

POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Teleinformatyka [TIN]

SPECJALNOŚĆ: Projektowanie Sieci Teleinformatycznych [TIP]

PRACA DYPLOMOWA
INŻYNIERSKA

Tytuł pracy w języku polskim
Projekt i implementacja aplikacji webowej i
mobilnej, katalogującej informacje o daniach w
restauracjach

Tytuł pracy w języku angielskim
Design and implementation of a web and mobile
aplication, storing data about dishes in restaurants

AUTOR:

Jakub Cembrzyński

PROWADZĄCY PRACĘ:
dr. inż. Wojciech Kmiecik, W4K2

OCENA PRACY:

Słownik pojęć i skrótów:

- Front-end – internetowy interfejs użytkownika
- Back-end – aplikacja serwerowa
- API (ang. application programming interface) – interfejs programistyczny
- DOCKER – oprogramowanie służące do konteneryzacji aplikacji
- DOS (ang. Denial Of Service) – atak mający na celu przerwanie usługi
- DDOS (ang. Distributed Denial Of Service) – rozproszony atak DOS
- JSON (ang. JavaScript Object Notation) – ustandaryzowany format danych
- JWT (ang. JSON Web Token) – format tokenów autoryzacyjnych
- Framework – platforma programistyczna do budowy aplikacji
- SPA (ang. Single Page Application) – model budowania strony internetowych
- MPA (ang. Multi Page Application) - model budowania strony internetowych
- MVC (ang. Model View Controller) – programistyczny wzorzec architektoniczny
- Atak metodą siłową – rodzaj ataku hakerskiego
- Atak z wykorzystaniem tęczowych tablic - rodzaj ataku hakerskiego
- Funkcja haszująca, funkcja skrótu – jednostronna funkcja kryptograficzna
- Sniffing – rodzaj ataku hakerskiego

Spis treści

Słownik pojęć i skrótów:	2
1. Wstęp	5
2. Przegląd i analiza istniejących rozwiązań.....	6
2.1. Rozwiązania Front-endowe	6
2.2. Rozwiązania Back-endowe.....	8
2.3. Rozwiązania bazo-danowe	10
3. Analiza potrzeb użytkowników.....	12
3.1. Interesariusze	12
3.2. Bezpośredni użytkownicy aplikacji	13
3.3. Przewidziane środowisko implementacyjne produktu.....	15
3.4. Przypadki użycia	17
3.5. Wymagania funkcjonalne	17
3.6. Wymagania нефункционалне	22
4. Projekt aplikacji.....	24
4.1. Baza danych.....	24
4.2. Webowy interfejs użytkownika (ang. Front-end application)	26
4.3. Aplikacja serwerowa (ang. Back-end application)	29
5. Implementacja	31
5.1. Utworzenie konta przez użytkownika	32
5.2. Użytkownik loguje się do systemu	32
5.3. Wyświetlanie mapy użytkownikowi	33
5.4. Wyszukiwanie restauracji w systemie, która ma w swojej karcie podane przez użytkownika danie.....	34
5.5. Zaprezentowanie użytkownikowi komentarzy i ocen innych użytkowników	35
5.6. Dodanie oceny i słownego komentarza do wybranego dania przez użytkownika	36
5.7. Rejestracja restauracji przez użytkownika	37

5.8.	Użytkownik dodaje nowe danie do menu swojej restauracji.....	37
5.9.	Użytkownik zmienia nazwę dania w menu swojej restauracji	38
5.10.	Użytkownik zmienia cenę dania w menu swojej restauracji	39
5.11.	Użytkownik usuwa danie z menu swojej restauracji.....	39
5.12.	Użytkownik ma wgląd w informację o ocenach dań w swojej restauracji ...	40
5.13.	Administrator blokuje użytkownika	41
5.14.	Administrator odblokowuje użytkownika.....	41
5.15.	Hasła w bazie danych są przechowywane w formie niejawnej	41
5.16.	Baza danych jest zabezpieczona hasłem.....	42
5.17.	Połączenia poza siecią lokalną są szyfrowane	42
5.18.	Aplikacja tworzy logi z wykonywanych operacji.	43
5.19.	Aplikacja jest kompatybilna z urządzeniami mobilnymi.....	43
6.	Podsumowanie.....	45
7.	Bibliografia	45
	Spis tabel i rysunków	47

1. Wstęp

Rozwój informatyki jest napędzany chęcią konsumentów do ułatwiania życia, co leży w ludzkiej naturze. Każdy z nas nie lubi lub nie może sobie pozwolić na stratę czasu w życiu codziennym. Przez co coraz chętniej korzystamy z restauracji, aby zaoszczędzić czas na gotowaniu. Dobrze świadczy o tym wzrost popularności takich aplikacji jak pyszne.pl, czy pizzaportal. Aplikacje te pozwalają na szybsze oraz sprawniejsze zamawianie jedzenia na dowóz. Co jednak jeżeli chcielibyśmy zjeść coś w na miejscu, w restauracji? Google maps pozwala nam wyszukać restaurację w pobliżu wraz z ich oceną, na podstawie zadowolenia konsumentów. Jednakże aplikacja nie pozwala nam dokładnie sprawdzić menu danej restauracji. Można sobie wyobrazić sytuację, że włoska restauracja nie serwuje naszego ulubionego makaronu. Wtedy tracimy czas na dojście do tego miejsca lub w lepszej sytuacji tracimy czas na szukanie strony danego miejsca, tylko po to, żeby się dowiedzieć, że należy szukać dalej. Załóżmy jednak, że restauracja serwuje nasz ulubiony makaron. Sprawdzamy ocenę i jest ona dla nas na tyle wysoka, że decydujemy się odwiedzić dane miejsce. Jednak po zjedzeniu, okazuje się, że danie do najlepszych nie należało. Przyczyną takiej sytuacji może być fakt, że ocena restauracji nie zawsze może oznaczać, że każda pozycja w karcie jest takiej samej jakości. Można sobie wyobrazić sytuację, gdzie tylko 3 najpopularniejsze dania z karty są świetne, a reszta przeciętna. Większość klientów oceni bardzo dobrze dane miejsce, gdzie tak naprawdę ich ocena dotyczy jedynie części z dań jakie dana restauracja oferuje. To tak jakby idąc na film do kina, klient podejmowałby decyzję o wyborze jedynie na podstawie oceny miejsca, na którą składa się znacznie więcej rzeczy, niż pozycje w repertuarze.

Motywuując się tym celem mojej pracy był projekt i implementacja aplikacji internetowej i mobilnej, która katalogując dane o daniach w restauracjach, pozwoli rozwiązać te problemy. Pozwala ona użytkownikowi znaleźć restaurację, gdzie zje danie, które go interesuje. Dodatkowo może zobaczyć jak inni klienci danej restauracji ocenili konkretne danie, dzięki czemu może podjąć świadomą decyzję, co przełoży się na zaoszczędzenie czasu i zmniejszy frustrację spowodowaną rozczarowanie.

Moja praca składa się z siedmiu rozdziałów. W pierwszym rozdziale opisałem motywację do stworzenia pracy. W drugim rozdziale porównałem dostępne technologie pozwalające na implementację tego typu aplikacji. Trzeci rozdział jest poświęcony analizie potrzeb użytkowników, co pozwoliło stworzyć listę wymagań funkcjonalnych oraz

niefunkcjonalnych. Rozdział 4 zawiera projekt aplikacji. W rozdziale 5 zawarłem sposób implementacji wymagań z rozdziału trzeciego. Rozdział 6 zawiera podsumowanie, które kończy pracę. Rozdział 7 zawiera bibliografię.

2. Przegląd i analiza istniejących rozwiązań

Opierając się na podstawowych wymaganiach i trendach w projektowaniu aplikacji webowych składa z trzech modułów, każdy odpowiedzialny za inne funkcjonalności.

Pierwszym z nich jest moduł odpowiedzialny za kontakt z użytkownikiem (ang. Front-end application). Jest to aplikacja, której zadaniem jest pobieranie oraz udostępnianie danych użytkownikowi. Drugim modułem jest baza danych, której zadaniem jest przechowywanie danych, z których korzysta cała aplikacja. Trzecim modułem jest aplikacja, której zadaniem jest pośredniczenie pomiędzy bazą danych, a interfejsem użytkownika (ang. Back-end application).

Po analizie aktualnie wykorzystywanych rozwiązań na podstawie danych udostępnionych przez portal stackoverflow [1], który skupia programistów z całego świata, można wyróżnić następujące rozwiązania przeznaczone do tworzenia aplikacji webowych:

2.1. Rozwiązania Front-endowe

Z listy wybrałem 3 frameworki do języka javascript, które pozwalają na budowanie interfejsów użytkownika. Są to rozwiązania wspierające model SPA (ang. Single Page Application). Jest to model, który opiera się na tym, że po wejściu na główny link pobiera się cała aplikacja z wszystkimi widokami. Skrypty umożliwiają renderowanie odpowiednich widoków oraz pobieranie danych dokładnie wtedy, kiedy one są potrzebne. Takie rozwiązanie zapewnia nieporównywalną płynność w korzystaniu z aplikacji do architektury MPA (ang. Multi Page Application), gdzie każdy widok pobierany jest oddzielnie, powodując ładowanie pomiędzy nimi. Minusem SPA w porównaniu do MPA jest to, że w momencie, gdy aplikacja urośnie do dużych rozmiarów wejście na stronę po raz pierwszy będzie zabierać więcej czasu, niż gdyby zostało zastosowane podejście MPA.

Z oczywistych powodów do działania SPA wymagane jest wsparcie javascript w przeglądarce.

W modelu SPA jest widoczny podział pomiędzy front-endem, a back-endem, a w MPA oba moduły są mocno ze sobą związane. [2]

Analizując powyższe plusy i minusy obu modeli budowania aplikacji front-endowych, zdecydowałem się na SPA. Podejście zapewnia lepsze doświadczenie użytkownika, a wymóg działania Javascript w przeglądarce spełnia zdecydowana większość aktualnie wykorzystywanych przeglądarek. Rozdzielenie front-endu i back-endu pozwala na wykorzystywanie API przez inne aplikacje.

