieć również klucze obce, które wiążą tabele między sobą. Dzięki takiej konstrukcji dane są zorganizowane w logiczny i przejrzysty sposób. [5]

MySQL jest zoptymalizowany, wydajny oraz skalowalny. Wykorzystuje się go przede wszystkim tam, gdzie dużo klientów korzysta z aplikacji. Większość stron internetowych oraz systemów e-commerce korzysta z tego właśnie rozwiązania. Ma zastosowanie w różnych platformach programistycznych, zapewniając przy tym w bezpieczny sposób przechowywać dane poprzez mechanizm kontroli dostępu i autoryzacji z odpowiednim szyfrowaniem.

3.5 JavaScript

To język skryptowy, stworzony z myślą o aplikacjach frontendowych. Przeznaczony jest do programowania interaktywnych stron internetowych. Nie potrzebuje kompilatora, gdyż jest językiem interpretowanym. Dzięki temu sprawniej pracuje się, gdyż rezultat zaimplementowanej strony internetowej widać od razu po jej odświeżeniu.

Brak typowania - nie trzeba deklarować typów zmiennych, co przyspiesza pisanie kodu, ale niestety ryzyko popełnienia błędów jest większe. Wspierany jest przez wszystkie nowoczesne przeglądarki. Pozwala na obsługę zdarzeń i wspiera asynchroniczność, która jest niezbędna, gdy operacje w programie muszą oczekiwać na siebie. W wygodny sposób można także tworzyć animację obiektów, które mogą być wyświetlone przez przeglądarkę. [4]

3.6 TypeScript

Język TypeScript jest nadzbiorem JavaScript. Wprowadza statyczne typowanie i przez to pozwala bardziej kontrolować pracę ze zmiennymi. Łatwiej jest zdiagnozować błędy, a projekty stają się bardziej przejrzyste. Typescript jest transpilowany do JavaScript, a to przyczynia się do możliwości pracy z tym językiem, tam gdzie mógłby być użyty JavaScript. Ma lepsze wsparcie dla Visual Studio Code, które jest zintegrowanym środowiskiem programistycznym - IDE (ang. *Integrated Development Environment*). Środowisko to służy do tworzenia, modyfikowania, testowania i konserwacji oprogramowania. Ułatwia to pracę programisty poprzez: autouzupełnianie, podpowiedzi i refaktoryzację kodu. [4]

3.7 React

To popularna biblioteka JavaScript używana do budowania interfejsów użytkownika (*UI*). Powstała za sprawą programisty Facebooka - Jordana Walke. Obecnie jest wspierana przez Meta oraz społeczność open source.

React wprowadza pojęcie komponentów będących modułami wielokrotnego użytku. Biblioteka korzysta z wirtualnego DOM (ang. *Document Object Model*), który zwiększa wydajność. Przy każdym zapisie następuje odświeżenie serwera i aktualizowane są tylko te komponenty, które uległy zmianie. Wprowadzono składnie JSX (ang. *JavaScript XML*) oraz TSX (ang. *Typescript XML*), dlatego można pisać w składni przypominającej HTML. Wprowadzono także mechanizm stanowy - *State* , który pozwala przechowywać w pamięci po stronie klienta dane w sposób dynamiczny. Dodano również właściwość - *Properties*. Umożliwia przekazywanie danych do komponentów w sposób dynamiczny.
Korzystać można także z *Hooków*, czyli z kontroli nad stanami przez możliwość zarządzania ich cyklem życia. [3]

3.8 REST API

To architektura umożliwiająca na komunikację między aplikacjami przy udziale protokołu HTTP. W praktyce pozwala na łatwą komunikację między klientem, a serwerem w aplikacjach webowych. Zapytania do serwera i powrotne odpowiedzi do klienta nie przechowują informacji o poprzednim stanie. Pozwala to tworzyć skalowalne aplikacje webowe, gdyż zapytania są od siebie niezależne. [6] Żądania HTTP (ang. *Hypertext Transfer Protocol*) przesyłane są w modelu klient-serwer przy udziale protokołu TCP/IP.

Składają się one z trzech części:

- metody określającej rodzaj żądania,
- nagłówka zawierającego informacje o żądaniu klienta,
- adresu URL (ang. *Uniform Resource Locator*) będącym informacją o lokalizacji przesyłanego zasobu.

Tabela 3.2 Metody żądania HTTP [opracowanie własne]

Metoda	**Manipulacja zasobem**
GET	pobranie
POST	dodanie
PUT	aktualizacja pełnego obiektu danych
PATCH	aktualizacja fragmentu danych
DELETE	usunięcie

W odpowiedzi na żądanie HTTP otrzymujemy trzycyfrowy kod informujący o przebiegu żądania. Jeśli jest wewnętrzny błąd po stronie serwera to kod jest w postaci 5XX, a jeśli problem znajduje się po stronie klienta to kod ma postać 4XX.
Poza tym są jeszcze kody typu: 3XX - przekierowanie, 2XX - przetworzono poprawnie, 1XX - serwer zaakceptował żądanie.

3.9 Protokół TCP/IP

TCP (ang *Transmission Control Protocol*) ustanawia połączenie transmitujące dane, rozdziela dane na segmenty i numeruje je, a następnie wysyła. Jeśli dane nie zostały poprawnie odebrane, to jeszcze raz próbuje wysłać, aby zapewnić kompletność przepływu danych.

IP (ang. *Internet Protocol*) ma za zadanie odpowiednio zaadresować pakiety danych między urządzeniami w sieci. Urządzenia mają unikalny adres, który pozwala zlokalizować je w łatwy sposób. [7]

3.10 OAuth 2.0.

To standard autoryzacji pozwalający aplikacjom na uzyskanie dla użytkownika ograniczonego dostępu do zasobów na serwerze HTTP. Serwer na określony czas wydaje token dostępu do zasobów. Tokeny mogą być wykorzystywane zamiast danych użytkownika do autoryzacji żądań. [8]

Proces autoryzacji:

1. Użytkownik przez aplikację kierowany jest do serwera autoryzacyjnego.
2. Następuje logowanie się użytkownika ze zgodą na dostęp do zasobów.
3. Serwer autoryzacyjny wysyła kod autoryzacyjny do aplikacji.
4. Przesyłany jest przez aplikację kod do serwera autoryzacyjnego.
5. Zwracany jest token dostępu przez serwer autoryzacyjny.

Po procesie autoryzacji token używany jest do autoryzacji żądań kierowanych do zasobów serwera.

Ten standard jest bardzo często wykorzystywany w aplikacjach webowych oraz mobilnych. Jest bardzo bezpieczny i zachowuje zgodność z zasadami ochrony prywatności.

4. Implementacja systemu

4.1 Serwer aplikacji

Serwer ma za zadanie zarządzać danymi zawartymi w bazie danych oraz prowadzić optymalną komunikację z klientem (interfejs użytkownika).
Przyjmuje i przetwarza żądania od klienta, a także gwarantuje bezpieczeństwo dla zasobów.

Poniższe funkcjonalności są zarządzane i przetwarzane przez serwer po tym jak użytkownik poprzez interfejs wyśle żądanie, które może zawierać dane.

