



**Politechnika
Śląska**

PROJEKT INŻYNIERSKI

Bazodanowa aplikacja webowa organizująca robienie zakupów, śledzenie wydatków oraz tworzenie harmonogramów żywieniowych.

Konrad RDUCH

Nr albumu: 300412

Kierunek: Informatyka

Specjalność: Grafika komputerowa i oprogramowanie

PROWADZĄCY PRACĘ

dr inż. Jacek Szedel

KATEDRA Algorytmiki i Oprogramowania

Wydział Automatyki, Elektroniki i Informatyki

Gliwice 2025

Tytuł pracy

Bazodanowa aplikacja webowa organizująca robienie zakupów, śledzenie wydatków oraz tworzenie harmonogramów żywieniowych.

Streszczenie

Współcześnie ważnym trendem jest aktywne podejście do dbania o swoje zdrowie. Elementem tego trendu jest dbanie o zdrową dietę. Stosowanie odpowiedniej diety pozwala na bardziej świadome realizowanie zakupów, a to z kolei przyczynia się do bardziej optymalnego wykorzystania zakupionych produktów. Pomimo istnienia wielu narzędzi pomagających w utrzymaniu diety i planowaniu zakupów, co jednak nie oznacza, że konsumenci nadal napotykają na różne trudności i niedogodności występujące w dostępnych systemach i aplikacjach. Brakuje mianowicie rozwiązania, łączącego możliwość planowania posiłków i realizowania celów żywieniowych oraz robienia zakupów. Celem niniejszej pracy było wykonanie aplikacji pomagającej użytkownikowi w robieniu zakupów, śledzeniu wydatków oraz przygotowywaniu harmonogramów żywieniowych. Opracowana aplikacja umożliwia użytkownikowi planowanie posiłków, tworzenie list zakupowych oraz tworzenie celów żywieniowych. Do budowy aplikacji zastosowano nowoczesne frameworki, takie jak Angular dla warstwy frontend i Spring Framework dla warstwy backend. Do przechowywania danych została użyta relacyjna baza danych MySQL. Dodatkowo aplikacja została skonteneryzowana za pomocą oprogramowania Docker. Rezultatem końcowym pracy jest aplikacja, pozwalająca użytkownikowi na efektywne zarządzanie swoją dietą i zakupami spożywczymi. Aplikacja oferuje intuicyjny interfejs, możliwość tworzenia celów żywieniowych, list zakupowych oraz planowanie posiłków.

Słowa kluczowe

Dieta, Lista zakupowa, Przepis, Aplikacja internetowa, Spring, Angular

Thesis title

A database-driven web application to organize shopping, expense tracking and diet scheduling.

Abstract

An important trend today is to take a proactive approach to taking care of one's health. Part of this trend is taking care of a healthy diet. Following a proper diet allows you to make more conscious purchases, which in turn contributes to a more optimal use of the products you buy. Despite the existence of several tools to help maintain a diet and plan purchases, however, that does not mean that consumers still face various difficulties

and inconveniences found in the available systems and applications – there is a lack of a solution, combining the ability to plan meals and achieve nutritional goals, as well as shopping. The purpose of the present study was to make an application that helps the user shop, track expenses and prepare nutritional schedules. The developed application allows the user to plan meals, create shopping lists and create nutritional goals. Modern frameworks such as Angular for the frontend layer and Spring Framework for the backend layer were used to build the application. A MySQL relational database was used to store data. In addition, the application was containerized using Docker software. The result of the work is an application that allows users to effectively manage their diet and grocery shopping. The application offers an intuitive interface, and the ability to create dietary goals, shopping lists and meal planning.

Key words

Diet, Shopping list, Recipes, Web application, Spring, Angular

Spis treści

1	Wstęp	1
2	Analiza tematu	3
2.1	Opis dziedziny przedmiotowej	3
2.1.1	Wpływ diety na zdrowie	3
2.1.2	Optymalizacja zakupów, a marnowanie żywności	4
2.1.3	Najważniejsze funkcje aplikacji wspierającej zarządzanie dietą i zakupami	5
2.2	Istniejące rozwiązania	6
2.2.1	Aplikacja Fitatu	6
2.2.2	Aplikacja Yazio	6
2.2.3	Aplikacja Listonic	6
3	Wymagania i narzędzia	7
3.1	Wymagania	7
3.1.1	Wymagania нефункционалне	7
3.1.2	Model przypadków użycia	8
3.2	Narzędzia	21
3.2.1	Spring Framework	22
3.2.2	MySQL	22
3.2.3	Angular	22
3.2.4	Bootstrap	22
3.2.5	Docker	23
3.2.6	Postman	23
3.3	Metodyka pracy nad projektem	23
4	Specyfikacja zewnętrzna	25
4.1	Zagadnienia administracyjne	25
4.1.1	Wymagania sprzętowe	25
4.1.2	Wymagania systemowe	25
4.1.3	Konfiguracja serwera	26

4.1.4	Wstępna konfiguracja systemu	27
4.2	Użytkowanie aplikacji	27
4.2.1	Role użytkowników	27
4.2.2	Scenariusze działania systemu	28
5	Specyfikacja wewnętrzna	45
5.1	Architektura systemu	45
5.1.1	Zarys architektury	45
5.1.2	Model klas	46
5.2	Wybrane fragmenty kodu	48
5.2.1	Przykładowa encja	48
5.2.2	Przykładowy kontroler	51
5.2.3	Przykładowe repozytorium	53
5.2.4	Przykładowy serwis - warstwa backend	53
5.2.5	Przykładowy serwis - warstwa frontend	55
5.2.6	Przykładowy interceptor	56
5.2.7	Przykładowy komponent	57
6	Weryfikacja i walidacja	59
6.1	Metodyka testów	59
6.2	Przebieg testów	59
6.2.1	Testy modułowe	60
6.2.2	Testy integracyjne	61
6.2.3	Testy systemowe	62
6.2.4	Testy akceptacyjne	62
7	Podsumowanie i wnioski	63
	Bibliografia	65
	Spis skrótów i symboli	69
	Źródła	71
	Lista dodatkowych plików, uzupełniających tekst pracy	73
	Spis rysunków	76
	Spis tabel	77
	Spis listingów	79

Rozdział 1

Wstęp

Współczesne społeczeństwo coraz lepiej uświadamia sobie znaczenie zdrowego stylu życia. Odpowiednie odżywianie, regularna aktywność oraz zarządzanie produktami spożywczymi przyczyniają się do poprawy jakości życia. Globalna pandemia COVID-19 pokazała, jak ważne jest to zagadnienie. Podczas pandemii ludzie musieli pozostawać w domu, a brak aktywności fizycznej wzmocniony przez negatywne skutki uboczne zakażenia wirusem skutkował pogorszeniem się stanu zdrowotnego i psychicznego społeczeństwa. Po wygaśnięciu pandemii jeszcze więcej osób zaczęło poszukiwać skutecznych metod realizacji różnego rodzaju diet i planowania posiłków – długotrwały brak aktywności fizycznej zwiększył liczbę osób borykających się z problemem nadwagi. Bardziej świadome podejście do kwestii diety przyczynia się jednocześnie do bardziej rozsądnego robienia zakupów spożywczych i ogranicza marnowanie żywności.

Istnieje wiele rozwiązań wspomagających zdrowe odżywianie i robienie zakupów. Popularne aplikacje, takie jak "Yazio" czy "Fitatu" oferują szereg funkcjonalności, które pomagają w liczeniu kalorii i planowaniu posiłków. Istnieją również rozwiązania, które pomagają użytkownikowi w planowaniu i robieniu zakupów. Rozwiązania takie jak "Listonic" umożliwiają użytkownikowi tworzenie własnych list zakupowych. Brakuje jednak rozwiązania łączącego funkcje planowania celów żywieniowych, planowania diety oraz optymalizacji procesu zakupów. Stanowi to motywację do stworzenia aplikacji, która umożliwi użytkownikowi dostęp do zdrowych przepisów, wyliczy wartości odżywcze posiłków, a także pomoże kontrolować koszty i przygotuje odpowiednią listę zakupów.

Celem niniejszej pracy jest zaprojektowanie i implementacja aplikacji webowej wspierającej planowanie posiłków oraz optymalizację zakupów artykułów spożywczych. Aplikacja ma umożliwić użytkownikowi przeglądanie listy z przepisami oraz listy z artykułami spożywczymi. Użytkownik będzie mógł dodawać przepisy do swojego kalendarza, które będą automatycznie zapisywane w harmonogramie zakupowym i żywieniowym użytkownika. w harmonogramie żywieniowym znajdą się zaplanowane posiłki, które użytkownik będzie mógł ocenić, skomentować i dodać do nich zdjęcie przygotowanej potrawy. Harmonogram zakupowy będzie pokazywał użytkownikowi listę zakupów, które musi wy-

konać w danym dniu. Użytkownik będzie miał także możliwość tworzenia własnych list zakupowych, niepowiązanych z zaplanowanym posiłkiem. Aplikacja umożliwi użytkownikowi tworzenie celów żywieniowych, które są nastawione na utrzymanie, zmniejszenie lub zwiększenie wagi ciała, obwodu brzucha i poziomu tkanki tłuszczowej. Aplikacja będzie śledziła postęp użytkownika, liczyła dzienne zapotrzebowanie kaloryczne wraz z rozbićciem na makroskładniki i będzie proponowała użytkownikowi przepisy do realizowanego celu. Użytkownik będzie mógł aktualizować swoje parametry ciała, na przykład wagę ciała, a aplikacja będzie tworzyła historię pomiarów. Aplikacja zostanie stworzona przy pomocy specjalnych narzędzi w architekturze Klient-Server.

Praca składa się z siedmiu rozdziałów. Pierwszy rozdział zawiera niniejszy wstęp. Drugi rozdział przedstawia analizę tematu. Omówiono w nim wpływ diety na zdrowie, a także odniesiono się do problemu marnowania żywności. Dodatkowo przedstawiono wybrane aplikacje związane z planowaniem diety i robieniem zakupów. w rozdziale trzecim opisano wymagania funkcjonalne i нефункционалне aplikacji oraz opisano wykorzystane narzędzia i metodykę prac nad projektem. Czwarty rozdział zawiera specyfikację zewnętrzną, w której opisano zagadnienia administracyjne, takie jak wymagania sprzętowe i systemowe, konfiguracje serwera oraz wstępną konfigurację systemu. w rozdziale opisano także użytkowanie aplikacji, przedstawiając dostępne w programie role użytkowników oraz scenariusze działania systemu. Piąty rozdział zawiera specyfikację wewnętrzną. Opisano w nim architekturę systemu, model klas i wybrane fragmenty kodu. Szósty rozdział opisuje metodykę testów z użyciem modelu "V". Przedstawiono przebieg testów oraz to, jakie błędy udało się w nich ujawnić. Siódmy rozdział zawiera podsumowanie i wnioski.

Rozdział 2

Analiza tematu

2.1 Opis dziedziny przedmiotowej

Niniejszy podrozdział poświęcono omówieniu dziedziny przedmiotowej, w której funkcjonować będzie projektowana aplikacja. W pierwszej części podrozdziału odniesiono się do motywacji autora płynącej z przekonania o tym, jak ważne w obecnych czasach jest zachowanie odpowiedniej diety, i z tego, że nawet stosunkowo proste rozwiązania informatyczne mogą pomóc w zdrowym odżywianiu się. W części drugiej opisano kolejny ważny fakt związany z tym, że przemyślane podejście do diety i odpowiednie planowanie zakupów może pomóc w ograniczeniu marnowania żywności, które ma miejsce często wtedy, gdy zakupy nie są odpowiednio zaplanowane. W podrozdziale przedstawiono także potencjalne funkcje aplikacji wspomagającej zarządzanie dietą i zakupami.

2.1.1 Wpływ diety na zdrowie

Zdrowie człowieka w dużym stopniu zależy od stosowanej przez niego diety. Organizm ludzki do prawidłowego funkcjonowania potrzebuje energii, dostarczanej z pożywienia. Ważne jest to, żeby w spożywanym posiłku zapewnić odpowiednią ilość makroskładników i mikroskładników. Niedobór ilości składników odżywczych lub ich nadmiar może prowadzić do problemów ze zdrowiem [13].

Zła dieta cechuje się spożywaniem nadmiernej ilości wysoko przetworzonych produktów, cukrów oraz soli, które prowadzą do szeregu problemów zdrowotnych, takich jak otyłość, nadciśnienie tętnicze czy choroby układu krążenia. Dobra dieta cechuje się różnorodnością produktów, które dostarczają niezbędne składniki odżywcze oraz wykluczają szkodliwe.

Według rekomendacji Światowej Organizacji Zdrowia dotyczących odżywiania, należy dbać o odpowiedni bilans energetyczny i masę ciała, zastąpić tłuszcze nasycone tłuszczami nienasyconymi, ograniczyć spożywanie cukrów prostych i soli oraz zwiększyć spożycie warzyw, owoców i produktów pełnoziarnistych.

Dobra dieta zawiera warzywa i owoce, które posiadają dużo witamin oraz minerałów. Dodatkowo zawierają także błonnik pokarmowy, który reguluje pracę układu trawienego oraz obniża poziom cholesterolu, co skutkuje zmniejszeniem ryzyka wystąpienia chorób sercowo-naczyniowych. Produkty pełnoziarniste i zbożowe dostarczają węglowodanów złożonych, które są podstawowym źródłem energii, a także pomagają utrzymać stabilny poziom cukru we krwi. Ważnym elementem dobrej diety jest też odpowiednia ilość białka w organizmie. Białko pełni funkcję budulcową i regeneracyjną oraz dostarcza aminokwasy, których organizm nie jest w stanie samodzielnie syntetyzować. Źródłem białka są ryby, mięso, jajka, nasiona roślin strączkowych oraz orzechy. Nadmiar tłuszczów nasyconych może przyczynić się do rozwoju otyłości, miażdżycy lub cukrzycy typu 2. Rekomendowane jest spożywanie zdrowych tłuszczów z produktów żywnościowych, takich jak oliwa z oliwek, awokado lub orzechy, wspierających pracę układu hormonalnego. Niedobór witamin z grupy B, magnezu, cynku lub kwasów omega-3 może prowadzić do wystąpienia stanów depresyjnych oraz zaburzeń lękowych.

2.1.2 Optymalizacja zakupów, a marnowanie żywności

Rozwój nauki oraz postęp technologiczny znacząco usprawnił proces produkcji żywności, dzięki czemu współcześnie ludzie nie muszą zmagać się z problemem jej niedoboru, pomijając oczywiście sytuacje patologiczne. Problem związany z niedoborem żywności w krajach rozwiniętych odwrócił się i obecnie istnieje problem nadmiernej podaży i złego jej wykorzystania. Problem marnowania żywności ma zarówno wymiar ekologiczny, ekonomiczny, jak i społeczny i stanowi wyzwanie dla współczesnego świata. Szacuje się, że rocznie na świecie marnowana jest jedna trzecia całkowitej produkcji żywności przeznaczonej do spożycia przez ludzi.

Zasadniczym powodem marnowania żywności jest niepoprawne planowanie zakupów, gdzie konsumenci kupują więcej jedzenia, niż potrzebują. Następnym powodem jest kupowanie produktów żywnościowych na różnego typu promocjach, w których przy zakupie większej ilości danego artykułu spożywczego dostaje się rabat, a często większa ilość produktów nie jest kupującemu potrzebna.

W celu ograniczenia ilości marnowanego jedzenia można odpowiednio zoptymalizować proces robienia zakupów. Obejmuje on świadome podejście do planowania posiłków, sporządzanie list zakupowych oraz bieżące monitorowanie stanu zapasów. Kluczowym elementem tego procesu jest tworzenie list zakupowych w oparciu o zaplanowane posiłki, dzięki któremu można uniknąć zakupu niepotrzebnego produktu.

Optymalizacja procesu robienia zakupów ma znaczenie globalne. Przy produkcji żywności wykorzystuje się zasoby naturalne, takie jak woda, energia i ziemia. Wyrzucenie jedzenia skutkuje zmarnowaniem zasobów użytych w ich produkcji. Na przykład do wyprodukowania jednego kilograma chleba wykorzystuje się 1600 litrów wody. Problem także

stanowi żywność rozkładająca się na wysypisku, która emituje gazy cieplarniane.

Jak widać z powyższych obserwacji, stworzenie rozwiązania, które połączy w sobie funkcje planowania i monitorowania diety z funkcjami planowania zakupów i wdrożenie takiego rozwiązania w gospodarstwie domowym, może być bardzo pożyteczne i to w różnych aspektach.

2.1.3 Najważniejsze funkcje aplikacji wspierającej zarządzanie dietą i zakupami

Aplikacja wspomagająca użytkownika w zarządzaniu dietą i zakupami powinna oferować użytkownikowi szereg funkcjonalności, ułatwiających realizację celów żywieniowych, tworzenie harmonogramu żywieniowego oraz organizację zakupów. Poniżej przedstawiono kluczowe funkcje potencjalnej aplikacji wspierającej zarządzanie dietą i zakupami:

Dostęp do bazy zdrowych przepisów. Aplikacja powinna umożliwić użytkownikowi wyszukiwanie zdrowych przepisów, które uwzględniają jego preferencje dietetyczne oraz ograniczenia kaloryczne.

Informacje o koszcie przygotowania dania. Aplikacja powinna obliczać i pokazywać użytkownikowi szacunkowy koszt przygotowania dania.

Organizacja planu żywieniowego. Aplikacja powinna umożliwić użytkownikowi tworzenie własnych harmonogramów żywieniowych, ograniczając przy tym marnowanie żywności spowodowane niepotrzebnym zakupem produktów.

Lista zakupów powiązana z planem. Aplikacja powinna automatycznie tworzyć listy zakupowe zaplanowanych posiłków. Użytkownik mógłby edytować listy oraz oznaczać z nich produkty jako zakupione. Dodatkowo istniałaby możliwość dodania listy zakupowych do ulubionych oraz utworzenia listy zakupowej niepowiązanej z zaplanowanym posiłkiem.

Organizacja celów żywieniowych. Aplikacja powinna umożliwić użytkownikowi tworzenie własnych celów żywieniowych, w zależności od aktualnych parametrów ciała, takich jak waga czy poziom tkanki tłuszczowej. Sugerowane listy zakupowe powinny być proponowane przez aplikację w zależności od prowadzonego celu żywieniowego.

Monitorowanie postępów zdrowotnych. Aplikacja mogłaby śledzić postęp użytkownika, poprzez tworzenie podsumowań wraz z wykresami, które dają użytkownikowi wiadomość zwrotną co do jego postępu.

Celem pracy jest stworzenie aplikacji wspierającej użytkownika w planowaniu posiłków i związanych z nimi zakupów, wpływając tym samym na optymalne wykorzystanie zakupionych produktów i wydatków związanych z artykułami spożywczymi. Narzędzie ma na celu wspierać zdrowe nawyki żywieniowe, a także ułatwiać użytkownikowi osiąganie zaplanowanych celów żywieniowych.

2.2 Istniejące rozwiązania

W poniższym podrozdziale przedstawiono i opisano wybrane, funkcjonujące w sieci rozwiązania z zakresu zarządzania dietą i planowania zakupów. Każda z nich oferuje różnorodną funkcjonalność ułatwiającą planowanie posiłków, a także śledzenie aktywności fizycznej.

2.2.1 Aplikacja Fitatu

Aplikacja Fitatu [1] wspiera użytkownika w zarządzaniu dietą i podażą kalorii. Posiada ona bogatą bazę artykułów spożywczych oraz przepisów. Aplikacja posiada wbudowany skaner kodów kreskowych, ułatwiający dodawanie produktów. Posiada także polskie miary użyteczne, takie jak łyżka lub garść. Dodatkowo posiada opcję udostępniania jadłospisów znajomym, licznik spożycia wody i możliwość dodania celów żywieniowych. Aplikacja dostępna jest w wersji webowej oraz mobilnej. Współpracuje także ze smart zegarkami Apple Watch.

