Politechnika Warszawska

WYDZIAŁ MATEMATYKI
I NAUK INFORMACYJNYCH

Praca dyplomowa inżynierska

na kierunku Informatyka i Systemy Informacyjne

Opracowanie i implementacja systemu wielowarstwowego służącego do oceny dań oraz restauracji

Wojciech Klusek

Numer albumu 305943

Aleksander Kuś

Numer albumu 305855

promotor

dr inż. Paweł Kotowski

WARSZAWA 2023

Streszczenie

Opracowanie i implementacja systemu wielowarstwowego służącego do oceny dań oraz restauracji

Praca ta zawiera szczegółowy opis systemu "FoodReview" do oceny dań i restauracji. System ma na celu umożliwienie użytkownikom ocenę zarówno restauracji, jak i dań dla popularnych restauracji sieciowych. Składa się on z aplikacji mobilnej, webowej oraz serwerowej. Aplikacja mobilna, przeznaczona dla klientów i gości, udostępnia możliwość oceny dań i restauracji oraz możliwość przeglądania ocen innych użytkowników w celu wyboru dania. Udostępniony jest również interfejs webowy jako panel administracyjny, skąd administratorzy będą mogli zarządzać elementami systemu oraz przeglądać statystyki. Połączenie wyżej wymienonych aplikacji z bazą danych zapewnia aplikacja serwerowa.

W poniższym dokumencie na wstępie opisany jest problem, który system ma rozwiązać. Następnie omówione są podobne rozwiązania istniejące na rynku i przedstawione jest rozwiązanie problemu oraz podział prac w projekcie. Następnie znajduje się opis systemu od strony biznesowej, który zawiera historyjki użytkowników, przypadki użycia oraz wymagania niefunkcjonalne.

Po opisie biznesowym opisana jest architektura systemu wraz z podziałem na moduły oraz opisem komunikacji między nimi. Dalej omówione są wykorzystane technologie w każdym z modułów oraz przedstawione są wybrane widoki aplikacji mobilnej i webowej. Następnie znajduje się opis testów systemu, które dzielą się na testy jednostkowe, integracyjne oraz akceptacyjne. Po testach zawarta jest instrukcja instalacji dla każdego z modułów.

Na koniec przedstawione są wnioski dotyczące ogólnej oceny projektu, możliwości rozwoju systemu oraz opis potencjalnych zagrożeń oraz problemów dotyczących zaimplementowanego systemu.

Słowa kluczowe: system wielowarstwowy, ocena dań i restauracji, aplikacja mobilna, aplikacja webowa, web API, ASP.NET, Angular, React Native

Abstract

Development and implementation of a multi-layered system to rate meals and restaurants

This thesis provides a detailed description of the "FoodReview" system for rating dishes and restaurants. The system is designed to allow users to rate both restaurants and dishes for popular restaurant chains. The described solution consists of a mobile application, a web application and a server application. The mobile application, designed for customers and guests, will provide the ability to rate dishes and restaurants, as well as the ability to view other users' ratings in order to select a dish. A web interface is also provided as an administrative panel, from where administrators will be able to manage system elements and view statistics. The connection of the above-mentioned applications with the database is provided by the server application.

In the following document at the beginning, the problem the system is designed to solve is described. Then similar solutions existing on the market are discussed. Later, the solution to the presented problem and the division of work in the project are presented. Then there is a description of the system from the business side, which includes user stories, use cases and non-functional requirements.

The business description is followed by a description of the system architecture with a breakdown of the modules and a description of the communication between them. The technologies used in each module are discussed next, followed by selected views of the mobile and web applications. Next, system tests are described, which are divided into unit, integration and acceptance tests. After the tests, installation instructions for each module are included.

Finally, conclusions are presented on the overall evaluation of the project, the possibility of developing the system, and a description of potential risks and problems with the implemented system.

Keywords: multi-layered system, dishes and restaurants rating, mobile application, web application, web API, ASP.NET, Angular, React Native

Spis treści

1. Wstęp

W tym rozdziale dokładnie opiszemy problem, który chcieliśmy rozwiązać projektując nasz system. Opowiemy o istniejących rozwiązaniach i ich wadach oraz przedstawimy, jak nasz system adresuje postawiony problem. Podamy również podział prac w pisaniu i projektowaniu samego systemu oraz tego dokumentu.

1.1. Opis problemu

W dzisiejszych czasach ważną rolę w wyborze restauracji, w której chcemy zjeść, pełnią oceny i recenzje użytkowników w Internecie. Dzięki nim możemy szybko odrzucić lokale ze słabszymi ocenami i skierować się w kierunku lepszych restauracji. Niestety we wszystkich popularnych aplikacjach do wystawiania takich recenzji użytkownik ma możliwość jedynie oceny danego lokalu, a nie poszczególnych dań. Co za tym idzie, użytkownik nie wie, jakie dania są najlepsze w danej restauracji. Sprawia to pewne problemy nowym klientom restauracji, których często wita duże menu nad kasami z dużą ilością dań, z którego wybór może być problematyczny.

Innym problemem jest to, że klienci często wystawiają oceny restauracji na podstawie dań, które jedli, co może być mylące, patrząc na ocenę restauracji. W takim wypadku tylko jakość popularnych dań decyduje o końcowej ocenie restauracji. Istnieje również możliwość, że restauracja ma kilka dobrych dań i resztę słabszych pozycji, a kierując się tylko oceną restauracji nie wiemy, które danie wybrać, aby uniknąć gorszych potraw. Jeżeli chcemy dowiedzieć się czegoś więcej o poszczególnych daniach, jesteśmy zmuszeni szukać w komentarzach pod ocenami, co nie jest zbyt wygodne.

1.2. Istniejące rozwiązania

Na rynku istnieją już serwisy służące do oceny restauracji. Najpopularniejszym z nich jest bez wątpienia Google Maps, jednak to rozwiązanie posiada wszystkie wady opisane w poprzedniej sekcji. Brak ocen poszczególnych dań może utrudnić nam wybór właściwej restauracji.

Innymi przykładami serwisów umożliwiających ocenę restauracji są serwisy takie jak Pyszne.pl, służące do zamawiania jedzenia z możliwością oceny restauracji. Często wyświetlają one najpopularniejsze dania oraz oceny restauracji, lecz również nie możemy znaleźć informacji o ocenie wybranego dania, co przy zamawianiu jedzenia w dowozie również jest bardzo ważne. Zaletą tych serwisów jest wyświetlanie popularnych restauracji w okolicy co pozwala nam znaleźć dobrze oceniane jedzenia w pobliżu miejsca, w którym przebywamy.

1.3. Proponowane rozwiązanie

Jako odpowiedź na wyżej opisane problemy zaprojektowaliśmy system, w którym istnieje możliwość oceny zarówno restauracji, jak i samych dań w popularnych restauracjach sieciowych. System składa się z trzech części:

- Aplikacja mobilna - przeznaczona dla klientów i gości,
- Aplikacja webowa - przeznaczona dla administratorów,
- Serwer - program komunikujący się z dwoma wyżej wymienionymi aplikacjami i bazą danych.

W aplikacji mobilnej użytkownicy mogą oceniać dania i restauracje w wielostopniowej skali oraz mają dostęp do opinii innych użytkowników w celu wybrania najlepszego dania w restauracji. Na głównej stronie dostępny jest ”feed” z listą proponowanych dań na podstawie ostatnich ocen użytkowników. Aplikacja oferuje wyszukiwanie restauracji, dań i użytkowników z możliwością filtrowania wyników po tagach oraz sortowania. Dostępna jest także funkcjonalność zarządzania profilem dla zalogowanych użytkowników.

Udostępniony jest również interfejs webowy jako panel administracyjny, który daje możliwość zarządzania listą restauracji, dań, użytkowników oraz ocen. Administratorzy będą musieli dbać o aktualność bazy danych dań oraz dodawać nowe restauracje, aby aplikacja mogła być wykorzystywana przez jak największe grono użytkowników. Konieczne będzie również moderowanie użytkowników, usuwając oceny zawierające niechciane treści oraz blokując użytkowników wielokrotnie łamiących regulamin. W tym panelu dostępne są również statystyki systemu.

System skierowany jest głównie do klientów popularnych restauracji sieciowych. Osoby bywające często w takich lokalach będą mogły oceniać dania, które im smakowały, i potem, patrząc na swoje oceny, zamówić to samo po jakimś czasie. Aplikacja może być również wykorzystywana przez ludzi po raz pierwszy idących do danej restauracji i chcących się zorientować, co warto

wybrać.

1.4. Podział prac

Z uwagi na znajomość różnych technologii przez członków zespołu, dokonaliśmy następujący podział przy projektowaniu i implementacji opisywanego systemu:

- Wojciech Klusek - Aplikacja mobilna dla klientów, część aplikacji serwerowej komunikująca się z aplikacją mobilną, część aplikacji serwerowej odpowiadająca za autoryzację, autentykację i logowanie użytkowników.
- Aleksander Kuś - Aplikacja webowa dla administratorów, część aplikacji serwerowej komunikująca się z aplikacją webową, konteneryzacja i integracja projektu z serwisami Azure.

Przy pisaniu tego dokumentu podzieliliśmy się następująco:

- Wojciech Klusek - abstrakt, architektura systemu, widoki aplikacji mobilnej i webowej, testy systemu.
- Aleksander Kuś - wstęp, opis biznesowy, wykorzystywane technologie, wnioski.

2. Opis biznesowy

W tym rozdziale przedstawiony zostanie opis biznesowy systemu, a w nim opis wymagań funkcjonalnych w postaci szczegółowo opisanych historyjek użytkownika, diagram przypadków użycia z podziałem na moduły oraz opis wymagań niefunkcjonalnych. Skorzystaliśmy z metody FURPS [2], czyli modelu klasyfikacji wymagań dzielących je na obszary funkcjonalności, używalności, niezawodności, wydajności i wsparcia.

2.1. Historyjki użytkowników

Zaczniemy od określenia wymagań funkcjonalnych systemu, opisujących zadania, które nasz system powinien wykonywać. Do tego celu najpierw użyjemy historyjek użytkowników, które jasno określają wymagane zachowanie systemu z punktu widzenia poszczególnych aktorów. W naszym systemie warto rozróżnić użytkowników zalogowanych i niezalogowanych (odpowiednio klient i gość) oraz administratorów.

2.1.1. Klient i gość

Dla zwykłych użytkowników, korzystających z systemu za pomocą aplikacji mobilnej, wyszczególniliśmy następujące historyjki użytkownika:

1. Jako gość chcę móc założyć konto, aby oceniać dania i restauracje - zakładanie konta odbywa się poprzez podanie adresu email, loginu i hasła, w aplikacji mobilnej. Posiadanie konta w systemie jest warunkiem koniecznym wystawiania ocen dań i restauracji.

2. Jako klient/gość chcę móc przeglądać oceny dań i restauracji, aby móc wybrać najlepsze jedzenie - przeglądanie odbywa się w aplikacji mobilnej, gdzie istnieje możliwość sortowania i filtrowania po tagach.