Wybrane rozwiązanie prezentują się następująco:

- React.js

React jest aktualnie najczęściej i najchętniej wykorzystywanym rozwiązaniem aplikacji front-endowych. Jest utrzymywany przez Facebook'a i ma bardzo dużą społeczność, dzięki czemu można w łatwy sposób znaleźć rozwiązania problemów. Dzięki technologii Virtual-DOM komponenty są szybko renderowane, a możliwość importowania dodatkowych pakietów zapewnia skalowalność aplikacji. [3]

- AngularJS

Jest również bardzo popularne rozwiązanie, które daje najwięcej możliwości z omawianej trójki, dzięki wsparciu dla modelu MVC (ang. Model-View-Controller). Framework jest wspierane przez Google oraz również ma dużą społeczność. Jednak AngularJS jest bardzo złożonym frameworkiem, który wymaga dużo wiedzy od programisty, aby osiągnąć zamierzone funkcjonalności. AngularJS bardzo dobrze sprawdza się w dużych i złożonych projektach. [3]

- Vue.js

Jest to najnowsze z wymienionych rozwiązań, dzięki czemu można było rozwiązać, niektóre z problemów tworzenia aplikacji w dwóch pozostałych frameworkach. Efektem tego jest bardzo prosta budowa aplikacji, która nie sprawia problemów początkującym programistą. Dodatkowo wykorzystano koncepty, wykorzystywane w React'cie i Angularze, dzięki czemu można w łatwy sposób zmienić technologię na Vue.js. [3]

W mojej pracy zdecydowałem się na Vue.js. Jest to nowe rozwiązanie, które w szybkim tempie zyskuje uznanie, przez co można założyć, że jest rozwiązaniem przyszłościowym. Dodatkowym czynnikiem wpływającym na moją decyzję, że jest implementacja w

aplikacji w Vue jest najprostsza z wymienionej trójki. Skutkuje to dużą szybkością w tworzeniu aplikacji.

2.2.Rozwiązania Back-endowe

Najpopularniejszym typem aplikacji webowej pośredniczącej między interfejsem użytkownika, a bazą danych (ang. WebAPI) jest implementacja oparta o zasady architektoniczne REST (ang. Representational state transfer). REST jest zbiorem dobrych praktyk architektonicznych w budowaniu serwisów internetowych. Wyróżnia się ich 6:

1. Architektura klient – serwer
2. Bezstanowość (ang. Statelessness)
3. Cacheability
4. Odseparowanie warstw (ang. Layered system)
5. Kod na żądanie (ang. Code on demand)
6. Jednolite interfejsy (ang. Uniform interfaces)

Interfejs webowy oparty o REST jest prosty w implementacji, ale także w obsłudze. Dzieje się tak, ponieważ wszystkie operacje możliwe do wykonania na API oparte są o podstawowe metody dostępne w protokole HTTP:

1. GET – służąca do pobierania danych
2. POST – służąca do umieszczania danych określonego typu
3. PUT – służąca do nadpisywania danych
4. DELETE – służąca do usuwania danych

Dodatkową zaletą oparcia operacji na protokole HTTP jest obsługa błędów. Protokół dostarcza kody, z których można dowolnie korzystać. [4]

Decydując się na oparciu back-endu omawianego serwisu na architekturze REST, wybrane zostały frameworki, które pozwalają na stworzenie takiej aplikacji.

- Express.js

Jest to framework webowy oparty o środowisko uruchomieniowe Node.js należące do MIT. Jak nazwa wskazuje kod tworzy się w tym przypadku w języku skryptowym JavaScript. Framework jest prostym rozwiązaniem do budowania interfejsów webowych. Rozwiązanie ma otwarte źródło oraz dużą społeczność, która zapewnia wsparcie w problemach, które można napotkać.[5] Minusem Express.js jest jego słaba wydajność.

Aktualne trendy pokazują, że podczas tworzenia API dąży się do tego, aby było ono jak najmniejsze i jak najszybsze. Express jest oparty na wielu modułach Node.js, które zabierają zasoby na serwerze. 0

- Django

Django jest frameworkiem do tworzenia interfejsów webowych w języku skryptowym python. Rozwiązanie ma otwarte źródło i jest w pełni darmowe. Ideą stojącą za Django jest zmniejszenie czasu implementacji aplikacji poprzez udostępnianie wielu gotowych funkcji. Dzięki temu programista może skupić się na wymaganiach funkcjonalnych i implementować je szybko i prosto, co z kolei poprawia czytelność kodu. Twórcy frameworka zadbali również o funkcję służącą do zabezpieczania aplikacji, tak aby można było to osiągnąć w prosty sposób. Twórcy twierdzą, że aplikacje napisane za pomocą Django dobrze się skalują, a także sam framework daje możliwość implementacji aplikacji o szerokim zakresie funkcjonalności. [7]

- Spring

Jest to framework, który zapewnia model programistyczny do tworzenia aplikacji webowych opartych na języku Java. Aplikacje napisane za pomocą Springa korzystają z JVM, przez co uruchamiają się na większości środowisk. Główne technologie zaimplementowane w framework (ang. Core technologies) pomagają w tworzeniu aplikacji. Jest to między innymi wsparcie dla wstrzykiwania zależności (ang. Dependency injection), obsługi wydarzeń (ang. Event handling), walidacja (ang. Validation) oraz konwersja typów (ang. Type conversion). Udostępnione jest również wsparcie dla testów oraz zaimplementowane są takie wzorce projektowe jak model: model – widok – kontroler (ang. MVC – Spring MVC). 0

- ASP.NET Core

Jest to framework do tworzenia aplikacji webowych stworzony przez firmę Microsoft. Do działania wykorzystuje platformę .NET i uruchamia się na wielu platformach, w tym na Windows, Linux, macOS oraz Docker. Według „Tech Power Web Framework Benchmarks” ASP.NET Core wygrywa pod kątem szybkości obsługi zapytań z innymi wiodącymi frameworkami 0. Rozwiązanie wspiera między innymi wstrzykiwanie zależności, obsługę wydarzeń, asynchroniczność, a także wzorce projektowe takie jak model – widok – kontroler (ang. MVC). Dostępna jest również dokumentacja, tworzona

przez Microsoft, a także, dzięki dużej społeczności, rozwiązania popularnych problemów na forach internetowych. [10]

W mojej pracy zdecydowałem się na ASP.NET Core. Framework jest darmowy i ma otwarte źródło, co pozwoli zmniejszyć koszty aplikacji. Czynnikiem przesądzającym za tym wyborem jest duża wydajność i wsparcie dla platformy Docker, które znacznie ułatwia wdrożenie aplikacji na środowisko produkcyjne.

2.3. Rozwiązania bazo-danowe

W tym przypadku należy zacząć od podjęcia decyzji, czy baza danych ma być relacyjna, czy nie. Bazy NoSQL'owe odróżnia przede wszystkim to, że nie korzystają one z języka SQL, a także, że nie trzeba definiować schematu danych, tak jak w bazach relacyjnych. Kolejną różnicą jest to, że bazy SQL skalują się wertykalnie poprzez zwiększanie wydajności, czy pojemności serwera, który udostępnia bazę. Bazy NoSQL'owe skalują się horyzontalnie poprzez dodawanie kolejnych serwerów (ang. Sharding), przez co są one rozproszone. Wśród najpopularniejszych rozwiązań znajdują się oba typy baz danych. 0

- **MySQL**

MySQL jest multiplatformowym, otwartym systemem opartym o język SQL. Posiada dużą społeczność, dzięki temu, że jest długo na rynku. Jest to darmowe rozwiązanie, które pozwoli ograniczyć koszty całej aplikacji. Dodatkowo MySQL, w przeciwieństwie do innych baz danych wspiera horyzontalne skalowanie, które pozwoli rozproszyć bazę danych. 0

- **PostgreSQL**

Jest to hybrydowe rozwiązanie pomiędzy typową relacyjną, a nie relacyjną bazą danych. Tak samo jak MySQL jest otwartym i darmowym systemem. Idealnie nadaje się do sytuacji, gdzie przechowywane dane nie zawsze będą tego samego formatu. Jest kompatybilne z wieloma platformami oraz z wieloma językami programowania. 0

- Microsoft SQL server

Dużym plusem rozwiązania jest fakt, że należy ono do Microsoftu. Korporacja zapewnia świetny support, dokumentację oraz naprawę błędów. MSSQL korzysta z T-SQL, który różni się nieco od „czystego” SQL. Rozwiązanie wspiera tylko platformy Windows oraz Linux. 0

- SQLite

SQLite jest bardzo ciekawym, multiplatformowym rozwiązaniem implementacji relacyjnej bazy danych opartej na języku SQL. Czynnikiem odróżniającym SQLite jest fakt, że cała baza danych jest przechowywana w jednym pliku, który znajduje się na tej samej maszynie, na której uruchomiona jest aplikacja. Plusem tego rozwiązania jest to, że wymaga ono znacznie mniej zasobów od innych rozwiązań. Jest to jednak okupione minusami. SQLite dla dużych serwisów jest mniej wydajnym rozwiązaniem niż bardziej konwencjonalne aplikacje. Fakt umiejscowienia bazy danych przy aplikacji może być zarówno plusem, jak i minusem. Dla architektury składających się z jednego modułu jest to idealne rozwiązanie, które pozwoli zmniejszyć wymagane zasoby. Przykładem dobrego wykorzystania SQLite są urządzenia Internetu rzeczy, które muszą przechowywać w swojej pamięci jakieś dane. [12]

- MongoDB

MongoDB jest przykładem NoSQL’owej bazy danych opartej na dokumentach, o formie podobnej do plików formatu JSON. Udostępnione są biblioteki do najpopularniejszych języków programowania, które pozwalają w bardzo łatwy sposób mapować obiekty w kodzie na dokumenty MongoDB. Fakt, że jest to baza NoSQL’owa sprawia, że w sytuacji, gdy przechowywane są duże ilości tego samego typu danych rozwiązanie sprawdza się bardzo dobrze. Przykładem może być serwis internetowy z recenzjami filmów. Będzie on przechowywał głównie dokumenty zawierające recenzje wraz z komentarzami. Brak oparcia bazy o język SQL uniemożliwia tworzenie relacji między encjami. Możliwe jest łączenie dokumentów za pomocą odpowiednika polecenia SQL „join” jednak, nie zapewnia ono takiej wydajności jak w przypadku relacyjnych rozwiązań. 0

Analizując wymienione rozwiązania zdecydowałem się na MySQL. Jest darmową bazą danych, która działa na zasadzie klient – serwer. Oparta jest na języku SQL i pozwala tworzyć relację, z których skorzystam w pracy. Dodatkowo zapewni wymaganą

wydajność i możliwość dowolnego skalowania, jeżeli aplikacja rozrośnie się do dużych rozmiarów.