2.2.2 Aplikacja Yazio

Aplikacja Yazio [2] udostępnia funkcjonalność zależnie od wykupionego planu taryfowego. Użytkownicy korzystający z darmowej wersji mogą skorzystać z dopasowanych planów treningowych, skupiających się na budowaniu mięśni lub utracie wagi. Aplikacja pozwala na skanowanie kodów kreskowych oraz zapamiętuje wyszukiwane posiłki. Dodatkowo monitoruje aktywność fizyczną. Wersja płatna rozszerza działanie aplikacji o plany żywieniowe, oferuje dodatkowe zdrowe przepisy oraz rozszerza analizę statystyk ciała i składników odżywczych. Dodatkowo monitoruje postęp użytkownika oraz ciśnienie krwi.

2.2.3 Aplikacja Listonic

Aplikacja Listonic [3] jest narzędziem pomagającym w robieniu zakupów. Pozwala użytkownikowi na tworzenie list zakupowych i udostępnianie ich innym osobom. Aplikacja synchronizuje zmiany w czasie rzeczywistym i powiadamia o nowo utworzonych listach oraz dodanych produktach. Posiada asystenta sztucznej inteligencji, usprawniającego tworzenie list. Dodatkowo istnieje możliwość dodawania produktów za pomocą mowy oraz dodawania zdjęć i określenia ilości produktów. Aplikacja jest dostępna w wersji webowej lub w wersji na urządzenia mobilne.

Rozdział 3

Wymagania i narzędzia

3.1 Wymagania

W niniejszym rozdziale zaprezentowano wymagania zidentyfikowane na etapie analizy systemu. W opisie uwzględniono zarówno wymagania funkcjonalne, jak i нефunkcjonalne. Wymagania нефunkcjonalne zaprezentowano w postaci ich krótkiej charakterystyki, wymagania funkcjonalne przedstawiono za pomocą modelu przypadków użycia w formie odpowiedniego diagramu UML (ang. Unified Modeling Language) oraz opisu tabelarycznego.

3.1.1 Wymagania нефunkcjonalne

Poniżej przedstawiono wymagania нефunkcjonalne, które wzięto pod uwagę podczas projektowania systemu oraz wyboru narzędzi.

Bezpieczeństwo. Dane użytkownika aplikacji powinny być zabezpieczone oraz chronione przed atakami z zewnątrz. System powinien posiadać mechanizm autoryzacji oparty na standardzie JWT (ang. JSON Web Token, gdzie JSON oznacza JavaScript Object Notation), który zapewni bezpieczne zarządzanie sesjami użytkowników. Należy także zastosować podział na role, ograniczający dostęp do określonych funkcjonalności lub zasobów systemu. Ponadto hasła użytkowników w bazie danych muszą być szyfrowane przez funkcje haszujące, na przykład takie jak BCrypt.

Integralność danych. System powinien zapewniać spójność danych użytkownika. W tym celu wykorzystano relacyjną bazę danych, której odpowiednio skonfigurowane klucze obce zapewniają ochronę przed modyfikacją rekordów. Usuwanie rekordów powinno być zabezpieczone mechanizmem transakcji, który w razie błędu wycofa operacje. Warstwy backend i frontend powinny zawierać mechanizmy walidacji danych, które chronią system przed nieprawidłowymi wartościami pól.

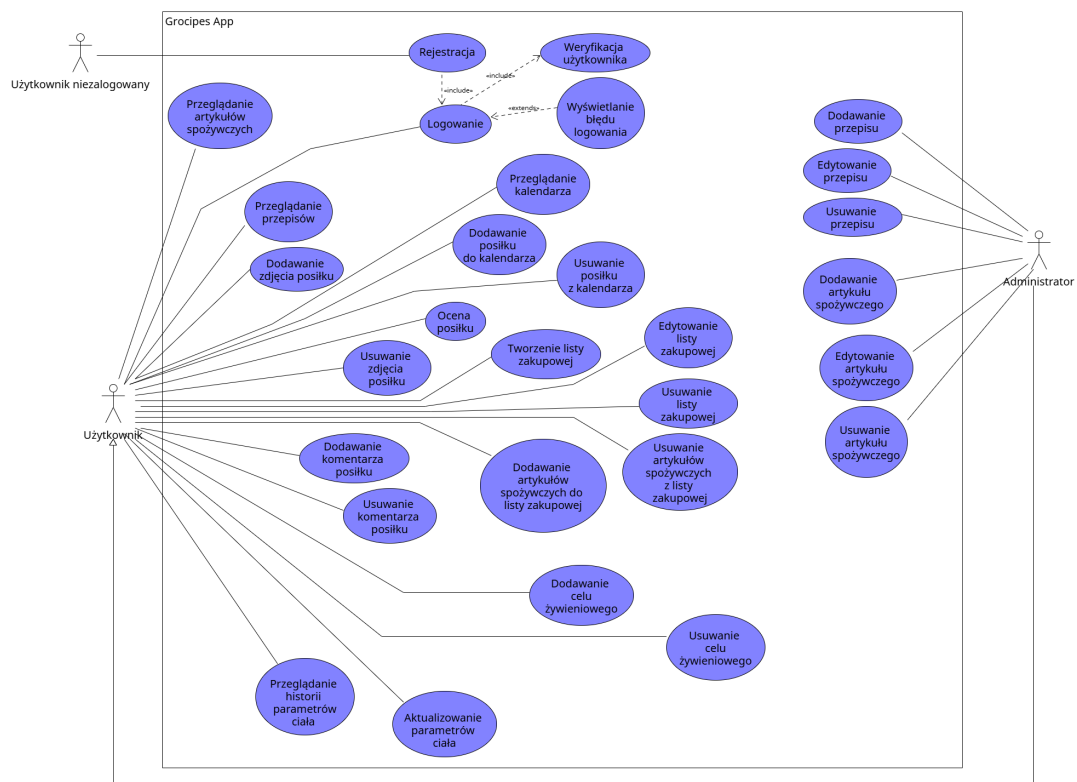
Rozszerzalność. Aplikacja powinna zapewniać łatwość w rozbudowie bez konieczności modyfikowania istniejącego kodu. W tym celu zastosowano w warstwie frontend

modułową architekturę kodu realizowaną za pomocą frameworku Angular. Warstwa backend opiera się na architekturze opartej na wzorcu Controller-Service-Repository. Dodatkowo REST (ang. Representational State Transfer) API (ang. Application Programming Interface) zapewnia niezależność i separację warstw.

Użyteczność System musi zapewniać intuicyjny interfejs użytkownika, który ułatwi użytkownikowi poruszanie się po aplikacji. Ponadto należy także zapewnić to, że użytkownicy będą mogli korzystać z aplikacji na różnych urządzeniach. Celem aplikacji jest pomoc użytkownikowi w codziennych czynnościach, dlatego musi być dostępna w każdej chwili.

3.1.2 Model przypadków użycia

Podrozdział ten ma na celu szczegółowe omówienie diagramu przypadków użycia (rysunek 3.1), który ilustruje funkcjonalności systemu z perspektywy użytkownika. Diagram obrazuje interakcje między aktorami a poszczególnymi przypadkami użycia. Dodatkowo, w tabelach 3.1 - 3.32 znajdują się opisy przypadków, zawierające ich identyfikator, nazwę, aktorów uczestniczących, dokładny opis operacji oraz warunki początkowe i końcowe.



Rysunek 3.1: Diagram przypadków użycia.

W Tabeli 3.1 został opisany przypadek użycia utworzenia konta przez niezarejestrowanego użytkownika.

Tabela 3.1: Rejestracja

Identyfikator	PU-1
Nazwa	Rejestracja
Aktor	Użytkownik niezarejestrowany
Opis	Użytkownik naciska przycisk <i>Switch to Sign Up</i> . Uzupełnia pola formularza rejestracyjnego wpisując swoje dane. Następnie klika przycisk <i>Sign Up</i> na dole formularza.
Warunki wstępne	Każde konto posiada tylko jeden adres mailowy.
Warunki końcowe	Utworzenie konta.

W Tabeli 3.2 został opisany przypadek użycia zalogowania się do aplikacji przez użytkownika.

Tabela 3.2: Logowanie

Identyfikator	PU-2
Nazwa	Logowanie
Aktor	Użytkownik
Opis	Użytkownik podaje adres mailowy oraz hasło. Następnie klika przycisk <i>Login</i> .
Warunki wstępne	Użytkownik utworzył konto w aplikacji
Warunki końcowe	Weryfikacja hasła.

W Tabeli 3.3 został opisany przypadek użycia weryfikacji użytkownika po zalogowaniu się do aplikacji.

Tabela 3.3: Weryfikacja użytkownika

Identyfikator	PU-3
Nazwa	Weryfikacja użytkownika
Aktor	Użytkownik
Opis	Aplikacja weryfikuje dane logowania podane przez użytkownika, sprawdzając czy istnieją w bazie danych. Następnie sprawdza rolę i przyznaje dostęp do odpowiednich funkcjonalności.
Warunki wstępne	Użytkownik podał prawidłowy adres mail i hasło
Warunki końcowe	Zalogowanie się do aplikacji.

W Tabeli 3.4 został opisany przypadek użycia wyświetlania użytkownikowi błędu logowania.

Tabela 3.4: Wyświetlanie błędu logowania

Identyfikator	PU-4
Nazwa	Wyświetlanie błędu logowania
Aktor	Użytkownik
Opis	Aplikacja po nieprawidłowej weryfikacji, wyświetla użytkownikowi błąd logowania. Użytkownik proszony jest o ponowne zalogowanie się do aplikacji.
Warunki wstępne	Błędna weryfikacja użytkownika.
Warunki końcowe	Użytkownik nie zostaje zalogowany do systemu.

W Tabeli 3.5 został opisany przypadek użycia przeglądania artykułów spożywczych przez użytkownika.

Tabela 3.5: Przeglądanie artykułów spożywczych

Identyfikator	PU-5
Nazwa	Przeglądanie artykułów spożywczych
Aktor	Użytkownik
Opis	Użytkownik wybiera na pasku nawigacji przycisk <i>Groceries</i> . Aplikacja wyświetla z bazy danych listę artykułów spożywczych. Wybór konkretnego produktu wyświetli szczegółowe informacje.
Warunki wstępne	Użytkownik zalogował się do aplikacji.
Warunki końcowe	Wyświetlenie listy artykułów spożywczych.

W Tabeli 3.6 został opisany przypadek użycia przeglądania przepisów przez użytkownika.

Tabela 3.6: Przeglądanie przepisów

Identyfikator	PU-6
Nazwa	Przeglądanie przepisów
Aktor	Użytkownik
Opis	Użytkownik wybiera na pasku nawigacji przycisk <i>Recipes</i> . Aplikacja wyświetla z bazy danych listę przepisów. Wybór konkretnego przepisu wyświetli szczegółowe informacje.
Warunki wstępne	Użytkownik zalogował się do aplikacji.
Warunki końcowe	Wyświetlenie listy przepisów

W Tabeli 3.7 został opisany przypadek użycia przeglądania kalendarza przez użytkownika.

Tabela 3.7: Przeglądanie kalendarza

Identyfikator	PU-7
Nazwa	Przeglądanie kalendarza
Aktor	Użytkownik
Opis	Użytkownik naciska na logo znajdujące się na pasku nawigacji. Zostaje przekierowany na stronę główną aplikacji, gdzie wyświetlany jest kalendarz. Przyciski < lub > pozwalają użytkownikowi zmieniać miesiąc, o jeden do tyłu lub do przodu. Wybranie konkretnego dnia wyświetla szczegóły harmonogramu żywieniowego oraz zakupowego przypisane do tego dnia.
Warunki wstępne	Użytkownik zalogował się do aplikacji.
Warunki końcowe	Wyświetlenie kalendarza.

W Tabeli 3.8 został opisany przypadek użycia dodawania posiłku do kalendarza użytkownika.

Tabela 3.8: Dodawanie posiłku do kalendarza

Identyfikator	PU-8
Nazwa	Dodawanie posiłku do kalendarza
Aktor	Użytkownik
Opis	Użytkownik po wybraniu konkretnego przepisu, ustawia date spożycia posiłku i naciska przycisk <i>Add to calendar</i> . Aplikacja dodaje przepis do harmonogramu żywieniowego oraz zakupowego.
Warunki wstępne	Użytkownik wybrał konkretny przepis z listy.
Warunki końcowe	Posiłek zostaje zapisany w kalendarzu.

W Tabeli 3.9 został opisany przypadek użycia dodawania zdjęcia posiłku przez użytkownika.

Tabela 3.9: Dodawanie zdjęcia posiłku

Identyfikator	PU-9
Nazwa	Dodawanie zdjęcia posiłku
Aktor	Użytkownik
Opis	Użytkownik wybiera konkretny dzień z kalendarza. Aplikacja wyświetla szczegółowy widok harmonogramu żywieniowego dla wybranego dnia. Użytkownik naciska na <i>Select file</i> i wybiera plik ze zdjęciem potrawy. Zapisanie następuje po naciśnięciu przycisku <i>Send</i> .
Warunki wstępne	Użytkownik zapisał danie w harmonogramie żywieniowym.
Warunki końcowe	Dodanie zdjęcia posiłku.

W Tabeli 3.10 został opisany przypadek użycia oceny posiłku przez użytkownika.

Tabela 3.10: Ocena posiłku

Identyfikator	PU-10
Nazwa	Ocena posiłku
Aktor	Użytkownik
Opis	Użytkownik wybiera konkretny dzień z kalendarza. Aplikacja wyświetla szczegółowy widok harmonogramu żywieniowego dla wybranego dnia. Użytkownik ocenia posiłek zaznaczając gwiazdki w skali od 0 do 10.
Warunki wstępne	Użytkownik zapisał danie w harmonogramie żywieniowym.
Warunki końcowe	Dodanie oceny posiłku.

W Tabeli 3.11 został opisany przypadek użycia usuwania zdjęcia posiłku przez użytkownika.

Tabela 3.11: Usuwanie zdjęcia posiłku

Identyfikator	PU-11
Nazwa	Usuwanie zdjęcia posiłku
Aktor	Użytkownik
Opis	Użytkownik wybiera konkretny dzień z kalendarza. Aplikacja wyświetla szczegółowy widok harmonogramu żywieniowego dla wybranego dnia. Użytkownik naciska na <i>X</i> i zdjęcie zostaje usunięte
Warunki wstępne	Użytkownik dodał zdjęcie posiłku.
Warunki końcowe	Usunięcie zdjęcia.

W Tabeli 3.12 został opisany przypadek użycia dodawania komentarza posiłku przez użytkownika.

Tabela 3.12: Dodawanie komentarza posiłku

Identyfikator	PU-12
Nazwa	Dodawanie komentarza posiłku
Aktor	Użytkownik
Opis	Użytkownik wybiera konkretny dzień z kalendarza. Aplikacja wyświetla szczegółowy widok harmonogramu żywieniowego dla wybranego dnia. Użytkownik w polu tekstowym komentuje posiłek. Naciska przycisk <i>Send</i> , który zapisuje komentarz.
Warunki wstępne	Użytkownik zapisał danie w harmonogramie żywieniowym.
Warunki końcowe	Dodanie komentarza posiłku.

W Tabeli 3.13 został opisany przypadek użycia usuwania komentarza posiłku przez użytkownika.

Tabela 3.13: Usuwanie komentarza posiłku

Identyfikator	PU-13
Nazwa	Usuwanie komentarza posiłku
Aktor	Użytkownik
Opis	Użytkownik wybiera konkretny dzień z kalendarza. Aplikacja wyświetla szczegółowy widok harmonogramu żywieniowego dla wybranego dnia. Użytkownik naciska przycisk <i>Remove opinion</i> i usuwa komentarz posiłku.
Warunki wstępne	Użytkownik dodał komentarz posiłku.
Warunki końcowe	Usunięcie komentarza posiłku.

W Tabeli 3.14 został opisany przypadek użycia usuwania posiłku z kalendarza.

Tabela 3.14: Usuwanie posiłku z kalendarza

Identyfikator	PU-14
Nazwa	Usuwanie posiłku z kalendarza
Aktor	Użytkownik
Opis	Użytkownik wybiera konkretny dzień z kalendarza. Aplikacja wyświetla szczegółowy widok harmonogramu żywieniowego dla wybranego dnia. Użytkownik naciska przycisk <i>Remove meal</i> usuwając danie z kalendarza.
Warunki wstępne	Użytkownik zapisał danie w harmonogramie żywieniowym.
Warunki końcowe	Usunięcie posiłku z kalendarza.

W Tabeli 3.15 został opisany przypadek użycia tworzenia listy zakupowej przez użytkownika.

Tabela 3.15: Tworzenie listy zakupowej

Identyfikator	PU-15
Nazwa	Tworzenie listy zakupowej
Aktor	Użytkownik
Opis	Użytkownik wybiera na pasku nawigacji przycisk <i>Shopping list</i> . Aplikacja przekierowuje go do widoku z listami zakupowymi, gdzie naciska na przycisk <i>+</i> aby rozpocząć tworzenie nowej listy zakupowej. Następnie użytkownik wypełnia formularz, gdzie ustawia nazwę listy, datę i godzinę zakupów, kolor karty oraz dodaje artykuły spożywcze. Naciskając na <i>Save</i> zapisuje listę, lub rezygnuje z tworzenia listy wybierając przycisk <i>Cancel</i> .
Warunki wstępne	Użytkownik zalogował się do aplikacji.
Warunki końcowe	Utworzenie listy zakupowej.

W Tabeli 3.16 został opisany przypadek użycia edytowania listy zakupowej przez użytkownika.

Tabela 3.16: Edytowanie listy zakupowej

Identyfikator	PU-16
Nazwa	Edytowanie listy zakupowej
Aktor	Użytkownik
Opis	Użytkownik wybiera na pasku nawigacji przycisk <i>Shopping list</i> . Aplikacja przekierowuje go do widoku list zakupowych, gdzie wybiera listę zakupową do edycji i naciska przycisk z ikoną edycji, który przenosi użytkownika do formularza edycji. Następnie użytkownik modyfikuje formularz, gdzie zmienia nazwę listy, datę i godzinę zakupów, kolor karty oraz artykuły spożywcze. Naciskając na <i>Save</i> zapisuje zmiany. W przeciwnym wypadku naciska przycisk <i>Cancel</i> odrzucając zmiany.
Warunki wstępne	Użytkownik utworzył listę zakupową.
Warunki końcowe	Zaktualizowanie listy zakupowej.

W Tabeli 3.17 został opisany przypadek użycia usuwania listy zakupowej przez użytkownika.

Tabela 3.17: Usuwanie listy zakupowej

Identyfikator	PU-17
Nazwa	Usuwanie listy zakupowej
Aktor	Użytkownik
Opis	Użytkownik wybiera na pasku nawigacji przycisk <i>Shopping list</i> . Aplikacja przekierowuje go do widoku list zakupowych, gdzie wybiera listę zakupową do usunięcia i naciska przycisk <i>X</i> , który usuwa listę zakupową.
Warunki wstępne	Użytkownik utworzył listę zakupową.
Warunki końcowe	Usunięcie listy zakupowej.

W Tabeli 3.18 został opisany przypadek użycia dodawania artykułów spożywczych do listy zakupowej.

Tabela 3.18: Dodawanie artykułów spożywczych do listy zakupowej

Identyfikator	PU-18
Nazwa	Dodawanie artykułów spożywczych do listy zakupowej
Aktor	Użytkownik
Opis	Użytkownik wybiera na pasku nawigacji przycisk <i>Shopping list</i> . Aplikacja przekierowuje go do widoku list zakupowych, gdzie wybiera listę zakupową do edycji i naciska przycisk z ikoną edycji, który przenosi użytkownika do formularza edycji. Następnie użytkownik naciska przycisk <i>Add Product</i> po czym w formularzu pojawia się nowe pole, w którym może wybrać produkt, ilość oraz jednostkę. Naciskając na <i>Save</i> zapisuje zmiany. W przeciwnym wypadku naciska przycisk <i>Cancel</i> odrzucając zmiany.
Warunki wstępne	Użytkownik utworzył listę zakupową.
Warunki końcowe	Dodanie artykułów spożywczych do listy zakupowej.

W Tabeli 3.19 został opisany przypadek użycia usuwania artykułów spożywczych z listy zakupowej.