3. Jako klient/gość chcę móc wyszukiwać dania, restauracje i użytkowników, aby wybrać dobrze oceniane danie z menu - wyszukiwanie danych odbywa się w aplikacji mobilnej, gdzie istnieje osobna strona do wyszukiwania podzielona na sekcję dla dań, restauracji oraz

użytkowników. Po wyszukaniu restauracji wyświetla nam się lista wszystkich dostępnych dań wraz z ocenami poszczególnych dań oraz samej restauracji.

4. Jako klient/gość chcę móc przeglądać ”feed” z daniami, aby zobaczyć sugerowane dania - ”Feed” wyświetlany jest na głównej stronie aplikacji mobilnej i zawiera najczęściej oceniane dania z różnych restauracji. Może on być pomocny użytkownikowi przy wyborze restauracji i dania.

5. Jako klient chcę móc zarządzać swoim profilem, aby był aktualny - ekran zmiany danych profilowych znajduje się w aplikacji mobilnej i oferuje możliwość zmiany podstawowych danych i zdjęcia.

6. Jako klient chcę móc usunąć swoje konto, aby usunąć swoje dane z systemu - opcja dostępna z aplikacji mobilnej. Po jej wybraniu dane klienta są kasowane z bazy danych systemu i nie są potem przechowywane w żaden sposób.

2.1.2. Administrator

Dla administratora historyjki skupiają się na zadaniach związanych z zarządzaniem systemem:

1. Jako administrator chcę móc zarządzać restauracjami, aby baza restauracji była aktualna i szeroka - możliwość dodawania, edycji danych oraz usuwania restauracji udostępniona będzie z aplikacji webowej. Po utworzeniu restauracji będzie ona niewidoczna dla użytkowników, aby administrator miał czas na dodanie dań i poprawę danych. Po zakończeniu procesu dodawania administrator może zmienić status restauracji na ”widoczna”.

2. Jako administrator chcę móc zarządzać daniami, aby lista dań w systemie pokrywała się z menu restauracji - funkcjonalność ta będzie dostępna z poziomu aplikacji webowej. Lista dań powinna być uaktualniana w miarę często z uwagi na częste zmiany menu restauracji sieciowych (promocje sezonowe itp.).

3. Jako administrator chcę móc zarządzać użytkownikami, aby móc blokować użytkowników łamiących regulamin - w aplikacji webowej znajduje się lista wszystkich zarejestrowanych użytkowników. Ważnym zadaniem administratora będzie kontrolowanie zachowania użytkowników i treści wystawianych przez nich ocen, aby blokować użytkowników zamieszczających nieodpowiednie treści.

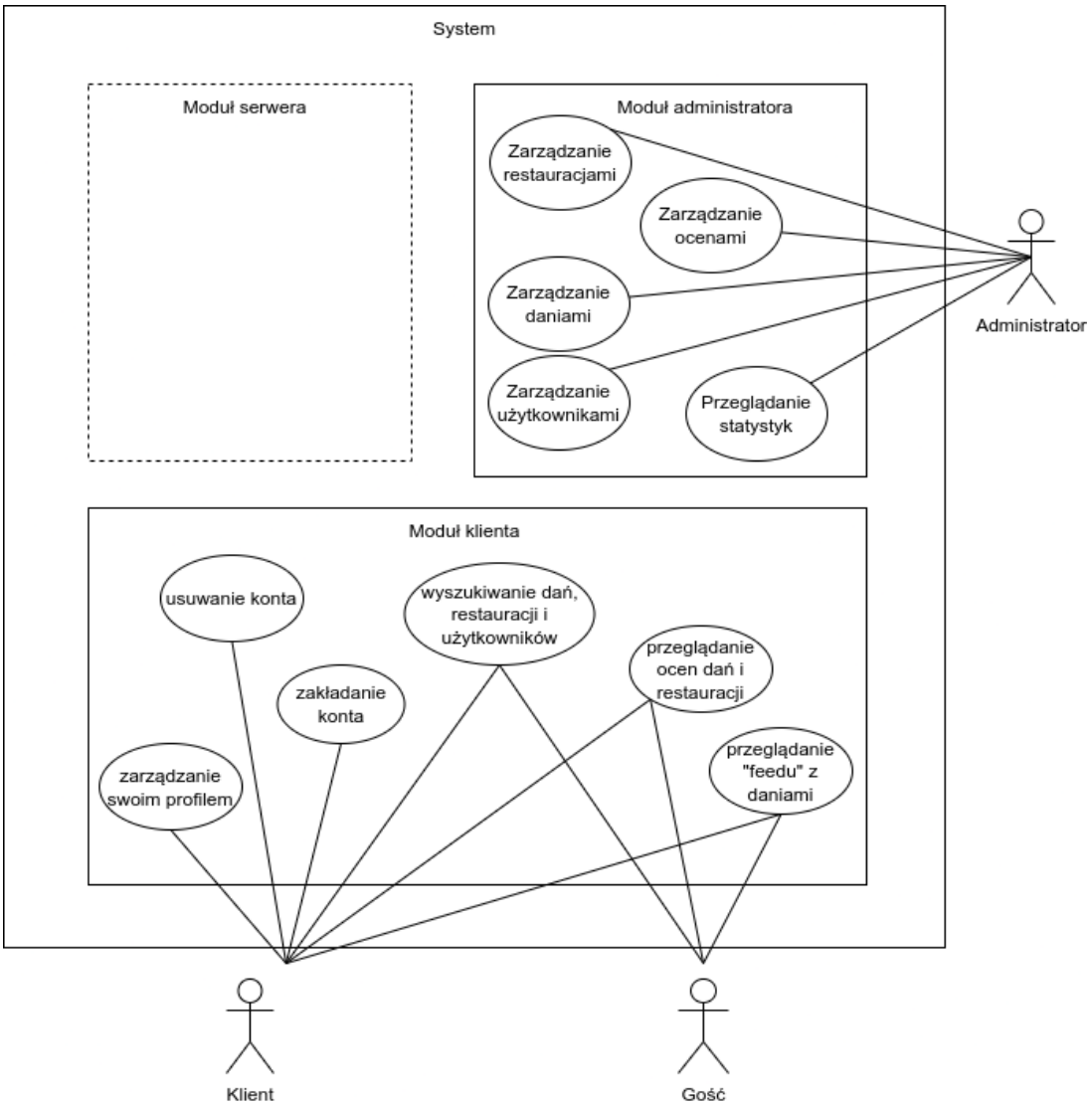
4. Jako administrator chcę móc zarządzać ocenami użytkowników, aby usuwać te zawierające nieodpowiedni opis - oceny zawierające w opisie nieodpowiednie słowa i zwroty powinni

być usuwane przez administratora, a użytkownicy wielokrotnie łamiący regulamin powinni być blokowani. Funkcjonalność ta dostępna jest w tabeli ocen w aplikacji webowej.

5. Jako administrator chcę móc przeglądać statystyki dań i restauracji, aby móc odpowiednio nimi zarządzać - funkcjonalność dostępna w aplikacji webowej.

2.2. Diagram przypadków użycia

Innym sposobem przedstawienia wymagań funkcjonalnych jest diagram przypadków użycia, który pokazuje, w jaki sposób użytkownicy mogą korzystać z różnych części systemu. Przygotowaliśmy taki diagram dla naszej aplikacji z podziałem na moduły klienta, administratora i serwera oraz wyszczególniając trzech aktorów: klienta, gościa i administratora.



Rysunek 2.1: Diagram przypadków użycia w systemie.

2.3. Wymagania niefunkcjonalne

Następnie przedstawiamy wymagania niefunkcjonalne systemu, czyli jak system powinien działać. Do ich przedstawienia użyliśmy tabeli z opisanymi wcześniej obszarami używalności, niezawodności, wydajności i wsparcia.

Obszar wymagań	Nr wymagania	Opis
Używalność	1	Kolorystyka aplikacji webowej i mobilnej powinna być spójna.
Używalność	2	Funkcjonalności systemu powinny być dostępne przez przeglądarki Google Chrome (od wersji 107.0.5304.92) na ekranach o rozdzielczości 1920x1080 oraz na telefonach z systemem Android (od wersji 8.0) i iOS (od wersji 15.0).
Używalność	3	Aplikacja webowa i mobilna powinna być intuicyjna w użyciu.
Niezawodność	4	Aplikacja powinna być dostępna cały czas z możliwymi przerwami technicznymi trwającymi maksymalnie 2 godziny co najwyżej raz na 2 tygodnie.
Niezawodność	5	Aplikacja webowa i mobilna nie powinny być bezpośrednio zależne od siebie.
Niezawodność	6	Awaria aplikacji klienckich nie powinna mieć wpływu na dane przechowywane na serwerze.
Wydajność	7	Wszystkie ekrany w aplikacji mobilnej i webowej powinny się ładować nie dłużej niż 5 sekund.
Wydajność	8	Aplikacja mobilna nie powinna powodować znacznego zużycia baterii.
Wsparcie	9	System powinien się skalować wraz ze wzrostem liczby użytkowników i danych.
Wsparcie	10	Logi aplikacji powinny być zapisywane w celu naprawy awarii.

Tabela 2.1: Tabela wymagań niefunkcjonalnych.

3. Architektura systemu

W tym rozdziale przedstawiona zostanie architektura systemu wraz z opisem modułów i komponentów systemu oraz opisem wewnętrznej komunikacji.

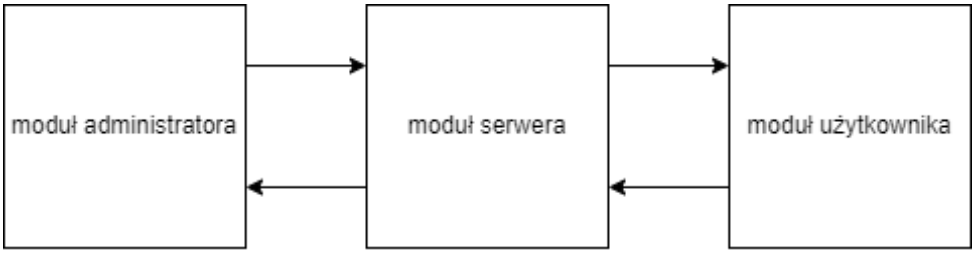
3.1. Podział na moduły

System składa się z trzech modułów:

1. Moduł użytkownika - aplikacja mobilna
2. Moduł administratora - aplikacja webowa
3. Moduł serwera - serwer WWW

Podział na pierwsze 2 moduły wynika wprost z podziału użytkowników systemu na role: użytkownik oraz administrator. Dostęp do modułu administratora uzyskiwany jest poprzez logowanie do konta o przypisanej roli w aplikacji webowej, natomiast dostęp do modułu użytkownika uzyskiwany jest poprzez pobranie aplikacji oraz opcjonalnie zalogowanie się na wcześniej zarejestrowane konto w celu wystawienia oceny lub zarządzania profilem. Moduł serwera jest modułem pośredniczącym między użytkownikiem i administratorem, a bazą danych.

Struktura modułów systemu znajduje się na rysunku poniżej:



Rysunek 3.1: Moduły systemu.

3.1.1. Moduł użytkownika

Moduł ten wykorzystywany jest przez użytkowników aplikacji mobilnej do interakcji z systemem. Dzięki niemu użytkownicy mogą: przeglądać dania i restauracje ocenione przez innych użytkowników oraz je oceniać.