3. Analiza potrzeb użytkowników

Przed przystąpieniem do projektowania aplikacji należy zdefiniować interesariuszy, a wśród nich nabywców aplikacji, czyli użytkowników, którzy będą z niej korzystać. Wśród interesariuszy znajdują się również inne podmioty, które mogą być potencjalnie zainteresowane aplikacją. Kolejnym krokiem jest stworzenie listy ograniczeń projektu oraz zaplanowanie środowiska implementacyjnego i środowiska pracy aplikacji. Pierwsze z nich definiuje fizyczny i logiczny sposób instalacji aplikacji, a drugie środowisko, w których użytkownicy będą z aplikacji korzystać. Kolejnym krokiem jest stworzenie diagramu przypadków użycia, który definiuje aktorów, czyli podmioty wykonujące dane czynności w systemie, oraz przypadki użycia, czyli operacje wykonywane w systemie przez aktorów. Na podstawie przypadków użycia tworzy się listę wymagań funkcjonalnych i niefunkcjonalnych. Poniższy rozdział napisałem korzystając z Wzorca specyfikacji wymagań Volere. [13]

3.1. Interesariusze

Interesariuszami aplikacji są osoby związane z branżą gastronomiczną oraz ludzie korzystający z jej usług. Kolejną grupą interesariusze są reklamodawcy związani z tą branżą.

- Nabywcą aplikacji jest osoba zarządzająca restauracją, która chce dotrzeć do nowych klientów oraz uzyskać od nich oceny konkretnych dań w swoim menu. Na ich podstawie będzie mogła podjąć różne decyzje biznesowe, co w rezultacie pozwoli zwiększyć zyski oraz zadowolenie klientów swojej restauracji.
- Drugą grupą interesariuszy są osoby korzystające z usług branży gastronomicznej. Aplikacja pozwoli im skrócić proces planowania posiłku w restauracji, dzięki wyszukiwarce restauracji oraz ocenom innych użytkowników.
- Reklamodawcy związani z branżą gastronomiczną będą mogli umieszczać reklamy na stronie internetowej, skierowane do klientów branży gastronomicznej.
- Kolejnym typem interesariuszy są portale do zamawiania jedzenia przez Internet. Będą one mogły skorzystać z ocen zebranych przez aplikację do umieszczenia ich w swoich systemach.

- Następną grupą interesariuszy są osoby prowadzące blogi oraz dziennikarze piszący do gazet, związanych z branżą gastronomiczną. Dzięki wyszukiwarce będą oni mogli szybciej dotrzeć do restauracji, o interesujących ich profilu. Przykładem takiego artykułu może być „10 najlepszych makaronów na starówce we Wrocławiu”.

3.2. Bezpośredni użytkownicy aplikacji

W projektowanej aplikacji można wyróżnić następujące typy użytkowników, którzy będą z nią mieli bezpośredni kontakt. Podstawowe informacje o nich są zebrane w postaci tabel od 1 do 3.

Tabela 1 Informacje o typie użytkownika – klient branży gastronomicznej

Nazwa użytkownika	Klient branży gastronomicznej
Rola użytkownika	Wyszukiwanie restauracji, dodawanie ocen
Znajomość dziedzinowa przedmiotu sprawy	Nowicjusz
Doświadczenie w sferze technologii	Nowicjusz
Inne cechy charakteryzujące użytkownika	Jest to użytkownik w różnej grupie wiekowej, o niskich umiejętnościach technicznych. Zazwyczaj posługuje się językiem ojczystym i angielskim w stopniu przynajmniej komunikatywnym. Korzysta z aplikacji za pomocą komputera, tabletu lub telefonu.

Tabela 2 Informacje o typie użytkownika – osoba zarządzająca restauracją

Nazwa użytkownika	Osoba zarządzająca restauracją
Rola użytkownika	Tworzenie i edycja menu restauracji, przeglądanie ocen klientów
Znajomość dziedzinowa przedmiotu sprawy	Ekspert
Doświadczenie w sferze technologii	Nowicjusz

Inne cechy charakteryzujące użytkownika	Użytkownik w wieku produkcyjnym (Polska 18-60/65), o niskich umiejętnościach technicznych, ale o wysokiej znajomości branży gastronomicznej. Posługuje się językiem ojczystym oraz językiem angielskim zazwyczaj w stopniu wyższym niż komunikatywny. Korzysta z aplikacji za pomocą komputera znajdującego się w miejscu pracy.
---	--

Tabela 3 Informacje o typie użytkownika – administrator aplikacji

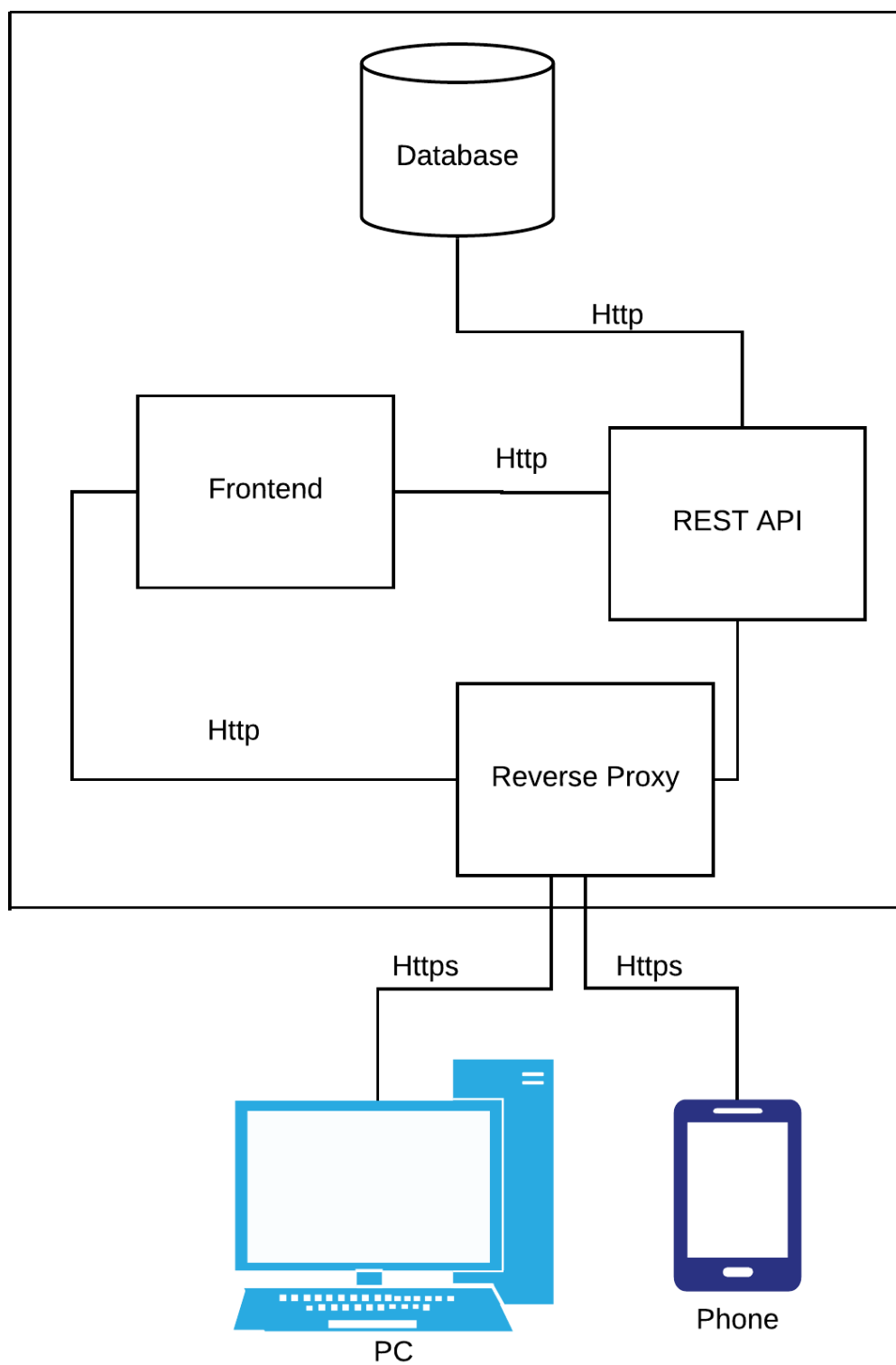
Nazwa użytkownika	Administrator
Rola użytkownika	Utrzymywanie systemu, usuwanie komentarzy niezgodnych z regulaminem, blokowanie użytkowników
Znajomość dziedzinowa przedmiotu sprawy	Nowicjusz
Doświadczenie w sferze technologii	Ekspert
Inne cechy charakteryzujące użytkownika	Osoba w wieku produkcyjnym (Polska 18-60/65), o wysokich umiejętnościach technicznych i bardzo dobrej znajomości języka angielskiego. Posiada wykształcenie techniczne związane z branżą IT. Zazwyczaj nie posiada wiedzy związanej z branżą gastronomiczną. Korzysta z aplikacji za pomocą komputera znajdującego się w miejscu pracy.

3.3.Przewidziane środowisko implementacyjne produktu

Rysunek 1 przedstawia środowisko implementacyjne systemu. Oprócz wcześniej wspomnianych elementów, czyli aplikacji front-endowej, REST API oraz bazy danych występuje tutaj aplikacja spełniająca zadanie reverse proxy. Jest to aplikacja pośrednicząca w wymianie zapytań pomiędzy klientem, a resztą systemu, stanowiąc jedyną możliwość kontaktu z aplikacjami z zewnętrznej sieci. Jej zastosowanie w środowisku implementacyjnym ma spełniać przede wszystkim dwie role: zwiększać bezpieczeństwo i ułatwiać skalowanie aplikacji. NGINX, którego zamierzam wykorzystać w środowisku implementacyjnym zapewnia ochronę przed atakami typu DoS i DDoS, poprzez ograniczanie ruchu pochodzącego z jednego adresu IP. W związku z tym, że cały ruch przechodzi przez reverse proxy, można go dowolnie kierować poprzez konfigurację tylko jednej aplikacji. Pozwala to na swobodę w zmianach architektury systemu, stojącego za reverse proxy.