Tabela 3.19: Usuwanie artykułów spożywczych z listy zakupowej

Identyfikator	PU-19
Nazwa	Usuwanie artykułów spożywczych z listy zakupowej
Aktor	Użytkownik
Opis	Użytkownik wybiera na pasku nawigacji przycisk <i>Shopping list</i> . Aplikacja przekierowuje go do widoku list zakupowych, gdzie wybiera listę zakupową do edycji i naciska przycisk z ikoną edycji, który przenosi użytkownika do formularza edycji. Następnie użytkownik naciska przycisk <i>X</i> przy produkcie, który ma zostać usunięty. Naciskając na <i>Save</i> zapisuje zmiany. W przeciwnym wypadku naciska przycisk <i>Cancel</i> odrzucając zmiany.
Warunki wstępne	Użytkownik utworzył listę zakupową.
Warunki końcowe	Usunięcie artykułów spożywczych z listy zakupowej

W Tabeli 3.20 został opisany przypadek użycia dodawania celu żywieniowego.

Tabela 3.20: Dodawanie celu żywieniowego

Identyfikator	PU-20
Nazwa	Dodawanie celu żywieniowego
Aktor	Użytkownik
Opis	Użytkownik wybiera na pasku nawigacji przycisk <i>Nutritional goals</i> . Aplikacja przekierowuje go do widoku z celami żywieniowymi, gdzie naciska na przycisk <i>+</i> aby rozpocząć tworzenie celu żywieniowego. Następnie użytkownik wypełnia formularz, gdzie ustawia nazwę celu, typ docelową wagę, obwód brzucha, procent tkanki tłuszczowej i datę rozpoczęcia celu. Aplikacja oblicza szacowany czas trawienia celu i ustawia datę ukończenia celu. Naciskając na <i>Save</i> zapisuje cel, lub rezygnuje z tworzenia celu wybierając przycisk <i>Cancel</i> .
Warunki wstępne	Użytkownik zalogował się do aplikacji.
Warunki końcowe	Dodanie celu żywieniowego.

W Tabeli 3.21 został opisany przypadek użycia usuwania celu żywieniowego.

Tabela 3.21: Usuwanie celu żywieniowego

Identyfikator	PU-21
Nazwa	Usuwanie celu żywieniowego.
Aktor	Użytkownik
Opis	Użytkownik wybiera na pasku nawigacji przycisk <i>Nutritional goals</i> . Aplikacja przekierowuje go do widoku z celami żywieniowymi, gdzie naciska na przycisk <i>X</i> aby usunąć cel żywieniowy.
Warunki wstępne	Użytkownik dodał cel żywieniowy
Warunki końcowe	Usunięcie celu żywieniowego.

W Tabeli 3.22 został opisany przypadek użycia aktualizowania parametrów ciała użytkownika.

Tabela 3.22: Aktualizowanie parametrów ciała

Identyfikator	PU-22
Nazwa	Aktualizowanie parametrów ciała
Aktor	Użytkownik
Opis	Użytkownik wybiera na pasku nawigacji przycisk <i>Profile</i> . Aplikacja przekierowuje go do widoku z informacjami o użytkowniku, gdzie naciska na przycisk <i>Update body parameters</i> . Aplikacja wyświetla użytkownikowi formularz, gdzie podaje swoją wagę, wzrost, obwód brzucha, procent tkanki tłuszczowej i aktywność fizyczną. Naciskając na <i>Save</i> zapisuje zmiany, lub z nich rezygnuje wybierając przycisk <i>Cancel</i> .
Warunki wstępne	Użytkownik zalogował się do aplikacji.
Warunki końcowe	Aktualizacja parametrów ciała użytkownika.

W Tabeli 3.23 został opisany przypadek użycia przeglądania historii parametrów ciała.

Tabela 3.23: Przeglądanie historii parametrów ciała

Identyfikator	PU-23
Nazwa	Przeglądanie historii parametrów ciała
Aktor	Użytkownik
Opis	Użytkownik wybiera na pasku nawigacji przycisk <i>Profile</i> . Aplikacja przekierowuje go do widoku z informacjami o użytkowniku, gdzie naciska na przycisk <i>Body parameters history</i> . Aplikacja wyświetla użytkownikowi listę wszystkich pomiarów parametrów ciała użytkownika.
Warunki wstępne	Użytkownik zalogował się do aplikacji.
Warunki końcowe	Wyświetlenie historii parametrów ciała.

W Tabeli 3.24 został opisany przypadek użycia dodawania przepisu przez administratora.

Tabela 3.24: Dodawanie przepisu

Identyfikator	PU-24
Nazwa	Dodawanie przepisu
Aktor	Administrator
Opis	Administrator wybiera na pasku nawigacji przycisk <i>Admin panel</i> . Aplikacja przekierowuje go do panelu admina, gdzie wybiera opcję <i>Manage recipes</i> , która przenosi go do widoku, w którym może zarządzać przepisami. Naciska przycisk <i>Add recipe</i> , który wyświetla formularz tworzenia przepisu. Administrator ustawia tytuł, typ posiłku, adres URL(ang. Uniform Resource Locator) zdjęcia przepisu, opis, metode przygotowania oraz dodaje składniki przepisu. Zapisanie przepisu następuje po naciśnięciu przycisku <i>Save</i> . Przycisk <i>Cancel</i> anuluje dodanie przepisu.
Warunki wstępne	Administrator zalogował się do aplikacji.
Warunki końcowe	Dodanie przepisu.

W Tabeli 3.25 został opisany przypadek użycia edytowania przepisu przez administratora.

Tabela 3.25: Edytowanie przepisu

Identyfikator	PU-25
Nazwa	Edytowanie przepisu
Aktor	Administrator
Opis	Administrator wybiera na pasku nawigacji przycisk <i>Admin panel</i> . Aplikacja przekierowuje go do panelu admina, gdzie wybiera opcję <i>Manage recipes</i> , która przenosi go do widoku, w którym może zarządzać przepisami. Wybiera konkretny przepis i naciska przycisk <i>Edit</i> , który wyświetla formularz edycji przepisu. Administrator zmienia tytuł, typ posiłku, adres URL zdjęcia przepisu, opis, metode przygotowania oraz dodaje składniki przepisu. Zapisanie zmian następuje po naciśnięciu przycisku <i>Save</i> . Przycisk <i>Cancel</i> anuluje zmiany.
Warunki wstępne	Administrator zalogował się do aplikacji.
Warunki końcowe	Edytowanie przepisu.

W Tabeli 3.26 został opisany przypadek użycia usuwania przepisu przez administratora.

Tabela 3.26: Usuwanie przepisu

Identyfikator	PU-26
Nazwa	Usuwanie przepisu
Aktor	Administrator
Opis	Administrator wybiera na pasku nawigacji przycisk <i>Admin panel</i> . Aplikacja przekierowuje go do panelu admina, gdzie wybiera opcję <i>Manage recipes</i> , która przenosi go do widoku, w którym może zarządzać przepisami. Wybiera konkretny przepis i naciska na przycisk <i>X</i> usuwając przepis.
Warunki wstępne	Administrator zalogował się do aplikacji.
Warunki końcowe	Usunięcie przepisu.

W Tabeli 3.27 został opisany przypadek użycia dodawania artykułu spożywczego przez administratora.

Tabela 3.27: Dodawanie artykułu spożywczego

Identyfikator	PU-27
Nazwa	Dodawanie artykułu spożywczego
Aktor	Administrator
Opis	Administrator wybiera na pasku nawigacji przycisk <i>Admin panel</i> . Aplikacja przekierowuje go do panelu admina, gdzie wybiera opcję <i>Manage products</i> , która przenosi go do widoku, w którym może zarządzać produktami. Naciska przycisk <i>Add product</i> , który wyświetla formularz tworzenia artykułu. Administrator ustawia nazwę, wagę, jednostkę, cenę, adres URL zdjęcia produktu, liczbę kalorii i wartości odżywcze artykułu spożywczego. Zapisanie produktu następuje po naciśnięciu przycisku <i>Save</i> . Przycisk <i>Cancel</i> anuluje dodanie produktu.
Warunki wstępne	Administrator zalogował się do aplikacji.
Warunki końcowe	Dodanie artykułu spożywczego.

W Tabeli 3.28 został opisany przypadek użycia edytowania artykułu spożywczego przez administratora.

Tabela 3.28: Edytowanie artykułu spożywczego

Identyfikator	PU-28
Nazwa	Edytowanie artykułu spożywczego
Aktor	Administrator
Opis	Administrator wybiera na pasku nawigacji przycisk <i>Admin panel</i> . Aplikacja przekierowuje go do panelu admina, gdzie wybiera opcję <i>Manage products</i> , która przenosi go do widoku, w którym może zarządzać produktami. Wybiera konkretny artykuł i naciska przycisk <i>Edit</i> , który wyświetla formularz edycji produktu. Administrator zmienia nazwę, wagę, jednostkę, cenę, adres URL zdjęcia produktu, liczbę kalorii i wartości odżywcze. Zapisanie zmian następuje po naciśnięciu przycisku <i>Save</i> . Przycisk <i>Cancel</i> anuluje zmiany.
Warunki wstępne	Administrator zalogował się do aplikacji.
Warunki końcowe	Edytowanie artykułu spożywczego.

W Tabeli 3.29 został opisany przypadek użycia usuwania artykułu spożywczego przez administratora.

Tabela 3.29: Usuwanie artykułu spożywczego

Identyfikator	PU-29
Nazwa	Usuwanie artykułu spożywczego
Aktor	Administrator
Opis	Administrator wybiera na pasku nawigacji przycisk <i>Admin panel</i> . Aplikacja przekierowuje go do panelu admina, gdzie wybiera opcję <i>Manage products</i> , która przenosi go do widoku, w którym może zarządzać przepisami. Wybiera konkretny produkt i naciska na przycisk <i>X</i> usuwając artykuł spożywczy.
Warunki wstępne	Administrator zalogował się do aplikacji.
Warunki końcowe	Usunięcie artykułu spożywczego.

3.2 Narzędzia

Poniżej przedstawiono opis narzędzi użytych podczas tworzenia aplikacji. Omówiono ich funkcjonalność, rolę w projekcie oraz wpływ na rozwój aplikacji. W tym podrozdziale opisano biblioteki, frameworki oraz oprogramowanie, które wspierają zarządzanie bazami danych, projektowanie interfejsu użytkownika, a także testowanie i zarządzanie aplikacją.

3.2.1 Spring Framework

Spring Framework to framework do tworzenia aplikacji webowych oparty na języku Java [9]. Został wykorzystany do stworzenia warstwy backend aplikacji. Odpowiada za zarządzanie logiką biznesową i bazą danych oraz obsługuje żądania klienta. Framework dostarcza narzędzia ułatwiające uruchomienie aplikacji. Obsługuje technologie JPA (ang. Java Persistence API) i Hibernate, które zarządzają bazą danych. Dostarcza interfejsy, które posiadają wbudowane metody do zarządzania encjami. Zarządza uwierzytelnianiem i autoryzacją. Wykorzystanie Spring Framework znacząco przyspieszyło proces tworzenia warstwy backend aplikacji. Poprawiło czytelność oraz skalowalność kodu.

3.2.2 MySql

MySQL to system zarządzania bazą danych. Umożliwia przechowywanie, modyfikacje, usuwanie i wyszukiwanie danych. Dane są trzymane w tabelach, które mogą być powiązane relacyjnie za pomocą kluczy głównych i obcych. MySQL zapewnia obsługę transakcji oraz optymalizację zapytań SQL (ang. Structured Query Language) [7]. Wykorzystanie MySQL wpłynęło pozytywnie na integralność danych, bezpieczeństwo oraz skalowalność aplikacji, umożliwiając efektywne zarządzanie dużymi zbiorami danych oraz ich integrację z frameworkiem Spring.

3.2.3 Angular

Angular to framework do tworzenia SPA (ang. Single Page Application). Wspierany jest przez Google oraz oparty na języku TypeScript [8]. SPA zawiera tylko jeden plik HTML (ang. HyperText Markup Language) i dynamicznie zmienia zawartość aplikacji bez konieczności przeładowywania strony. Angular zapewnia dwukierunkowe wiązanie danych, które skutkuje synchronizacją danych między modelem TypeScript a widokiem HTML. Angular pozwala również na tworzenie komponentów, które mają za zadanie powiązać pliki CSS (ang. Cascading Style Sheets), HTML i TypeScript w jeden element, co ułatwia jego ponowne wykorzystanie. Posiada wbudowany mechanizm routingu umożliwia zarządzanie nawigacją pomiędzy widokami oraz mechanizm obsługi żądań HTTP (ang. Hypertext Transfer Protocol). Praca z pomocą frameworku Angular w sposób znaczący przyspieszyła tworzenie warstwy frontend aplikacji. Dodatkowo użycie języka TypeScript, który posiada system typów statycznych, poprawiło jakość kodu poprzez eliminację błędów na etapie kompilacji.

3.2.4 Bootstrap

Bootstrap to front-endowa biblioteka CSS. Zawierająca zestaw gotowych komponentów, takich jak przyciski, formularze, paski postępu i inne. Automatycznie dostosowuje

zawartość aplikacji do rozmiarów ekranu monitora lub urządzenia mobilnego. Bootstrap w aplikacji skraca czas tworzenia interfejsu użytkownika. Dodatkowo Bootstrap doskonale współpracuje z frameworkiem Angular.

3.2.5 Docker

Docker to otwarte oprogramowanie służące do tworzenia, uruchamiania i zarządzania aplikacjami w kontenerach. Kontenery to lekkie, samowystarczalne środowiska posiadające tylko niezbędne pliki do działania aplikacji. Kontenery są zarządzane przez maszynę wirtualną programu Docker, która zapewnia izolację oraz zarządzanie zasobami systemowymi. Aplikacje są opakowane w obrazy, które są tworzone zgodnie z plikiem Dockerfile. Użycie Dockera pozwoliło na uruchomienie warstw frontend, backend i bazy danych w osobnych kontenerach. Takie podejście w pozytywny sposób wpłynęło na elastyczność oraz przenośność aplikacji.

3.2.6 Postman

Postman to narzędzie do testowania API napisane w stylu REST. Pozwala na tworzenie i wysyłanie zapytań HTTP do serwera takich jak GET, POST, PUT i DELETE. Umożliwia grupowanie zapytań i automatyzację testów. Praca z Postmanem zwiększyła efektywność tworzenia endpointów, zapewniając, że dany endpoint działa prawidłowo.

3.3 Metodyka pracy nad projektem

W ramach wstępnych działań nad projektem określony został problem, który aplikacja ma rozwiązać. Zidentyfikowano także docelową grupę użytkowników. Przeprowadzono analizę potrzeb i wymagań funkcjonalnych, mających na celu określenie oczekiwań użytkowników wobec aplikacji. Następnie opracowano ogólną strukturę projektu. Kolejnym krokiem było zidentyfikowanie przypadków użycia i opracowanie modelu klas oraz modelu bazy danych. Po zakończeniu projektowania aplikacji przystąpiono do implementacji. Podczas tworzenia aplikacji poświęcono szczególną uwagę na jakość kodu oraz na zgodność z wymaganiami projektu. Po zakończeniu implementacji przeprowadzono szczegółowe testy, w tym testowanie punktów końcowych przy użyciu narzędzia Postman, testy jednostkowe, integralne oraz testy akceptacyjne.

Rozdział 4

Specyfikacja zewnętrzna

4.1 Zagadnienia administracyjne

W bieżącym podrozdziale omówiono wymagania sprzętowe i systemowe niezbędne do poprawnego działania opracowanej aplikacji. Odniesiono się zarówno do wymagań sprzętowych, jak i systemowych.

4.1.1 Wymagania sprzętowe

Wymagania sprzętowe określono na podstawie dokumentacji oprogramowania Docker i MySQL. Zarówno wymagania minimalne, jak i rekomendowane określono, biorąc pod uwagę najwyższe wartości podane dla każdego ze wspomnianych programów.

Minimalne wymagania sprzętowe :

- Procesor: 2-rdzeniowy, wspierający wirtualizację.
- Pamięć RAM: 4 GB.

Rekomendowane wymagania sprzętowe:

- Procesor: 4-rdzeniowy, wspierający wirtualizację.
- Pamięć RAM: 8 GB.

4.1.2 Wymagania systemowe

Ze względu na użycie programu Docker, opracowana aplikacja działa na różnych platformach systemów operacyjnych; są one następujące:

- Windows 10/11 (z dodatkiem WSL, ang. Windows Subsystem for Linux),
- Linux – dystrybucja Ubuntu,

- MacOS .

Niezależnie od systemu operacyjnego w kontenerach Docker powinno zostać uruchomione następujące oprogramowanie:

- Java Development Kit (JDK): Wersja 17 (zalecane OpenJDK),
- Node.js: Wersja 20.5.0 lub nowsza,
- Docker Desktop: 4.29.0 lub nowsza (host),
- MySQL Server: Wersja 8.x lub nowsza,
- Angular CLI: Wersja 16.x lub nowsza.

4.1.3 Konfiguracja serwera

Instalacja dla systemu operacyjnego Windows 10/11. Pierwszym krokiem konfiguracji dla systemu Windows jest pobranie instalatora Docker dla systemu Windows znajdującego się na stronie głównej tegoż programu. Następnie należy uruchomić plik "Docker Desktop Installer.exe"z instalatorem i postępować zgodnie z instrukcjami kreatora instalacji. Kolejnym etapem jest uruchomienie aplikacji Docker Desktop. Przed uruchomieniem tej aplikacji należy sprawdzić w ustawieniach BIOS (ang. Basic Input/Output System) komputera, czy procesor ma włączoną wirtualizację. Następnie należy otworzyć program PowerShell z uprawnieniami administratora i przejść do folderu, w którym znajduje się plik "compose.yaml"używając komendy "cd <ścieżka do pliku>"i wykonać komendę "docker-compose up -d", która uruchomi kontener z aplikacją.

Instalacja dla systemu operacyjnego MacOS. Pierwszym krokiem jest pobranie instalatora dla systemu MacOS znajdującego się na stronie głównej programu Docker. Następnie należy uruchomić plik "Docker.dmg"z instalatorem i postępować zgodnie z instrukcjami kreatora instalacji. Następnie należy otworzyć terminal i przejść do folderu, w którym znajduje się plik "compose.yaml"używając do tego polecenia "cd <ścieżka do pliku>"i wykonać komendę "docker-compose up -d", startującą kontener.

Instalacja dla systemu operacyjnego Linux Ubuntu. Pierwszym krokiem jest uruchomienie terminala i użycie komendy "sudo apt-get update"oraz "sudo apt-get install ./docker-desktop-amd64.deb", zainstaluje to program Docker. Następnie należy przejść do folderu, w którym znajduje się plik "compose.yaml"używając polecenia "cd <ścieżka do pliku>"i wykonać komendę "docker-compose up -d", która uruchamia kontener.

Oprogramowanie Docker samo skonfiguruje wszystkie niezbędne zależności i pliki potrzebne do prawidłowego działania aplikacji. W celu wyświetlenia zawartości aplikacji należy otworzyć przeglądarkę internetową i wpisać w pasku wyszukiwania "localhost:80".

4.1.4 Wstępna konfiguracja systemu

Baza danych powstaje automatycznie z mapowania obiektowo-relacyjnego, klasy z adnotacją @Entity są konwertowane na tabele bazodanowe przez Java Persistence API i Hibernate, dzięki czemu zarządzanie bazą odbywa się za pomocą warstwy backend. Użytkownik może skonfigurować w pliku "compose.yml" zmienne środowiskowe :

'SPRING_DATASOURCE_PASSWORD=' oraz 'MYSQL_ROOT_PASSWORD='

które zmieniają hasło do bazy danych.

4.2 Użytkowanie aplikacji

W tym podrozdziale opisano role użytkowników określając ich uprawnienia (por. rozdz. 3). Szczegółowo opisano instrukcje obsługi aplikacji, która pozwoli na efektywne korzystanie z systemu.

4.2.1 Role użytkowników

Użytkownik niezarejestrowany. Użytkownik nieposiadający konta; ma on ograniczony dostęp do funkcjonalności i zasobów systemu. Aplikacja udostępnia użytkownikowi jedynie widok autoryzacji, umożliwiając rejestrację lub zalogowanie się do serwisu. Użytkowanie pełnego dostępu do aplikacji przez użytkownika niezarejestrowanego następuje po pomyślnym przejściu etapu rejestracji.