Funkcjonalności dla zwykłego użytkownika (zalogowany):

1. Wprowadzanie danych (wzrost, waga, płeć, aktywność fizyczna) pozwalających na ułożenie systemu monitorującego procesem zmiany masy ciała: redukcja masy ciała, wzrost masy ciała i stabilizacja masy ciała.
2. Umożliwienie wybierania z listy dostępnych produktów spożywczych przez podanie ich ilości (w gramach, w sztukach) z podziałem na pory posiłków (śniadanie, lunch, obiad, kolacja).
3. Wgląd na tabele wybranych produktów przez użytkownika, które przedstawiają makroskładniki produktów (w gramach) oraz ich kaloryczność (w kcal).
   Dodatkowo wyświetlane są tabele, które zawierają zsumowane ilości makroskładników oraz kalorie dla poszczególnych posiłków z podziałem na ich pory. Sumy te wyświetlane też są dla całego dnia. Generują się także wykresy kołowe dla powyższych sum, które zawierają proporcje udziału: tłuszczów, węglowodanów i białka.
4. Zapisywanie posiłków dla wybranego dnia z ustalonego przedziału dni. Następnie generowany jest wykres, który przedstawia udział makroskładników lub kalorii dla danego przedziału dni.
5. Wprowadzenie swojej masy ciała (w kilogramach) dla wybranego dnia. Tworzenie wykresu, który ilustruje jak zmienia się masa ciała w określonym przedziale dni.
6. Przypominanie użytkownikowi co określony czas o potrzebie nawodnienia organizmu.

7. Możliwość korzystania z portalu społecznościowego, który pozwala na wysyłanie wiadomości do innych użytkowników oraz na dzieleniu się przepisami kulinarnymi.

Funkcjonalności dla administratora (zalogowany):

1. Możliwość przeglądania treści wysyłanych przez użytkowników na portalu społecznościowym.
2. Uprawnienie do usuwania, bądź zawieszania kont użytkowników na pewien czas.
3. Uprawnienie do dodawania lub usuwania produktów spożywczych z bazy danych.

Funkcjonalności dla zwykłego użytkownika oraz administratora (niezalogowany):

1. Rejestracja konta poprzez podanie imienia, loginu i hasła (dane są zapisywane do bazy danych wraz z ich szyfrowaniem).
2. Logowanie się do aplikacji po wcześniejszym zarejestrowaniu konta (tworzony jest token JWT, który umożliwia autoryzowany dostęp do zasobów aplikacji).

4.2 Wybrane elementy implementacyjne wraz z ich analizą

4.2.1 Tworzenie nowego użytkownika

Nowy użytkownik tworzony jest poprzez interfejs rejestracyjny dla nowego konta. Konieczne jest podanie: imienia, loginu i hasła. Login oraz hasło muszą być unikalne, zgodne z zaleceniami. Po pomyślnej rejestracji, użytkownik otrzymuje swój własny spersonalizowany obszar aplikacji.

```
@PostMapping("/register")
public ResponseEntity<?> postRegisterData(@RequestBody User user) {
    try {
        if (userRepository.existsByLogin(user.getLogin())||
                userRepository.existsByPassword(user.getPassword())) {
            return ResponseEntity.status(HttpStatus.CONFLICT)
                    .body("The user with the given login or password already exists");

        }
        else{
            log.info(user.getLogin());
            user.setPassword(passwordEncoder.encode(user.getPassword()));
            userService.saveUser(user);
            return ResponseEntity.ok( body: "Successfully registered");
        }
    } catch (Exception e) {
        log.error("Invalid JSON data: {}", e.getMessage());
        return ResponseEntity.status(HttpStatus.BAD_REQUEST)
                    .body("Invalid JSON data");
    }
}
```

Rys. 4.1 Rejestracja użytkownika [opracowanie własne]

Metoda *postRegisterData* przechwytuje żądanie od klienta przy rejestracji. Jeśli nie ma duplikatów loginu lub hasła w bazie danych to tworzone jest konto dla użytkownika wraz z haszowaniem hasła. Serwer zwraca do klienta odpowiedni komunikat dotyczący pomyślności zakończenia procesu rejestracji.

4.2.2 Logowanie użytkownika wraz z tworzeniem tokenu JWT

Mechanizm logowania weryfikuje, czy użytkownik o podanym loginie i haśle znajduje się w systemie, jeśli tak to tworzony jest token JWT. Pomyślne zakończenie procesu umożliwia przejście do obszaru aplikacji dla użytkownika.

```
private static final String SECRET_KEY  1 usage
        = "Yf3Epj1cK3N6f8QkP6U4h1L9g5Z7s3D2W6C9j8H5f2V7m4B1G6H9e2J5k8M1N4R7";

private static final Key KEY = Keys.hmacShaKeyFor(SECRET_KEY  1 usage
        .getBytes(StandardCharsets.UTF_8));

Rename usages
@PostMapping("/loginUser")
public ResponseEntity<?> loginUser(@RequestBody User user) {


    User existingUser = userRepository.findByLogin(user.getLogin());

    if (existingUser != null && existingUser.getPassword().equals(user.getPassword())) {
        long currentTimeMillis = System.currentTimeMillis();

        String token = Jwts.builder()
                .setSubject(existingUser.getLogin())
                .claim( name: "userId", existingUser.getUserId())
                .claim( name: "roles",  value: "user")
                .setIssuedAt(new Date(currentTimeMillis))
                .setExpiration(new Date(currentTimeMillis + 1000 * 60 * 60))
                .signWith(KEY, SignatureAlgorithm.HS512)
                .compact();
        log.info(token);
        return ResponseEntity.ok(new AuthResponse(token));
    } else {
        return ResponseEntity.status(401).body("Invalid login or password");
    }
}
```

Rys. 4.2 Logowanie dla zwykłego użytkownika [opracowanie własne]

Metoda *loginUser* odbiera żądanie przy logowaniu zwykłego użytkownika.
Sprawdzane jest, czy podany użytkownik znajduje się w bazie danych.
Jeśli tak to tworzony jest token JWT zawierający login użytkownika, rolę użytkownika, ID użytkownika, czas powstania tokenu oraz czas zakończenia jego żywotności.

Zwracany jest z serwera do klienta powyższy właśnie token, a jeśli były podane błędne dane do logowania to odpowiednio powiadamiany jest o tym klient.

4.2.3 Zabezpieczenie autoryzacyjne po stronie interfejsu użytkownika

Po pomyślnym zalogowaniu, utworzony przez serwer token JWT odbierany jest przez klienta (interfejs użytkownika). Następnie zapisywany jest w pamięci przeglądarki *localStorage*, a dane są zatrzymane do momentu ręcznego usunięcia ich z aplikacji. Wykorzystano do tego procesu filtr, który przenosi użytkownika do określonych ścieżek dostępu (obszar użytkownika) tylko wtedy, gdy znajduje się token w pamięci klienta. Wyjątkiem jest strona logowania i rejestracji. Przy wylogowaniu się, token jest usuwany z *localStorage* i użytkownik zostaje skierowany do ekranu logowania.

```
interface PrivateRouteProps {
  children: ReactNode;
}

const PrivateRoute: React.FC<PrivateRouteProps> = ({ children }) => {
  const token = localStorage.getItem('token');
  if (!token) {
    return <Navigate to="/logIn" />;
  }
  return <>{children}</>;
};

export default PrivateRoute;
```

Rys. 4.3 Filtr autoryzujący [opracowanie własne]

```
<Route path="/register" element={<Register />} />
<Route path="/logIn" element={<Login />} />
<Route path="/dietDayHome"
element={<PrivateRoute><DietDayHome /></PrivateRoute>} />
<Route path="/searchIngredient/breakfast"
element={<PrivateRoute><SearchIngredientPage /></PrivateRoute>} />
```

Rys. 4.4 Przykład ustawienia filtra na ścieżkach dostępu [opracowanie własne]