3.1.2. Moduł administratora

Moduł ten wykorzystywany jest przez administratorów korzystających z aplikacji internetowej do interakcji z systemem. Dzięki niemu administratorzy mogą zarządzać restauracjami, daniami, ocenami, użytkownikami oraz przeglądać statystyki systemu.

3.1.3. Moduł serwera

Moduł serwera jest centralnym modułem w tym systemie. Zajmuje się przechowywaniem, modyfikowaniem i udostępnianiem danych o daniach, restauracjach, użytkownikach oraz ocenach. Pośredniczy w komunikacji między modułem administratora, a modułem użytkownika.

3.2. CQRS

CQRS, czyli Command Query Responsibility Segregation, to wzorzec projektowy, którego głównym założeniem jest podział operacji aplikacji na wyszukiwanie i modyfikowanie danych [13]. Powstał w wyniku obserwacji, że często operacje wyszukiwania i modyfikacji danych są różnie zbudowane, działają na różnych kolumnach w bazie danych, mają zupełnie inne potrzeby walidacji przychodzących danych, dane przez nie zwracane znacząco się różnią i mają różną wydajność. W odpowiedzi na te problemy stworzono właśnie przedstawiany wzorzec, czyli CQRS. Z założenia operacje w aplikacji podzielone są na dwie grupy:

- komendy - powinny być skoncentrowane na działaniach, a nie na danych, na przykład ”Usuń restaurację”, ”Zmień dane dania” itd.,
- kwerendy - nigdy nie modyfikują bazy danych, a jedynie zwracają odpowiednie DTO (Data Transfer Object).

Przy użyciu tego wzorca możliwy jest nawet podział bazy danych na dwie części, odpowiedzialne za odpowiednio czytanie oraz modyfikację danych. Bazy te mogą mieć nawet różne schematy (na przykład baza odpowiedzialna za czytanie może posiadać schemat zoptymalizowany pod kątem kwerend). W takim przypadku obie bazy danych muszą być ze sobą zsynchronizowane,

na przykład przy użyciu mechanizmu zdarzeń generowanych przez bazę do modyfikacji danych po każdej takiej modyfikacji. Baza danych odpowiadająca za czytanie może też być kopią tylko do odczytu. Takich kopii w opisywej konfiguracji może być wiele. Zastosowanie tego wzorca daje nam:

- skalowalność niezależnych od siebie części - operacje do odczytu i zapisu mogą być skalowane niezależnie,
- zoptymalizowane schematy do zapisu i odczytu,
- bezpieczeństwo - łatwiej chronić przed niepożądanymi danymi tylko endpointy do zapisu.

Istnieją jednak pewne negatywne strony tego rozwiązania, takie jak:

- większa złożoność aplikacji,
- synchronizacja baz danych poprzez wiadomości.

3.3. Mediator

Wzorzec mediator jest wzorcem projektowym, który ma za zadanie zapewnić sprawną komunikację pomiędzy różnymi klasami lub obiektami bez bezpośredniego połączenia [8].

Wzorzec ten polega na utworzeniu dodatkowej, pośredniczącej klasy lub obiektu, który będzie odpowiedzialny za przekazywanie informacji pomiędzy wyżej wymienionymi obiektami. Dzięki temu można uniknąć tworzenia złożonych relacji pomiędzy poszczególnymi klasami oraz zapewnić lepszą modularność i elastyczność systemu. Wzorzec mediator jest szczególnie przydatny w przypadku, gdy istnieje wiele klas, które ze sobą wzajemnie komunikują się i wymieniają informacje, a ich liczba i złożoność uniemożliwiają utrzymanie ich wszystkich w jednym kodzie lub projekcie.

3.4. Architektura serwera

Moduł serwera będzie udostępniał CQRS (Command Query Responsibility Segregation) API. W naszym systemie nie posunęliśmy się aż tak daleko, aby definiować osobne bazy dla dwóch typów zapytań, ale stosujemy wyżej opisany podział na komendy i kwerendy. Dzięki takiemu podziałowi nie musimy walidować zapytań będących kwerendami, ze względu na założenie, że nie modyfikują one danych.

Struktura aplikacji serwera będzie podzielona na 3 projekty:

1. Kontrakty - zawierają opis przyjmowanych i zwracanych danych w zapytaniach,
2. Domena - klasy odwzorowujące zachowanie aplikacji,
3. Serwisy - wykorzystywane do komunikacji z bazą danych.

W momencie otrzymania zapytania kontrakt przesyła je dalej wykorzystując obiekt mediatora. Jeśli zapytanie jest komendą, to po przejściu walidacji trafia do odpowiedniego serwisu, a jeśli jest kwerendą, to bezpośrednio trafia do serwisu, który, wykorzystując domenę i logikę w niej zawartą, będzie zwracał dane z bazy lub je modyfikował.

3.5. Komunikacja w systemie

Opisany system dzieli się na trzy moduły: moduł użytkownika, administratora, serwera. Kluczowym dla komunikacji jest ostatni z nich, gdyż pośredniczy w każdym przekazie informacji między modułami wewnątrz aplikacji. Tylko moduł serwera danych posiada CQRS API i odbiera zapytania HTTP. Pozostałe moduły, aby modyfikować bazę danych, muszą wysłać zapytanie żądające tej zmiany i otrzymać odpowiedź w postaci potwierdzenia jej zajścia.

Bezpośrednia komunikacja między modułami użytkownika i administratora nie zachodzi. Moduły, aby operować na aktualnych danych, muszą je uzyskać z modułu serwera. Każde wysyłane zapytanie przez administratora musi zawierać w nagłówku otrzymany po zalogowaniu token (uzyskany z modułu serwera), w którym przechowywane są dane o ID użytkownika oraz jego roli.

Dla zapytań z modułu użytkownika token w nagłówku, uzyskany po zalogowaniu się, jest wymagany jedynie w zapytaniach dotyczących wystawiania ocen oraz zarządzania profilem. Przechowywane w tokenie ID użytkownika oraz jego rola określa czy jednostka ma wystarczające uprawnienia do wykonania danego zapytania lub jakie dane należy zwrócić. Przykładowo, jeżeli użytkownik wysyła zapytanie o wyświetlenie danych swojego profilu, na podstawie jego ID zawartego w tokenie zostaną zwrócone odpowiednie dane.

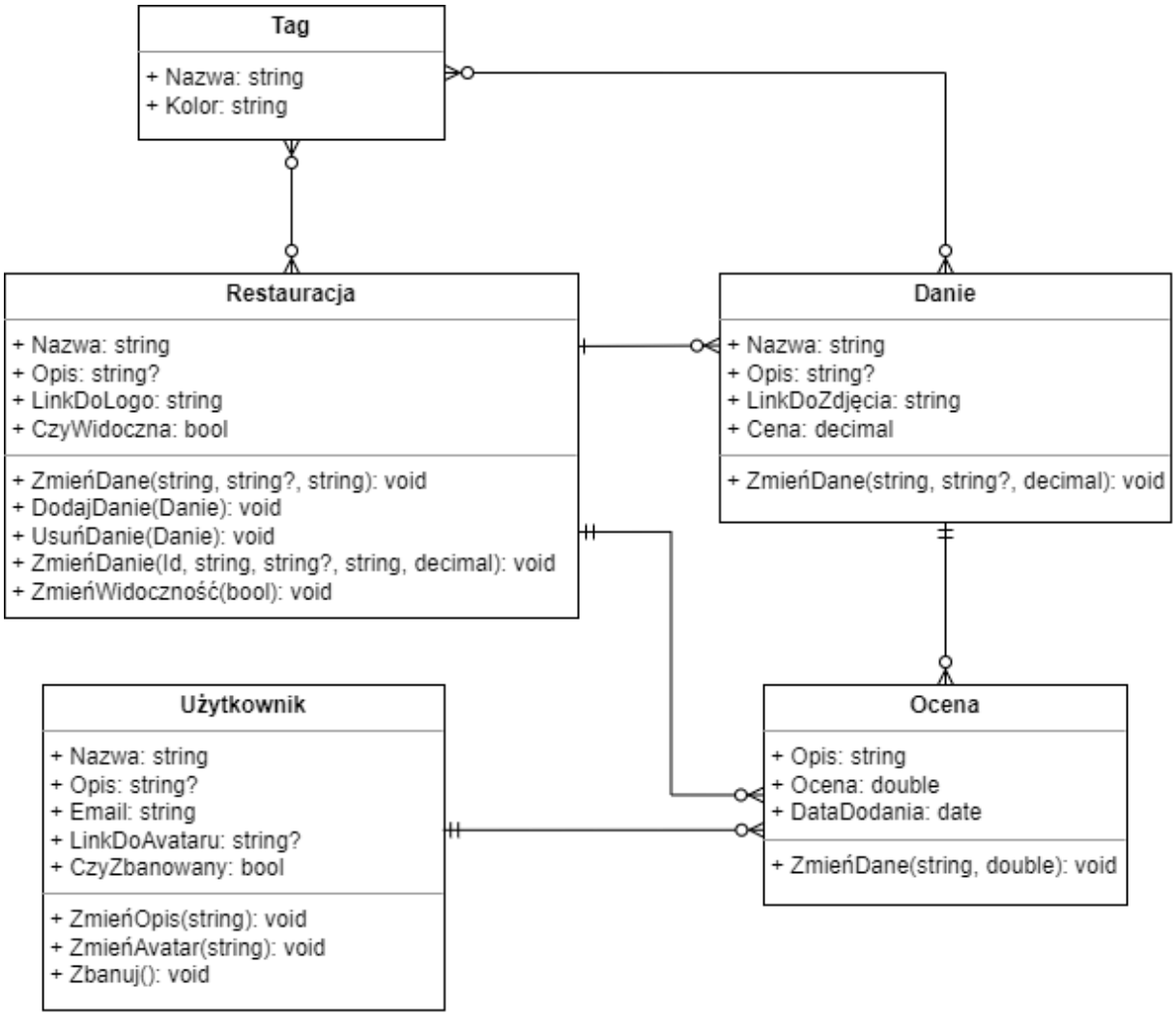
Zapytania mapowane są w następujący sposób:

- Kwerendy - /query/pełna przestrzeń nazw/
- Komendy - /command/pełna przestrzeń nazw/

gdzie pełna przestrzeń nazw to przestrzeń nazw klasy wraz z jej nazwą. Zapytania będą wysyłane z czasownikiem HTTP POST.

3.6. Komponenty systemu

Poniżej przedstawiony jest diagram klas dla opisywanego systemu, przedstawiający strukturę danych w bazie danych.



Rysunek 3.2: Diagram klas

4. Wykorzystane technologie

Ten rozdział zawiera opis wykorzystanych technologii w całej aplikacji. Dla każdego z trzech modułów przygotowaliśmy tabelę z nazwą technologii i jej przeznaczeniem, a pod nią dokładny opis, jak wykorzystujemy daną technologię i dlaczego zdecydowaliśmy się na jej użycie. Na koniec opisaliśmy wykorzystywane przez nas usługi chmurowe platformy Azure.