Użytkownik. Osoba z utworzonym kontem oraz zalogowana w systemie. Posiada dostęp do większości funkcjonalności aplikacji. Aplikacja pozwala Użytkownikowi na wyszukiwanie lub przeglądanie przepisów i artykułów spożywczych oraz na zapisanie przepisu w harmonogramie. Serwis udostępnia kalendarz, który wyświetla harmonogramy żywieniowy i zakupowy po wybraniu konkretnego dnia. Dodatkowo w harmonogramie zakupowym istnieje opcja dodania zdjęcia gotowej potrawy oraz oceny i stosownego komentarza. Użytkownik ma możliwość tworzenia list zakupowych z istniejących artykułów spożywczych. Produkty na liście można dodawać, usuwać, a także zaznaczać lub odznaczać. W celu zapewnienia szybkiego dostępu do listy, Użytkownik może dodać listę do tzw. ulubionych. Aplikacja udostępnia Użytkownikowi możliwość tworzenia celów żywieniowych, które pomagają zmniejszyć, zwiększyć lub utrzymać wagę. Użytkownikowi udostępniany jest podgląd profilu, gdzie wyświetlane są informacje, takie jak adres mail, płeć, data urodzenia i aktualne parametry, takie jak np. masa ciała. Dodatkowo aplikacja pozwala na aktualizację parametrów ciała oraz na przeglądanie historii dokonanych pomiarów ciała.

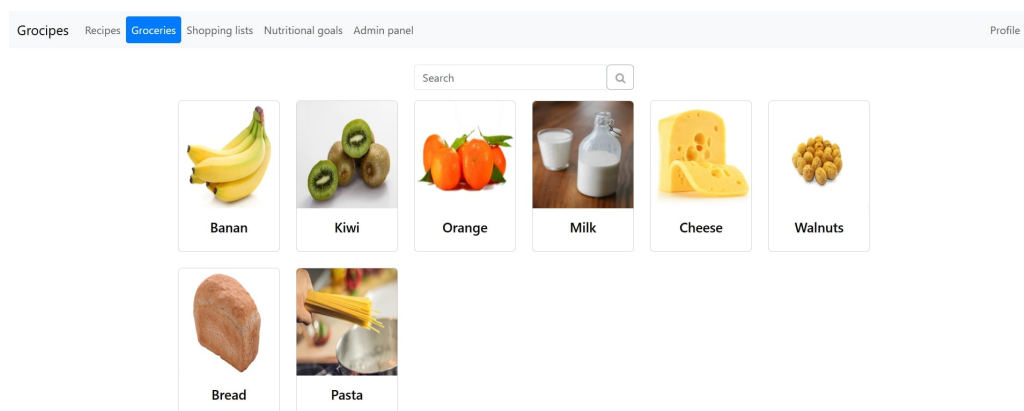
Administrator. Użytkownik posiadający dodatkowe uprawnienia pozwalające na zarządzanie przepisami i artykułami spożywczymi, które widzi Użytkownik. Administrator

może dodawać nowe przepisy i produkty, a także odpowiada za ich usuwanie i modyfikację zawartości.

4.2.2 Scenariusze działania systemu

W poniższym podrozdziale przedstawione zostały scenariusze działania systemu. W scenariuszach tych opisano sposób, w jaki użytkownicy mogą zrealizować dany przypadek użycia. Każdy scenariusz zilustrowano odpowiednimi zrzutami ekranów.

Przeglądanie artykułów spożywczych. Użytkownik loguje się do aplikacji. System po poprawnej weryfikacji wyświetla użytkownikowi stronę główną serwisu¹. Następnie wybiera na pasku nawigacji przycisk "Groceries", wyświetlający widok listy artykułów spożywczych w formie kart, co ilustruje zrzut ekranu przedstawiony na rysunku 4.1.




Rysunek 4.1: Zrzut ekranu przedstawiający widok listy artykułów spożywczych.

Naciśnięcie na kartę konkretnego artykułu spożywczego przenosi użytkownika do widoku, w którym są wyświetlane szczegółowe informacje, takie jak średnia cena, liczba kalorii produktu i wartości odżywcze na 100 gram, co ilustruje zrzut ekranu przedstawiony na rysunku 4.2.

¹W opisie scenariuszy założono, że każdy użytkownik rozpoczyna interakcję od zalogowania się do aplikacji.

Grocipes Recipes Groceries Shopping lists Nutritional goals Admin panel Profile

Kiwi1.89 PLN




Weight	80 g	
Nutritional values per 100g		
Calories	60 kcal	
Potassium	0.312 g	5 %
Protein	1.1 g	0 %
Carbohydrates	15 g	4 %
Sodium	0.003 g	0 %
Cholesterol	0 g	0 %
Fat	0.5 g	0 %

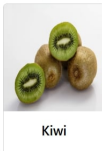
Rysunek 4.2: Zrzut ekranu przedstawiający widok szczegółów produktu.

Dodatkowo użytkownik w celu zaoszczędzenia czasu może wyszukać produkt z listy. Wpisuje nazwę interesującego go artykułu spożywczego lub część nazwy w pole wyszukiwania i naciska na przycisk z ikoną lupy. Ilustrują to zrzuty ekranu przedstawione na rysunku 4.3 i 4.4.

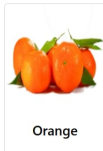
Grocipes Recipes Groceries Shopping lists Nutritional goals Admin panel Profile




Banan




Kiwi



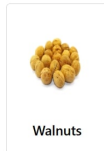
Orange




Milk




Cheese



Walnuts

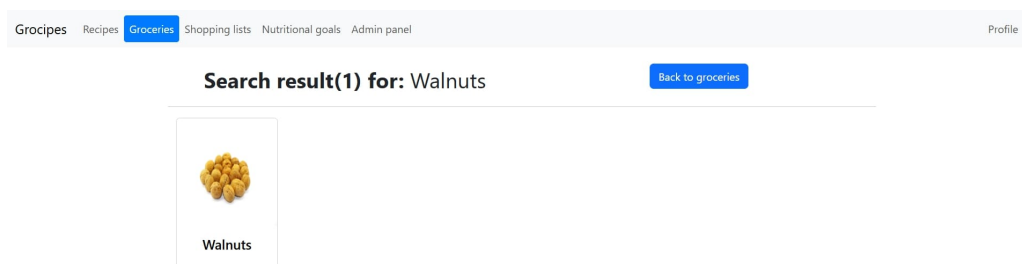


Bread



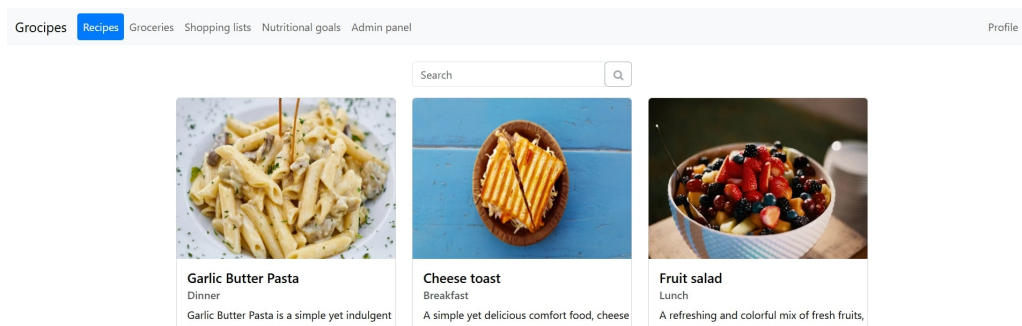
Pasta

Rysunek 4.3: Zrzut ekranu przedstawiający wyszukiwanie produktu.



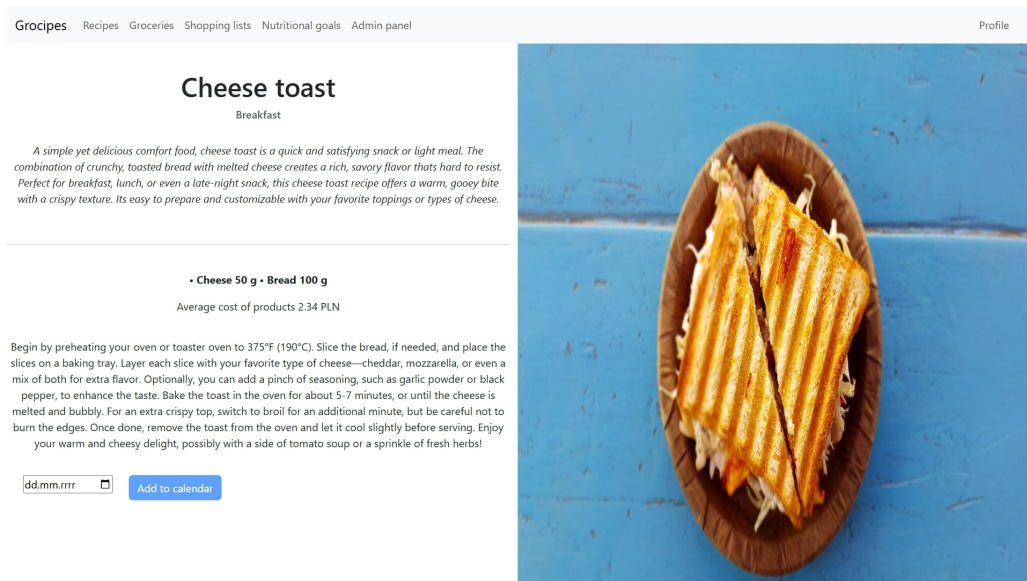
Rysunek 4.4: Zrzut ekranu przedstawiający wyszukanie produktu.

Przeglądanie przepisów. Użytkownik loguje się do aplikacji. System po poprawnej weryfikacji wyświetla użytkownikowi stronę główną serwisu. Następnie wybiera na pasku nawigacji przycisk "Recipes", wyświetlający widok listy z przepisami w formie kart, co ilustruje zrzut ekranu przedstawiony na rysunku 4.5.



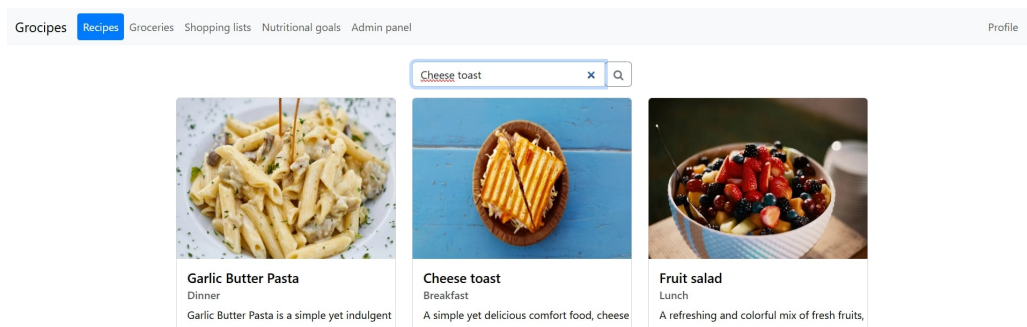
Rysunek 4.5: Zrzut ekranu przedstawiający widok listy z przepisami.

Naciśnięcie na kartę z konkretnym przepisem przenosi użytkownika do widoku, w którym są wyświetlane szczegółowe informacje, takie jak opis, składniki przepisu, średnia cena przygotowania dania oraz metoda przygotowania produktu, co ilustruje zrzut ekranu przedstawiony na rysunku 4.6.

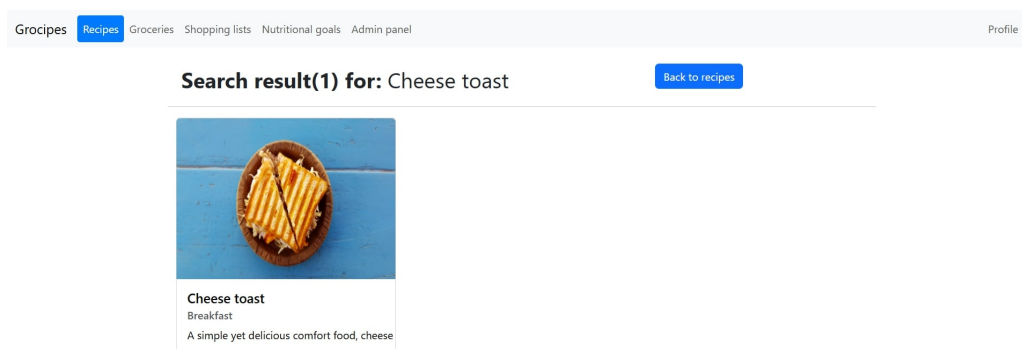


Rysunek 4.6: Zrzut ekranu przedstawiający widok szczegółu przepisu.

Dodatkowo użytkownik w celu zaoszczędzenia czasu może wyszukać przepis z listy. Wpisuje nazwę interesującego go dania lub część nazwy tego dania w pole wyszukiwania i naciska na przycisk z ikoną lupy. Ilustrują to zrzuty ekranu przedstawione na rysunku 4.7 i 4.8.

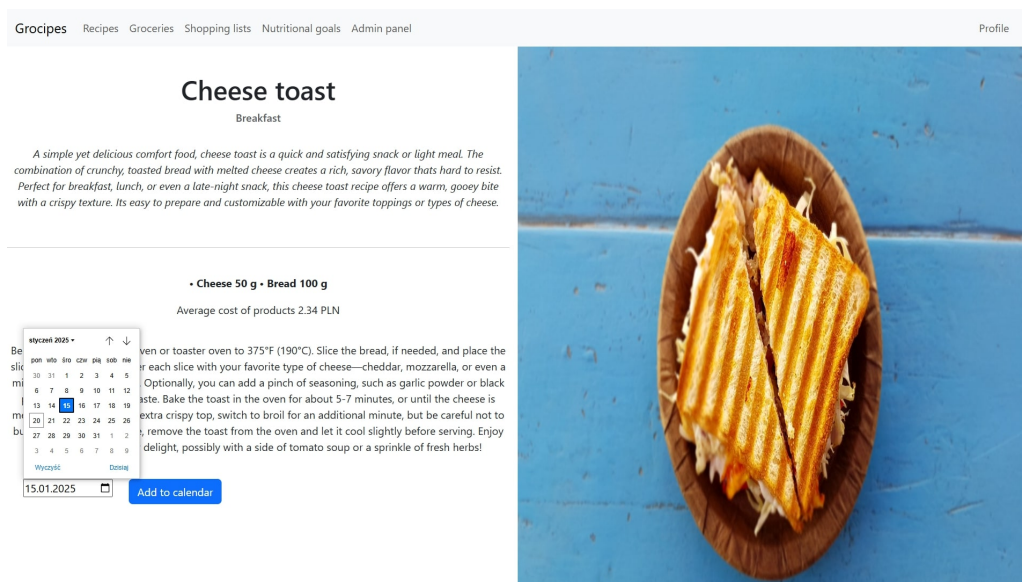


Rysunek 4.7: Zrzut ekranu przedstawiający wyszukiwanie przepisu.



Rysunek 4.8: Zrzut ekranu przedstawiający wyszukanie przepisu.

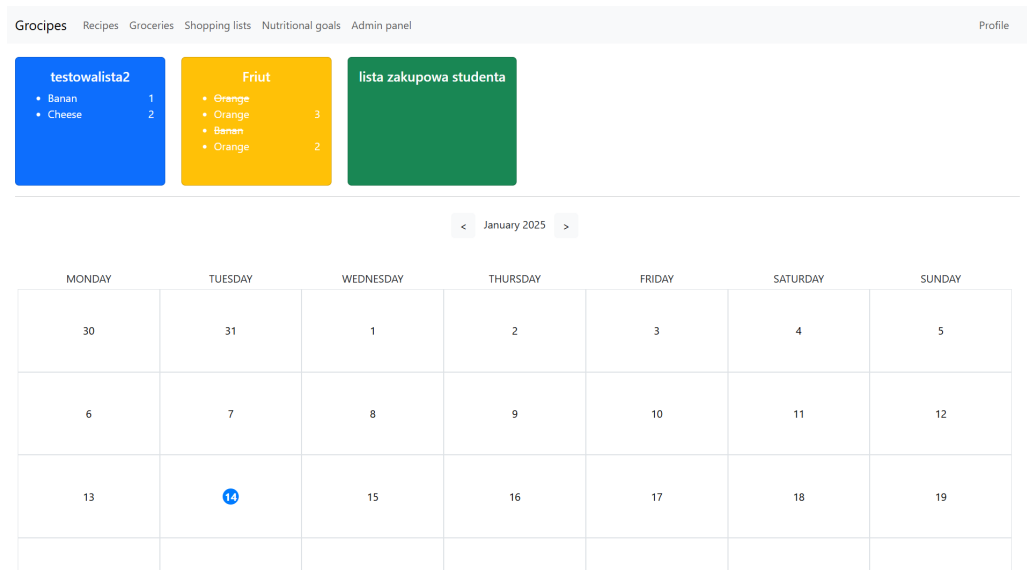
Dodawanie przepisu do harmonogramu żywieniowego i zakupowego. Użytkownik loguje się do aplikacji. System po poprawnej weryfikacji wyświetla użytkownikowi stronę główną serwisu. Następnie wybiera na pasku nawigacji przycisk "Recipes", wyświetlający widok listy z przepisami w formie kart. Użytkownik naciska na kartę z interesującym go przepisem, po czym aplikacja przenosi go do podglądu przepisu. Wybiera w selektorze daty termin spożycia posiłku, który znajduje się w lewym dolnym rogu i naciska na przycisk "Add to calendar". Ilustruje to zrzut ekranu przedstawiony na rysunku 4.9.



Rysunek 4.9: Zrzut ekranu przedstawiający dodawanie przepisu do harmonogramu żywieniowego i zakupowego.

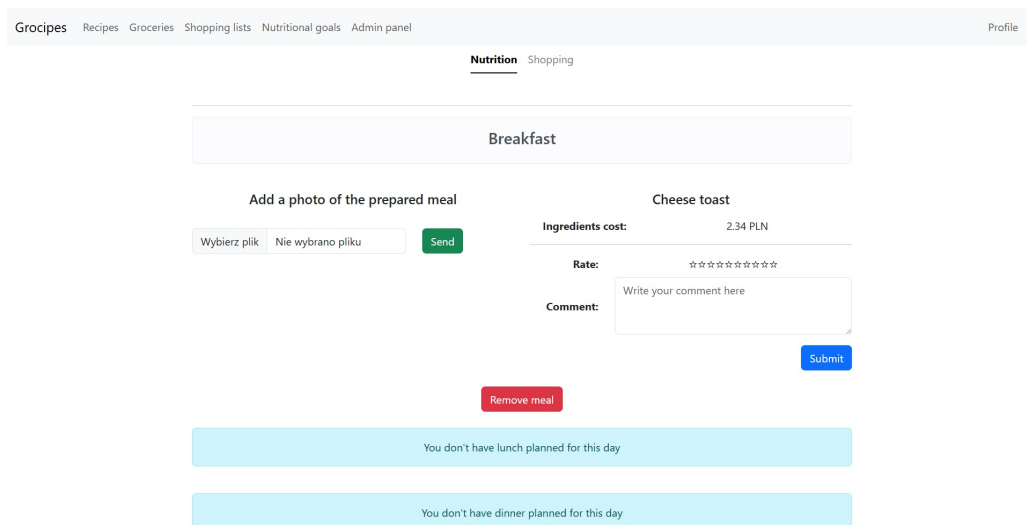
Przeglądanie harmonogramu żywieniowego i zakupowego. Użytkownik loguje się do aplikacji. System po poprawnej weryfikacji wyświetla użytkownikowi stronę główną

serwisu. Następnie klika na siatkę kalendarza, wybierając konkretny dzień, co ilustruje zrzut ekranu przedstawiony na rysunku 4.10.