4.2.4 Dodawanie wszystkich składników z obiadu dla danego dnia

```
@PostMapping("/postDayOfEatingForDinnerIngredients/{day}")
public ResponseEntity<?> postDayOfEatingForDinnerIngredients(@PathVariable Integer day) {
    try {
        List<DateOfEating> existingDaysWithDinner = dayOfEatingRepository
                .findByDayAndDinnerForDateIdNotNull(day);
        List<Dinner> dinnerList = dinnerRepository.findAll();
        List<DinnerForDate> dinnerForDateList = dinnerList.stream().map(dinner -> {
            DinnerForDate dinnerForDate = new DinnerForDate();
            dinnerForDate.setMealIngredientId(dinner.getMealIngredientId());
            dinnerForDate.setMealQuantityOfGrams(dinner.getMealQuantityOfGrams());
            dinnerForDate.setUserId(dinner.getUserId());
            return dinnerForDate;
        }).collect(Collectors.toList());
        dinnerForDateRepository.saveAll(dinnerForDateList);
        List<DinnerForDate> allDinerForDate = dinnerForDateRepository.findAll();
        List<Integer> ids=new ArrayList<>();
        for(int i=allDinerForDate.size() - dinnerForDateList.size();i<allDinerForDate.size(); i++){
            ids.add(allDinerForDate.get(i).getDinnerForDateId());
        }
        if (!existingDaysWithDinner.isEmpty()) {
            dayOfEatingRepository.deleteAll(existingDaysWithDinner);
        }

        if (!dinnerList.isEmpty()) {
            for (Integer id : ids) {
                dayOfEatingService
                        .saveDayOfEatingForIngredients(day, month: 2, year: 2024,
                                breakfastForDateId: null, lunchForDateId: null, id, snacksForDateId: null);
            }
        }
        return ResponseEntity.ok( body: "Entries for dinner have been updated for the day " + day + ".");
    } catch (Exception e) {
        log.error("\n" + "An error occurred: {}", e.getMessage());
        return ResponseEntity.status(HttpStatus.BAD_REQUEST)
                .body("Dinner entries for the day could not be updated " + day + ".");
    }
}
```

Rys. 4.5 Dodawanie wszystkich składników z obiadu dla danego dnia [opracowanie własne]

Na rysunku 4.5 przedstawiona jest metoda *postDayEatingForDinnerIngredients*, która zarządza zapisywaniem składników obiadu do danego dnia. Metoda sprawdza, czy dany dzień ma już przypisane składniki. Jeśli tak, to usunięte zostają wszystkie składniki z tego dnia.

Proces zapisywania składników:

1. Składniki z aktualnego widoku dnia dla obiadu, zapisywane są do encji przechowującej na stałe składniki. Dane z aktualnego widoku dnia zostają usunięte na stałe, po tym jak użytkownik zatwierdzi usunięcie.
2. Następnie z tej encji składniki przypisywane są po identyfikatorze do wybranego dnia.

Ma to za zadanie oddzielić warstwę symulacji związaną z zapisem na wybrany dzień od warstwy dla aktualnego dnia.

4.2.5 Wysyłanie i odbieranie wiadomości na forum społecznościowym

```
@ResponseBody
@GetMapping("/getIncomingMessages/{toUserId}")
public ResponseEntity<List<Message>> getIncomingMessages(@PathVariable Integer toUserId) {
    try {
        return ResponseEntity.ok(messageRepository.findAllByToUserId(toUserId));
    } catch (Exception e) {
        log.error("Error fetching users list", e);
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    }
}

@ResponseBody
@GetMapping("/getSentMessages/{fromUserId}")
public ResponseEntity<List<Message>> getSentMessages(@PathVariable Integer fromUserId) {
    try {
        return ResponseEntity.ok(messageRepository.findAllByFromUserId(fromUserId));
    } catch (Exception e) {
        log.error("Error fetching users list", e);
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    }
}

@PostMapping("/postMessage")
public ResponseEntity<?> postMessage(@RequestBody Message message) {
    try {
        messageService.saveMessage(message.getFromUserId(), message.getToUserId(), message.getText());
        return ResponseEntity.ok( body: "Successfully sended message");
    } catch (Exception e) {
        log.error("Invalid JSON data: {}", e.getMessage());
        return ResponseEntity.status(HttpStatus.BAD_REQUEST)
                .body("Invalid JSON data");
    }
}
```

Rys. 4.6 Wysyłanie i odbieranie wiadomości na forum społecznościowym [opracowanie własne]

Implementacja pokazana na rysunku 4.6 przedstawia metody związane z żądaniami dotyczących wiadomości wysyłanych między użytkownikami, które są następujące:

- *getIncomingMessage* - metoda zwracająca przez żądanie listę wszystkich wiadomości, które wysłane zostały do określonego użytkownika o identyfikatorze *toUserId*. Wynikiem metody są wiadomości przychodzące do aktualnie zalogowanego użytkownika.
- *getSentMessages* - metoda, która przez żądanie zwraca listę wszystkich wiadomości wysłanych od danego użytkownika o identyfikatorze *fromUserId*.
  Wynikiem metody są wiadomości wysłane przez użytkownika, który aktualnie jest zalogowany.
- *postMessage* - metoda odpowiedzialną za wysyłanie wiadomości i zapisywanie jej do bazy danych. Przy zapisie do bazy danych podawana jest informacja o nadawcy i adresacie wiadomości wraz z jej treścią.

4.4 Architektura aplikacji

4.4.1 Opis architektury

Architektura została zaprojektowana z myślą o zoptymalizowaniu procesów przepływu danych w sieci między warstwami aplikacji. Zgodnie z techniką tworzenia nowoczesnych aplikacji webowych, architektura systemu została podzielona na następujące warstwy aplikacji:

- aplikacja serwerowa,
- aplikacja internetowa,
- serwer bazodanowy.

Interfejs aplikacji internetowej pobiera dane od użytkownika. Pobrane dane zostają przekazywane do serwera aplikacji przez żądania HTTP. Dane te przed wysłaniem umieszczane są w formacie JSON (ang. *JavaScript Object Notation*). Następnie po ich odebraniu, serwer aplikacji zarządza procesami przekształcania tych danych, aby mogły zostać umieszczone w bazie danych na serwerze bazodanowym. Z bazy danych, aplikacja serwerowa może również pobierać dane albo je aktualizować. Do aplikacji internetowej wracają z powrotem niezbędne dane z serwera aplikacji.

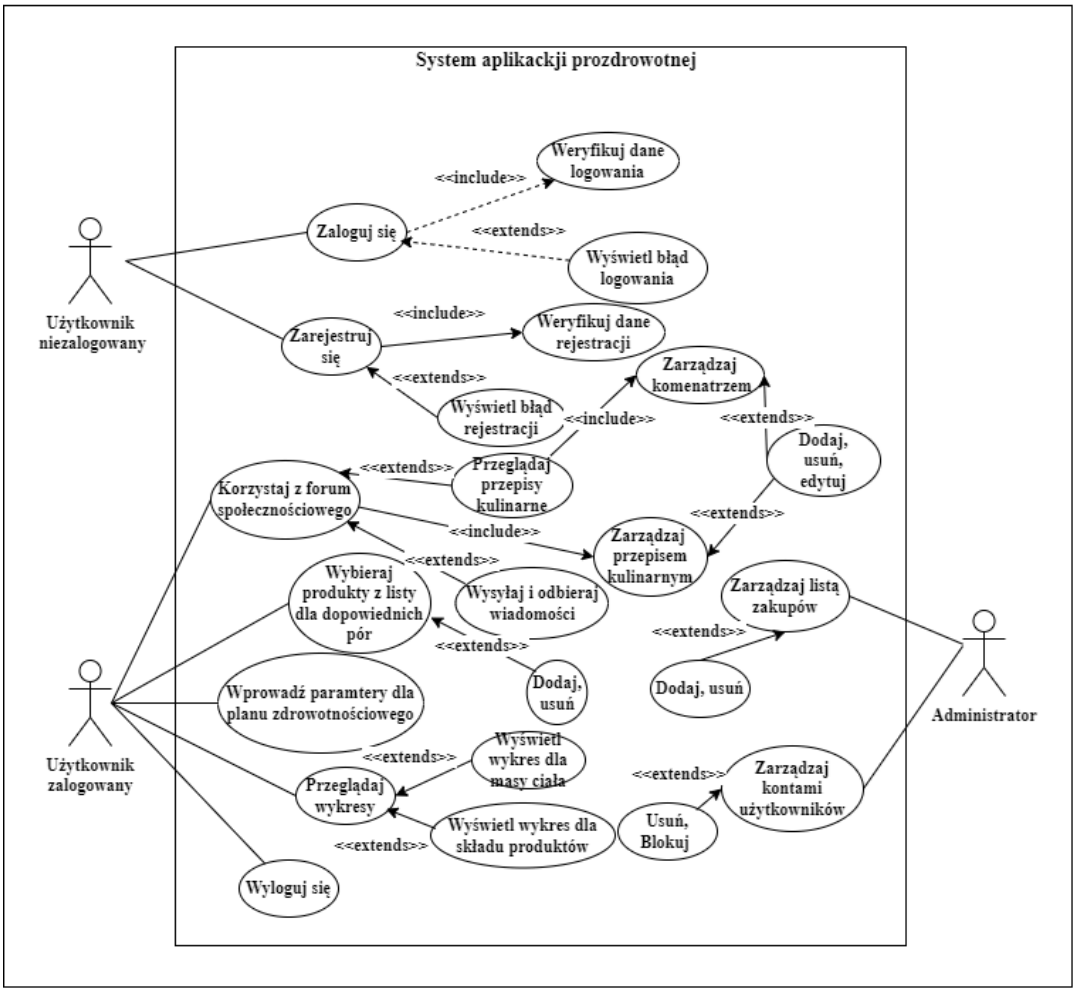
W aplikacji wykorzystano relacyjne bazy danych, gdyż pozwalają zachować spójność danych oraz ich integralność. Schemat bazy danych wraz z relacjami został zaimplementowany bezpośrednio w skrypcie pliku z rozszerzeniem sql.