Dodatkową zaletą korzystania z tego typu aplikacji jest zwiększenie wydajności całego systemu. Wśród stosowanych technik można wyróżnić następujące:

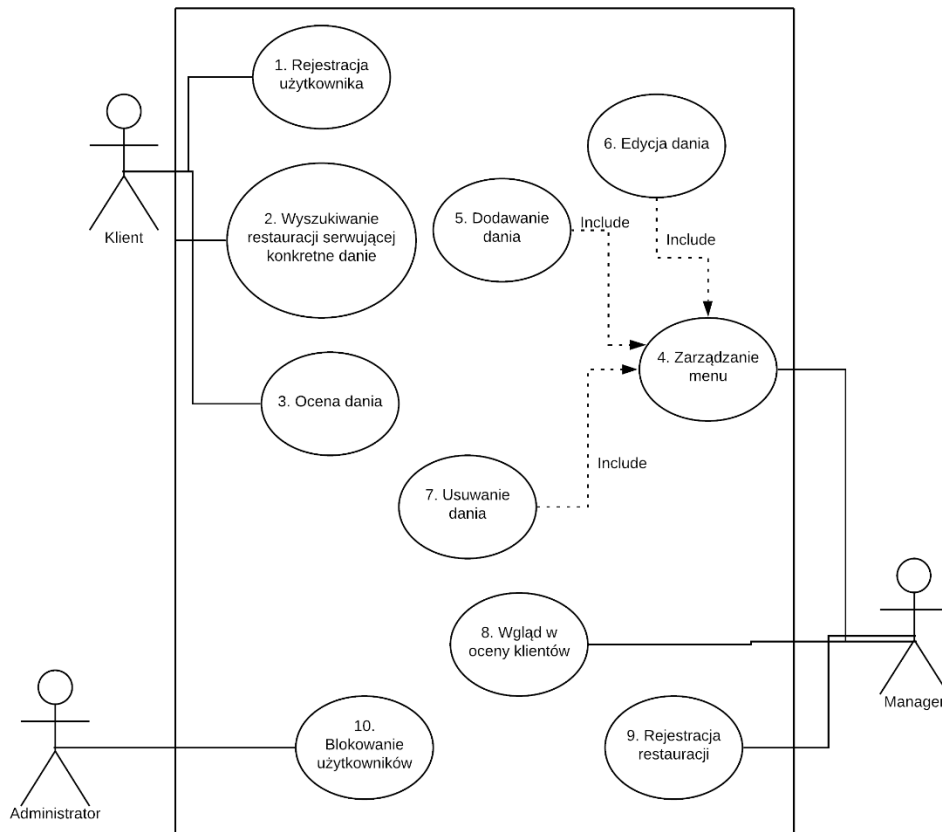
- Kompresja odpowiedzi – reverse proxy poprzez kompresję gzip zmniejsza ilość przesyłanych danych
- „SSL Termination” – aplikacja stanowiąc styk z siecią zewnętrzną pozwala na niestosowanie szyfrowanego połączenia pomiędzy pozostałymi aplikacjami w systemie, co widać na poniższym diagramie. Zakłada się, że w sieci wewnętrznej połączenia szyfrowane nie są konieczne, ponieważ do tego ruchu mają dostęp tylko osoby upoważnione.
- „Caching” – aplikacje typu reverse proxy mają możliwość tymczasowego przechowywania odpowiedzi systemu na konkretne zapytania. Dzięki temu, gdy system będzie miał obsłużyć kilka dokładnie takich samych zapytań, tylko na pierwsze z nich odpowie odpowiednia aplikacja, a na pozostałe zapytania odpowiedź wyśle reverse proxy, tym samym odciążając pozostałe aplikacje z wykonywania tych samych operacji. [14]



Rysunek 1 Schemat środowiska implementacyjnego

3.4.Przypadki użycia

Diagram umieszczony na rysunku 2 prezentuje 3 typy użytkowników jako aktorzy, a także przypisane do nich przypadki użycia. Definiują on kto w aplikacji może wykonywać określone akcje.



Rysunek 2 Diagram przypadków użycia

3.5.Wymagania funkcjonalne

Wymagania wobec aplikacji dzielą się na dwa typy: funkcjonalne i нефункционалне. Pierwszy typ wymagań dotyczy podstawowych funkcji jakie ma spełniać aplikacja, czyli na co będzie pozwalać użytkownikom. W przypadku mojej aplikacji zdefiniowałem wymagania za pomocą kart wymagań, które są zaprezentowane w postaci tabel od 4 do 22.

Tabela 4 Karta wymagań funkcjonalnych: Utworzenie konta przez użytkownika

Nr wymagania: 1	Typ wymagania: Funkcjonalne	Nr. przypadku użycia: 1
Opis: Utworzenie konta przez użytkownika.		
Przesłanka: Użytkownik musi posiadać konto w systemie, aby można było go zidentyfikować oraz przypisać mu odpowiednią dla niego rolę.		
Kryterium spełnienia: Po podaniu adresu email oraz hasła konto użytkownika zapisuje się w bazie danych.		

Tabela 5 Karta wymagań funkcjonalnych: Użytkownika loguje się do systemu

Nr wymagania: 2	Typ wymagania: Funkcjonalne	Nr. przypadku użycia: 1
Opis: Użytkownik loguje się do systemu.		
Przesłanka: Użytkownik w celu potwierdzenia swojej tożsamości podaje unikalny login i hasło do systemu.		
Kryterium spełnienia: Po podaniu prawidłowej pary: login i hasło, użytkownik dostaje dostęp do funkcjonalności odpowiednich do jego roli w systemie.		

Tabela 6 Karta wymagań funkcjonalnych: Wyświetlanie mapy użytkownikowi

Nr wymagania: 3	Typ wymagania: Funkcjonalne	Nr. przypadku użycia: 2
Opis: Wyświetlanie mapy użytkownikowi.		
Przesłanka: Wyświetlenie pustej mapy sugeruje użytkownikowi, że wyszukane dane będą na niej zaprezentowane.		
Kryterium spełnienia: Mapa wyświetla się poprawnie na stronie internetowej.		

Tabela 7 Karta wymagań funkcjonalnych: Wyszukiwanie restauracji w systemie, która ma w swojej karcie podane przez użytkownika danie

Nr wymagania: 4	Typ wymagania: Funkcjonalne	Nr. przypadku użycia: 2
Opis: Wyszukiwanie restauracji w systemie, która ma w swojej karcie podane przez użytkownika danie.		
Przesłanka: Użytkownik powinien mieć możliwość wyszukiwania restauracji, a następnie przeglądania ich na mapie.		
Kryterium spełnienia: Po wpisaniu dania, które widnieje w bazie danych, na mapie zaprezentowane są restauracje, które posiadają wyszukiwane danie w karcie.		

Tabela 8 Karta wymagań funkcjonalnych: Zaprezentowanie użytkownikowi komentarzy i ocen innych użytkowników

Nr wymagania: 5	Typ wymagania: Funkcjonalne	Nr. przypadku użycia: 2
Opis: Zaprezentowanie użytkownikowi komentarzy i ocen innych użytkowników.		
Przesłanka: Użytkownik na podstawie komentarzy i ocen może podjąć lepszą dla niego decyzję, korzystając z doświadczeń innych użytkowników.		
Kryterium spełnienia: Komentarze oraz oceny na temat szukanego dania w wybranej przez niego restauracji są mu zaprezentowane w czytelny sposób.		

Tabela 9 Karta wymagań funkcjonalnych: Dodanie oceny i słownego komentarza do wybranego dania przez użytkownika

Nr wymagania: 6	Typ wymagania: Funkcjonalne	Nr. przypadku użycia: 3
Opis: Dodanie oceny i słownego komentarza do wybranego dania przez użytkownika.		
Przesłanka: Użytkownicy powinni mieć możliwość dzielenia się opiniami na temat dań.		
Kryterium spełnienia: Zalogowany użytkownik ma możliwość dodania komentarza i oceny do dania.		

Tabela 10 Karta wymagań funkcjonalnych: Rejestracja restauracji przez użytkownika

Nr wymagania: 7	Typ wymagania: Funkcjonalne	Nr. przypadku użycia: 9
Opis: Rejestracja restauracji przez użytkownika.		
Przesłanka: Użytkownik powinien mieć możliwość rejestracji swojej restauracji, aby móc dotrzeć do klienta		
Kryterium spełnienia: Po podaniu wymaganych danych: loginu, hasła, nazwy restauracji, oraz jej adresu system dodaje do bazy danych konto użytkownika oraz restaurację, do niego przypisaną.		

Tabela 11 Karta wymagań funkcjonalnych: Użytkownik dodaje nowe danie do menu swojej restauracji

Nr wymagania: 8	Typ wymagania: Funkcjonalne	Nr. przypadku użycia: 4,5
Opis: Użytkownik dodaje nowe danie do menu swojej restauracji.		
Przesłanka: W celu dodania faktycznego menu swojej restauracji, użytkownik powinien móc dodawać dania w systemie.		
Kryterium spełnienia: Po wpisaniu nazwy i ceny system dodaje do bazy danych danie w restauracji użytkownika.		

Tabela 12 Karta wymagań funkcjonalnych: Użytkownik zmienia nazwę dania w menu swojej restauracji

Nr wymagania: 9	Typ wymagania: Funkcjonalne	Nr. przypadku użycia: 4,6
Opis: Użytkownik zmienia nazwę dania w menu swojej restauracji.		
Przesłanka: W przypadku pomyłki użytkownik powinien móc edytować nazwę dania w swoim menu.		
Kryterium spełnienia: Po podaniu nowej nazwy dania system aktualizuje informacje w bazie danych.		

Tabela 13 Karta wymagań funkcjonalnych: Użytkownik zmienia cenę dania w menu swojej restauracji

Nr wymagania: 10	Typ wymagania: Funkcjonalne	Nr. przypadku użycia: 4,6
Opis: Użytkownik zmienia cenę dania w menu swojej restauracji.		
Przesłanka: W przypadku pomyłki użytkownik powinien móc edytować cenę dania w swoim menu.		
Kryterium spełnienia: Po podaniu nowej ceny dania system aktualizuje informacje w bazie danych.		

Tabela 14 Karta wymagań funkcjonalnych: Użytkownik usuwa danie z menu swojej restauracji

Nr wymagania: 11	Typ wymagania: Funkcjonalne	Nr. przypadku użycia: 4,7
Opis: Użytkownik usuwa danie z menu swojej restauracji.		
Przesłanka: Kiedy restauracja nie serwuje już dania, użytkownik powinien mieć możliwość usunięcia dania z systemu, aby nie wprowadzać w błąd klientów.		
Kryterium spełnienia: Po wywołaniu odpowiedniej akcji system usuwa danie z bazy danych.		

Tabela 15 Karta wymagań funkcjonalnych: Użytkownik ma wgląd w informację o ocenach dań w swojej restauracji.

Nr wymagania: 12	Typ wymagania: Funkcjonalne	Nr. przypadku użycia: 8
Opis: Użytkownik ma wgląd w informację o ocenach dań w swojej restauracji.		
Przesłanka: W celach analizy biznesowej użytkownik powinien mieć możliwość przeglądania informacji na temat ocen klientów.		
Kryterium spełnienia: Użytkownik widzi informacje na temat ocen konkretnych dań.		