4.1. Aplikacja serwerowa

Aplikacja serwerowa jest kluczowa z punktu widzenia całego systemu. Jest to również obszar, w którym mamy największe doświadczenie pod względem tworzenia aplikacji. Poniżej przedstawiamy wybrane technologie do stworzenia tego modułu:

Język programowania	C# 9.0
Framework API	ASP.NET
Rozwiązania chmurowe	Azure
Baza danych	Microsoft SQL Server
ORM	Entity Framework Core
Uwierzytelnianie	Identity Server 4
Walidacja	Fluent Validation
Testy jednostkowe	XUnit
Komunikacja między kontraktami a serwisami	MediatR
Konteneryzacja	Docker

Tabela 4.1: Technologie aplikacji serwerowej

Największe znaczenie na kształt tego modułu miał wybór frameworka API, czyli w naszym przypadku ASP.NET. Jest to open-sourcowe rozwiązanie firmy Microsoft, które pozwala w łatwy sposób tworzyć aplikacje API [6]. Wybór tej technologii był łatwy z uwagi za niemal nieustanne wykorzystanie języka C# w całym toku studiów. Wykonaliśmy już kilka projektów wykorzystu-

jącą ASP.NET, więc mamy już doświadczenie z budową i projektowaniem takich aplikacji.

Naszym systemem zarządzania bazą danych (DBMS) został Microsoft SQL Server, który znamy już z wykorzystania w poprzednich projektach. Jest to rozwiązanie dobrze działające z ASP.NET oraz łatwe do wdrożenia na platformie Azure, o czym powiemy na końcu tego rozdziału.

Dostęp do bazy danych zapewnił nam Entity Framework Core, czyli open-sourcowe narzędzie ORM (object-relational mapping) firmy Microsoft [14]. Pozwala ono łatwo tworzyć strukturę bazy danych przy podejściu code-first. Dzięki niej możemy przeszukiwać zawartośc bazy danych oraz ją modyfikować z poziomu języka C# bez konieczności pisania kwerend w języku SQL.

W kwestii autoryzacji, autentykacji, logowania i rejestracji użytkowników użyliśmy IdentityServer 4. Jest to zaawansowane rozwiązanie dla platformy .NET, udostępniające wszystkie wymienione wyżej funkcje [12]. Logowanie i uwierzytelnianie odbywa się za pomocą tokenów JWT, dzięki którym możemy łatwo pracować z naszym API z poziomu aplikacji mobilnej i webowej. Dzięki zastosowaniu tej technologii mamy pewność o bezpieczeństwie naszego API.

Ważnym aspektem napisania dobrego API jest walidacja przychodzących danych w zapytaniach od strony aplikacji frontendowych. Do tego celu użyliśmy paczki FluentValidation, która pozwala na zdefiniowanie kompleksowych reguł, jakie muszą spełniać przychodzące dane.

Do testów jednostkowych wykorzystaliśmy bibliotekę XUnit, a do zrealizowania wzorca mediator w całej aplikacji użyliśmy paczki MediatR.

4.2. Aplikacja mobilna

Aplikacja mobilna jest wykorzystywana przez zalogowanych użytkowników oraz gości do przeglądania i oceniania dań/restauracji. Wykorzystane zostały w niej następujące technologie:

Język programowania	TypeScript
Framework	React Native
Narzędzia	Expo
Biblioteka komponentów	React Native UI Kitten

Tabela 4.2: Technologie aplikacji mobilnej.

React Native jest frameworkiem do tworzenia aplikacji mobilnych dla platform Android i iOS z wykorzystaniem języka JavaScript/Typescript [19]. Pozwala on na tworzenie natywnych aplikacji mobilnych z wykorzystaniem biblioteki React [18]. Jest szczególnie przydatny w przypadku chęci wypuszczenia aplikacji jednocześnie na platformę Android oraz IOS, gdzyż nie istnieje koniecz-

ność tworzenia osobnych wersji aplikacji dla każdej z nich. Sama aplikacja została napisana w języku TypeScript, będącym rozwinięciem języka JavaScript o statyczną kontrolę typów. Zastosowanie tej technologii pozwala na wczesne wykrywanie błędów podczas kompilacji, co ułatwia unikanie wielu niespodziewanych zachowań trudnych do wykrycia w inny sposób.

Wykorzystaliśmy również narzędzia Expo, które umożliwiają tworzenie aplikacji mobilnych bez konieczności instalowania specjalistycznych narzędzi i środowisk programistycznych, takich jak Android Studio czy Xcode [10]. Zamiast tego, Expo udostępnia aplikację mobilną, która pozwala na testowanie i uruchamianie aplikacji bezpośrednio na urządzeniach mobilnych.

Jako bibliotekę do komponentów wykorzystaliśmy React Native UI Kitten, która udostępnia szeroki zakres komponentów interfejsu użytkownika, takich jak: przyciski, pola tekstowe, nawigacja, formularze itp., które można w prosty sposób wykorzystać w aplikacji mobilnej [1].

4.3. Aplikacja webowa

Aplikacja webowa jest wykorzystywana przez administratorów do zarządzania systemem. Zostały w niej zastosowane następujące technologie:

Język programowania	TypeScript
Framework	Angular
Biblioteka komponentów	Angular Material
Rozwiązania chmurowe	Azure
Konteneryzacja	Docker

Tabela 4.3: Technologie aplikacji webowej

Framework Angular to bardzo popularne rozwiązanie, pozwalajace na tworzenie nowoczesnych aplikacji webowych typu SPA (single-page application) [3]. Doświadczenie z wcześniejszych projektów sprawiło, że wybór tej technologii okazał się prosty. Podobnie jak w przypadku aplikacji mobilnej zastosowaliśmy język TypeScript.

Wykorzystaliśmy również bibliotekę komponentów Angular Material, która dostarcza wiele ładnie wyglądających kontrolek, z których można budować aplikacje. Udostępniane komponenty są uniwersalne i wysokiej jakości [15]. Bez użycia bibliteki komponentów musielibyśmy ręcznie stylizować wszystkie komponenty HTML, co byłoby bardzo uciążliwe.

Integracja tak napisanego frontendu z naszym serwerem nie stanowiła większych problemów z uwagi na zastosowanie standardowego protokołu HTTP. Z wykorzystaniem klasy HttpClient

i skonfigurowaniem nagłówków naszych zapytań, aby zawierały token JWT z IdentityServera, mogliśmy łatwo komunikować się z modułem serwera.

4.4. Rozwiązania chmurowe Azure

Przy implementacji systemu skorzystaliśmy z wielu rozwiązań dostępnych na platformie Azure, które pozwoliły łatwo wdrożyć naszą aplikację.

4.4.1. Wykorzystanie Dockera

Przy budowaniu modułów naszego systemu postanowiliśmy skorzystać z kontenerów dockerowych. Jest to technologia pozwalająca "zapakować" wszystkie potrzebne biblioteki i zależności w paczki zwane kontenerami, aby uruchamiać daną aplikację na dowolnym komputerze z zainstalowanym Dockerem. Jest to rozwiązanie powszechnie używane, oferujące wydajność i elastyczność aplikacji [4]. W naszym przypadku mamy dwa kontenery - jeden dla aplikacji serwerowej, drugi dla aplikacji webowej. Rozróżniamy również dwa środowiska - środowisko lokalne i środowisko produkcyjne (platforma Azure).

4.4.2. Użyte serwisy Azure

Aby wdrożyć nasze kontenery na platformie Azure korzystamy z następujących serwisów:

- Resource Group - grupa zasobów niezbędna w serwisie Azure do korzystania z rozwiązań tej platformy.

- Azure SQL Server i Azure SQL Database - hostowana produkcyjna baza danych. Do niej właśnie łączy się aplikacja serwerowa. Na platformie Azure w łatwy sposób można stworzyć bazę korzystającą z Microsoft SQL Server.

- Azure Key Vault - miejsce przechowywania "sekretów", czyli wartości, które nie powinny być przechowywane w repozytorium. Przykładami są choćby "connection string" do bazy danych czy też adres API, do którego ma się łączyć aplikacja webowa. Wartości te są automatycznie wstrzykiwane do kontenerów z naszymi aplikacjami.

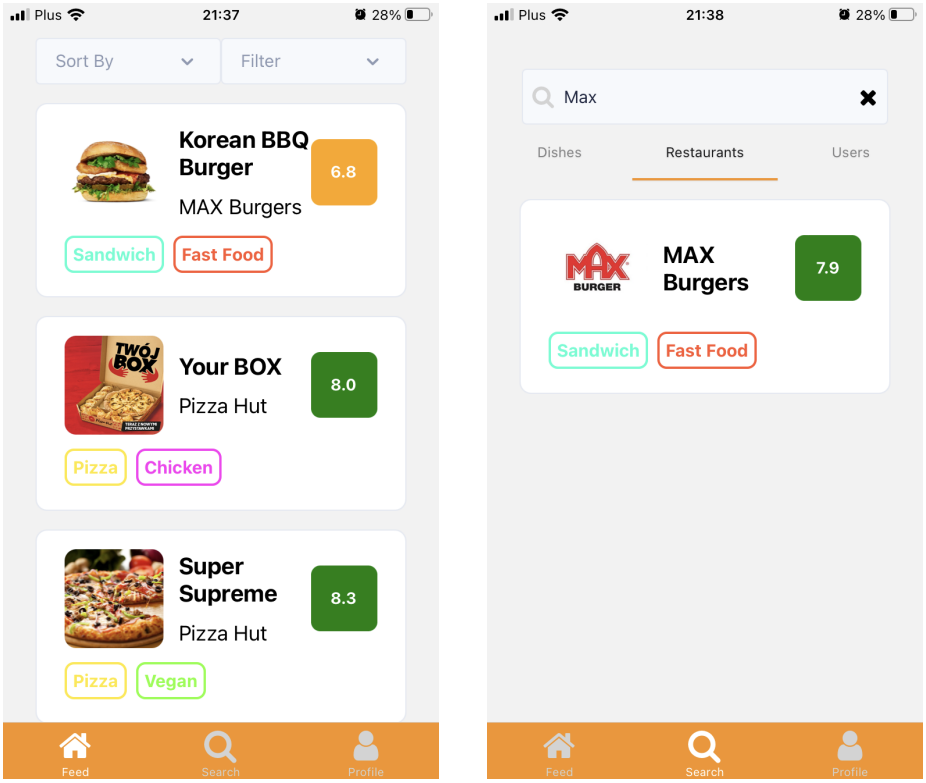
- Azure App Service - serwis pozwalający na hosting aplikacji webowych. W naszym przypadku hostujemy aplikację serwerową i webową w osobnych serwisach. Dzięki hostowaniu nasze aplikacje są dostępne z poziomu Internetu.

Do automatycznego wdrażania naszych aplikacji używamy potoków (Azure Pipelines). Po wprowadzeniu zmian na głównej gałęzi w repozytorium uruchamia się potok, który buduje i wysyła kontenery dockerowe do portalu DockerHub [11]. Następnie sygnalizuje serwisom App Service, aby pobrały najnowszą wersję kontenerów i ją załadowały. Po restarcie serwisów nasze aplikacje zostają zmodyfikowane.

5. Widoki aplikacji mobilnej i webowej

W tym rozdziale przedstawimy, jak wyglądają interfejsy użytkownika aplikacji mobilnej oraz webowej. Kolorystyka aplikacji mobilnej i webowej zostały zsynchronizowane, co daje poczucie spójności systemu.