Warstwa związana z interfejsem użytkownika kontroluje poprawność wprowadzanych danych oraz odpowiednio informuje, gdy warunki te nie zostaną spełnione.
Dodatkowo może przekształcać otrzymane dane z serwera. Wpływa to korzystnie na optymalność, gdyż odciąża to przepływ danych w sieci oraz serwer.

4.4.2 Diagram przypadków użycia

Diagram przypadków użycia (ang. *Use Case Diagram*) ilustruje interakcję użytkowników aplikacji z jej funkcjonalnościami.

Użytkownik, który nie jest zalogowany ma dostęp jedynie do obszaru rejestracji i logowania. Administrator ma uprawnienia do zmiany listy produktów oraz zarządzania kontami zwykłych użytkowników. Użytkownik zalogowany ma dostęp do całości obszaru aplikacji z wyjątkiem obszaru zarezerwowanego dla administratora.
Użytkownik ten może korzystać z forum, na którym publikuje przepisy. Posiada dostęp do funkcjonalności związanych z głównym przeznaczeniem aplikacji. Wszystko to umożliwia lepsze dbanie o zdrowie poprzez wgląd na wartości liczbowe związane z ilością makroskładników i kaloryczności posiłków. Dodatkowo jest możliwa kontrola masy ciała.

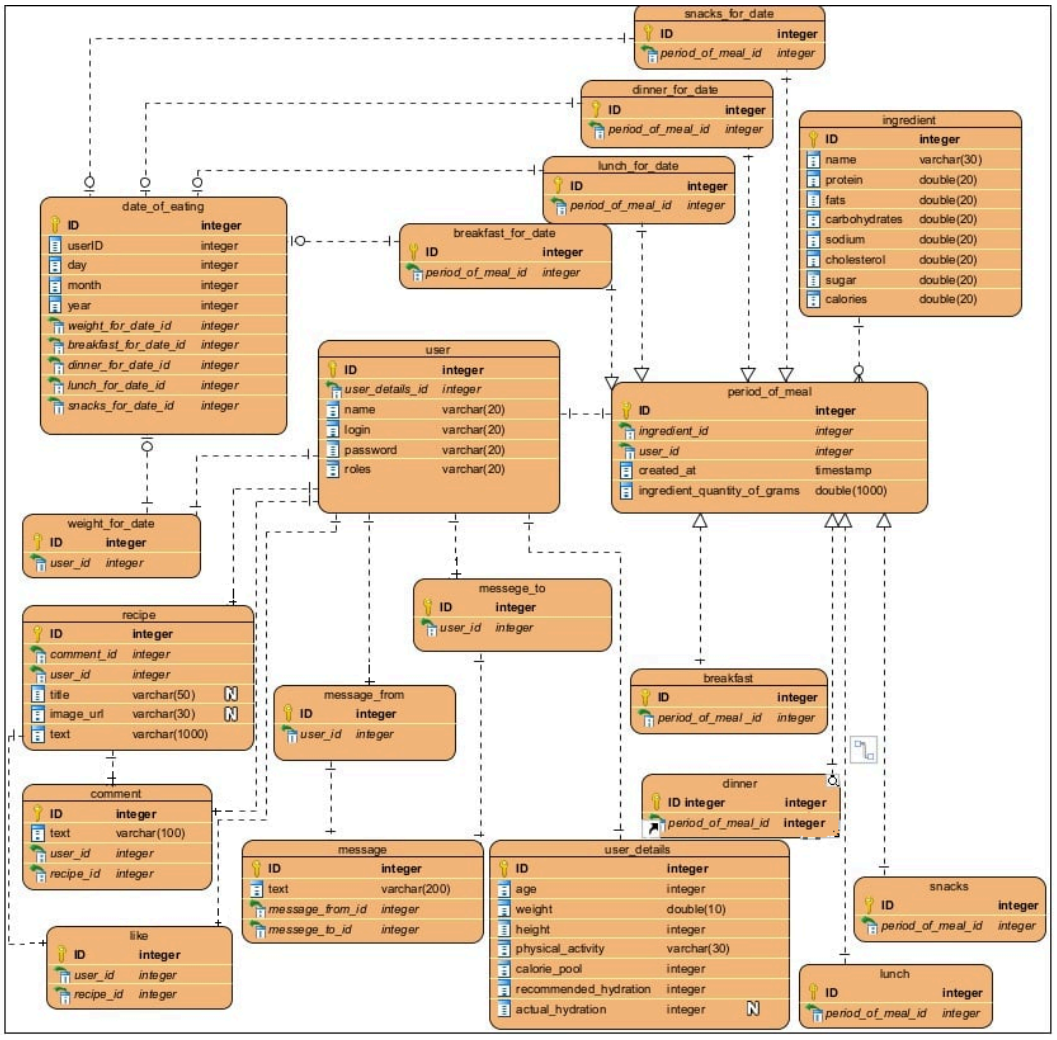


Rys. 4.7 Diagram przypadków użycia [opracowanie własne]

4.4.3 Diagram relacji encji

Diagram encji (ang. *Entity Relationship Diagram*) obrazuje strukturę danych dla całej aplikacji wraz z ich relacjami. Relacja to informacja w jaki sposób dane powiązane są między sobą. Tabela składa się z rekordów, które mają określony typ danych. Tabele zawierają dane dotyczące między innymi: użytkownika, listy składników do wyboru. Użytkownik tworzy posiłki na aktualny dzień albo wybiera posiłki na dany dzień z narzuconego zakresu.

Model zapisu danych na dzień z odpowiedniego zakresu, jest symulacją, gdyż aplikacja nie działa cały czas w tle i sama nie aktualizuje przez to danych na każdy dzień.



Rys. 4.8 Diagram relacji encji [opracowanie własne]

5. Interfejsy użytkownika

Przedstawione zostaną poszczególne widoki aplikacji składające się na interfejs użytkownika wraz z krótkim opisem ich funkcjonalności.