Tabela 16 Karta wymagań funkcjonalnych: Administrator blokuje użytkownika.

Nr wymagania: 13	Typ wymagania: Funkcjonalne	Nr. przypadku użycia: 10
Opis: Administrator blokuje użytkownika.		
Przesłanka: W przypadku użytkownika łamiącego regulamin lub złośliwego oprogramowania podającego się za użytkownika administrator systemu powinien móc zablokować konkretnego użytkownika.		
Kryterium spełnienia: Administrator po podaniu odpowiednich danych blokuje użytkownika w systemie. Zablokowany użytkownik nie może się zalogować.		

Tabela 17 Karta wymagań funkcjonalnych: Administrator odblokowuje użytkownika.

Nr wymagania: 14	Typ wymagania: Funkcjonalne	Nr. przypadku użycia: 10
Opis: Administrator odblokowuje użytkownika.		
Przesłanka: W przypadku pomyłki administratora proces blokady powinien być odwracalny.		
Kryterium spełnienia: Administrator ma możliwość, po podaniu odpowiednich danych, na odblokowanie wcześniej zablokowanego użytkownika.		

3.6. Wymagania нефunkcjonalne

Drugim typem wymagań wobec projektowanej aplikacji są wymagania нефunkcjonalne. Dotyczą one takich dziedzin jak bezpieczeństwo aplikacji, polityka przechowywania danych w bazie danych, wyglądu, utrzymania aplikacji w środowisku implementacyjnym, czy doświadczenia użytkownika. W przypadku mojej aplikacji prezentują się one następująco:

Tabela 18 Karta wymagań niefunkcjonalnych: Hasła w bazie danych są przechowywane w formie niejawnej

Nr wymagania: 16	Typ wymagania: Niefunkcjonalne	Nr. przypadku użycia: brak
Opis: Hasła w bazie danych są przechowywane w formie niejawnej.		
Przesłanka: Hasła muszą być zabezpieczone przed ewentualnym wyciekiem.		
Kryterium spełnienia: Hasła są zabezpieczone funkcją haszującą.		

Tabela 19 Karta wymagań niefunkcjonalnych: Baza danych jest zabezpieczona hasłem

Nr wymagania: 17	Typ wymagania: Niefunkcjonalne	Nr. przypadku użycia: brak
Opis: Baza danych jest zabezpieczona hasłem.		
Przesłanka: Dostęp do bazy danych powinien być ograniczony ze względu na przechowywanie w niej danych klientów.		
Kryterium spełnienia: Nie da się w prosty sposób wyciągnąć danych z bazy bez podania odpowiedniego loginu i hasła.		

Tabela 20 Karta wymagań niefunkcjonalnych: Połączenia poza siecią lokalną są szyfrowane

Nr wymagania: 18	Typ wymagania: Niefunkcjonalne	Nr. przypadku użycia: brak
Opis: Połączenia poza siecią lokalną są szyfrowane.		
Przesłanka: Aplikacja kliencka będzie wymieniać z aplikacją serwerową dane, które są narażone na sniffing.		
Kryterium spełnienia: Połączenia poza siecią lokalną są zabezpieczone protokołem HTTPS.		

Tabela 21 Karta wymagań niefunkcjonalnych: Aplikacja tworzy logi z wykonywanych operacji.

Nr wymagania: 19	Typ wymagania: Niefunkcjonalne	Nr. przypadku użycia: brak
Opis: Aplikacja tworzy logi z wykonywanych operacji.		
Przesłanka: W środowisku produkcyjnym nie można zajrzeć do kodu i sprawdzić, w którym miejscu aplikacja przestała działać. Logi pozwolą na wykrycie miejsca usterki.		
Kryterium spełnienia: Aplikacja tworzy plik z logami, które zawierają datę operacji oraz jej typ.		

Tabela 22 Karta wymagań niefunkcjonalnych: Aplikacja jest kompatybilna z urządzeniami mobilnymi

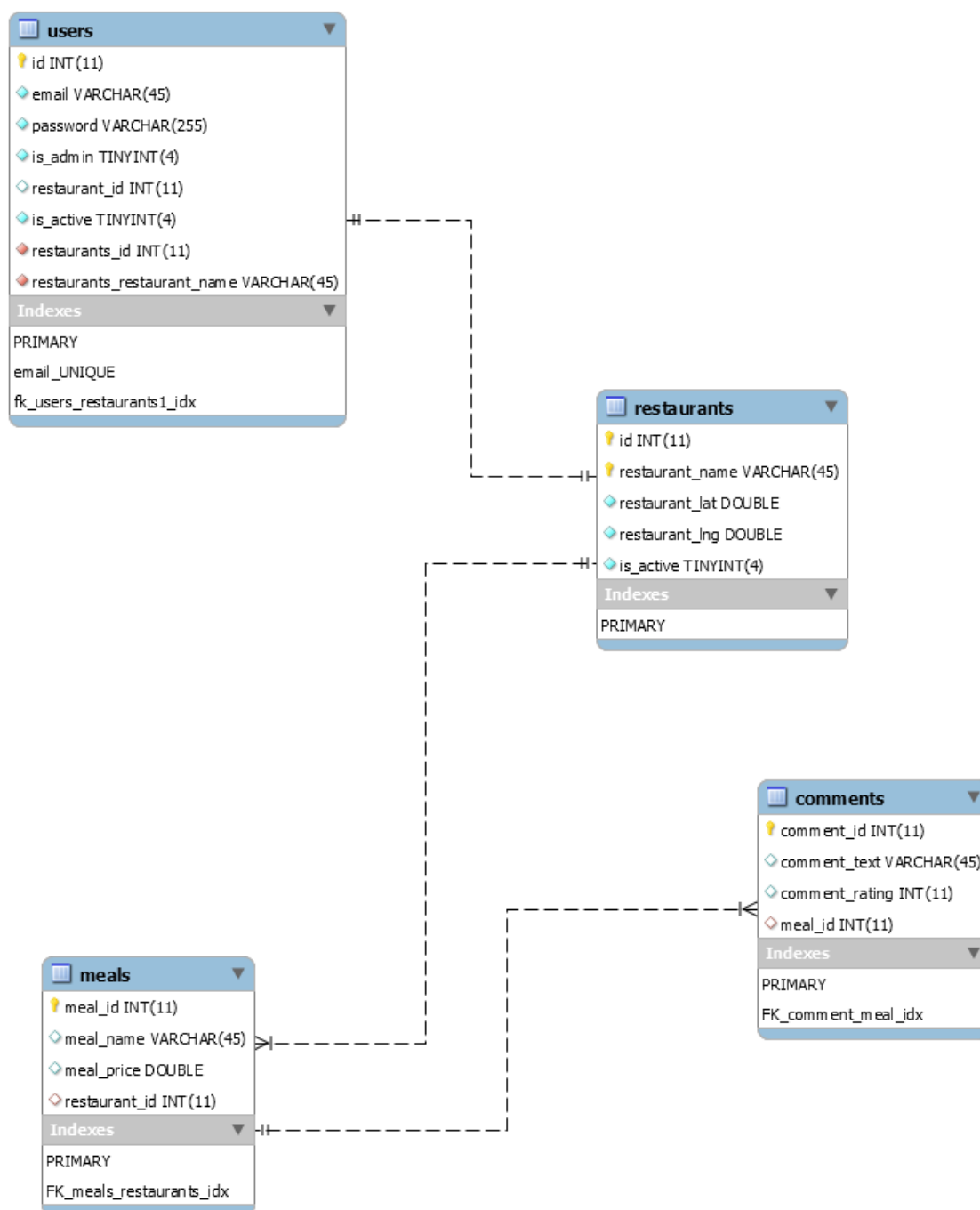
Nr wymagania: 20	Typ wymagania: Niefunkcjonalne	Nr. przypadku użycia: brak
Opis: Aplikacja jest kompatybilna z urządzeniami mobilnymi.		
Przesłanka: Znaczna część użytkowników – klientów korzysta głównie z urządzeń mobilnych.		
Kryterium spełnienia: Aplikacja pozostaje funkcjonalna na urządzeniach mobilnych.		

4. Projekt aplikacji

Przed przystąpieniem do prac nad implementacją aplikacji należy ją najpierw zaprojektować. Uprzednie przemyślenie architektury aplikacji pozwala uniknąć błędów oraz pomaga lepiej dodawać kolejne funkcjonalności. Wyróżniłem bazę danych, sieciowy interfejs użytkownika (ang. Front-end) oraz aplikację serwerową (ang. Back-end). Poniżej przedstawiłem schematy architektury każdego z wymienionych modułów oraz zastosowane rozwiązania implementacyjne.

4.1. Baza danych

Po przeanalizowaniu wymagań aplikacji wobec bazy danych przybrała ona postać przedstawioną na rysunku 3.



Rysunek 3 Schemat bazy danych

Jak widać na diagramie baza składa się z 4 tabel:

- Users

Jest to tabela przechowująca dane o zarejestrowanych użytkownikach, w tym dane służące do logowania, czyli email oraz hasło. Hasło jest przechowywane w formie niejawnej za pomocą funkcji skrótu SHA-256. Pozwala ona stworzyć ciąg znaków z hasła użytkownika. Dzięki temu w bazie nie przechowywane są hasła

w formie jawnej. Zwiększa to ich bezpieczeństwo, ponieważ jedynym sposobem na odwrócenie działania funkcji skrótu jest stosowanie jej na losowe ciągi znaków, do momentu uzyskania szukanego hasła.

Oprócz danych logowania tabela przechowuje informacje o roli użytkownika. Jeżeli jest on administratorem flaga „is_admin” przyjmuje wartość 1, w przeciwnym wypadku przyjmuje wartość 0. Jeżeli użytkownik jest właścicielem restauracji posiada on pole „restaurant_id” przyjmuje wartość różną od 0, a w przeciwnym wypadku użytkownik uznawany za zwykłego użytkownika. Dodatkowo pole „restaurant_id” jest kluczem obcym odnoszącym się do tabeli restaurants.

- Restaurants

Tabela przechowuje podstawowe informacje o restauracjach. Na pole z nazwą restauracji nałożone jest wymaganie unikalności, aby uniknąć dwóch restauracji o tej samej nazwie. Zamiast przechowywać adres w bazie danych przechowywane są tylko długość i szerokość geograficzna. Wynika to z faktu, że mapa dla języka JavaScript udostępniana przez google korzysta właśnie z tych informacji do umiejscowienia pinezki na mapie.