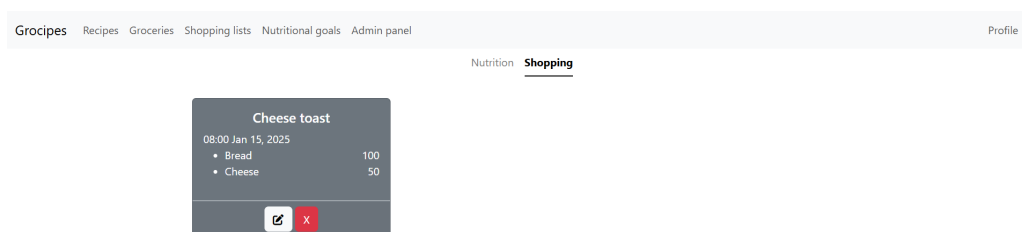
Rysunek 4.10: Zrzut ekranu przedstawiający widok strony głównej.

Po naciśnięciu aplikacja wyświetla użytkownikowi harmonogram żywieniowy, co ilustruje zrzut ekranu przedstawiony na rysunku 4.11. Harmonogram żywieniowy zawiera informacje o spożywanych posiłkach w wybranym dniu. Posiłki są podzielone na śniadanie, obiad i kolację. Każdy posiłek można ocenić, skomentować i dodać do niego zdjęcie gotowego posiłku. Wyświetlana jest także informacja o średniej cenie przygotowania posiłku i nazwie. Posiłek można usunąć z harmonogramu żywieniowego naciskając na przycisk "Remove meal".



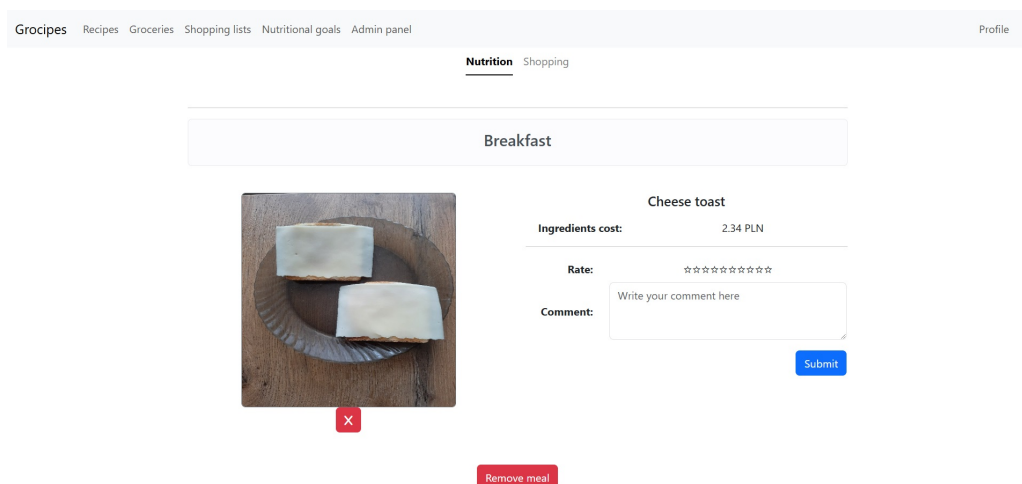
Rysunek 4.11: Zrzut ekranu przedstawiający widok harmonogramu żywieniowego.

Użytkownik może przełączyć widok na harmonogram zakupowy naciskając na zakładkę "Shopping", co ilustruje zrzut ekranu przedstawiony na rysunku 4.12. Harmonogram zakupowy wyświetla listy zakupowe użytkownika w wybranym dniu.



Rysunek 4.12: Zrzut ekranu przedstawiający widok harmonogramu zakupowego.

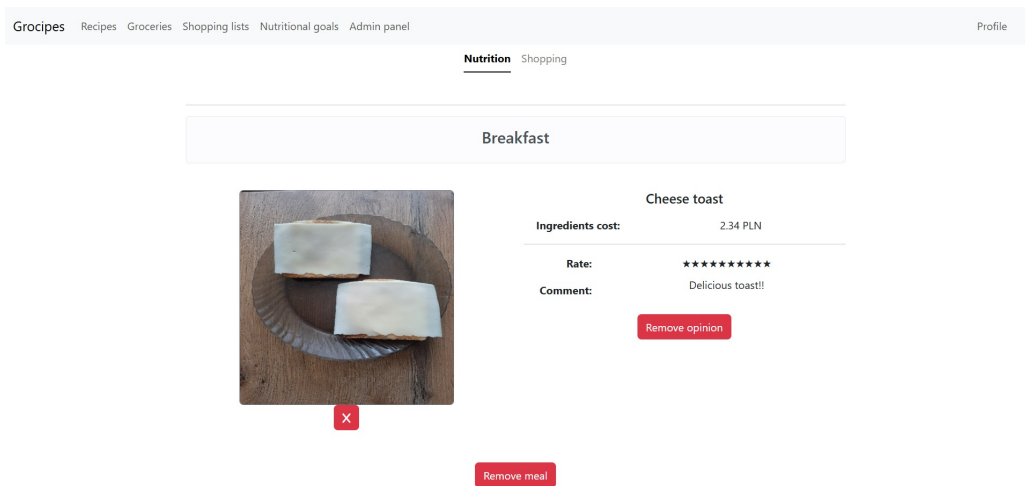
Dodanie zdjęcia gotowego posiłku w harmonogramie żywieniowym. Użytkownik loguje się do aplikacji. System po poprawnej weryfikacji wyświetla użytkownikowi stronę główną serwisu. Następnie klika na siatkę kalendarza wybierając konkretny dzień. Aplikacja wyświetla harmonogram żywieniowy, co ilustruje zrzut ekranu przedstawiony na rysunku 4.11. Użytkownik naciska przycisk "Wybierz plik", wybiera zdjęcie i naciska przycisk "Send" zapisując obraz w bazie danych, co ilustruje zrzut ekranu przedstawiony na rysunku 4.13. Dodatkowo użytkownik może usunąć zdjęcie naciskając na przycisk "X".



Rysunek 4.13: Zrzut ekranu przedstawiający dodanie zdjęcia gotowego posiłku.

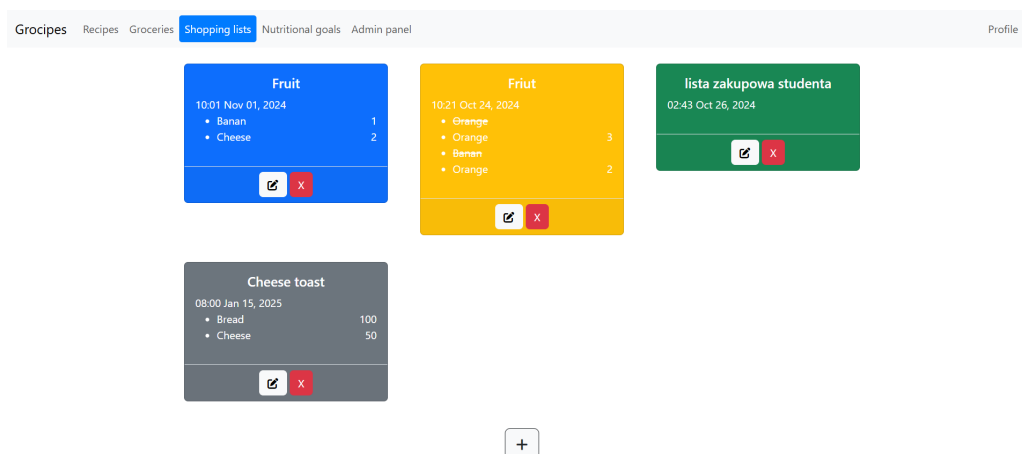
Dodanie oceny i komentarza przepisu w harmonogramie żywieniowym.

Użytkownik loguje się do aplikacji. System po poprawnej weryfikacji wyświetla użytkownikowi stronę główną serwisu. Następnie klika na siatkę kalendarza wybierając konkretny dzień. Aplikacja wyświetla mu harmonogram żywieniowy, co ilustruje zrzut ekranu przedstawiony na rysunku 4.11. Użytkownik ocenia posiłek w skali od 1 do 10 wybierając liczbę gwiazdek oraz pisze stosowny komentarz w polu tekstowym i klika przycisk "Submit" zapisujący komentarz w bazie danych, co ilustruje zrzut ekranu przedstawiony na rysunku 4.14. Dodatkowo użytkownik może usunąć komentarz naciskając na przycisk "Remove opinion".



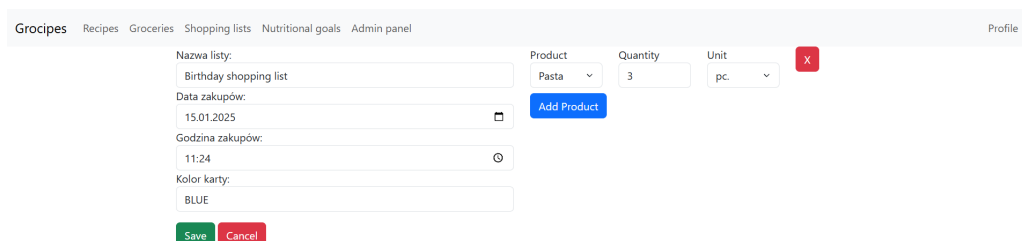
Rysunek 4.14: Zrzut ekranu przedstawiający dodanie oceny i komentarza posiłku.

Tworzenie listy zakupowej. Użytkownik loguje się do aplikacji. System po poprawnej weryfikacji wyświetla użytkownikowi stronę główną serwisu. Następnie wybiera na pasku nawigacji przycisk "Shopping lists". Aplikacja wyświetla użytkownikowi widok z wszystkimi listami zakupowymi użytkownika, co ilustruje zrzut ekranu przedstawiony na rysunku 4.15. Dodanie nowej listy zakupowej następuje po naciśnięciu przycisku "+".



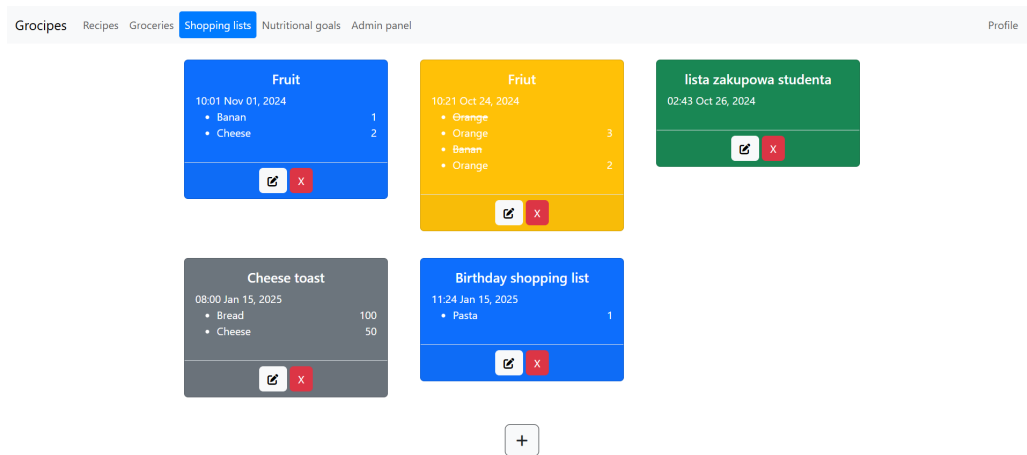
Rysunek 4.15: Zrzut ekranu przedstawiający widok listy z zakupami.

Następnie aplikacja wyświetla użytkownikowi formularz utworzenia listy zakupowej, gdzie wybiera on nazwę listy, datę i godzinę zrobienia zakupów, kolor karty i artykuły spożywcze. Dodanie produktu następuje po naciśnięciu przycisku "Add Product", aplikacja dynamicznie tworzy pole z wyborem istniejącego w bazie produktu i wybraniem jego ilości. Ilustruje to zrzut ekranu przedstawiony na rysunku 4.16.



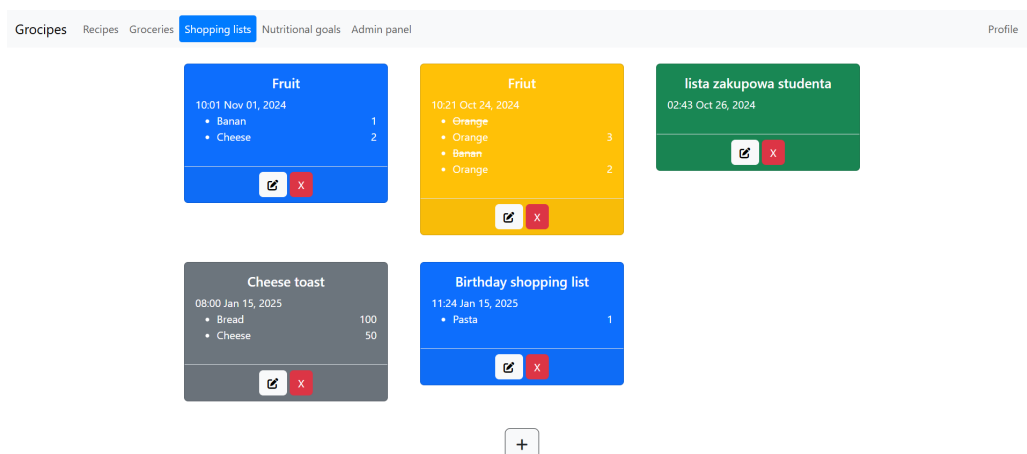
Rysunek 4.16: Zrzut ekranu przedstawiający formularz tworzenia listy zakupowej.

Wybór przycisku "Cancel" anuluje operację utworzenia listy. Utworzenie listy następuje po naciśnięciu przycisku "Save". Następnie aplikacja wyświetla użytkownikowi widok z listami zakupowymi wraz z nowo utworzoną listą, co ilustruje zrzut ekranu przedstawiony na rysunku 4.17.



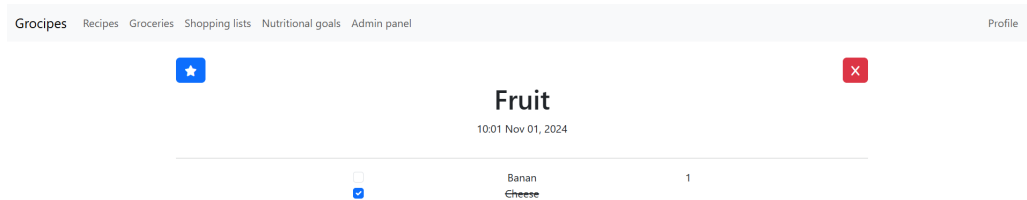
Rysunek 4.17: Zrzut ekranu przedstawiający widok listy zakupowej po utworzeniu nowej listy zakupowej.

Zarządzanie listą zakupową. Użytkownik loguje się do aplikacji. System po poprawnej weryfikacji wyświetla użytkownikowi stronę główną serwisu. Następnie wybiera na pasku nawigacji przycisk "Shopping lists". Aplikacja wyświetla użytkownikowi widok z wszystkimi listami zakupowymi użytkownika, co ilustruje zrzut ekranu przedstawiony na rysunku 4.18. Użytkownik usuwa konkretną listę zakupową naciskając na niej przycisk "X". Naciśnięcie ikonki edycji na liście wyświetla użytkownikowi formularz utworzenia listy z załadowanymi danymi wybranej listy, co ilustruje zrzut ekranu przedstawiony na rysunku 4.16. Zapisanie zmian następuje po naciśnięciu przycisku "Save", a odrzucenie zmian po naciśnięciu przycisku "Cancel".



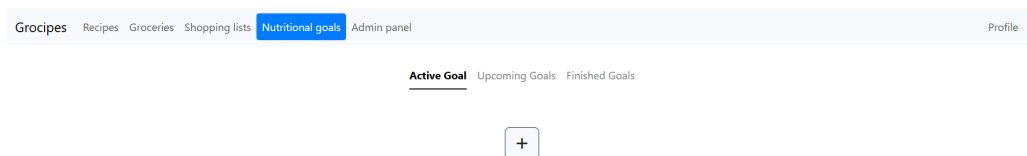
Rysunek 4.18: Zrzut ekranu przedstawiający widok listy zakupowej.

Naciśnięcie na konkretną listę zakupową wyświetla użytkownikowi widok ze szczegółowymi informacjami, co ilustruje zrzut ekranu przedstawiony na rysunku 4.19. Użytkownik widzi nazwę listy, datę zrobienia zakupów oraz produkty, które znajdują się w liście, które można zaznaczać i odznaczać. Dodatkowo użytkownik może oznaczyć listę naciskając na przycisk z ikonką gwiazdki w celu dodania listy zakupowej do ulubionych, które są wyświetlane na stronie głównej.



Rysunek 4.19: Zrzut ekranu przedstawiający widok podglądu listy zakupowej.

Tworzenie celu żywieniowego. Użytkownik loguje się do aplikacji. System po poprawnej weryfikacji wyświetla użytkownikowi stronę główną serwisu. Następnie wybiera na pasku nawigacji przycisk "Nutritional goals". Aplikacja wyświetla widok z celami żywieniowymi, co ilustruje zrzut ekranu przedstawiony na rysunku 4.20.

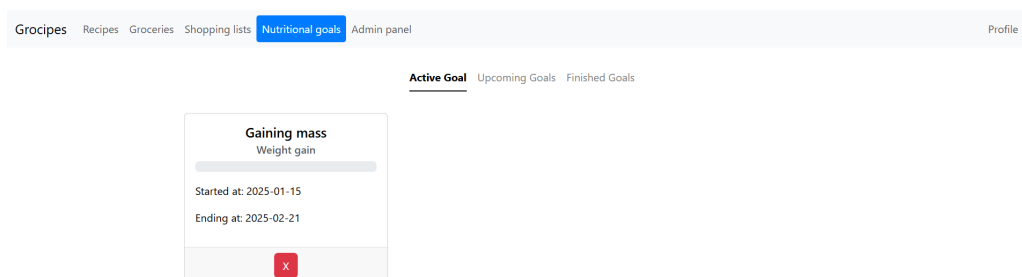


Rysunek 4.20: Zrzut ekranu przedstawiający widok aktywnego celu żywieniowego.

Naciśnięcie przycisku "+" wyświetli użytkownikowi formularz utworzenia celu żywieniowego, w którym uzupełnia pola, takie jak nazwa celu, typ celu, docelowa waga, obwód brzucha i procent tkanki tłuszczowej. Ilustruje to zrzut ekranu przedstawiony na rysunku 4.21. Aplikacja pozwala użytkownikowi zmieniać wartości docelowe parametrów ciała w zależności od wybranego typu celu. Wybranie opcji "Weight maintenance" pozwala tylko na ustawienie nazwy i wybranie daty początkowej i końcowej. Opcja "Weight loss" pozwala tylko zmniejszać parametry ciała, a opcja "Weight gain" na ich zwiększanie. Dodatkowo aplikacja dla opcji "Weight gain" i "Weight loss" oblicza średni czas trwania celu i po wybraniu przez użytkownika daty rozpoczęcia uzupełnia datę końcową. Anulowanie operacji następuje po naciśnięciu przycisku "Cancel". Naciśnięcie przycisku "Save" zapisuje cel w systemie.

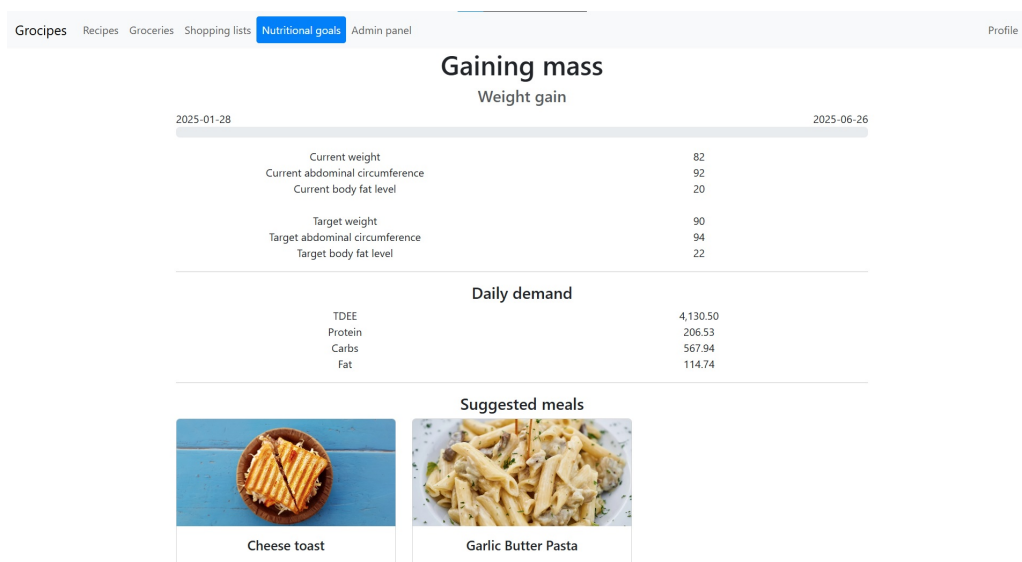
Rysunek 4.21: Zrzut ekranu przedstawiający formularz utworzenia celu żywieniowego.