5.1 Ekran logowania i rejestracji

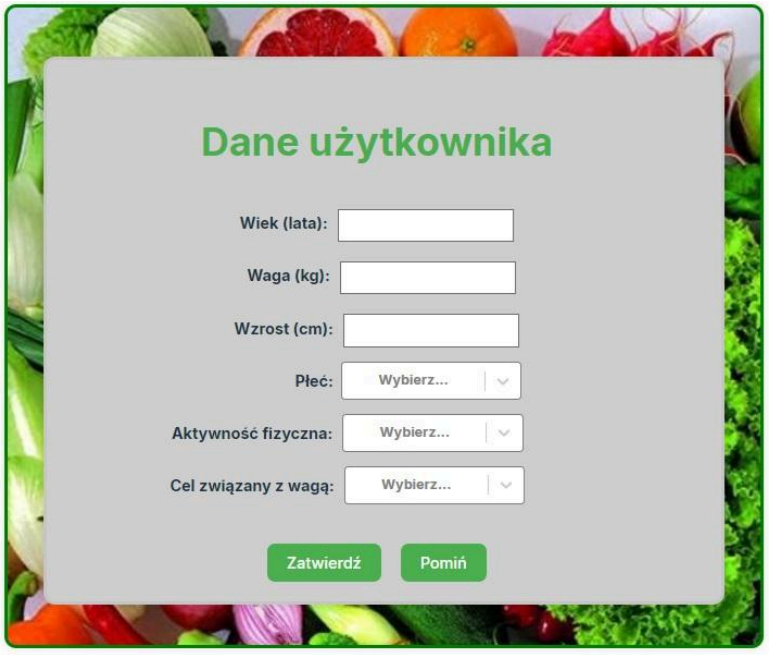
Rys. 5.1 Ekran rejestracji [opracowanie własne]

Rys. 5.2 Ekran logowanie [opracowanie własne]

Przy nieudanej rejestracji wynikającej z próby stworzenia konta użytkownika, którego dane byłyby duplikacją danych logowania istniejącego już konta, pojawia się alert na górze ekranu przeglądarki. Przy rejestracji nowego konta wybiera się uprawnienia administratora lub zwykłego użytkownika. Ilość znaków loginu i hasła musi mieścić się w ustalonym zakresie.

Przy niepomyślnym logowaniu wynikającym z podania nieprawidłowych danych logowania, pojawia się również alert na górze ekranu przeglądarki.

5.2 Ekran wprowadzania danych użytkownika

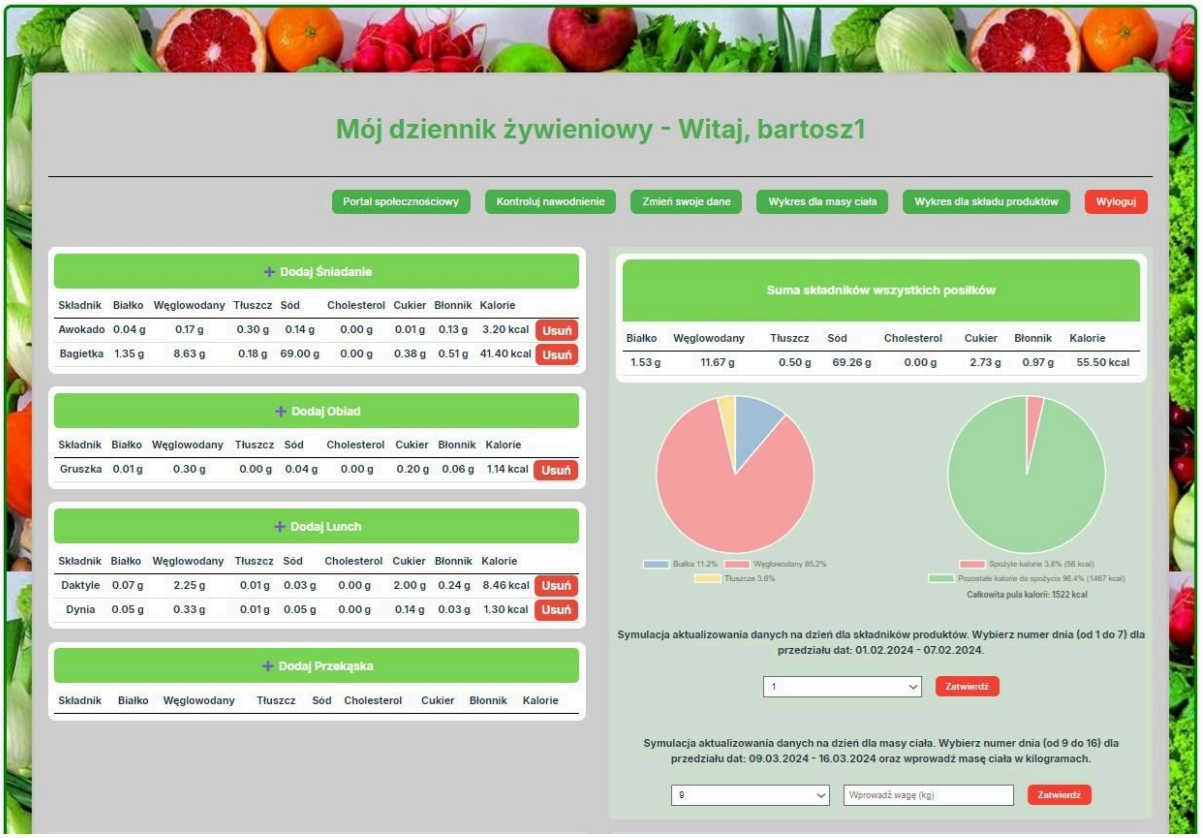


Rys. 5.3 Ekran wprowadzania danych użytkownika [opracowanie własne]

Na rysunku 5.3 zobrazowany jest ekran, na którym użytkownik uzupełnia formularz pozwalający z podanych danych wyliczyć sugerowaną wartość kalorii.
Pozwala to użytkownikowi lepiej kontrolować ilość spożywanych kalorii, aby zrealizować cel związany z redukcją, wzrostem bądź stabilizacją masy ciała.

Także na podstawie danych obliczana jest docelowa ilość płynów (w litrach), które użytkownik powinien uzupełnić w celu odpowiedniego nawodnienia organizmu. Możliwe jest pominięcie procesu wprowadzania danych, jeśli użytkownik wcześniej wypełniał ten formularz. Użytkownikowi umożliwione jest nadpisywanie danych.

5.3 Ekran komponowania posiłków.



Rys. 5.4 Ekran komponowania posiłków cz. I [opracowanie własne]

Rys. 5.5 Ekran komponowania posiłków cz. II [opracowanie własne]

Rysunek 5.4 oraz rysunek 5.5 przedstawia ekran, na którym po wyborze odpowiednich składników z listy, przypisywane są one do poszczególnych pór spożywania posiłków. Jest również opcja ich usuwania. Użytkownik ma wgląd w ilościowy udział makroskładników danego składnika wraz z jego kalorycznością. Dodatkowo obliczane są sumy tych makroskładników i kalorie, które przyporządkowane są do odpowiedniej pory spożycia posiłku oraz dla całego dnia. Dla każdej z sum, generowane są wykresy proporcji udziału tłuszczów, białka i węglowodanów. Tworzony jest także wykres dla dnia, ukazujący wartość sumy związanej ze spożytymi kaloriami wraz z sugerowanym dla nich limitem.

Na rysunku 5.4. widać po prawej stronie funkcjonalności odpowiadające za zapisywanie spożytych posiłków i masy ciała dla wybranego dnia. Zapis ten jest możliwy tylko dla dni z narzuconego zakresu. Nadpisywane są dane dla dnia, który był użyty wcześniej przy zapisie. Po prawej stronie pod tekstem powitalnym są umieszczone przyciski menu głównego.