- Meals

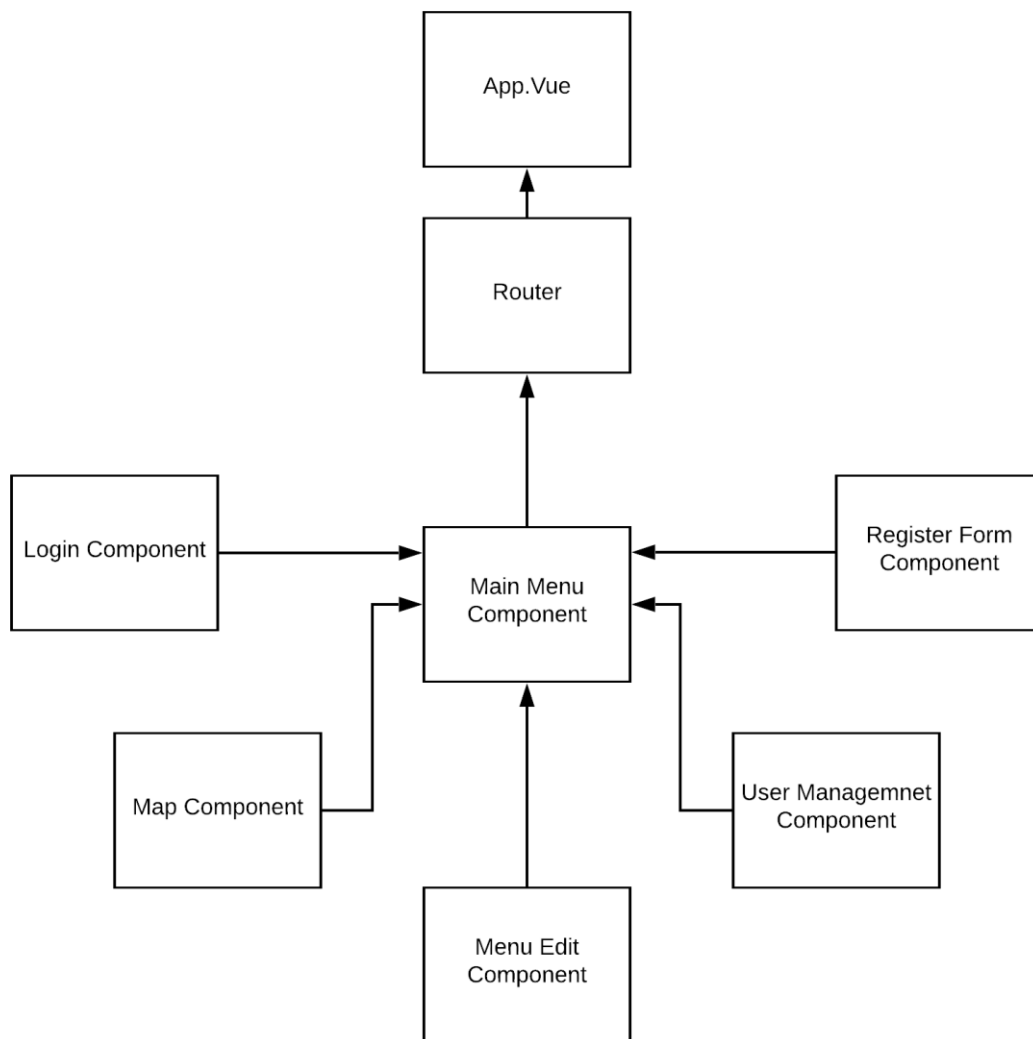
Tabela przechowuje informację o wszystkich dostępnych daniach w systemie. Menu danej restauracji definiowane jest poprzez wszystkie wpisy z tabeli „meals”, które zawierają w polu „restaurant_id” id danej restauracji.

- Comments

Tabela przechowuje wszystkie komentarze dodane przez użytkowników. Na komentarz składa się krótki tekst oraz ocena liczbowa w zakresie od 1 do 5.

4.2. Webowy interfejs użytkownika (ang. Front-end application)

Sieciowy interfejs użytkownika zbudowany jest w oparciu o architekturę SPA (ang. Single Page Application). Głównym modułem aplikacji jest plik App.vue, który korzystając z obiektu typu Router wyświetla odpowiednie komponenty aplikacji zależnie od podanej ścieżki. Zamyśl przedstawiony jest na rysunku 4.



Rysunek 4 Schemat ideowy interfejsu użytkownika

Podstawowym komponentem aplikacji, czyli widocznym po wejściu na adres strony, jest „Main Menu Component”. Jest to obiekt zawierający układ strony i wskazuje Router’owi, gdzie wyświetlać kolejne komponenty.

Domyślnym widokiem wyświetlanym przez Router w komponencie menu głównego jest mapa z wyszukiwarką restauracji. Pozwala on wyszukać danie, a następnie wyświetla wyniki wyszukiwania na mapie. Po kliknięciu na wyświetloną pinezkę użytkownik ma dostęp do informacji o daniu oraz jeżeli jest zalogowany może ocenić wybrane danie.

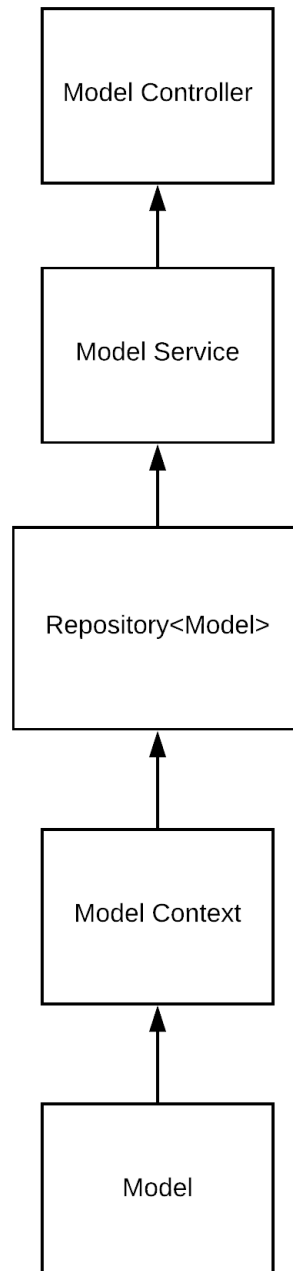
Aplikacja zawiera dwa formularze: pierwszy służący do logowania, a drugi służący do rejestracji użytkowników. Formularz rejestracyjny jest podzielony na dwie części. Pierwsza z nich pozwala zarejestrować tylko użytkownika, a druga użytkownika wraz z restauracją.

Dla użytkowników udostępnione są dwa widoki służące do edycji wpisów w bazie danych. Pierwszy udostępniony jest tylko dla użytkownika posiadającego restaurację i pozwala na przeglądanie menu oraz jego edycję. Drugi jest udostępniony dla administratora i pozwala blokować użytkowników.

Tak jak wspomniałem aplikacja rozróżnia typ użytkownika, a następnie reaguje na jego rolę w aplikacji. Dzieje się to po stronie interfejsu oraz po stronie aplikacji serwerowej. W celu zapisania informacji o zalogowaniu tworzone są tokeny JWT (ang. Json Web Token). Użytkownik po podaniu danych logowania otrzymuje indywidualny token, który pozwala na potwierdzenie swojej tożsamości bez ponownego podawania danych logowania. Tokeny przechowywane są w pamięci przeglądarki za pomocą Local Storage, który jest słownikiem typu klucz – wartość.

4.3. Aplikacja serwerowa (ang. Back-end application)

Aplikacja serwerowa jest zaprojektowana jako REST API. W związku z tym ma ona działać na zasadzie zapytanie-odpowiedź. W tym celu zostały zaimplementowane następujące typy klas współpracujących ze sobą: kontroler (ang. controller), serwis (ang. service), repozytorium (ang. repository), kontekst (ang. kontekst). Schemat ideowy aplikacji przedstawiony jest na rysunku 5



Rysunek 5 Schemat ideowy klas w aplikacji serwerowej

Każda z klas przedstawionych na schemacie ma inną rolę:

- Kontroler ma za zadanie odebrać zapytania, sprawdzić uprawnienia użytkownika, a po wykonaniu wszystkich operacji odesłać odpowiedź oznaczoną adekwatnym kodem HTTP.
- Serwis ma za zadanie wykonać odpowiednią logikę zapytania.
- Repozytorium oddziela część aplikacji odpowiedzialną za obsługę danych od reszty procesów. Metody tej klasy zwracają i przyjmują dane w takiej samej formie, niezależnie od tego w jaki sposób te dane są przechowywane.
- Kontekst odpowiada za operacje na bazie danych.

Wszystkie skupiają się na operacjach wokół jednego modelu danych. Model danych oznacza klasę obiektów zawierający tylko atrybuty, przedstawiający rzeczywisty obiekt. Przykładem takiego modelu będzie klasa `Comment.cs`, której implementację przedstawia rysunek 6.

```
public class Comment
{
    Odwołania: 3
    public int Rate { get; set; }
    Odwołania: 3
    public string Text { get; set; }
    Odwołania: 4
    public int MealId { get; set; }
    — odwołania
    public override string ToString() => $"0,{Text}\", \"{Rate}\", \"{MealId}\"";
}
```

Rysunek 6 Przykład klasy - modelu danych

Jak widać na zamieszczonym rysunku klasa zawiera tylko atrybuty, wraz z metodami pozwalającymi na odczyt i przypisanie wartości (ang. getters, setters), oraz metodę zamieniającą obiekt tej klasy na ciąg znaków.

Modele danych pozwalają zorganizować operacje na danych w aplikacji oraz ułatwiają operację na bazie danych, gdyż modele w kodzie zawierają te same atrybuty, co encje w bazie danych.