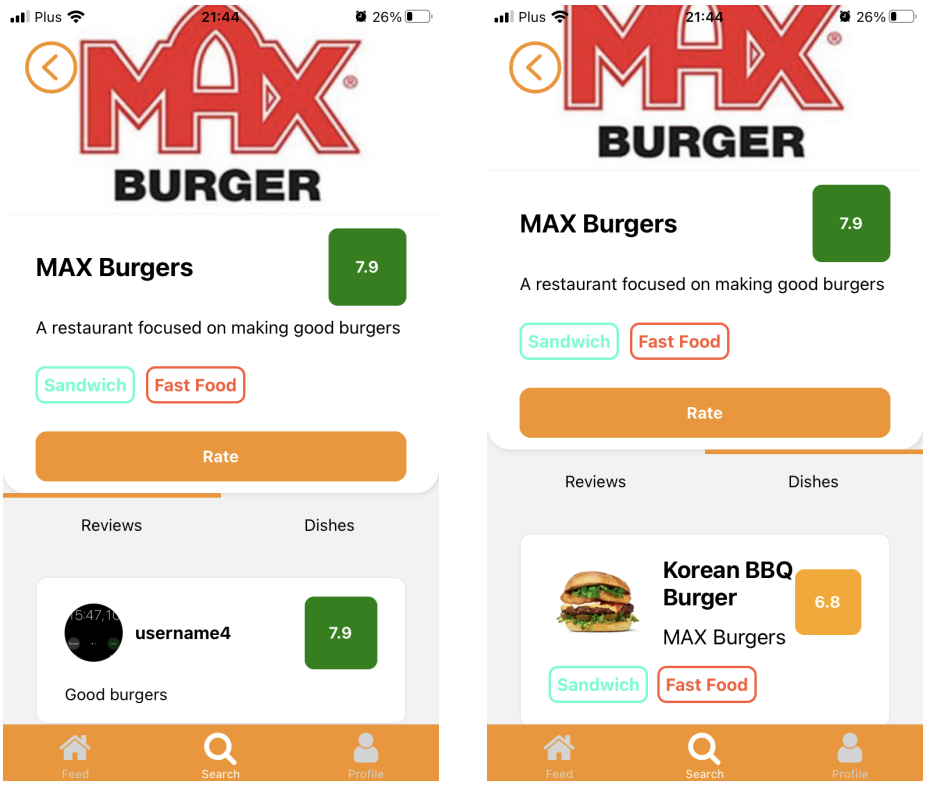
5.1. Aplikacja mobilna



Rysunek 5.1: Feed i wyszukiwarka w aplikacji mobilnej

Na rysunku 5.1 widać dwa główne ekrany aplikacji mobilnej. Po lewej znajduje się ”feed”, czyli lista dań z różnych restauracji ostatnio popularnych wśród użytkowników. Każde danie zawiera średnią ocenę oraz tagi. Listę można sortować oraz filtrować po tagach. Po prawej widoczny jest widok wyszukiwania z możliwością szukania dań, restauracji i użytkowników. Na dole widzimy

wstążkę z nawigacją pomiędzy ekranem ”feedu”, wyszukiwania i profilu użytkownika.

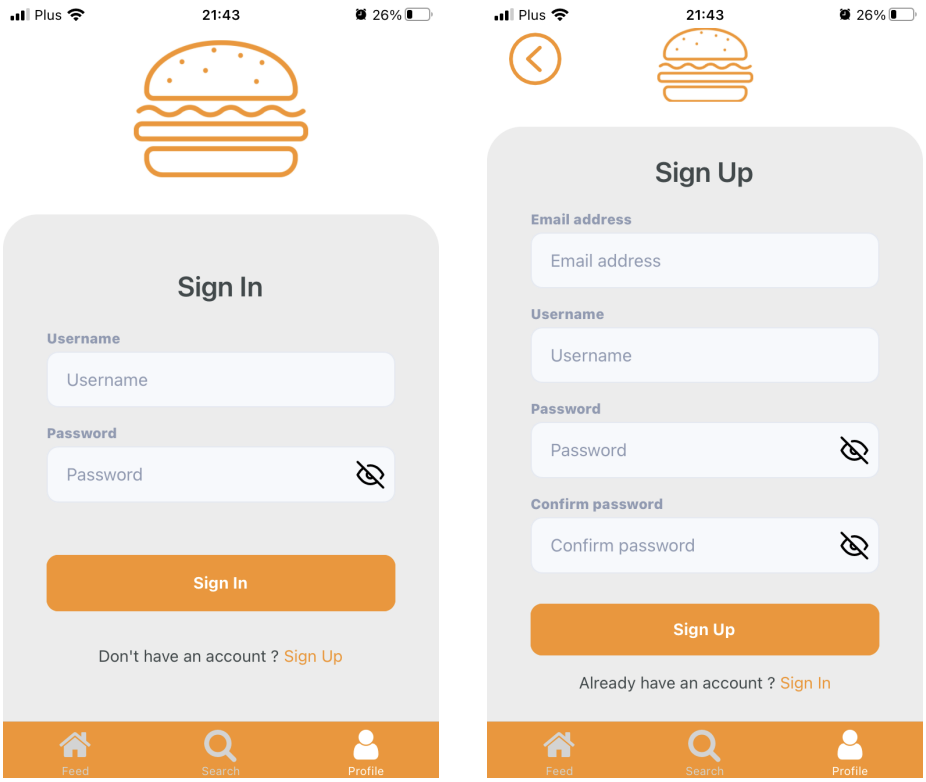


Rysunek 5.2: Detale restauracji w aplikacji mobilnej

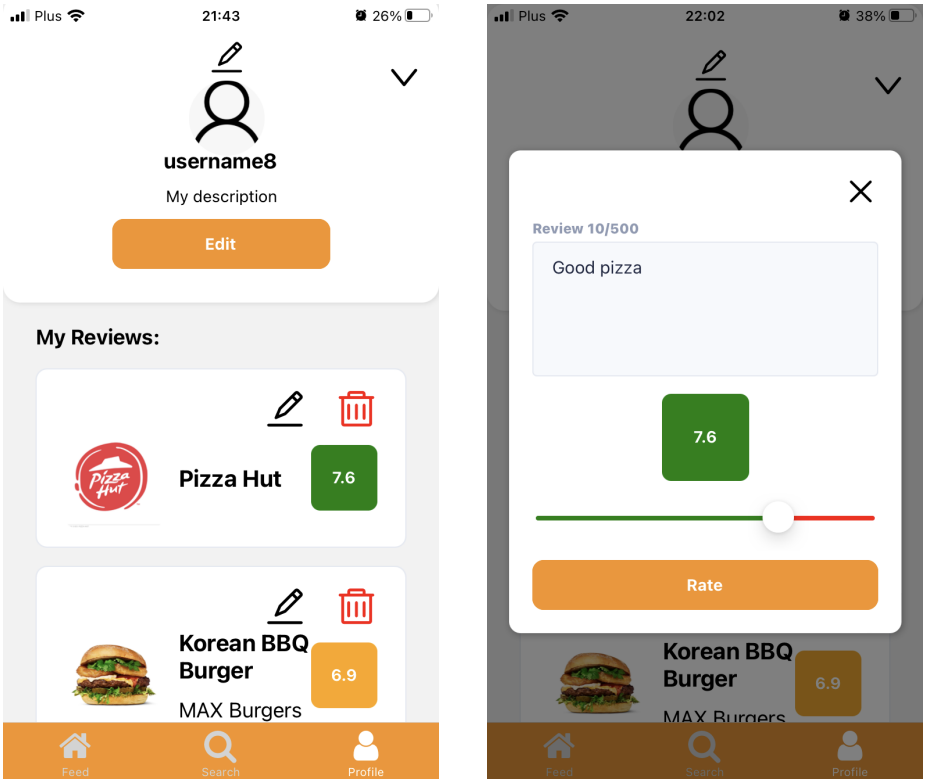
Po kliknięciu na restaurację z widoku ”feedu” lub wyszukiwania ukazuje nam się ekran detali restauracji widoczny na rysunku 5.2. Zawiera on logo restauracji, a także nazwę, opis, średnią ocenę oraz tagi. Pośrodku znajduje się przycisk udostępniający możliwość wystawienia oceny dla zalogowanych użytkowników. W dolnej części ekranu mamy listę recenzji użytkowników posortowaną po dacie dodania (lewy widok) oraz spis dań danej restauracji (prawy widok).

Ekran widoku dania wygląda podobnie do tych z rysunku 5.2 z tą różnicą, że nie ma zakładki z listą dań.

W celu zalogowania się należy przejść do trzeciej zakładki, to znaczy profilu użytkownika. Ukaże się nam ekran z rysunku 5.3. Po lewej stronie widoczny jest ekran logowania, gdzie należy podać poprawny login i hasło użytkownika. W przypadku braku konta należy nacisnąć opcję ”Sign Up”, po której użytkownik zostanie przeniesiony do ekranu rejestracji widocznego po prawej stronie rysunku 5.3. W celu rejestracji należy podać adres email, login i hasło. Formularz oferuje walidację danych w celu wymuszenia poprawności wysyłanych danych do serwera. Po poprawnym zalogowaniu lub zarejestrowaniu użytkownik uzyskuje dostęp do swojego profilu.



Rysunek 5.3: Logowanie i rejestracja w aplikacji mobilnej



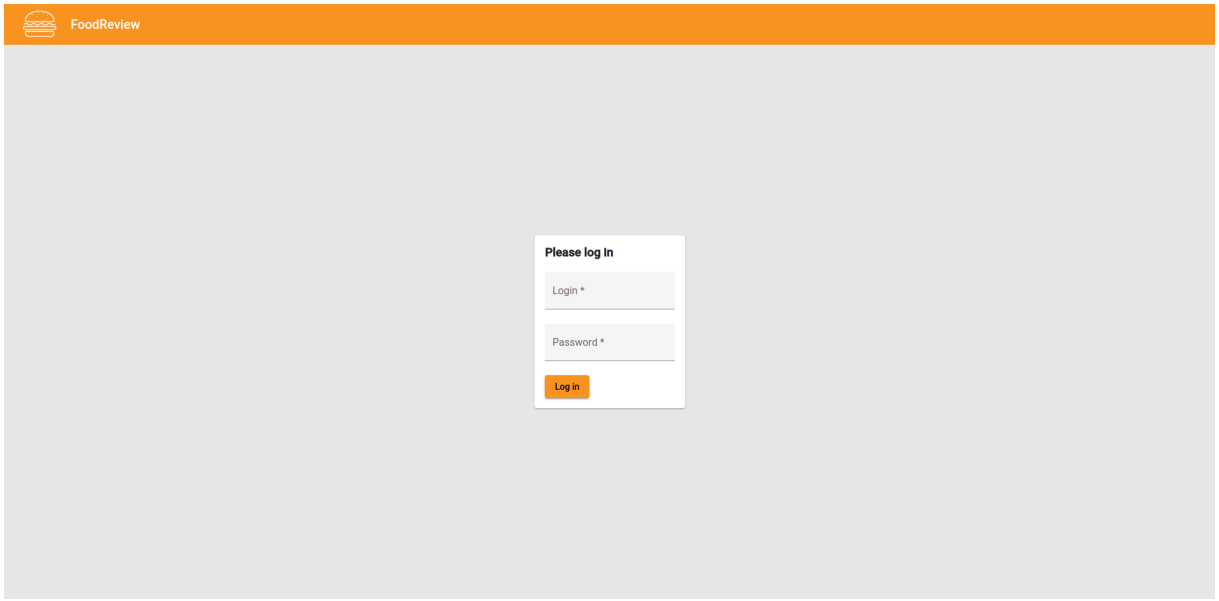
Rysunek 5.4: Profil i edycja oceny

Po zalogowaniu się użytkownik uzyskuje dostęp do ekranu użytkownika widocznego po lewej stronie rysunku 5.4. Znajdują się na nim takie informacje jak zdjęcie, nazwa użytkownika i opis. Istnieje możliwość edycji danych profilu, klikając na przycisk ”Edit”, oraz możliwość usunięcia konta. Poniżej widoczna jest lista ocen, jakie użytkownik wystawił restauracjom i daniom z możliwością ich edycji lub usunięcia.