Następnie aplikacja przenosi użytkownika do widoku z celami żywieniowymi, co widać na zrzucie ekranu przedstawionym na rysunku 4.22. Użytkownik może realizować tylko jeden cel w danym okresie. W widoku znajdują się zakładki "Active Goal", "Upcoming Goals", "Finished Goals", które porządkują cele. W zakładce "Active Goal" znajduje się obecnie wykonywany cel, zakładka "Upcoming Goals" zawiera cel, który się jeszcze nie rozpoczął, a zakładka "Finished Goals" cele, które użytkownik ukończył.



Rysunek 4.22: Zrzut ekranu przedstawiający widok po utworzeniu celu żywieniowego.

Naciśnięcie na cel wyświetla widok ze szczegółowymi informacjami, co ilustruje zrzut ekranu przedstawiony na rysunku 4.23. Użytkownik widzi swój postęp, obecne i docelowe parametry ciała oraz dzienne zapotrzebowanie kaloryczne wraz z podziałem na makroskładniki, takie jak białko, tłuszcze i węglowodany. Dodatkowo aplikacja proponuje przepisy dostosowane do zapotrzebowania żywieniowego.



Rysunek 4.23: Zrzut ekranu przedstawiający podgląd celu żywieniowego.

Aktualizacja parametrów ciała. Użytkownik loguje się do aplikacji. System po poprawnej weryfikacji wyświetla użytkownikowi stronę główną serwisu. Następnie wybiera na pasku nawigacji przycisk "Profile", który wyświetla informacje o profilu użytkownika. Ilustruje to zrzut ekranu przedstawiony na rysunku 4.24.

The screenshot shows a user profile page for 'Konrad Rduch'. At the top, there is a navigation bar with links: 'Grocipes', 'Recipes', 'Groceries', 'Shopping lists', 'Nutritional goals', 'Admin panel', and a 'Profile' button. Below the navigation bar is a large circular profile picture placeholder. Underneath the picture is the name 'Konrad Rduch'. The profile information is displayed in a table-like format:

Email	user1@wp.pl
Gender	Male
Birthday	2002-10-16

Below this is a section titled 'Body parameters' with another table:

Height	186 [cm]
Weight	82 [kg]
Body fat level	20 %
Abdominal circumference	92 [cm]
Physical activity	High active

At the bottom of the profile section, there are two buttons: 'Update body parameters' (green) and 'Body parameters history' (blue). Below these buttons is a 'Log out' button (red).

Rysunek 4.24: Zrzut ekranu przedstawiający widok profilu użytkownika.

Użytkownik naciska na przycisk "Update body parameters", aplikacja wyświetla formularz przedstawiony na rysunku 4.25. Następnie uzupełnia pola z wagą, wysokością, obwodem brzucha, poziomem tkanki tłuszczowej aktualnymi oraz wybiera stopnie aktywności fizycznej. Wybiera pomiędzy "High active", "Low active" i "Moderately active". Anulowanie operacji następuje po naciśnięciu przycisku "Cancel". Naciśnięcie przycisku "Save" aktualizuje zmiany w systemie. Następnie system przenosi użytkownika z powrotem do widoku profilu użytkownika.

The screenshot shows a form for updating body parameters. At the top, there is a navigation bar with links: 'Grocipes', 'Recipes', 'Groceries', 'Shopping lists', 'Nutritional goals', 'Admin panel', and a 'Profile' button. Below the navigation bar is the form itself, which contains the following fields:

- Weight[kg]: 82
- Height[cm]: 186
- Abdominal circumference[cm]: 92
- Body fat level[%]: 20
- Physical activity: High active

At the bottom of the form, there are two buttons: 'Save' (green) and 'Cancel' (red).

Rysunek 4.25: Zrzut ekranu przedstawiający formularz aktualizacji parametrów ciała.

Naciśnięcie na przycisk "Body parameters history" wyświetla użytkownikowi historię pomiarów ciała, co widać na zrzucie ekranu przedstawionym na rysunku 4.26.

Grocipes Recipes Groceries Shopping lists Nutritional goals Admin panel Profile				
Body parameters history				
Measurement taken at 10:53 Oct 30, 2024				
Physical activity: Moderately active				
Weight:	80			[kg]
Height:	186			[cm]
Abdominal circumference:	90			[cm]
Body fat level:	20			%
Measurement taken at 09:48 Nov 05, 2024				
Physical activity: High active				
Weight:	82			[kg]
Height:	186			[cm]
Abdominal circumference:	92			[cm]
Body fat level:	20			%

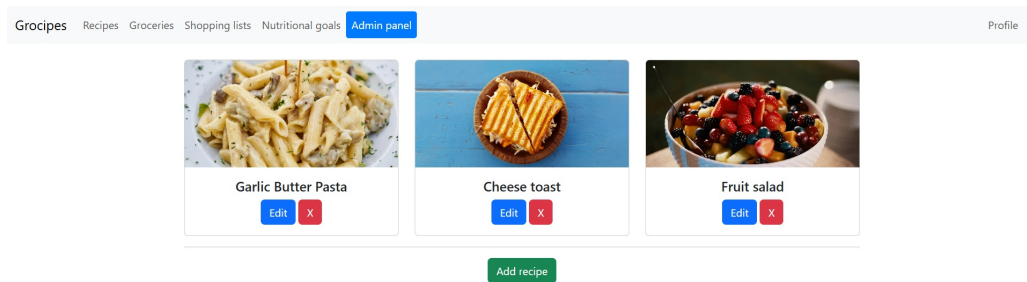
Rysunek 4.26: Zrzut ekranu przedstawiający historię pomiarów.

Zarządzanie przepisami - Administrator. Administrator loguje się do aplikacji. System po poprawnej weryfikacji wyświetla stronę główną serwisu. Następnie wybiera na pasku nawigacji przycisk "Admin panel", który wyświetla informacje o profilu użytkownika. Ilustruje to zrzut ekranu przedstawiony na rysunku 4.27.

Grocipes Recipes Groceries Shopping lists Nutritional goals Admin panel Profile				
Manage products				
Create product or modify or delete existing product				
Manage recipes				
Create recipe or modify or delete existing recipe				

Rysunek 4.27: Zrzut ekranu przedstawiający panel admina.

Administrator wybiera opcję "Manage recipes", aplikacja wyświetla widok zarządzający przepisami, w którym znajdują się wszystkie przepisy widoczne we serwisie. Ilustruje to zrzut ekranu przedstawiony na rysunku 4.28. Naciśnięcie przycisku "X" usuwa przepis z systemu. Naciśnięcie przycisku "Edit" przy wybranym przepisie wyświetla użytkownikowi formularz edycji z wypełnionymi wartościami. Przycisk "Save" zapisuje zmiany, a przycisk "Cancel" odrzuca zmiany.



Rysunek 4.28: Zrzut ekranu przedstawiający widok zarządzania przepisami.

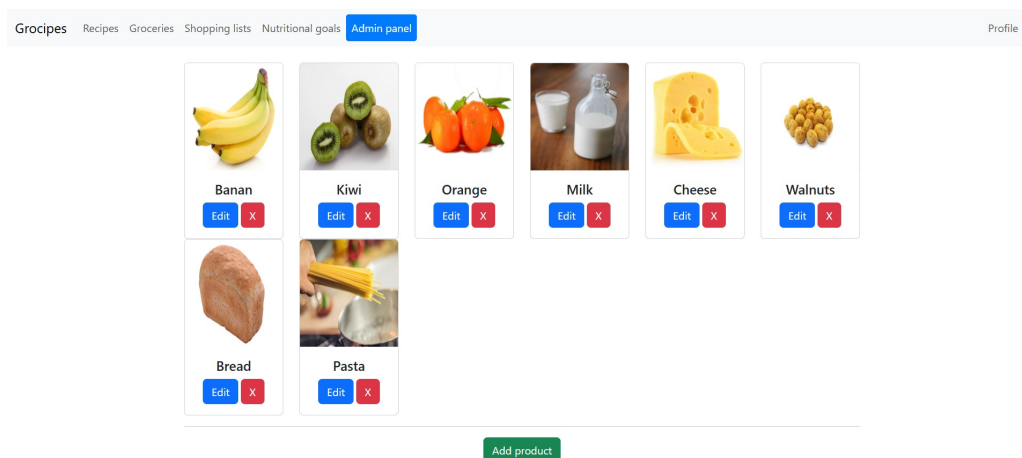
Administrator dodaje nowy przepis naciskając na przycisk "Add recipe", aplikacja wyświetla formularz utworzenia przepisu, co ilustruje zrzut ekranu przedstawiony na rysunku 4.29. Przycisk "Save" zapisuje nowy przepis w systemie, a przycisk "Cancel" anuluje operację.

 The screenshot shows the 'Add Product' form within the 'Admin panel'. The navigation bar is the same as in the previous image. The form contains several input fields: 'Title' (with a placeholder 'title'), 'Product' (a dropdown menu), 'Quantity' (with a placeholder 'quantity'), 'Unit' (a dropdown menu), and 'Type of Meal' (a dropdown menu). To the right of the 'Quantity' and 'Unit' fields is a red 'X' button. Below these fields is a blue 'Add Product' button. Further down are 'Image URL' (with a placeholder 'image url'), 'Description' (a large text area), and 'Preparation method' (a large text area). At the bottom of the form are two buttons: a green 'Save' button and a red 'Cancel' button.

Rysunek 4.29: Zrzut ekranu przedstawiający formularz utworzenia przepisu.

Zarządzanie artykułami spożywczymi - Administrator. Administrator loguje się do aplikacji. System po poprawnej weryfikacji wyświetla stronę główną serwisu. Następnie wybiera na pasku nawigacji przycisk "Admin panel", który wyświetla informacje o profilu użytkownika. Ilustruje to zrzut ekranu przedstawiony na rysunku 4.27. Administrator wybiera opcję "Manage products", aplikacja wyświetla widok zarządzający

artykułami spożywczymi, w którym znajdują się wszystkie produkty widoczne we serwisie. Ilustruje to zrzut ekranu przedstawiony na rysunku 4.31. Naciśnięcie przycisku "X" usuwa artykuł spożywczy z systemu. Naciśnięcie przycisku "Edit" przy wybranym produkcie wyświetla użytkownikowi formularz edycji z wypełnionymi wartościami. Przycisk "Save" zapisuje zmiany, a przycisk "Cancel" odrzuca zmiany.



Rysunek 4.30: Zrzut ekranu przedstawiający widok zarządzania artykułami spożywczymi.

Administrator dodaje nowy artykuł spożywczy naciskając na przycisk "Add recipe", aplikacja wyświetla formularz utworzenia produktu, co ilustruje zrzut ekranu przedstawiony na rysunku 4.29. Przycisk "Save" zapisuje nowy przepis w systemie, a przycisk "Cancel" anuluje operację.

The screenshot shows the 'Add recipe' form in the 'Admin panel'. The form is divided into two columns. The left column contains: Name (placeholder: name), Weight (placeholder: weight), Unit (dropdown menu), Price (placeholder: price), Image URL (placeholder: image url), and Calories (placeholder: calories). The right column contains: Fat (placeholder: 0), Cholesterol (placeholder: 0), Sodium (placeholder: 0), Carbohydrates (placeholder: 0), Protein (placeholder: 0), and Potassium (placeholder: 0). At the bottom, there are two buttons: 'Save' (green) and 'Cancel' (red).

Rysunek 4.31: Zrzut ekranu przedstawiający formularz utworzenia artykułu spożywczego.

Rozdział 5

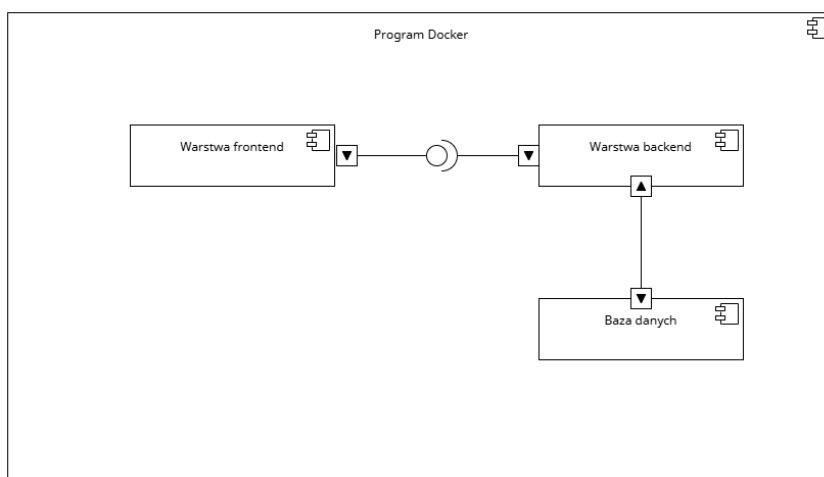
Specyfikacja wewnętrzna

5.1 Architektura systemu

System został zaprojektowany w oparciu o architekturę klient-serwer. Rolę klienta przyjmuje warstwa frontend, natomiast warstwa backend pełni rolę serwera. Warstwy zostały napisane przy pomocy nowoczesnych, dedykowanych narzędzi (por. rozdz. 3). Warstwy komunikują się za pomocą protokołu komunikacyjnego HTTP, zgodnie z zasadami architektury REST API [4].

5.1.1 Zarys architektury

W poniższym podrozdziale przedstawiono diagram komponentów, co ilustruje zrzut ekranu przedstawiony na rysunku 5.1. Diagram pokazuje, jak poszczególne komponenty systemu komunikują się między sobą.



Rysunek 5.1: Zrzut ekranu przedstawiający diagram komponentów.

W dalszej części podrozdziału poszczególne komponenty systemu zostały opisane wraz z ich rolą i wzajemnymi interakcjami.

Warstwa frontend. Odpowiada za część systemu związaną z interfejsem graficznym. Definiuje to, w jaki sposób dane zostaną zaprezentowane użytkownikowi – określa kolor tła, wygląd przycisków, rozmiar i styl czcionki oraz innych elementów interfejsu graficznego. Zapewnia obsługę interakcji z użytkownikiem, w tym pobieranie i walidację danych wprowadzonych do systemu. Warstwa frontend pobiera lub zapisuje informacje, wysyłając do warstwy backend odpowiednie zapytania, które spełniają wymogi odpowiedniego punktu końcowego API [12].

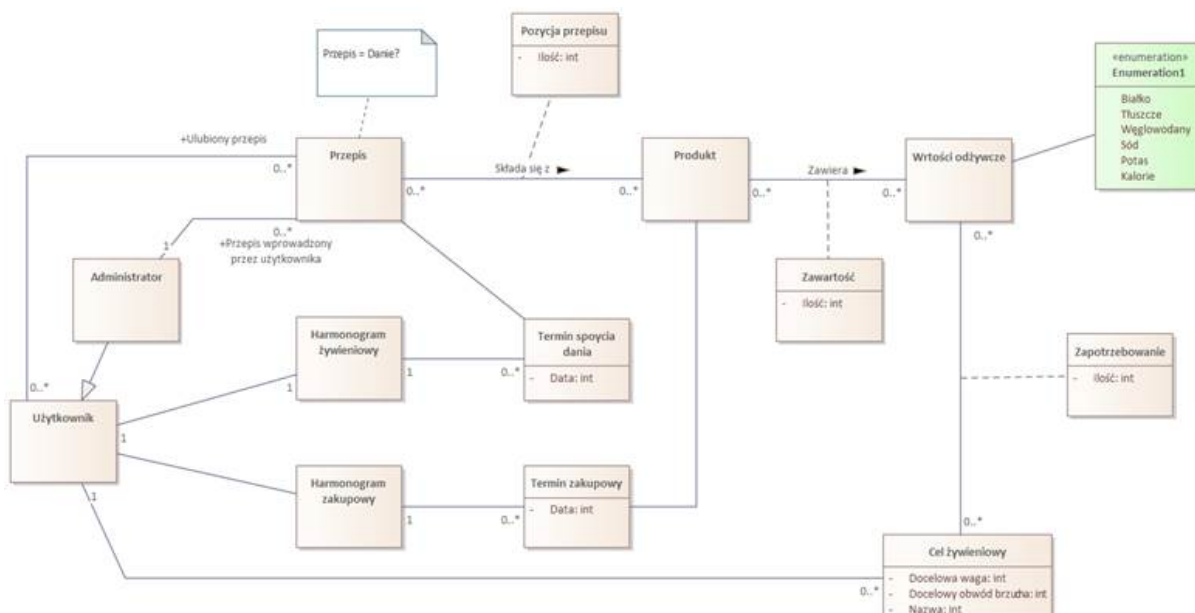
Warstwa backend. Odpowiada za część systemu związaną z zapleczem aplikacji. Obsługuje zapytania oraz realizuje logikę biznesową aplikacji. Zarządza bazą danych i komunikuje się z nią poprzez wykorzystanie Java Persistence API i Hibernate, służących do mapowania obiektowo-relacyjnego. Warstwa backend wystawia warstwie frontend wymagany interfejs punktu końcowego API [11].

Baza danych. Przechowuje i zapewnia spójność danych aplikacji oraz zapewnia bezpieczeństwo.

Program Docker. Odpowiada za konteneryzację aplikacji, dzięki której każdy komponent posiada własny kontener, co sprawia, że może być on uruchamiany oraz zarządzany niezależnie.

5.1.2 Model klas

Poniżej przedstawiono diagram klas UML (rysunek 5.2.), pokazujący strukturę logiczną systemu. W tej części zamieszczono także krótki opis każdej z klas. rysunku 5.2.



Rysunek 5.2: Zrzut ekranu przedstawiający diagram klas.

Użytkownik. Klasa reprezentująca użytkownika aplikacji zawiera informacje o danych osobowych, takich jak imię, nazwisko, adres e-mail, data urodzenia oraz parametry

ciała.

Administrator. Klasa reprezentująca administratora aplikacji. Rozszerza uprawnienia względem zwykłego użytkownika o zarządzanie przepisami i artykułami spożywczymi.

Przepis. Klasa reprezentująca przepis kulinarny. Przechowuje informacje, takie jak tytuł przepisu, opis, metoda przygotowania oraz typ posiłku. Zawiera także listę składników powiązanych z klasą "Produkt".

Pozycja przepisu. Klasa reprezentuje pojedynczy produkt w przepisie, zawiera informacje o ilości danego produktu w tym przepisie.

Harmonogram żywieniowy. Klasa tworząca i zarządzająca planem żywieniowym użytkownika. Powiązana jest z klasą "Termin spożycia dania".

Harmonogram zakupowy. Klasa tworząca i zarządzająca listą zakupową użytkownika. Powiązana jest z klasą "Produkty".

Termin spożycia dania. Klasa przechowuje datę i godzinę spożycia posiłku oraz komentarz i ocenę użytkownika. Wskazuje, kiedy użytkownik powinien spożyć dane danie. Klasa jest powiązana z "Harmonogram żywieniowy" i "Przepis".

Termin zakupowy. Klasa przechowuje datę i godzinę zrobienia zakupów. Klasa jest powiązana z "Harmonogram zakupowy" i "Produkt".

Produkt. Klasa reprezentująca artykuł spożywczy. Przechowuje informacje o nazwie przepisu, ilości kalorii, średniej cenie i wadze. Zawiera także dane o wartościach odżywczych powiązanych z klasą "Produkt".