W aplikacji zostało wykorzystanie wstrzykiwanie zależności (ang. dependency injection), korzystając z wsparcia użytej technologii – ASP.NET Core. Zależnością w

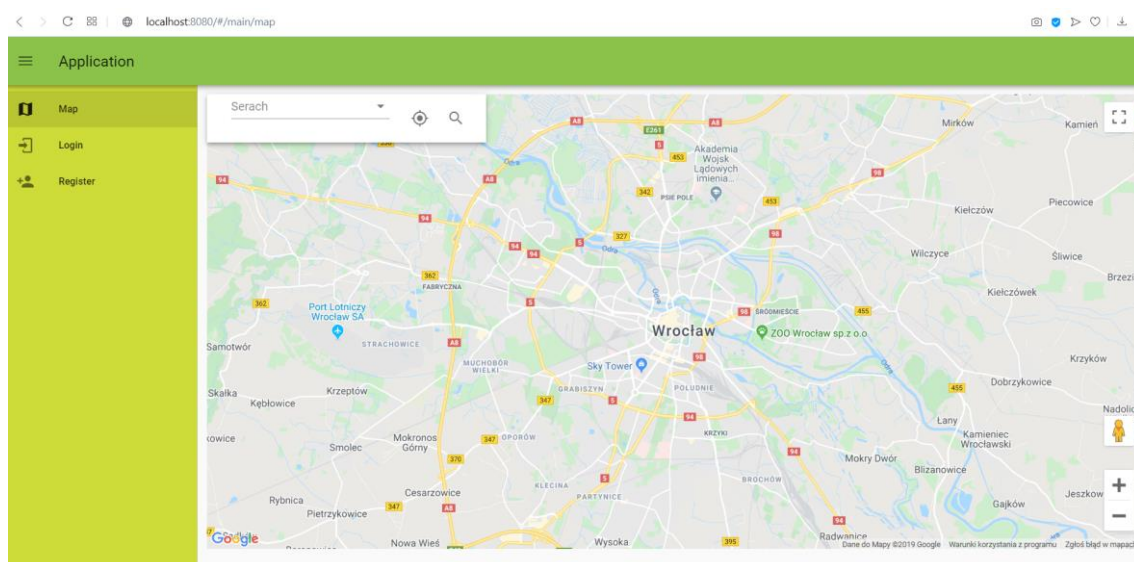
tym przypadku należy rozumieć obiekt, z którego korzysta inny obiekt. Najprostszym sposobem rozwiązania takiej sytuacji jest powołanie wymaganego obiektu, w obiekcie, który go potrzebuje. Jest to błędne podejście, które skutkuje powstawaniem problemów. Pierwszym z nich, fakt, że aby wymienić zależność należy zmodyfikować obiekt, który z niej korzysta. Samo w sobie jest to problematyczne, ponieważ wymaga zmian w działającym już fragmencie aplikacji. Problem jednak narasta w momencie, gdy aplikacja jest większa i zawiera drzewa zależności, czyli zależność, ma swoją zależność, która ma swoją zależność itd. W tym przypadku zmiana jednego elementu drzewa zależności wymaga zmian w wielu miejscach w kodzie aplikacji.

Rozwiązaniem tych problemów jest uzależnienie obiektu od abstrakcji – interfejsu lub klasy abstrakcyjnej. ASP.NET Core zapewnia serwis, który w momencie, gdy obiekt będzie chciał skorzystać z zależności powoła jej instancję, a następnie pozbędzie się jej z pamięci, gdy przestanie już być potrzebna. [15]

5. Implementacja

W tym rozdziale znajdują się wyniki testów funkcjonalnych aplikacji. Przypadki testowe to wymagania przedstawione wcześniej w rozdziale 3.5 oraz 3.6.

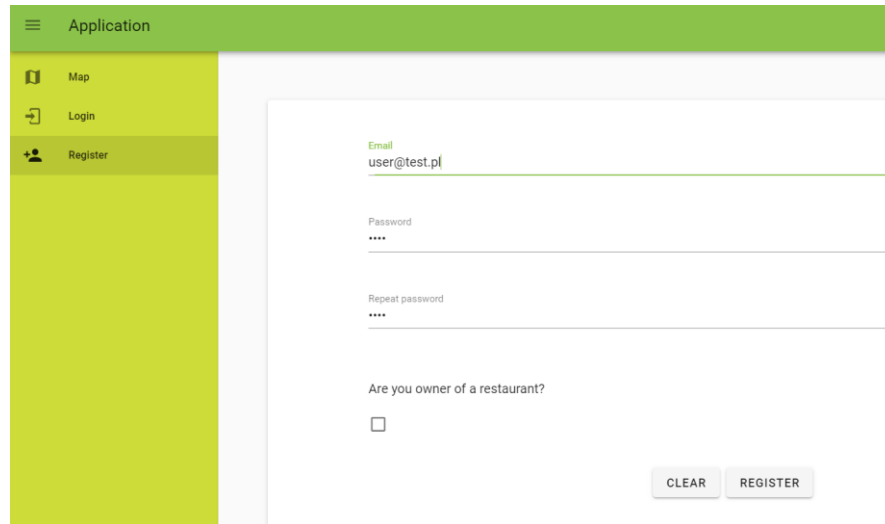
Po wejściu na adres aplikacji użytkownik widzi widok przedstawiony na rysunku 7.



Rysunek 7 Główny widok aplikacji

5.1. Utworzenie konta przez użytkownika

W celu utworzenia konta użytkownik wchodzi i wybiera z menu zakładkę Register, a następnie uzupełnia formularz, który przedstawiony jest na rysunku 8 i potwierdza operację przyciskiem Register.

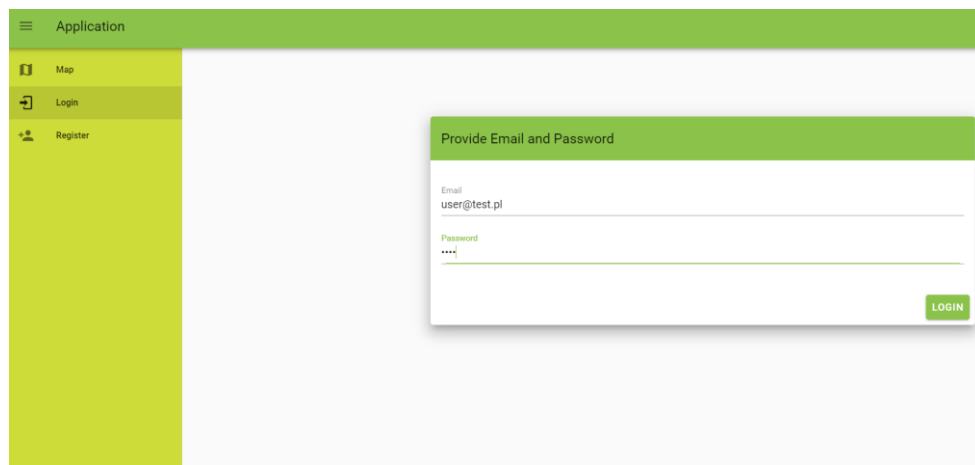


The screenshot shows a mobile application interface with a green header bar labeled 'Application'. On the left, there is a yellow sidebar menu with three options: 'Map', 'Login', and 'Register'. The 'Register' option is selected. The main content area displays a registration form with the following fields: 'Email' (containing 'user@test.pl'), 'Password' (masked with four dots), and 'Repeat password' (also masked with four dots). Below these fields is a checkbox labeled 'Are you owner of a restaurant?'. At the bottom right of the form are two buttons: 'CLEAR' and 'REGISTER'.

Rysunek 8 Widok formularza rejestracji użytkownika

5.2. Użytkownik loguje się do systemu

W celu zalogowania się użytkownik wybiera zakładkę Login z bocznego menu, a następnie podaje email oraz hasło. Formularz przedstawia rysunek 9.

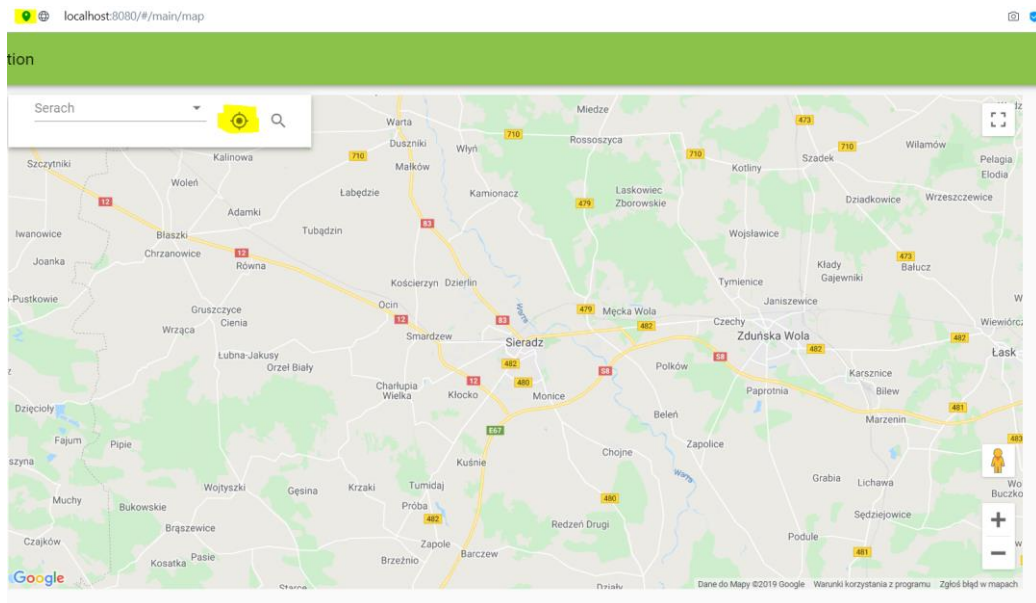


The screenshot shows the same mobile application interface as in Figure 8, but with the 'Login' option selected in the sidebar menu. A modal dialog box titled 'Provide Email and Password' is displayed in the center. It contains two input fields: 'Email' (containing 'user@test.pl') and 'Password' (masked with four dots). A green 'LOGIN' button is located at the bottom right of the modal.

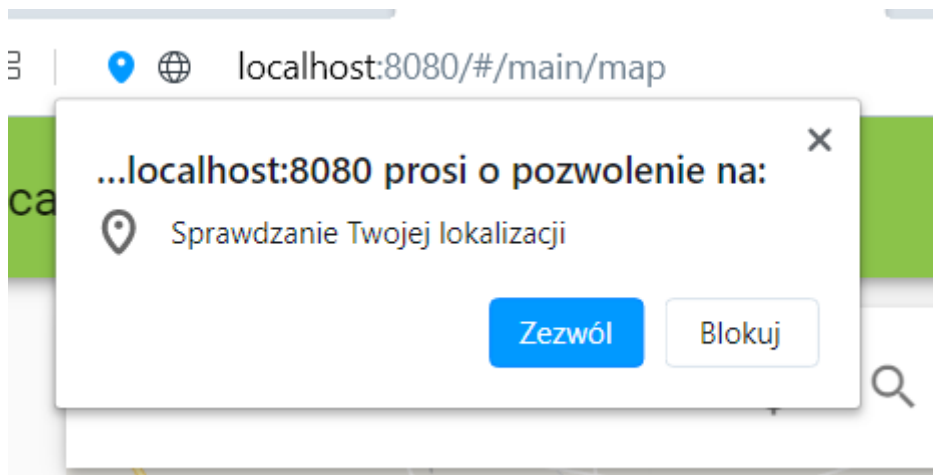
Rysunek 9 Widok formularza logowania się użytkownika

5.3. Wyświetlanie mapy użytkownikowi

Mapa wyświetlana jest na głównym widoku aplikacji. Dodatkowo można skorzystać z opcji auto lokalizacji, która wycentruje mapę na aktualną lokalizację urządzenia. Podczas pierwszego korzystania z tej funkcjonalności użytkownik proszony jest o zgodę na wykorzystanie lokalizacji jego urządzenia. Rysunki 10 i 11 przedstawiają opisane działania.