Po prawej stronie rysunku 5.4 znajduje się okienko do wystawiania oceny. Zawiera ono pole do wpisania komentarza oraz suwak do wystawienia oceny w skali 1-10 z dokładnością do 0.1.

5.2. Aplikacja webowa

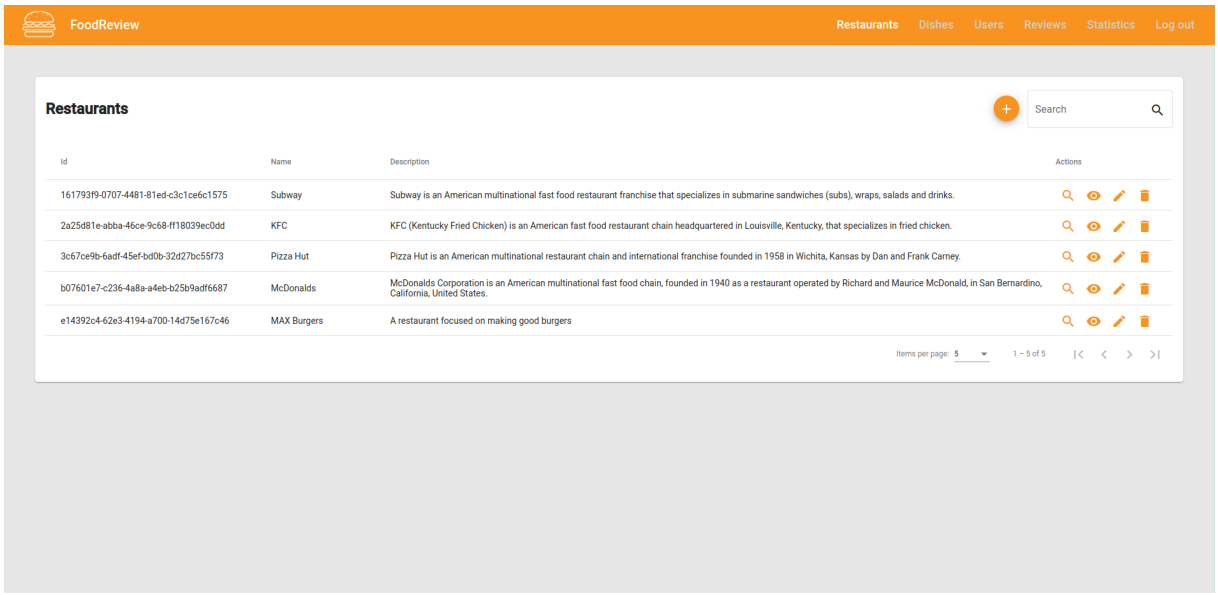
W tej sekcji przedstawimy widoki aplikacji webowej.



Rysunek 5.5: Ekran logowania w aplikacji webowej

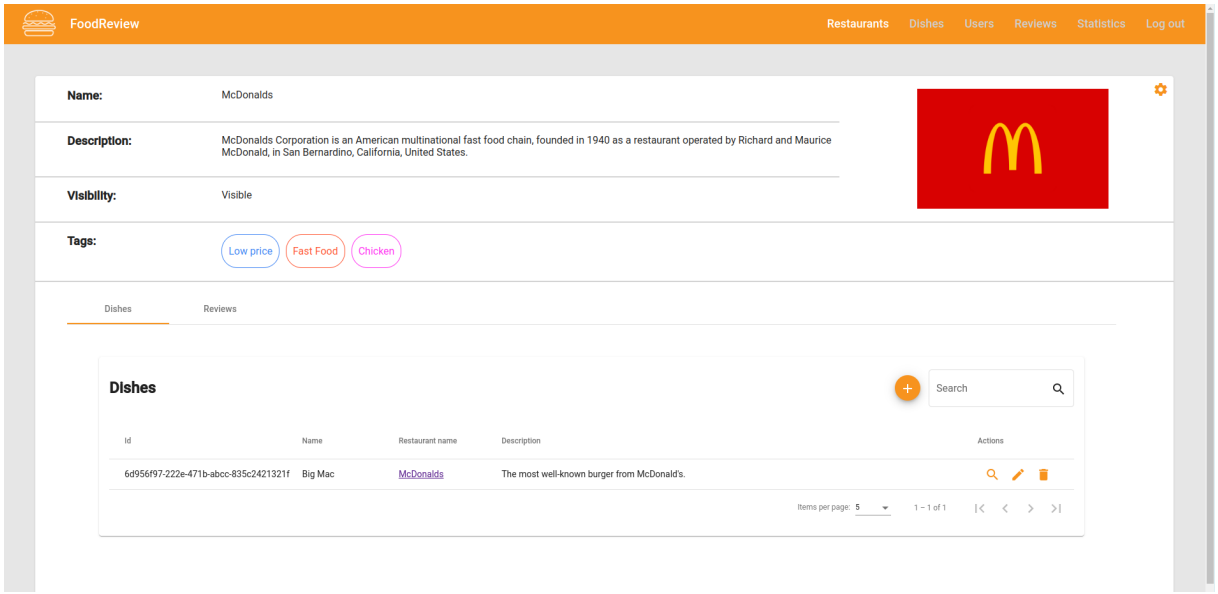
Na rysunku 5.5 znajduje się ekran logowania aplikacji webowej. Aby zalogować się do serwisu, należy podać login i hasło administratora (domyślne wartości to ”admin” i ”admin1234”).

Po zalogowaniu się ukazuje się ekran widoczny na rysunku 5.6. Jest to lista restauracji w bazie danych zawierająca identyfikator, nazwę i opis restauracji. W ostatniej kolumnie znajdują się przyciski akcji, to znaczy przejście do widoku detali, zmiana widoczności, edycja danych i usunięcie restauracji. Na wstążce na górze znajdują się przyciski do zmiany strony do widoku dań, użytkowników, ocen i statystyk oraz do wylogowania się.



Rysunek 5.6: Lista restauracji w aplikacji webowej

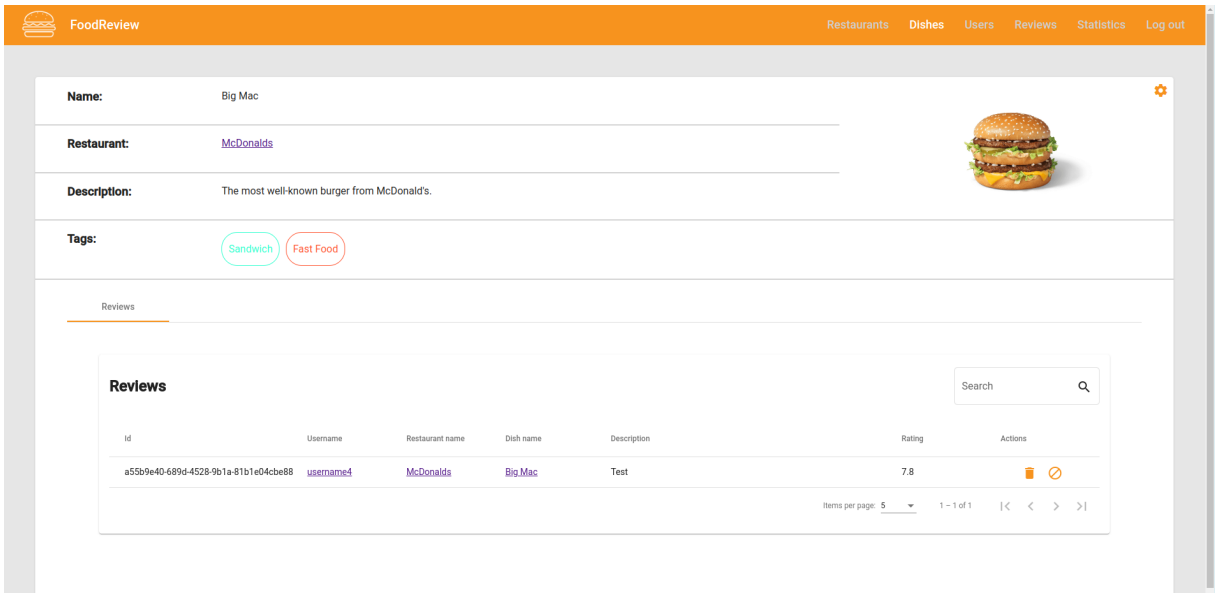
Widoki dań, użytkowników i ocen zawierają tabele bardzo podobne do tej ukazanej na rysunku 5.6. Wszystkie tabele w systemie obsługują sortowanie, paginację oraz możliwość filtrowania w polu ”Search”. Sortowanie, filtrowanie i paginacja odbywają się po stronie serwera, aby nie przesyłać niepotrzebnych danych.



Rysunek 5.7: Detale restauracji w aplikacji webowej

Po przejściu do widoku restauracji ukazuje nam się strona z rysunku 5.7. Widoczne jest na niej logo restauracji oraz dane, takie jak nazwa, opis, widoczność oraz tagi. W dolnej części znajduje się tabela z daniami oferowanymi przez daną restaurację oraz możliwość pokazania tabeli recenzji, jakie posiada dana restauracja. W prawym górnym rogu znajduje się przycisk z

dostępem do opcji zmiany widoczności, edycji danych i usunięcia restauracji.



Rysunek 5.8: Detale dania w aplikacji webowej

Po przejściu do widoku detali dania ukazuje nam się strona z rysunku 5.8 podobna do widoku restauracji 5.7. Mamy rysunek poglądowy dania oraz informacje, takie jak nazwa, restauracja, opis i tagi. Na dole znajduje się tabela z recenzjami dania, a w prawym górnym rogu przycisk oferujący możliwości edycji i usunięcia dania.

6. Testy systemu

W tym rozdziale przedstawione zostaną różne rodzaje testów, jakie zostały przeprowadzone na zaimplementowanym systemie.

6.1. Testy jednostkowe

Testy jednostkowe systemu są testami domeny w aplikacji serwera. Testowane są metody z domeny odzwierciedlające logikę biznesową aplikacji. Testy jednostkowe pokrywają następujące akcje:

- restauracje - dodawanie, edycja, zmiana widoczności;
- Dania - dodawanie, edycja, usuwanie;
- Oceny - dodawanie oceny restauracji/daniu, edycja;
- Użytkownicy - dodawanie, edycja.

Aby uruchomić testy jednostkowe w naszym projekcie należy przejść do katalogu backend/tests i wykonać komendę

```
dotnet test
```

Do wykonania tej komendy wymagane jest zainstalowanie odpowiednich bibliotek (.NET SDK).