5.4 Ekran wyboru składników na posiłek.

Rys. 5.6 Ekran wyboru składników na posiłek [opracowanie własne]

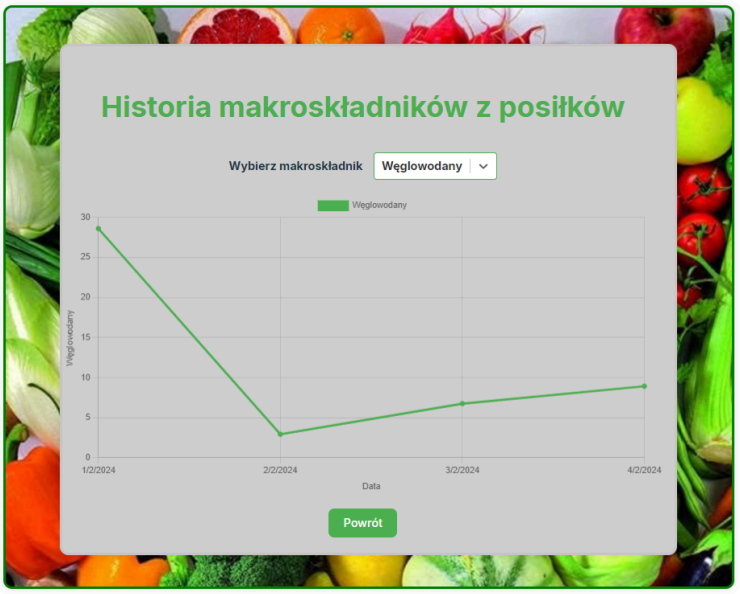
Rysunek 5.6 obrazuje ekran zawierający listę wszystkich dostępnych składników, z których użytkownik komponuje swoje posiłki. Po wybraniu składnika konieczne jest podanie jego wagi w gramach. Następnie użytkownik musi zatwierdzić swój wybór, aby składnik mógł zostać przyporządkowany do posiłku.

5.5 Ekran z wykresem zależności masy ciała od dni

Rys. 5.7 Ekran z wykresem zależności masy ciała od dni [opracowanie własne]

Na rysunku 5.7 przedstawiony jest ekran zawierający wykres dla masy ciała użytkownika z danego zakresu dni. Wykres skaluje się odpowiednio do wartości masy ciała, aby możliwe było jak najdokładniejsze zobrazowanie.

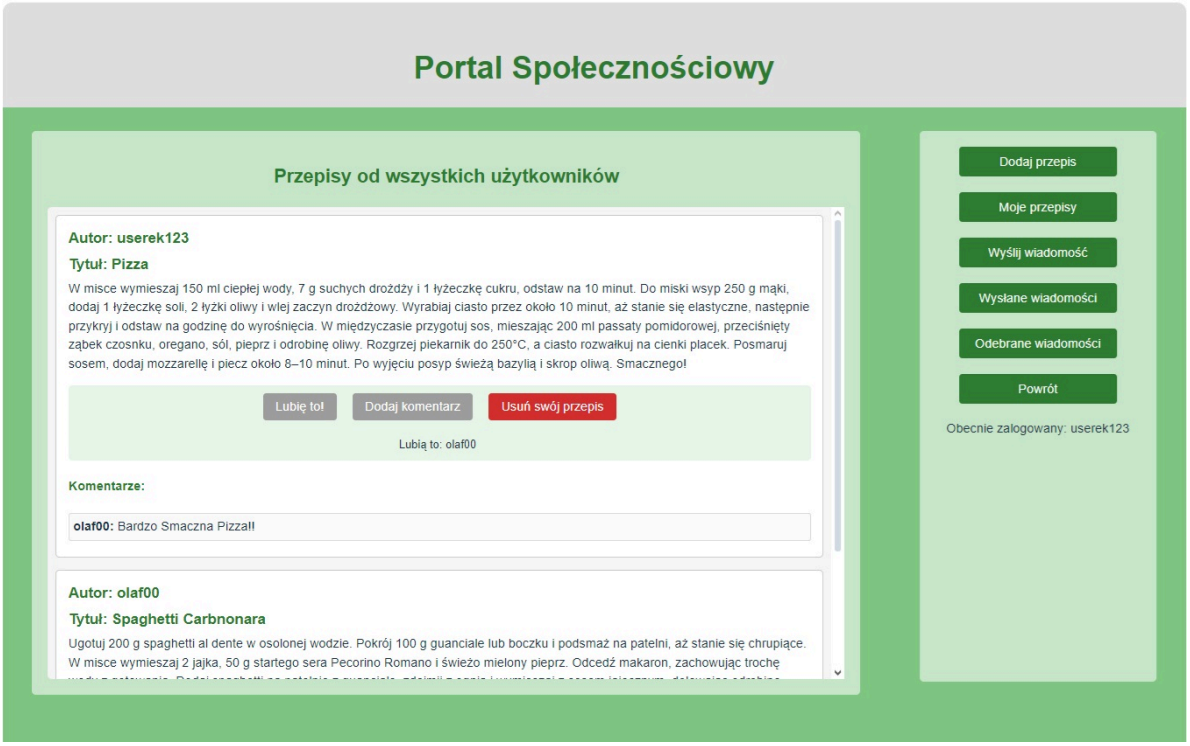
5.6 Ekran z wykresem dla spożytych składników



Rys. 5.8 Ekran z wykresem zależności ilości makroskładnika od dnia [opracowanie własne]

Na rysunku 5.8 został przedstawiony ekran zawierający wykres dla makroskładników wchodzących w skład sumy wszystkich posiłków użytkownika zapisanych na dany dzień. Użytkownik ma możliwość wybrania dla jakiego makroskładnika wykres ma zostać zobrazowany. Wykres skaluje się odpowiednio do ilości danego makroskładnika.

5.7 Ekran forum społecznościowego

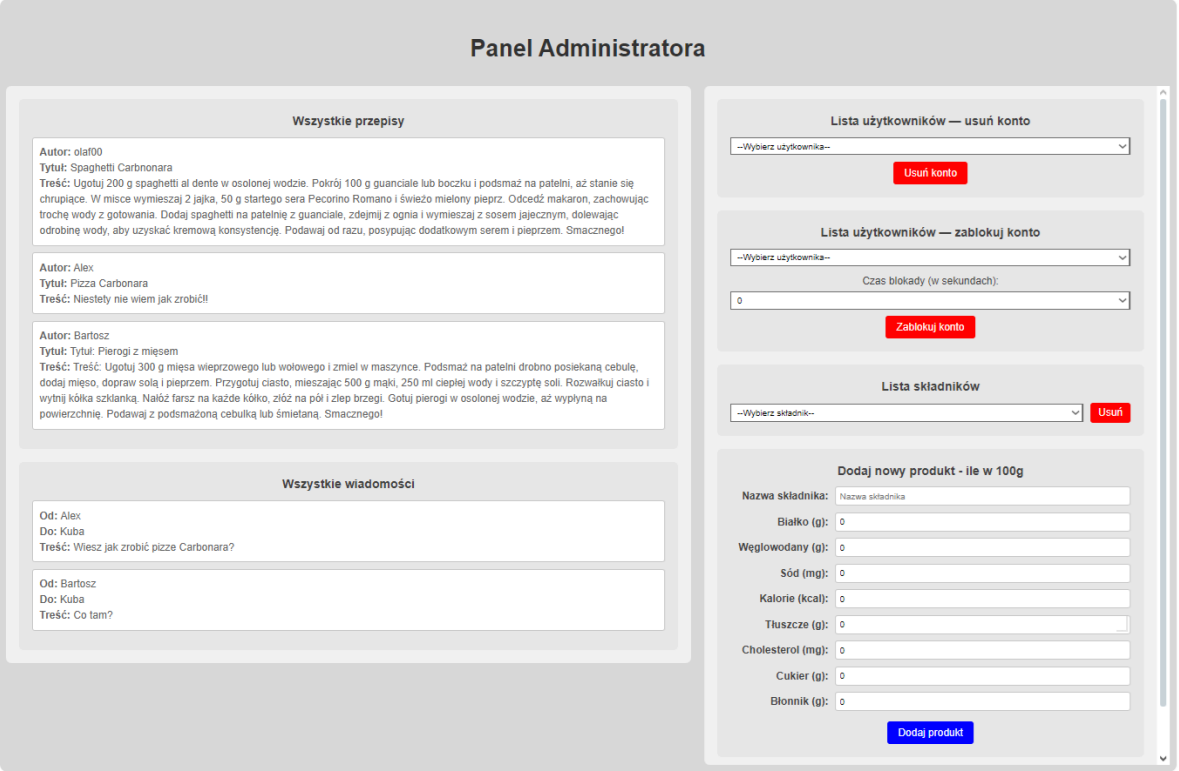


Rys. 5.9 Ekran forum społecznościowego [opracowanie własne]

Rysunek 5.9 przedstawia główny forum społecznościowego, na którym widoczne są wszystkie umieszczone posty wraz z polubieniami i komentarzami. Użytkownik ma uprawnienia do polubienia, bądź dania komentarza postowi, którego nie jest właścicielem. Użytkownik ma możliwość usuwać posty, wówczas gdy jest ich właścicielem.