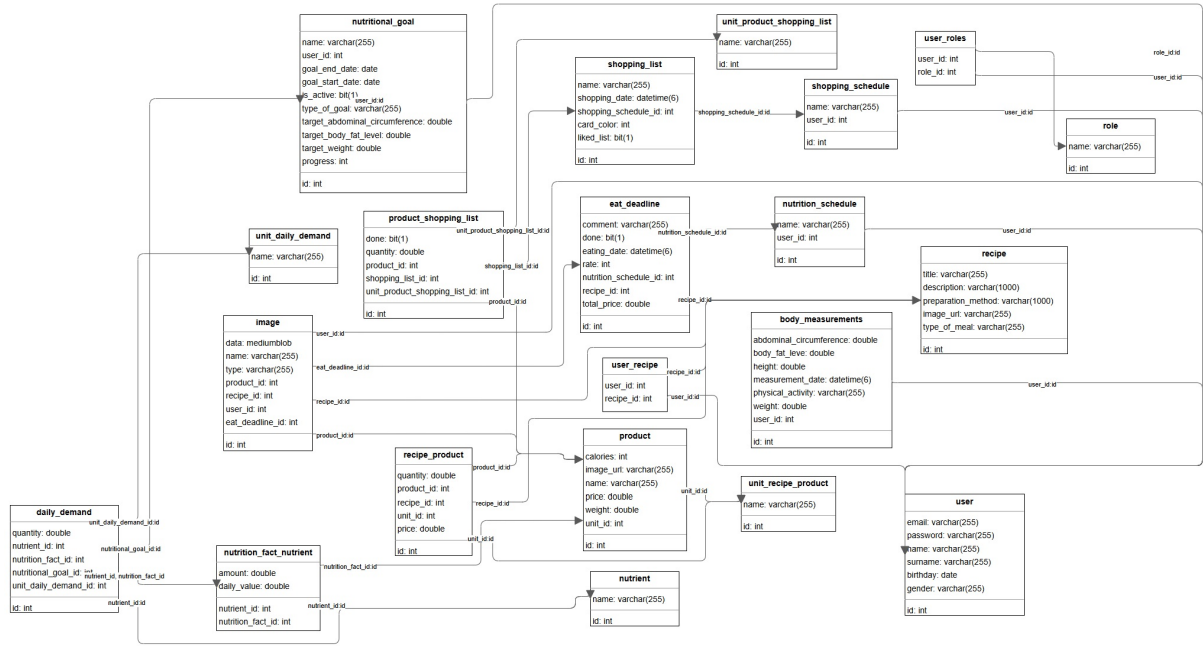
Zawartość. Klasa reprezentuje zawartość składników odżywczych w produkcie. Przechowuje informacje o ilości składnika w artykule spożywczym.

Wartość odżywcza. Klasa przechowująca informacje o nazwie składnika odżywczego.

Zapotrzebowanie. Klasa reprezentująca zapotrzebowanie użytkownika na składniki odżywcze w zależności od jego celów żywieniowych.

Cel żywieniowy. Klasa reprezentująca cel żywieniowy użytkownika. Przechowuje informację o nazwie celu, dacie rozpoczęcia i zakończenia, docelowych parametrach ciała oraz typie celu: czy jest to zmniejszanie, zwiększanie, lub też utrzymanie parametrów ciała.

Diagram związków encji, co ilustruje zrzut ekranu przedstawiony na rysunku 5.3, został zaprojektowany na podstawie diagramu klas. Klasy zostały odwzorowane na bazodanowe tabele.



Rysunek 5.3: Zrzut ekranu przedstawiający diagram związków encji.

5.2 Wybrane fragmenty kodu

Warstwy frontend i backend zostały zaprojektowane z wykorzystaniem narzędzi, stosujących inne modele architektury kodu, specjalnie przystosowanych do realizacji specyficznych zadań każdej z warstw, jednocześnie zapewniając wysoką jakość i czystość kodu [10].

Warstwa backend. Warstwa backend została stworzona za pomocą frameworka Spring Framework, który wymaga stosowania wzorca Controller-Service-Repository [5]. Wzorzec zapewnia separację odpowiedzialności, oddzielającą część związaną z zarządzaniem bazą danych, logiką biznesową oraz przyjmowaniem i wysyłaniem zapytań do warstwy frontend. Dzięki temu kod jest łatwiejszy w rozbudowie, testowaniu i utrzymaniu.

Warstwa frontend. Warstwa frontend została stworzona za pomocą frameworka Angular, który jest oparty na używaniu komponentów oraz serwisów. Komponenty wiążą pliki TypeScript, HTML i CSS w jeden element, zapewniając modularność oraz możliwość ponownego wykorzystania komponentu w aplikacji. Serwisy dbają o komunikację z API wystawionym przez warstwę backend za pomocą protokołu komunikacyjnego HTTP.

5.2.1 Przykładowa encja

Poniżej przedstawiono przykładową klasę encji. Ilustruje to fragment kodu przedstawiony na listingu 5.1. Oznaczenie klasy adnotacją @Entity sprawia, że klasa, która reprezentuje zapotrzebowanie żywieniowe, zostaje zamieniona na bazodanową tabelę, w której

pola pełnią rolę kolumn. Adnotacja @Id nad polem powoduje utworzenie klucza głównego, a adnotacja @ManyToOne utworzenie relacji wiele do jednego.

Listing 5.1: Przykładowa encja.

```

1 @Entity
2 public class DailyDemand {
3
4     @Id
5     @Column(name = "id", nullable = false)
6     @GeneratedValue(strategy = GenerationType.IDENTITY)
7     private Integer id;
8
9     private double quantity;
10
11     @ManyToOne
12     @JoinColumns({
13         @JoinColumn(name = "nutrition_fact_id",
14             referencedColumnName = "nutrition_fact_id"),
15         @JoinColumn(name = "nutrient_id",
16             referencedColumnName = "nutrient_id")
17     })
18     private NutritionFactNutrient nutrient;
19
20     @ManyToOne
21     @JoinColumn(name="nutritional_goal_id")
22     private NutritionalGoal nutritionalGoal;
23
24     @ManyToOne
25     @JoinColumn(name="unit_daily_demand_id")
26     private UnitDailyDemand unitDailyDemand;
27
28     public DailyDemand() {
29
30     }
31
32     public DailyDemand(Integer id, double quantity,
33         NutritionFactNutrient nutrient, NutritionalGoal
34         nutritionalGoal, UnitDailyDemand unitDailyDemand) {
35         this.id = id;

```

```
32     this.quantity = quantity;
33     this.nutrient = nutrient;
34     this.nutritionalGoal = nutritionalGoal;
35     this.unitDailyDemand = unitDailyDemand;
36 }
37 public Integer getId() {
38     return id;
39 }
40 public void setId(Integer id) {
41     this.id = id;
42 }
43 public double getQuantity() {
44     return quantity;
45 }
46 public void setQuantity(double quantity) {
47     this.quantity = quantity;
48 }
49 public NutritionFactNutrient getNutrient() {
50     return nutrient;
51 }
52 public void setNutrient(NutritionFactNutrient nutrient) {
53     this.nutrient = nutrient;
54 }
55 public NutritionalGoal getNutritionalGoal() {
56     return nutritionalGoal;
57 }
58 public void setNutritionalGoal(NutritionalGoal
    nutritionalGoal) {
59     this.nutritionalGoal = nutritionalGoal;
60 }
61 public UnitDailyDemand getUnitDailyDemand() {
62     return unitDailyDemand;
63 }
64 public void setUnitDailyDemand(UnitDailyDemand
    unitDailyDemand) {
65     this.unitDailyDemand = unitDailyDemand;
66 }
67 }
```

5.2.2 Przykładowy kontroler

Poniżej przedstawiono przykładową klasę kontrolera obsługującą klasę Produkt. Ilustruje to fragment kodu przedstawiony na listingu 5.2. Klasa obsługuje zapytania HTTP, takie jak POST tworzące nowe zasoby, GET pobierające dane, PUT i PATCH aktualizujące dane oraz DELETE usuwające dane. Każda funkcja jest oznaczona specjalną adnotacją, która realizuje odpowiednie metody HTTP. Dodatkowo w funkcji wykonywane są tylko metody serwisu, które oddzielają logikę biznesową aplikacji od przyjmowania danych z warstwy frontend.

Listing 5.2: Przykładowy kontroler.

```

1 @RestController
2 @RequestMapping("/products")
3 public class ProductController {
4     private final ProductService productService;
5     private final NutritionFactNutrientService
        nutritionFactNutrientService;
6
7     public ProductController(ProductService productService,
        NutritionFactNutrientService nutritionFactNutrientService
        ) {
8         this.productService = productService;
9         this.nutritionFactNutrientService =
            nutritionFactNutrientService;
10    }
11    @GetMapping("/{id}")
12    public GetProductsDTO getProduct(@PathVariable Integer id){
13        List<GetProductsDTO>productsDTOS = productService.
            getProducts();
14        GetProductsDTO foundProduct = productsDTOS.stream()
15            .filter(product -> product.getId().equals(id))
16            .findFirst()
17            .orElse(null);
18        return foundProduct;
19    }
20    @GetMapping("")
21    public List<GetProductsDTO>getProducts(){
22        return productService.getProducts();
23    }
24    @PostMapping("/add")

```

```
25     public ResponseEntity<Void> addProduct(@RequestBody
        ProductCreationDTO request){
26         ProductDTO productDTO = request.getProductDTO();
27         List<NutritionFactNutrientDTO> nutritionFactNutrientDTO
            = request.getNutritionFactNutrientDTO();
28
29         productService.addProduct(productDTO).map(ResponseEntity::ok).
            orElseGet(() -> ResponseEntity.badRequest().build());
30         Integer productId = productService.getProductIdByName(productDTO
            .getName());
31         for(int i = 0; i < nutritionFactNutrientDTO.size(); i++) {
32             nutritionFactNutrientDTO.get(i).setProductId(productId);
33         }
34         for(int i = 0; i < nutritionFactNutrientDTO.size(); i++){
            nutritionFactNutrientService.
                addNutrionFactNutrient(nutritionFactNutrientDTO.get(i
                ));
35         }return ResponseEntity.ok().build();}
36         @DeleteMapping("/delete/{id}")
37         public ResponseEntity<Void> deleteProduct(@PathVariable
            Integer id) {
38             NutritionFactNutrient
39             nutritionFactNutrientService.deleteNutritionFactsByProductId(id)
            ;
40             productService.deleteProductById(id);
41             return ResponseEntity.ok().build();}
42         @PatchMapping("edit/{id}")
43         public ResponseEntity<Void> editProduct(@PathVariable
            Integer id, @RequestBody ProductCreationDTO
            updatedProduct) {
44             Product product = productService.findById(id);
45             ProductDTO productDTO = updatedProduct.getProductDTO();
46             product.setId(id);
47             product.setName(productDTO.getName());
48             product.setWeight(productDTO.getWeight());
49             product.setPrice(productDTO.getPrice());
50             product.setImage_url(productDTO.getImage_url());
51             product.setCalories(productDTO.getCalories());
52             productService.save(product, productDTO.getUnitId());
```

```
53 nutritionFactNutrientService.deleteNutritionFactsByProductId(id)
    ;
54     for (NutritionFactNutrientDTO nutrientDTO :
        updatedProduct.getNutritionFactNutrientDTO()) {
55         nutrientDTO.setProductId(id);
56 nutritionFactNutrientService.addNutritionFactNutrient(nutrientDTO)
        ;} return ResponseEntity.ok().build();
57 }
58 };
```

5.2.3 Przykładowe repozytorium

Poniżej przedstawiono przykładowe repozytorium. Ilustruje to fragment kodu przedstawiony na listingu 5.3. Repozytorium dziedziczy po interfejsie `JpaRepository`, zapewniając podstawowe operacje CRUD (ang. Create, Read, Update, Delete) dla encji celu żywieniowego.

Listing 5.3: Przykładowe repozytorium.

```
1 public interface NutritionalGoalRepository extends JpaRepository
    <NutritionalGoal, Integer>
2 {
3     NutritionalGoal findNutritionalGoalById(Integer id);
4     void deleteById(Integer id);
5     List<NutritionalGoal> findNutritionalGoalsByUserId(
        Integer userId);
6
7 }
```

5.2.4 Przykładowy serwis - warstwa backend

Poniżej przedstawiono przykładową klasę serwisu obsługującą listę zakupową. Ilustruje to fragment kodu przedstawiony na listingu 5.4. Klasa serwisu zawiera metody realizujące logikę biznesową aplikacji. Wykorzystuje instancje repozytorium listy zakupowej, udostępniając metody, pozwalające na wyszukiwanie, modyfikowanie i usuwanie rekordów z bazodanowych tabel.

Listing 5.4: Przykładowy serwis warstwy backend.

```
1 @Service
2 public class ShoppingListService {
```

```
3     private final ShoppingListRepository shoppingListRepository ;
4     public ShoppingListService(ShoppingListRepository
        shoppingListRepository) {
5         this.shoppingListRepository = shoppingListRepository;
6     }
7     public void save(ShoppingList shoppingList){
8         shoppingListRepository.save(shoppingList);
9     }
10    public List<Object[]> getAllByUserId(Integer userId) {
11        return shoppingListRepository.
            findShoppingListsWithProductsByUserId(userId);
12    }
13    public List<ShoppingList> findByShoppingSchedule(
        ShoppingSchedule shoppingSchedule){
14        return shoppingListRepository.
            findShoppingListByShoppingList(shoppingSchedule);
15    }
16    public ShoppingListDTO findShoppingListDTOById(Integer id){
17        ShoppingList shoppingList = shoppingListRepository.
            findShoppingListById(id);
18        ShoppingListDTO shoppingListDTO = new
            ShoppingListDTO(shoppingList.getId(),
                shoppingList.getName(), shoppingList.
                    getShopping_date(), shoppingList.getCardColor(),
                    shoppingList.isLikedList(),
                    convertToProductShoppingListDTO(shoppingList.
                        getProductShoppingLists()));
19        return shoppingListDTO;}
20    public ShoppingList findShoppingListById(Integer id){
21        return shoppingListRepository.findShoppingListById(id);
22    }
23    private List<ProductShoppingListDTO>
        convertToProductShoppingListDTO(List<ProductShoppingList>
            productShoppingLists){
24        List<ProductShoppingListDTO> converted =
            productShoppingLists.stream()
25            .map(productShoppingList -> {
26                ProductShoppingListDTO dto = new
                    ProductShoppingListDTO(
```

27

```

        productShoppingList.getId(),

        productShoppingList.getProduct().
        getId(),
        productShoppingList.getProduct().
        getName(),
        productShoppingList.getProduct().
        getWeight(),
        productShoppingList.getProduct().
        getPrice(),
        productShoppingList.getProduct().
        getImage_url(),
        productShoppingList.getProduct().
        getCalories(),
28     productShoppingList.getShoppingList().getId(),
29     productShoppingList.getQuantity(),
        productShoppingList.
        getUnitProductShoppingList().getId(),
        productShoppingList.
        getUnitProductShoppingList().getName(),
30     productShoppingList.isDone()
31 );
32     return dto;
33 })
34     .collect(Collectors.toList()); return converted;}
35 @Transactional
36 public void deleteShoppingListById(Integer id) {
37     this.shoppingListRepository.deleteById(id);
38 }
39 }

```

5.2.5 Przykładowy serwis - warstwa frontend

Poniżej przedstawiono przykładową klasę serwisu warstwy frontend. Ilustruje to fragment kodu przedstawiony na listingu 5.5. Serwis odpowiada za komunikację z REST API warstwy backend przy użyciu protokołu HTTP. Każda funkcja wysyła odpowiednie zapytanie do punktu końcowego API.

Listing 5.5: Przykładowy serwis warstwy frontend.

```
1  @Injectable({ providedIn: 'root' })
2  export class EatDeadlineService {
3      eatDeadlineChanged = new BehaviorSubject<EatDeadline[]>([]);
4  eatDeadlineChanged$ = this.eatDeadlineChanged.asObservable();
5      constructor(private http: HttpClient,) { }
6      getAllEatDeadline(): Observable<EatDeadline[]>{
7          return this.http.get<EatDeadline[]>('http://localhost
           :8080/eatDeadline/getAllEatDeadline').pipe(tap(data
           =>{this.eatDeadlineChanged.next(data);}));
8      getEatDeadline(id: number): Observable<EatDeadline>{
9          return this.http.get<EatDeadline>('http://localhost
           :8080/eatDeadline/${id}');
10     addEatDeadline(dto: any){
11         const addDto = {
12     recipelId: dto.recipelId, eatingDate: dto.eatingDate, done: dto.done
           , rate: dto.rate, comment: dto.comment}
13         return this.http.post('http://localhost:8080/eatDeadline
           /add', addDto)
14     deleteEatDeadline(id: number){
15         return this.http.delete('http://localhost:8080/eatDeadline
           /delete/${id}')
16     commentMeal(id: number, comment: string){
17         const newComment = {comment: comment}
18         return this.http.put('http://localhost:8080/eatDeadline/
           comment/${id}', newComment);
19     rateMeal(id: number, rate: number){
20         const newRate = {rate: rate}
21         return this.http.put('http://localhost:8080/eatDeadline/
           rate/${id}', newRate);}
```

5.2.6 Przykładowy interceptor

Poniżej przedstawiono przykładowy interceptor autoryzacyjny. Ilustruje to fragment kodu przedstawiony na listingu 5.6. Interceptor przechwytuje żądania HTTP przed wysłaniem do serwera i dokleja do nich odpowiedni token autoryzacyjny, umożliwiając warstwie frontend na otrzymanie dostępu do punktów końcowych API warstwy backend.

Listing 5.6: Przykładowy interceptor.

```

1 @Injectable({ providedIn: 'root' })
2 export class AuthInterceptorService implements HttpInterceptor {
3     constructor(private authService: AuthService) { }
4     intercept(req: HttpRequest<any>, next: HttpHandler):
5         Observable<HttpEvent<any>> {
6         return this.authService.user.pipe(
7             take(1),
8             exhaustMap(user => {
9                 if (!user) {
10                     return next.handle(req);
11                 }
12                 const modifiedReq = req.clone
13                     ({
14                         setHeaders: {
15                             Authorization: 'Bearer ${user.token}
16                         }
17                     })
18                 return next.handle(modifiedReq);
19             })
20         );
21     }

```

5.2.7 Przykładowy komponent

Poniżej przedstawiono przykładowy komponent, który reprezentuje widok strony głównej aplikacji. Ilustruje to fragment kodu przedstawiony na listingach 5.7 i 5.8. Komponent zawiera plik HTML odpowiadający za strukturę wizualną widoku oraz plik TypeScript definiujący logikę działania komponentu, na przykład naciskanie na przyciski oraz pobieranie i zapisywanie danych.

Listing 5.7: Przykładowy komponent. Plik TypeScript.

```

1 @Component({
2     selector: 'app-homepage',
3     templateUrl: './homepage.component.html',
4     styleUrls: ['./homepage.component.css']
5 })

```

```
6 export class HomepageComponent implements OnInit, OnDestroy{
7   shoppingList: ShoppingList[] | undefined;
8   shoppingSchedule: ShoppingSchedule | undefined;
9   constructor(private shoppingListService: ShoppingListService)
10    { }
11   ngOnInit(): void {
12     this.shoppingListService.fetchShoppingLists().subscribe(
13       (data: ShoppingSchedule[]) => {
14         this.shoppingSchedule = data[0];
15         this.shoppingList = [...this.shoppingSchedule.
16           shoppingList];
17       });
18     if(this.shoppingList !== undefined){
19       this.shoppingListService.shoppingListChanged$.subscribe(
20         (data: ShoppingSchedule[]) => {
21           this.shoppingSchedule = data[0];
22           this.shoppingList = [...this.shoppingSchedule.
23             shoppingList];});}}
21   ngOnDestroy(): void {
22   }}
```

Listing 5.8: Przykładowy komponent. Plik HTML.

```
1 <div class="container-fluid" style="height: 100vh;">
2
3   <div class="row overflow-x-auto flex nowrap" style="min-
4     height: 12rem; max-height: 12rem; margin-top: 25px;">
5     <div class="col-auto" *ngFor="let sl of shoppingList">
6       <app-shop-list-item
7         [shopList]="sl"></app-shop-list-item>
8     </div></div>
9   <div class="row"><div class="col"><hr></div> </div>
10  <div class="row h-100">
11    <div class="col-12">
12      <app-calendar-grid></app-calendar-grid>
13    </div></div></div>
```

Rozdział 6

Weryfikacja i walidacja

6.1 Metodyka testów

Do procesu weryfikacji i walidacji systemu użyto podejścia opartego na Modelu-V [6]. Model ilustruje diagram w kształcie litery "V". Lewa strona litery "V" przedstawia kolejne etapy konstrukcji, a prawa strona kolejne etapy weryfikacji. Proces zaczyna się od górnej części lewej strony litery "V". Następnie schodzi w dół kolejno po etapach konstrukcji aż do zakończenia implementacji. Następnie przechodzi na prawą stronę litery "V", gdzie wykonuje się kolejne etapy weryfikacji, poruszając się w górę aż do końcowej akceptacji przez klienta. Etap weryfikacji wyróżnia testy modułowe, integracyjne, systemowe oraz akceptacyjne (model V może też być skonstruowany z innych bloków czynności i innych poziomów/etapów testowania).