6.2. Testy integracyjne

Testy integracyjne są testami endpointów aplikacji serwera symulującymi rzeczywiste zachowania użytkowników.

Testy integracyjne zostały napisane z użyciem oprogramowania Postman. Aby uruchomić testy integracyjne, należy zainstalować to oprogramowanie a następnie zaimportować kolekcje zapytań z folderu backend/integration-tests. Następnie należy dodać kilka zmiennych globalnych dostępnych w zakładce Environments -> Globals:

- api_address - http://localhost:8000/api
- auth_address - http://localhost:8000/auth
- user_username - nazwa założonego wcześniej użytkownika
- user_password - hasło założonego wcześniej użytkownika
- admin_username - admin
- admin_password - admin1234
- test_restaurant_name - Test restaurant
- test_restaurant_updated_description - Updated description
- test_user_username - Test username

Podane wartości są wartościami przykładowymi dla aplikacji działającej przez uruchomienie z docker compose (Sekcja 7.1). Po dodaniu zmiennych globalnych można uruchomić testy w programie Postman, klikając lewym przyciskiem na zaimportowaną kolekcję, wybierając zakładkę "Tests" i klikając "Run".

Opracowane przez nas testy integracyjne pokrywają następujące części systemu:

6.2.1. Restauracje i dania

1. Dodawanie restauracji - dodanie niewidocznej restauracji, a następnie sprawdzenie, czy wyświetla się ona administratorowi.
2. Zmiana widoczności - sprawdzenie, czy restauracja nie wyświetla się klientowi, a następnie dodanie przykładowego dania, zmiana widoczności i sprawdzenie, czy po zmianie widoczności zaczęła się ona wyświetlać wraz z dodanym daniem.
3. Edycja restauracji - edycja danych restauracji oraz sprawdzenie, czy szczegóły dotyczące restauracji zostały zmienione.
4. Usuwanie restauracji - usunięcie restauracji, a następnie sprawdzenie, czy dane dotyczące szczegółów restauracji przestały się wyświetlać oraz czy dania oraz oceny, które restauracja zawiera, zostały usunięte.

6.2.2. Użytkownicy

1. Rejestracja - podanie emaila, hasła oraz nazwy użytkownika w celu rejestracji oraz sprawdzenie, czy kod odpowiedzi jest poprawny.

2. Logowanie - sprawdzenie, czy system po podaniu poprawnego emaila oraz hasła zwraca token dostępu.

3. Zmiana opisu - edycja opisu oraz sprawdzenie, czy opis się zmienił na szczegółach uzytkownika.

4. Usuwanie konta - usunięcie konta i sprawdzenie, czy serwer zwraca błąd uwierzytelniania przy próbie zalogowania oraz weryfikacja, czy dane dotyczące użytkownika przestały się wyświetlać w jego szczegółach.

6.3. Testy akceptacyjne (scenariusze testowe)

Testy akceptacyjne służą do sprawdzenia, czy system spełnia określone wymagania funkcjonalne i jakościowe zgodnie z wymaganiami użytkownika końcowego. Poniżej przedstawione zostały testy akceptacyjne dla zaimplementowanego systemu:

6.3.1. Aplikacja klienta

Zarządzanie profilem

Po przejściu na widok profilu dla niezalogowanego użytkownika powinien wyświetlić się przycisk z napisem ”Sign In” oraz ”Sign Up”. Po wybraniu przycisku ”Sign Up” powinien wyświetlić się formularz z możliwością wpisania emaila, nazwy użytkownika, hasła oraz powtórzenia hasła. Po przesłaniu formularza aplikacja powinna przekierować użytkownika na stronę profilu. Na stronie profilu wyświetlany jest przycisk ”Edit”, po kliknięciu którego wyświetla się formularz, za pomocą którego możemy zmienić opis. Po zatwierdzeniu formularza aplikacja powinna przekierować użytkownika na stronę profilu z zaktualizowanymi danymi. Po wybraniu ikony edycji nad zdjęciem profilowym powinna nam się wyświetlić galeria, z której możemy wybrać nowe zdjęcie profilowe. Po kliknięciu ikony ustawień w prawym górnym rogu powinien się wyświetlić ”modal” z przyciskami ”Sign Out” oraz ”Delete my account”. Po wylogowaniu aplikacja powinna przekierować użytkownika na stronę profilu dla niezalogowanego użytkownika. Po usunięciu konta aplikacja również powinna przekierować użytkownika na stronę profilu dla niezalogowanego użytkownika, ale teraz ponowne zalogowanie z poprzednimi danymi nie powinno być możliwe.

Wyświetlanie "Feedu"

Po przejściu na widok "feedu" powinny nam się wyświetlić dania, które w ostatnim czasie otrzymały najwięcej ocen. Za pomocą przycisku "Sort" na górze ekranu można zmienić sortowanie (rosnąco lub malejąco) na:

- ostatnio najczęściej oceniane,
- najczęściej oceniane.

Można również filtrować dania po tagach. Po kliknięciu przycisku "Filter" wyświetli nam się lista tagów, które możemy zaznaczyć. Po wybraniu odpowiednich tagów powinny nam się wyświetlić tylko dania, które mają jeden z wybranych tagów. Po kliknięciu dowolnego kafelka z daniem powinniśmy zostać przekierowani do szczegółów dania.

Wyszukiwanie restauracji, dań i użytkowników

Po przejściu na widok wyszukiwania restauracji, dań, i użytkowników powinna wyświetlić się wyszukiwarka, po wpisaniu do której odpowiedniej frazy powinny nam się wyświetlić restauracje, dania lub użytkownicy, którzy w nazwie mają szukaną frazę. W momencie kliknięcia na kafelek restauracji, dania lub użytkownika aplikacja powinna nas przekierować na szczegóły wyszukiwanego obiektu, a strzałka wyświetlająca się w lewym górnym rogu powinna powodować powrót do widoku wyszukiwania.

Wystawienie oceny

Po wejściu w szczegóły dania lub restauracji wyświetli nam się przycisk "Rate", który spowoduje wyświetlenie formularza, gdzie można wybrać ocenę w skali 1-10 oraz dodać komentarz do swojej oceny. Po zatwierdzeniu formularza nasza ocena powinna się wyświetlić na liście ocen dania lub restauracji oraz, jeśli jest wystarczająco mało ocen, zmienić ogólną ocenę dania lub restauracji.

6.3.2. Aplikacja administratora

Po wejściu na stronę administratora wyświetlony będzie formularz do logowania z polami na nazwę użytkownika i hasło oraz przyciskiem "Log in". Po przesłaniu poprawnych danych powinna nam się ukazać strona główna panelu administratora.

Wyświetlanie listy restauracji, dań, użytkowników

W czterech pierwszych zakładkach powinny znajdować się tabele kolejno z restauracjami, daniami, użytkownikami i recenzjami. Tabele powinny obsługiwać paginację za pomocą przycisków w prawym dolnym rogu, sortowanie poprzez klikanie na nagłówki tabel (za wyjątkiem kolumn "Id" i "'Actions") oraz filtrowanie z użyciem pola "Search" w prawym górnym rogu tabeli. Po wpisaniu frazy do pola tabela zostanie przefiltrowana i wyświetlone będą tylko wyniki zawierające szukaną frazę w którejkolwiek kolumnie, za wyjątkiem kolumn "Id" i "'Actions". W piątej zakładce powinny znajdować się statystyki, czyli w naszym przypadku wykres pięciu najpopularniejszych restauracji pod względem ilości ocen.

Dodawanie restauracji

Na stronie restauracji powinien widoczny być przycisk "+". Po kliknięciu przycisku wyświetlany jest formularz do wypełnienia danych restauracji, takich jak nazwa, opis oraz lista tagów. Po wypełnieniu formularza i zapisaniu restauracja powinna zapisać się jako "niewidoczna". Następnie powinno dać się ją odnaleźć w tabeli restauracji i oznaczyć jako widoczną.

Dodawanie dań

W panelu szczegółów restauracji przy liście dań powinien być widoczny przycisk "+". Po jego kliknięciu wyświetlany będzie formularz na dane dania. Po ich wypełnieniu danie powinno zostać dodane do listy dań restauracji.

Zarządzanie użytkownikami

Na liście użytkowników w każdym wierszu powinien być widoczny przycisk do wyświetlenia szczegółów (przycisk z "lupą"). Po kliknięciu powinien otworzyć się panel ze szczegółami użytkownika. W panelu powinien znajdować się przycisk "Ban", po naciśnięciu którego użytkownik zostanie zablokowany, to znaczy nie będzie mógł się zalogować.

Usuwanie ocen

W panelach szczegółów dań i restauracji powinna znajdować się zakładka "Ratings", a w niej tabela z ocenami użytkowników. W każdym wierszu powinien znajdować się przycisk "Delete", który usuwa daną ocenę. Po jego kliknięciu ocena nie wyświetla się na liście ocen.

7. Instrukcja instalacji

Poniżej przedstawiamy, w jaki sposób zainstalować aplikację serwerową wraz z webową, a następnie mobilną.

7.1. Aplikacja serwerowa oraz webowa

W celu instalacji aplikacji serwerowej oraz webowej najlepiej skorzystać z narzędzia docker-compose, które w łatwy sposób pobierze i uruchomi wszystkie niezbędne komponenty. Aby zainstalować to narzędzie na komputerze, należy postępować zgodnie z oficjalną dokumentacją [5]. Po pobraniu narzędzia należy przejść do głównego katalogu projektu i wykonać komendę

```
docker compose up
```

Po wykonaniu tej komendy zbudowane zostaną dwa kontenery dockerowe (jeden dla aplikacji serwerowej, drugi dla aplikacji webowej) oraz pobrany zostanie kontener z bazą danych. Po zakończeniu budowania narzędzie uruchomi trzy kontenery, wykorzystując następujące porty:

- 80 dla aplikacji webowej,
- 8000 dla aplikacji serwerowej,
- 1433 dla bazy danych.

Ważne jest zapewnienie, aby na komputerze te porty nie były zajęte przez inne aplikacje, na przykład przez lokalne bazy danych często wykorzystujące domyślny port 1433 lub inne hostowane aplikacje webowe wykorzystujące domyślny port HTTP (80). Konfiguracja portów używanych przez kontenery oraz ustawienia narzędzia docker-compose znajduje się w pliku docker-compose.yaml w głównym katalogu projektu, natomiast instrukcje budowania kontenerów dockerowych znajdują się w plikach Dockerfile w katalogach /backend i /webapp.

Po wykonaniu powyższych instrukcji aplikacja serwerowa dostępna będzie pod adresem localhost:8000, a aplikacja webowa pod adresem localhost:80, lub po prostu localhost w przeglądarce.

Przy pierwszym uruchomieniu aplikacji konto administratora zostanie utworzone automatycznie z loginem ”admin”. Hasło i email do konta są konfigurowalne przez zmienne środowiskowe w pliku docker-compose.yaml. Domyślne hasło administratora to ”admin1234”.