Z panela głównego znajdują się przejścia do ekranów związanych z pozostałymi funkcjonalnościami forum społecznościowego.

5.8. Ekran funkcjonalności administratora



Rys. 5.10 Ekran funkcjonalności administratora [opracowanie własne]

Na rysunku 5.10 widoczny jest ekran z funkcjonalnościami dostępnymi wyłącznie dla administratora. Na tym ekranie znajduje się podgląd na wszystkie dodane przepisy przez użytkowników oraz na wysyłane przez nich wiadomości.

Administrator ma możliwość usunąć, bądź zablokować konto użytkownika. Także te uprawnienia umożliwiają usuwać lub dodawać produkt do listy produktów, z których użytkownik komponuje swoje posiłki.

6. Testy

Wybrane funkcjonalności aplikacji zostały przetestowane poprzez testy jednostkowe oraz testy integracyjne.

6.1 Testy jednostkowe

Testy jednostkowe używane są zazwyczaj do sprawdzenia pojedynczych fragmentów kodu np. metody, pojedynczych komponentów. Testy te weryfikują, czy algorytm danych funkcjonalności jest zgodny z wymaganiami postawionymi na etapie projektowania aplikacji. Zachowanie występujących w algorytmie interakcji między warstwami, zostaje zastąpione symulacją. Podawane jest jawnie, jak to zachowanie powinno przebiegać, aby spełniało wymagania. Dla Javy do symulowania zachowania tych interakcji wykorzystywana jest biblioteka Mockito.

6.1.1 Test poprawności obliczania docelowej wartości puli kalorii i nawodnienia

Po udanym zalogowaniu, użytkownik podaje swoje dane umożliwiające wyliczyć wartości weryfikowane przez test.

Docelowa pula kalorii (w kcal) obliczana jest na podstawie całkowitej przemiany materii z uwzględnieniem preferowanej zmiany masy ciała, bądź jej stabilizacji.
Na wynik całkowitej przemiany materii przyczynia się współczynnik aktywności fizycznej oraz wzór pośredniczący w obliczeniach, który określa podstawową przemianę materii. Podstawowa przemiana materii obliczana jest ze wzoru Harissa-Benedicta. Współczynnik aktywności fizycznej, według przyjętego standardu jest określeniem w sposób numeryczny podanego przez użytkownika rodzaju aktywności fizycznej. Wartość docelowego nawodnienia (w litrach), obliczana jest według jednego z przyjętych standardów, w którym przemnażana jest masa ciała (w *kg)* przez 40 ml.

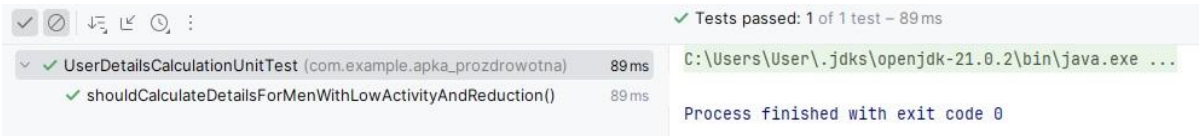
```java
@Test
public void shouldCalculateDetailsForMenWithLowActivityAndReduction() {
    // Given
    UserDetails userDetails = new UserDetails( weight: 80.0,  height: 180.0,  age: 30,
            physicialActivity: "low",  gender: "men",  weightPreference: "reduction");
    // When
    UserDetails result = userDetailsService.calculateUserDetails(userDetails);
    // Then
    assertThat(result.getRecommendedHydration()) AbstractDoubleAssert<capture of ?>
            .withFailMessage( newErrorMessage: "Expected recommended hydration to be 3.2," +
                    " but was %.2f", result.getRecommendedHydration()) capture of ?
            .isEqualTo( expected: 3.2);
    assertThat(result.getCaloriePool()) AbstractIntegerAssert<capture of ?>
            .withFailMessage( newErrorMessage: "Expected calorie pool to be 1423, but was %d",
                    result.getCaloriePool()) capture of ?
            .isEqualTo( expected: 1423);
}
```

Rys. 6.1 Test poprawności obliczania docelowej puli kalorii i nawodnienia [opracowanie własne]

Na rysunku 6.1. został przedstawiony test jednostkowy dla użytkownika będącego mężczyzną, który chce zredukować masę ciała przy zachowaniu niskiej aktywności fizycznej. Test został napisany z uwzględnieniem konwencji *Given-When-Then*.

Proces testowania *Given-When-Then* w oparciu o przykład z rysunku 6.1.:

1. *Given* - określane są rozpoczynające test warunki, które dotyczą danych użytkownika.
2. *When* - wywołana zostaje metoda obliczająca docelową wartość puli kalorii oraz nawodnienia.
3. *Then* - weryfikowana zostaje poprawność zwracanego wyniku przez wywołaną metodę. Wynik zostaje porównany z wartością, którą powinna zwrócić ta metoda.



Rys. 6.2 Rezultat testu poprawności obliczania docelowej puli kalorii i nawodnienia [opracowanie własne]

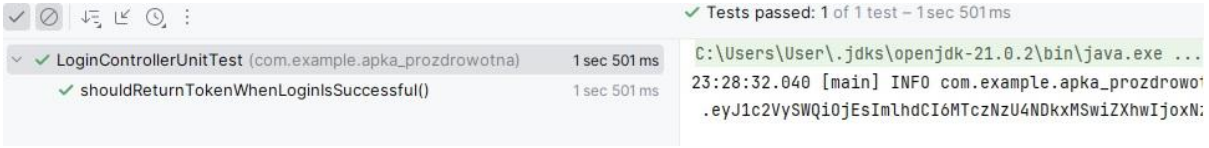
6.1.2 Test poprawności logowania

```
@Test
void shouldReturnTokenWhenLoginIsSuccessful() {
    // Given
    User mockUser = new User();
    mockUser.setUserId(1);
    mockUser.setLogin("testuser");
    mockUser.setPassword("password123");
    when(userRepository.findByLogin("testuser")).thenReturn(mockUser);
    User requestUser = new User();
    requestUser.setLogin("testuser");
    requestUser.setPassword("password123");
    // When
    ResponseEntity<?> response = loginController.loginUser(requestUser);
    // Then
    assertThat(response.getStatusCodeValue()) AbstractIntegerAssert<capture of ?>
            .withFailMessage( newErrorMessage: "Expected HTTP status 200, but got %d",
                    response.getStatusCodeValue()) capture of ?
            .isEqualTo( expected: 200);
    assertThat(response.getBody())
            .withFailMessage( newErrorMessage: "Expected response body to be an instance of AuthResponse," +
                    " but was %s", response.getBody())
            .isInstanceOf(LoginController.AuthResponse.class);
    LoginController.AuthResponse authResponse = (LoginController.AuthResponse) response.getBody();
    String token = authResponse.getToken();
    assertThat(token) AbstractStringAssert<capture of ?>
            .withFailMessage( newErrorMessage: "Expected token to be not null") capture of ?
            .isNotNull();
    String subject = Jwts.parser() JwtParserBuilder
            .setSigningKey(KEY)
            .build() JwtParser
            .parseClaimsJws(token) Jws<Claims>
            .getBody() Claims
            .getSubject();
    assertThat(subject) AbstractStringAssert<capture of ?>
            .withFailMessage( newErrorMessage: "Expected token subject to be 'testuser'," +
                    " but was '%s'", subject) capture of ?
            .isEqualTo( expected: "testuser");
    verify(userRepository, times( wantedNumberOfInvocations: 1)).findByLogin("testuser");
}
```

Rys 6.3 Test poprawności logowania [opracowanie własne]

Rysunek 6.3 obrazuje test jednostkowy sprawdzający algorytm wykonywany przez metodę odpowiadającą za logowanie użytkownika zakładając, że logowanie przebiegnie pomyślnie. Określane jest na wstępie dla podstawionych danych jak zachowywać mają się funkcje, które w warunkach rzeczywistych integruje się z bazą danych. Następnie wywoływana jest metoda *loginUser* dla kontrolera *loginControlle*r z danymi logowania, które są poprawne.