Testy modułowe. Testowanie pojedynczych jednostek systemu, na przykład klas lub modułów.

Testy integracyjne. Testowanie współdziałania między jednostkami, na przykład komunikacji pomiędzy warstwą frontend a warstwą backend.

Testy systemowe. Testowanie systemu pod kątem założonych wymagań wynikających z modelu przypadku użycia.

Testy akceptacyjne. Ostateczne testowanie funkcjonalności systemu, realizowane wraz z klientem.

6.2 Przebieg testów

W poniższym podrozdziale opisano przebieg testów oraz ujawniono błędy wykryte podczas procesu testowania.

6.2.1 Testy modułowe

W trakcie tworzenia warstwy backend, zostały utworzone testy jednostkowe dla poziomu repozytorium, mające na celu przetestowanie natywnych zapytań SQL. Natywne zapytania SQL były konieczne przy pobieraniu danych z wielu tabel. Po wykryciu błędów przeprowadzono proces debugowania kodu. Ujawniono dzięki temu błędy związane z :

- Niepoprawnym wynikiem zapytań. Spowodowane błędnym sformułowaniem warunku w klauzuli "WHERE".
- Błąd w składni zapytania SQL. Spowodowany pojawieniem się literówki w nazwie kolumny "height" w tabeli "BodyMeasurements".
- Problemem z typem danych. Spowodowane niezgodnym typem obiektu transferu danych a typem zdefiniowany w klasie encji.

Wszystkie wykryte błędy zostały naprawione, co zwiększyło niezawodność działania repozytoriów oraz pobierania danych z bazy.

Przykładowy test jednostkowy sprawdzający poprawność pobieranych danych ilustruje poniższy fragment kodu przedstawiony na listingu 6.1.

Listing 6.1: Przykładowy test jednostkowy.

```
1 @Test
2     void findAllEatDeadlinesByUserIdTest () {
3         Integer userId = userRepository.findAll().get(0).
4             getId();
5         List<EatDeadlineDTO> deadlines = eatDeadlineRepository.
6             findAllEatDeadline(userId);
7         assertNotNull(deadlines, "The deadline list should not
8             be null");
9         assertFalse(deadlines.isEmpty(), "The deadline list
10             should not be empty");
11         EatDeadlineDTO deadlineDTO = deadlines.get(0);
12         assertNotNull(deadlineDTO.getRecipeId(), "Recipe ID
13             should not be null");
14         assertEquals("Cheese toast", deadlineDTO.getTitle(), "
15             The title should be 'Cheese toast'");
16         assertEquals("Breakfast", deadlineDTO.getTypeOfMeal(), "
17             Meal type should be 'Breakfast'");
18         assertFalse(deadlineDTO.isDone(), "Status 'done' should
19             be false");
20         assertEquals(10, deadlineDTO.getRate(), "The rating
21             should be 10");
```

```

13     assertEquals("Delicious toast!!", deadlineDTO.getComment
14         (), "The comment should be 'Delicious toast!!'");
    }

```

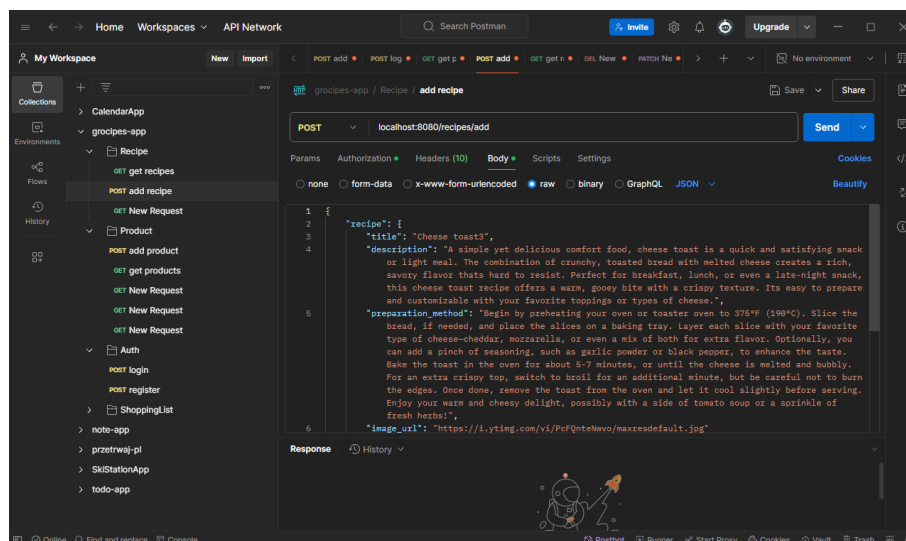
6.2.2 Testy integracyjne

W testach przeprowadzono weryfikację współdziałania warstwy frontend i backend oraz poprawności komunikacji między komponentami systemu. Gdzie między innymi przetestowano działanie mechanizmu autoryzacji oraz poprawność punktów końcowych API warstwy backend. Sprawdzając, czy poprawnie blokuje treści systemu w przypadku braku tokena JWT lub w przypadku jego wygaśnięcia oraz w przypadku przesłania formularza z danymi użytkownika, który nie istnieje w systemie.

Do testowania warstwy frontend użyto narzędzi debugowania przeglądarki internetowej. Ujawniono dzięki temu błędy związane z :

- Cross-Origin Resource Sharing. Spowodowane źle skonfigurowanymi ustawieniami bezpieczeństwa warstwy backend.
- Błąd w ścieżce URL. W serwisie komunikującym się z API podano niepoprawny adres URL.
- Błąd z przesyłaniem niepoprawnego JSON-a. W obiekcie JSON brakowało jednego z wymaganych pól przez co warstwa backend odbierała nieprawidłowe żądanie.

Do testowania warstwy backend użyto narzędzia Postman pozwalającego na wysyłanie zapytań do punktów końcowych API. Ilustruje to zrzut ekranu przedstawiony na rysunku 6.2. Gdzie sprawdzono, między innymi, czy dany punkt końcowy API poprawnie zapisuje, usuwa, edytuje lub pobiera dane z bazy danych, lub czy dany punkt końcowy wykona się bez odpowiedniego tokena JWT.



Rysunek 6.1: Interfejs Postmana.

Ujawniono dzięki temu błędy związane z :

- Błąd zapisywania danych. Spowodowany brakiem fragmentu kodu, zapisujący rekord w bazie danych.
- Błąd obliczania wartości TDEE (ang. Total Daily Energy Expenditure) i BMR (ang. Basal Metabolic Rate). Spowodowane przez ustawienie sztywnych wartości w argumentach funkcji.

Testy integracyjne pozwoliły na wykrycie kluczowych problemów oraz ich skuteczne rozwiązanie, co zwiększyło stabilność całego systemu.

6.2.3 Testy systemowe

W testach zweryfikowano, czy system spełnia wymagania funkcjonalne i нефunkcjonalne określone na etapie analizy. Przetestowano działanie każdego scenariusza użytkownika (por. rozdz. 4) oraz przypadku użycia (por. rozdz. 3).

Ujawniono dzięki temu błędy związane z :

- Podczas rejestracji data zapisywana w niepoprawnym formacie.
- Problem z walidacją formularza rejestracyjnego, skutkujący możliwością wprowadzenia niepoprawnego adresu mail
- Problem z wyświetlaniem elementów list zakupowych w interfejsie użytkownika po ich edycji lub usunięciu.

Powyższe błędy naprawiono.

6.2.4 Testy akceptacyjne

Zostały przeprowadzone wraz z promotorem wcielającym się w rolę klienta i miały na celu ostateczną weryfikację systemu. Przetestowano aplikację pod kątem zrealizowania funkcjonalności kluczowych elementów systemu. Sprawdzono intuicyjność interfejsu użytkownika oraz spełnienie wymagań opisanych w (por. rozdz. 3). Rezultat testów akceptacyjnych potwierdził, że system spełnia wszystkie określone wymagania.

Rozdział 7

Podsumowanie i wnioski

W ramach referowanej pracy inżynierskiej utworzona została aplikacja webowa pomagająca użytkownikowi w planowaniu posiłków, optymalizowaniu zakupów spożywczych i realizowaniu celów żywieniowych. System aplikacji umożliwia użytkownikowi przeglądanie przepisów i artykułów spożywczych, monitorowanie wartości odżywczych oraz tworzenie list zakupowych i celów żywieniowych. System został stworzony przy pomocy dwóch frameworków: frameworka Angular dla warstwy frontend oraz frameworka Spring dla warstwy backend. Dane aplikacji są trzymane w relacyjnej bazie danych MySQL. Aplikacja została umieszczona w kontenerze przy pomocy programu Docker. Dodatkowo aplikacja została przetestowana za pomocą podejścia opartego na modelu "V".

Niezależnie od spełnienia wszystkich wymagań i zrealizowania wszystkich zaplanowanych scenariuszy, istnieje szereg potencjalnych funkcji, o które można by rozszerzyć opracowaną aplikację. Przyszłe wersje aplikacji mogą poprawić działanie aktualnie zastosowanych rozwiązań (na przykład usprawnić algorytm sugerowania przepisów), a także wprowadzić nowe. Jednym z kierunków rozwoju jest rekomendowanie optymalnych planów dietetycznych dostosowanych do realizowanych celów użytkownika. Aplikacja mogłaby również wyświetlać i porównywać ceny artykułów spożywczych z różnych sklepów, co pozwoliłoby użytkownikowi na zakup produktu o najniższej cenie. Dodatkowo można by rozbudować aplikację o asystenta sztucznej inteligencji, który analizowałby nawyki żywieniowe użytkownika.

Podczas realizacji projektu napotkano kilka problemów technicznych, których rozwiązanie wymagało poświęcenia większej ilości czasu i uwagi. Jednym z nich był problem z odświeżaniem treści w interfejsie graficznym użytkownika, po usunięciu lub dodaniu elementu. Problem rozwiązano przy użyciu odpowiedniego mechanizmu aktualizacji, który przy wykryciu zmiany ponownie pobierał dane z bazy danych. Wystąpiły także problemy związane z konfiguracją zabezpieczeń, powodujące blokowanie wysyłanych zapytań do warstwy backend przez przeglądarkę internetową. Rozwiązaniem problemu było odpowiednie skonfigurowanie ustawień bezpieczeństwa warstwy backend. Ponadto przy dodawaniu biblioteki rysującej wykresy i statystyki w harmonogramach wystąpił problem

z instalacją, uniemożliwiający ich skuteczne dodanie do aplikacji.

Podczas realizacji pracy inżynierskiej autor rozwinął umiejętności związane z projektowaniem i implementacją aplikacji internetowych; wykorzystał w praktyce zdobytą w trakcie studiów wiedzę z zakresu inżynierii oprogramowania i baz danych. Utrwalił wiedzę na temat technologii takich, jak Angular i Spring Framework.

Bibliografia

- [1] URL: <https://www.fitatu.com/> (term. wiz. 29.01.2025).
- [2] URL: <https://www.yazio.com/en> (term. wiz. 29.01.2025).
- [3] URL: <https://listonic.com/pl/> (term. wiz. 29.01.2025).
- [4] URL: <https://www.ovhcloud.com/pl/learn/what-is-rest-api/> (term. wiz. 20.01.2025).
- [5] URL: <https://tom-collings.medium.com/controller-service-repository-16e29a4684e5> (term. wiz. 20.01.2025).
- [6] URL: <https://tester.milenabednarczyk.pl/modele-wytwarzania-oprogramowania/> (term. wiz. 23.01.2025).
- [7] Ben Forta. *SQL w mgnieniu oka. Opanuj język zapytań w 10 minut dziennie. Wydanie V*. Helion, 2020.
- [8] Adam Freeman. *TypeScript 4. Od początkującego do profesjonalisty. Wydanie II*. Helion, 2022.
- [9] Cay S. Horstmann. *Java. Podstawy. Wydanie XI*. Helion, 2019.
- [10] Robert C. Martin. *Czysty kod. Podręcznik dobrego programisty*. Helion, 2010.
- [11] Damian Maślanka. *Co to jest back-end i za co odpowiada?* 2022. URL: <https://webmakers.expert/blog/co-to-jest-back-end-i-za-co-odpowiada> (term. wiz. 20.01.2025).
- [12] Damian Maślanka. *Co to jest front-end i za co odpowiada?* 2022. URL: <https://webmakers.expert/blog/co-to-jest-front-end> (term. wiz. 20.01.2025).
- [13] Pitchford Paul. *Odżywianie dla zdrowia*. Wydawnictwo Galaktyka, 2008.

Dodatki

Spis skrótów i symboli

JSON - ang. *JavaScript Object Notation*

JWT - ang. *JSON Web Token*

SPA - ang. *Single page application*

URL - ang. *Uniform resource locator*

REST - ang. *Representational state transfer*

API - ang. *Application programming interface*

BIOS - ang. *Basic Input/Output System*

TDEE - ang. *Total Daily Energy Expenditure*

BMR - ang. *Basal Metabolic Rate*

UML - ang. *Unified Modeling Language*

HTTP - ang. *Hypertext Transfer Protocol*

HTML - ang. *HyperText Markup Language*

CSS - ang. *Cascading Style Sheets*

SQL - ang. *Structured Query Language*

WLS - ang. *Windows Subsystem for Linux*

CRUD - ang. *Create, Read, Update, Delete*

Źródła

```
1 services:
2   mysql:
3     image: 'mysql:latest'
4     environment:
5       - 'MYSQL_DATABASE=grocipesdb'
6       - 'MYSQL_ROOT_PASSWORD='
7     ports:
8       - '33062:3306'
9
10  backend:
11    build:
12      context: ./Backend/grocipes
13      dockerfile: Dockerfile
14    environment:
15      - 'SPRING_DATASOURCE_URL=jdbc:mysql://mysql:3306/
16        grocipesdb'
17      - 'SPRING_DATASOURCE_USERNAME=root'
18      - 'SPRING_DATASOURCE_PASSWORD='
19    ports:
20      - '8080:8080'
21    depends_on:
22      - mysql
23  frontend:
24    build:
25      context: ./Frontend/Grocipes
26      dockerfile: Dockerfile
27    ports:
28      - '80:80'
29    depends_on:
30      - backend
```

³¹ `volumes :`

³² `mysql-data :`

Lista dodatkowych plików, uzupełniających tekst pracy

W systemie do pracy dołączono dodatkowe pliki zawierające:

- kod źródłowy aplikacji,
- dane testowe,
- film pokazujący działanie opracowanego oprogramowania,

Spis rysunków

3.1	Diagram przypadków użycia.	8
4.1	Zrzut ekranu przedstawiający widok listy artykułów spożywczych.	28
4.2	Zrzut ekranu przedstawiający widok szczegółów produktu.	29
4.3	Zrzut ekranu przedstawiający wyszukiwanie produktu.	29
4.4	Zrzut ekranu przedstawiający wyszukanie produktu.	30
4.5	Zrzut ekranu przedstawiający widok listy z przepisami.	30
4.6	Zrzut ekranu przedstawiający widok szczegółu przepisu.	31
4.7	Zrzut ekranu przedstawiający wyszukiwanie przepisu.	31
4.8	Zrzut ekranu przedstawiający wyszukanie przepisu.	32
4.9	Zrzut ekranu przedstawiający dodawanie przepisu do harmonogramu ży- wienia i zakupowego.	32
4.10	Zrzut ekranu przedstawiający widok strony głównej.	33
4.11	Zrzut ekranu przedstawiający widok harmonogramu żywieniowego.	33
4.12	Zrzut ekranu przedstawiający widok harmonogramu zakupowego.	34
4.13	Zrzut ekranu przedstawiający dodanie zdjęcia gotowego posiłku.	34
4.14	Zrzut ekranu przedstawiający dodanie oceny i komentarza posiłku.	35
4.15	Zrzut ekranu przedstawiający widok listy z zakupami.	36
4.16	Zrzut ekranu przedstawiający formularz tworzenia listy zakupowej.	36
4.17	Zrzut ekranu przedstawiający widok listy zakupowej po utworzeniu nowej listy zakupowej.	37
4.18	Zrzut ekranu przedstawiający widok listy zakupowej.	37
4.19	Zrzut ekranu przedstawiający widok podglądu listy zakupowej.	38
4.20	Zrzut ekranu przedstawiający widok aktywnego celu żywieniowego.	38
4.21	Zrzut ekranu przedstawiający formularz utworzenia celu żywieniowego.	39
4.22	Zrzut ekranu przedstawiający widok po utworzeniu celu żywieniowego.	40
4.23	Zrzut ekranu przedstawiający podgląd celu żywieniowego.	40
4.24	Zrzut ekranu przedstawiający widok profilu użytkownika.	41
4.25	Zrzut ekranu przedstawiający formularz aktualizacji parametrów ciała.	41
4.26	Zrzut ekranu przedstawiający historię pomiarów.	42
4.27	Zrzut ekranu przedstawiający panel admina.	42

4.28	Zrzut ekranu przedstawiający widok zarządzania przepisami.	43
4.29	Zrzut ekranu przedstawiający formularz utworzenia przepisu.	43
4.30	Zrzut ekranu przedstawiający widok zarządzania artykułami spożywczymi.	44
4.31	Zrzut ekranu przedstawiający formularz utworzenia artykułu spożywczego.	44
5.1	Zrzut ekranu przedstawiający diagram komponentów.	45
5.2	Zrzut ekranu przedstawiający diagram klas.	46
5.3	Zrzut ekranu przedstawiający diagram związków encji.	48
6.1	Interfejs Postmana.	61

Spis tabel

3.1	Rejestracja	9
3.2	Logowanie	9
3.3	Weryfikacja użytkownika	9
3.4	Wyświetlanie błędu logowania	10
3.5	Przeglądanie artykułów spożywczych	10
3.6	Przeglądanie przepisów	10
3.7	Przeglądanie kalendarza	11
3.8	Dodawanie posiłku do kalendarza	11
3.9	Dodawanie zdjęcia posiłku	12
3.10	Ocena posiłku	12
3.11	Usuwanie zdjęcia posiłku	12
3.12	Dodawanie komentarza posiłku	13
3.13	Usuwanie komentarza posiłku	13
3.14	Usuwanie posiłku z kalendarza	14
3.15	Tworzenie listy zakupowej	14
3.16	Edytowanie listy zakupowej	15
3.17	Usuwanie listy zakupowej	15
3.18	Dodawanie artykułów spożywczych do listy zakupowej	16
3.19	Usuwanie artykułów spożywczych z listy zakupowej	16
3.20	Dodawanie celu żywieniowego	17
3.21	Usuwanie celu żywieniowego	17
3.22	Aktualizowanie parametrów ciała	18
3.23	Przeglądanie historii parametrów ciała	18
3.24	Dodawanie przepisu	19
3.25	Edytowanie przepisu	19
3.26	Usuwanie przepisu	20
3.27	Dodawanie artykułu spożywczego	20
3.28	Edytowanie artykułu spożywczego	21
3.29	Usuwanie artykułu spożywczego	21

Spis listingów

5.1	Przykładowa encja.	49
5.2	Przykładowy kontroler.	51
5.3	Przykładowe repozytorium.	53
5.4	Przykładowy serwis warstwy backend.	53
5.5	Przykładowy serwis warstwy frontend.	56
5.6	Przykładowy interceptor.	57
5.7	Przykładowy komponent. Plik TypeScript.	57
5.8	Przykładowy komponent. Plik HTML.	58
6.1	Przykładowy test jednostkowy.	60