7.2. Aplikacja mobilna

W celu uruchomienia aplikacji mobilnej na środowisku lokalnym należy:

1. Zainstalować node.js przez pobranie pakietu ze strony internetowej narzędzia [16].
2. Zainstalować narzędzia expo komendą: npm install -g expo-cli [10].
3. Przejść do folderu /mobile/src i uruchomić komendę: npm install.
4. Uruchomić aplikację komendą: npm start.
5. Po uruchomieniu serwera deweloperskiego, można zeskanować QR code za pomocą aplikacji Expo na telefonie, aby uruchomić aplikację na prawdziwym urządzeniu lub uruchomić wybrany emulator zgodnie z instrukcjami wyświetlanymi na ekranie.

8. Wnioski

W tym rozdziale podsumujemy naszą pracę i omówimy wnioski, jakie udało nam się wyciągnąć po zaprojektowaniu i zaimplementowaniu opisanego systemu. Przedstawimy również możliwości rozwoju systemu z użyciem ciekawych technologii.

8.1. Ogólna ocena projektu

Uważamy, że nasz projekt okazał się sukcesem. Rozwiązaliśmy problem opisany we wstępie, to jest brak możliwości oceny poszczególnych dań w restauracjach sieciowych. Utworzony przez nas system stanowi niezależną całość, działa sprawnie i intuicyjnie. Jesteśmy zadowoleni z wyglądu interfejsów zarówno aplikacji mobilnej, jak i webowej. Uważamy, że obie aplikacje byłyby intuicyjne w obsłudze dla potencjalnych użytkowników.

Wymagania funkcjonalne oraz niefunkcjonalne zostały w pełni pokryte. Jesteśmy zadowoleni z architektury naszego rozwiązania oraz z okazji użycia wielu ciekawych wzorców projektowych, takich jak mediator, wzorca CQRS do implementacji serwera czy też nowoczesnych technologii, takich jak IdentityServer. Była to dla nas bardzo dobra okazja do przećwiczenia projektowania i implementacji aplikacji API oraz aplikacji webowych i mobilnych. Jesteśmy również zadowoleni ze sprawnej komunikacji pomiędzy wszystkimi trzema modułami naszej aplikacji.

8.2. Możliwości rozwoju systemu

W tej sekcji opiszemy, w jaki sposób nasz system mógłby zostać rozwinięty i rozszerzony o nowe funkcjonalności.

8.2.1. Wspomaganie sztuczną inteligencją

Uważamy, że system mógłby bardzo skorzystać z wykorzystania wspomagania sztucznej inteligencji i uczenia maszynowego. Jednym z zastosowań takiej technologii byłaby personalizacja ”feedu” dla użytkowników na podstawie ich poprzednich ocen. Algorytm brałby pod uwagę po-

przednie oceny użytkownika oraz to, w jaki sposób zachowywali się inni użytkownicy do niego podobni. Na tej podstawie moglibyśmy stworzyć listę dań, którymi użytkownik prawdopodobnie byłby zainteresowany. Taka funkcjonalność dostępna byłaby tylko dla zalogowanych użytkowników z uwagi na korzystanie z ich historii ocen. Dla gości alternatywnym rozwiązaniem mogłoby być stworzenie listy na podstawie średnich ocen wszystkich użytkowników lub losowo wybranej grupy.

Innym zastosowaniem wyżej wspomnianych technologii jest wykrywanie wulgaryzmów i innych niestosownych wyrażeń w ocenach użytkowników. Oceny o takiej treści byłyby automatycznie blokowane przez system, który zapisywałby również liczbę wykroczeń wszystkich użytkowników. Po przekroczeniu pewnej liczby, system automatycznie blokowałby użytkownika. Moderacja użytkowników jest ważną częścią zapewniającą dobre działanie serwisu, a system wspomagany sztuczną inteligencją znacząco odciążyłby administratorów systemu.

Kolejnym zastosowaniem sztucznej inteligencji mógłby być system do wykrywania tzw. zjawiska "review bombingu" [9]. Polega ono na masowym wystawianiu przez użytkowników najczęściej negatywnych ocen dla danego dania bądź restauracji na podstawie różnych zewnętrznych czynników czy zdarzeń. Warto byłoby zabezpieczyć się przed takimi fałszywymi ocenami, ponieważ chcielibyśmy, aby tylko jakość produktu i świadczonych usług decydowała o ocenie. Sztuczna inteligencja mogłaby wykryć nietypowe ilości ocen wystawiane w krótkim czasie dla danych produktów i informować administratorów o takich zjawiskach, którzy podjęliby konkretne działania, na przykład zablokowanie możliwości oceniania danego dania lub restauracji na krótki czas.

Funkcjonalność uczenia maszynowego w naszym stacku technologicznym zapewniłaby paczka ML.NET.

8.2.2. Automatyczne pobieranie listy dań

W związku z tym, że menu restauracji sieciowych zmienia się dosyć gwałtownie, istnieje potrzeba aktualizacji naszej bazy danych. Istnieje możliwość stworzenia systemu, który pobierałby aktualną listę dań ze stron internetowych restauracji, a następnie przetworzył ją i zaktualizował naszą bazę danych. Jest to tak zwany "web scraping", czyli sposób pobierania danych ze stron internetowych [17].

W przypadku implementacji takiego systemu należałoby koniecznie zweryfikować prawne kwestie takiego rozwiązania, m.in. zgodność z ustawą o prawie autorskim. Należy również pamiętać, że strona każdej restauracji zbudowana jest inaczej i nasz algorytm musiałby być dostosowany do każdej z nich. Jednak w przypadku pomyślnego stworzenia takiego rozwiązania wartość dodana wynikająca z automatycznego odświeżania bazy danych jest bardzo duża.

W naszym przypadku paczka umożliwiająca takie operacje to HTML Agility Pack.

8.2.3. Wykorzystanie Elasticsearch

W procesie wyszukiwania dań/restauracji/użytkowników system można rozszerzyć o wykorzystanie Elasticsearch, który jest silnikiem wyszukiwania pełnotekstowego (baza danych). Umożliwia on zaawansowane wyszukiwanie obiektów, w tym nadawanie odpowiednich wag polom obiektów np. przy wyszukiwaniu restauracji tytuł może mieć większą wagę niż opis [7].

Wykorzystując Elasticsearch można również premiować wyniki, które mają większą liczbę ocen lub dopasowywać tekst niekoniecznie dokładnie, czy też zdefiniować synonimy np. przy szukanych frazach "Macca's", "McDonalds" będziemy w stanie znaleźć restaurację, która nazywa się McDonald's.

Uważamy, że wykorzystanie takiej technologii znacząco polepszyłoby komfort korzystania z aplikacji przez użytkowników z uwagi na to, że często popełniamy błędy próbując szybko wpisać nazwę dania lub restauracji do pola tekstowego, a taki system wyszukiwałby odpowiednie wyniki nie zważając na małe błędy w pisowni.

8.3. Potencjalne zagrożenia i problemy

W tej sekcji przedstawimy potencjalne problemy, na jakie można by się natknąć przy długotrwałym prowadzeniu naszego systemu oraz zewnętrzne czynniki, które mogłyby takiemu systemowi zagrozić.

8.3.1. Konieczność ciągłych aktualizacji bazy dań i restauracji

Jednym z najtrudniejszych zadań administratorów systemu byłoby bez wątpienia dbanie, aby lista dań wszystkich restauracji była aktualna, oraz dodawanie coraz to nowych restauracji sieciowych do bazy. Menu restauracji sieciowych często się zmienia z uwagi na promocje sezonowe, dodawanie i wycofywanie dań itp. W takim wypadku administratorzy musieliby śledzić na bieżąco strony internetowe wielu restauracji i dokonywać odpowiednich zmian w bazie danych. W naszej bazie danych przechowujemy również ceny poszczególnych dań, a więc również ten aspekt musiałby zostać pokryty przez administratorów.

Z pomocą mógłby przyjść wyżej opisany system wykorzystujący technologię "web scrapingu", jednak i to rozwiązanie nie jest bezproblemowe z uwagi na duży nakład pracy przy dodawaniu nowych restauracji (konieczność analizy kodu źródłowego stron internetowych z menu). Problem w takim wypadku byłby również z sieciówkami, które nie udostępniają swojego menu online.

8.3.2. Konieczność moderacji użytkowników

Z uwagi na sposób zachowania różnych ludzi w Internecie zadaniem administratorów byłaby również moderacja użytkowników. W przypadku wystawiania recenzji o nieodpowiedniej treści, na przykład takich zawierających wulgaryzmy, mowę nienawiści lub ataki na innych użytkowników. Takie oceny powinny zostać natychmiast usuwane z systemu. Użytkownicy wielokrotnie wstawiający takie oceny powinni być permanentnie blokowani.

Innym zagrożeniem są boty, czyli fałszywi użytkownicy sterowani przez jakieś oprogramowanie. Mogą być one używane przez firmy aby sztucznie podwyższać oceny swoich produktów lub obniżać oceny produktów konkurencyjnych. Często jest to wiele ocen o dokładnie takiej samej treści wystawianych przez różnych użytkowników. Potencjalnymi sposobami na taki problem są ograniczenie możliwości wystawiania ocen przez użytkowników zarejestrowanych w ostatnich kilku dniach, aby uniknąć masowego tworzenia kont przez tych samych użytkowników, oraz stosowanie mechanizmu CAPTCHA przy rejestracji nowych użytkowników.

W walce z takimi zachowaniami mogą pomagać opisane wcześniej systemy wspomagane sztuczną inteligencją, które oszczędziłyby uciążliwej pracy administratorom systemu, automatycznie blokując odpowiednich użytkowników i usuwając niechciane oceny.

8.3.3. Konkurencja z dużymi firmami

Systemy oceny restauracji nie są nowością w dzisiejszych czasach. Istnieje już wiele rozwiązań wykorzystywanych przez miliony użytkowników, które posiadających duże bazy restauracji. Przykładami takich systemów są Google Maps i Pyszne.pl. W przypadku sukcesu naszego systemu korporacje rozwijające własne rozwiązania mogłyby zainteresować się naszą funkcjonalnością i wprowadzić podobny system do swoich rozwiązań, przez co nasz produkt straciłby swoje znaczenie. Użytkownicy pewnie woleliby korzystać z wielomilionowych serwisów oferujących taką samą funkcjonalność, niż korzystać z osobnego systemu.

Bibliografia

[1] https://akveo.github.io/react-native-ui-kitten/

[2] https://analizait.pl/2014/metoda-furps-czyli-29-rzeczy-do-przemyslenia-w-kazdym-projekcie-it/

[3] https://angular.io/features

[4] https://docker.com/

[5] https://docs.docker.com/compose/install/

[6] https://dotnet.microsoft.com/en-us/apps/aspnet

[7] https://elastic.co/

[8] https://en.wikipedia.org/wiki/Mediator_pattern/

[9] https://en.wikipedia.org/wiki/Review_bomb

[10] https://expo.dev/

[11] https://hub.docker.com/

[12] https://identityserver4.readthedocs.io/en/latest/

[13] https://learn.microsoft.com/pl-pl/azure/architecture/patterns/cqrs

[14] https://learn.microsoft.com/en-us/ef/core/

[15] https://material.angular.io/

[16] https://nodejs.org/

[17] https://parp.gov.pl/component/content/article/83315:web-scraping-jak-prawidlowo-korzystac-z-tresci-dostepnych-w-internecie

[18] https://pl.reactjs.org/

[19] https://reactnative.dev/

Spis rysunków

Spis tabel