Po zakończeniu realizacji metody i zwróceniu przez nią danych, porównywane są ich wartości z poprawnymi wartościami spełniającymi założenie pomyślnego zalogowania.
W tym przypadku powinien być zwrócony status 200, który oznacza pomyślny przebieg metody. Także testowana metoda powinna zwrócić odpowiedni token JWT,
który zawiera w sobie poprawne dane logowania dla użytkownika.



Rys. 6.4 Rezultat testu logowania użytkownika [opracowanie własne]

6.2 Testy integracyjne

Testy integracyjne sprawdzają poprawność interakcji pomiędzy komponentami oraz modułami aplikacji. Testy te weryfikują, czy komunikacja między elementami odbywa się w skuteczny i oczekiwany sposób.

6.2.1 Test integracyjny serwera aplikacji z bazą danych

Istotną cechą aplikacji jest skuteczna komunikacja serwera aplikacji z bazą danych. Test integracyjny został podzielony na następujące testy, które sprawdzają poprawność:

- połączenia serwera aplikacji z bazą danych
- wyszukania istniejącego składnika w bazie danych
- zapisu składnika do bazy danych

```java
@Test
public void shouldSaveBreakfastWhenValidDataProvided() throws Exception {
    // Given
    IngradientsOptionResponseJSON request =
            new IngradientsOptionResponseJSON( label: "Ananas", quantityOfGrams: 70, userId: 1);

    // When
    boolean isIngredientInDatabaseBefore = mealIngredientRepository.
            existsByMealIngredientName("Ananas");

    mockMvc.perform(post( urlTemplate: "/postBreakfastIngredients")
                    .contentType(MediaType.APPLICATION_JSON)
                    .content(new ObjectMapper().writeValueAsString(request)))
            // Then
            .andExpect(status().isOk())
            .andExpect(jsonPath( expression: "$.mealIngredientId").value( expectedValue: "2"))
            .andExpect(jsonPath( expression: "$.mealQuantityOfGrams").value( expectedValue: 70))
            .andExpect(jsonPath( expression: "$.userId").value( expectedValue: 1));

    assertThat(isIngredientInDatabaseBefore) AbstractBooleanAssert<capture of ?>
            .withFailMessage( newErrorMessage: "Ingredient 'Ananas' " +
                    "is not found in the database.") capture of ?
            .isTrue();
}
```

Rys 6.5 Test integracyjny serwera aplikacji z bazą danych [opracowanie własne]

Rys 6.6 Rezultat testu integracyjny serwera aplikacji z bazą danych [opracowanie własne]

Rysunek 6.5. przedstawia test integracyjny metody z serwera aplikacji, która odpowiedzialna jest za dodawania wybranego składnika do encji w bazie danych związanej ze śniadaniem. Na początku jest sprawdzane, czy składnik znajduję się w bazie danych. Jeśli warunek jest spełniony, to składnik dodawany jest do encji *Breakfast*, do której funkcja zapisu znajduje się w metodzie pod endpointem *postBreakfastIngredients*.
W tej encji dodany składnik jest wiązany z identyfikatorem użytkownika, który wybrał dany składnik na śniadanie. Test weryfikuje pomyślność wykonanie żądania. Także sprawdzana jest poprawność zapisu danych pod encję *Breakfast*, gdyż oczekiwany wynik żądania powinien być obiektem, który został zapisany do bazy danych dla tej encji.

7. Podsumowanie i wnioski

Celem pracy dyplomowej było zaimplementowanie aplikacji internetowej dostarczającej funkcjonalności dla użytkowników, którzy chcą zmienić swoje nawyki żywieniowe poprzez odpowiedni wgląd na spożywane posiłki. Dodatkowo aplikacja miała umożliwić na dzielenie się przepisami między użytkownikami poprzez forum społecznościowe.

Realizacja tego projektu obejmowała zaprojektowanie, a także zaimplementowanie kompletnego systemu aplikacji internetowej i serwerowej wraz z bazą danych.

Zostały spełnione wszystkie założenia związane z wymaganiami funkcjonalnymi i niefunkcjonalnymi:

- Usprawnianie procesu stabilizacji lub zmiany masy ciała przez użytkownika - dla każdego użytkownika, obliczana jest zalecana pula kalorii i rekomendowane nawodnienie organizmu, a następnie te wartości ułatwiają manipulować masą ciała.
- Monitorowanie spożywanych posiłków - aplikacja umożliwia użytkownikowi tworzyć posiłki z wybranych produktów wraz ze szczegółowym wglądem na ich makroskładniki i kaloryczność.
- Generowanie wykresu - dla poszczególnego przedziału dni, tworzony jest odpowiednio wykres masy ciała użytkownika oraz wykres wartości makroskładników dla danych produktów wraz z ich kalorycznością.
- Forum społecznościowe - każdy użytkownik ma możliwość na udostępnianie swoich przepisów. Użytkownicy mogą korespondować między sobą, a także oceniać przepisy.
- Funkcje administratora - funkcje te pozwalają na manipulowanie listą dostępnych produktów, a także na blokowanie, bądź usunięcie konta użytkownika.

W ramach dalszego rozwoju, aplikacja przede wszystkim powinna wyjść poza obszar symulacji, który obecnie wynika z pewnych ograniczeń. Nowsza wersja aplikacji musi uwzględniać wymagania związane z ciągłym działaniem tej aplikacji w czasie rzeczywistym po wdrożeniu jej na serwer w chmurze. W tych warunkach jest zalecane,
aby dane były zapisywane w czasie rzeczywistym.

Kolejna funkcjonalność mogłaby dotyczyć integracji aplikacji ze sztuczną inteligencją. Możliwe jest zintegrowanie aplikacji z wirtualnym asystentem AI (ang. *Artificial Intelligence*).

Dzięki zaproponowanym modyfikacjom, aplikacja ma szansę spełnić oczekiwania współczesnych użytkowników.

Bibliografia

[1] Herbert Schildt, Java Kompendium programisty, Wydanie XII, 2023

[2] Craig Walls, Spring w akcji, Wydanie V, 2019

[3] Anthony Accomazzo, Nate Murray, Ari Lerner, Fullstack React, 2017

[4] Cherny Boris, Programowanie w TypeScript. Tworzenie skalowalnych aplikacji w JavaScript, 2020

[5] Vinicius M. Grippa, Sergey Kuzmichev, MySQL. Jak zaprojektować i wdrożyć wydajną bazę danych. Wydanie II, 2022

[6] Mark Masse, REST API Design Rulebook, 2012

[7] Andrew S. Tanenbaum, David J. Wetherall, Sieci komputerowe,
Wydanie V, 2012

[8] Adolfo Eloy Nascimento, OAuth 2.0 Cookbook, 2017

[9] JWT [Online] *https://jwt.io/*

[10] Maven [Online] *https://maven.apache.org*

Spis tabel

Spis rysunków