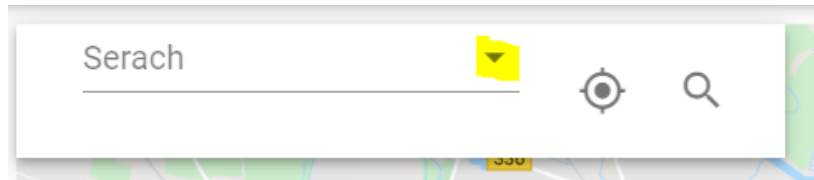
Rysunek 10 Widok mapy z wyszukiwarką



Rysunek 11 Prośba o zgodę na wykorzystanie lokalizacji

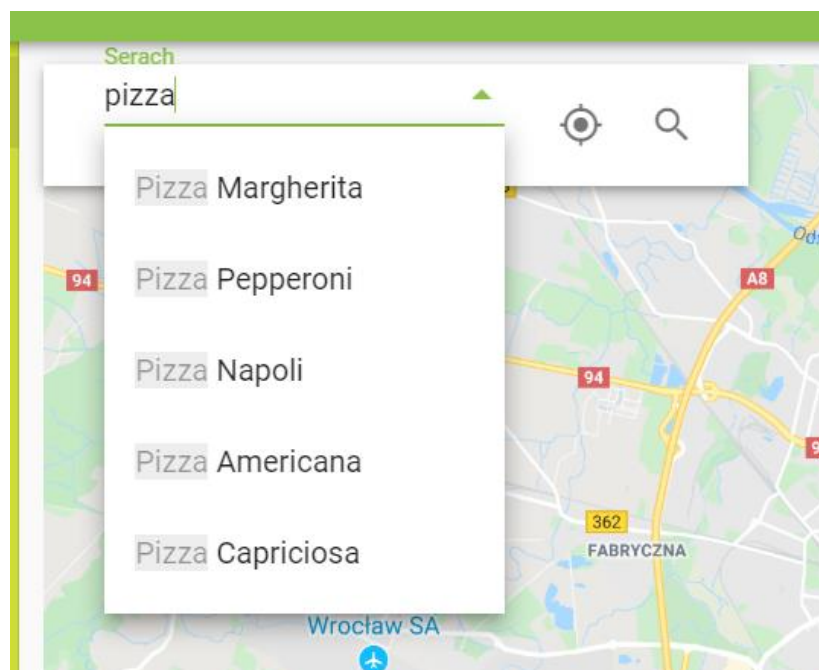
5.4. Wyszukiwanie restauracji w systemie, która ma w swojej karcie podane przez użytkownika danie.

W celu wyszukania dania użytkownik wpisuje w wyszukiwarkę danie, które chciałby znaleźć lub może rozwinąć listę dań poprzez kliknięcie na ikonę strzałki. Jej umiejscowienie przedstawia rysunek 12.



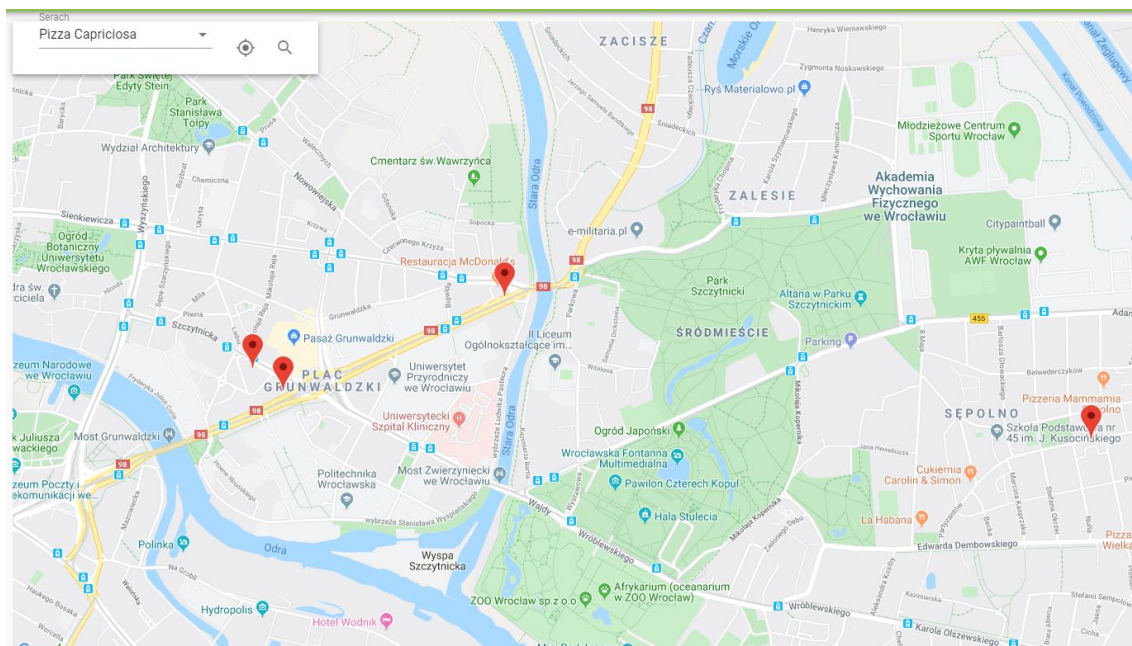
Rysunek 12 Wyszukiwarka restauracji

W obu przypadkach rozwinię się lista odpowiedzi. Im więcej znaków poda użytkownik, tym bardziej zawężona będzie lista, przedstawiona na rysunku 13.



Rysunek 13 Wyszukiwarka restauracji: lista odpowiedzi

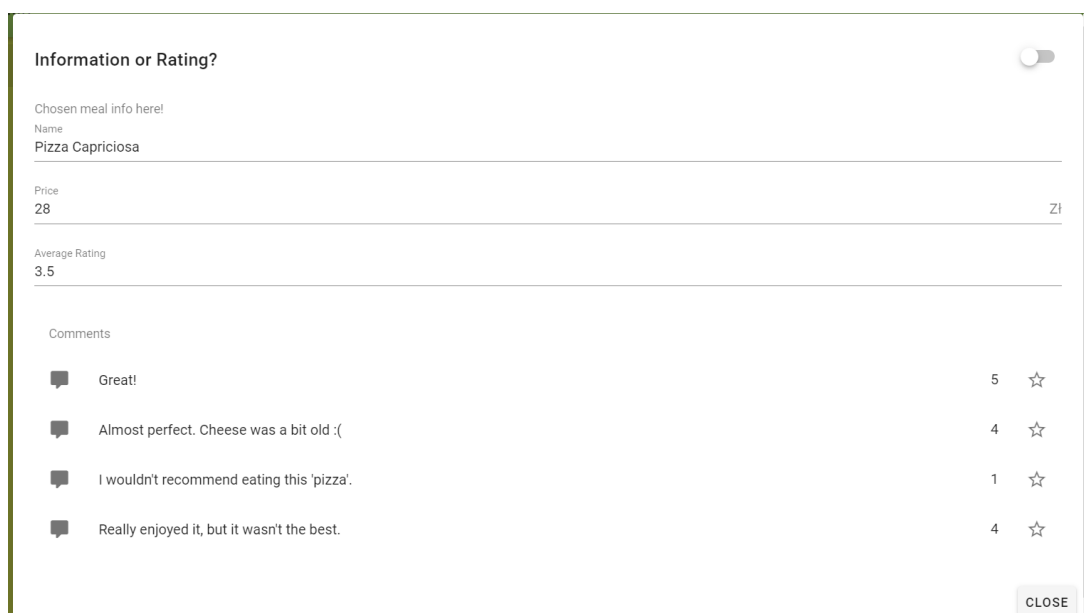
Po wybraniu dania z listy na mapie zostaną wyświetlone lokalizację restauracji, które mają szukane danie w karcie. Sposób prezentacji wyników wyszukiwania przedstawiony jest na rysunku 14.



Rysunek 14 Wyszukiwanie restauracji: Przedstawienie wyników wyszukiwania

5.5. Zaprezentowanie użytkownikowi komentarzy i ocen innych użytkowników

Po kliknięciu w pinę na mapie aplikacja wyświetla okno prezentujące cenę dania, średnią ocenę oraz listę wszystkich komentarzy użytkowników. Jego wygląd przedstawia rysunek 15.

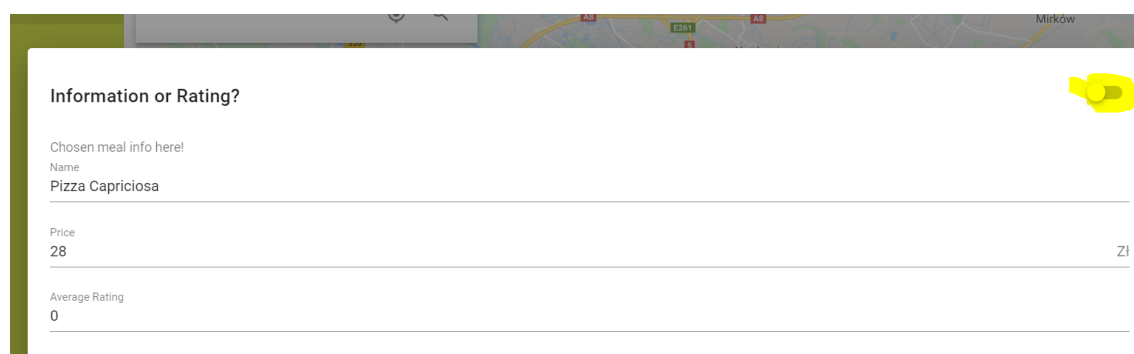


Rysunek 15 Widok informacji o daniu

Zestawienie ilości komentarzy ze średnią oceną pozwala użytkownikowi na dokonanie świadomego wyboru. Średnia 5.0 wynikająca z jednej oceny jest mniej wiarygodna od średniej 3.8 wynikającej z dużej liczby ocen. Pozwala to uniknąć sytuacji, w której właściciel restauracji usuwa słabo ocenione danie ze swojego menu, aby zaraz dodać nowe o takiej samej nazwie.

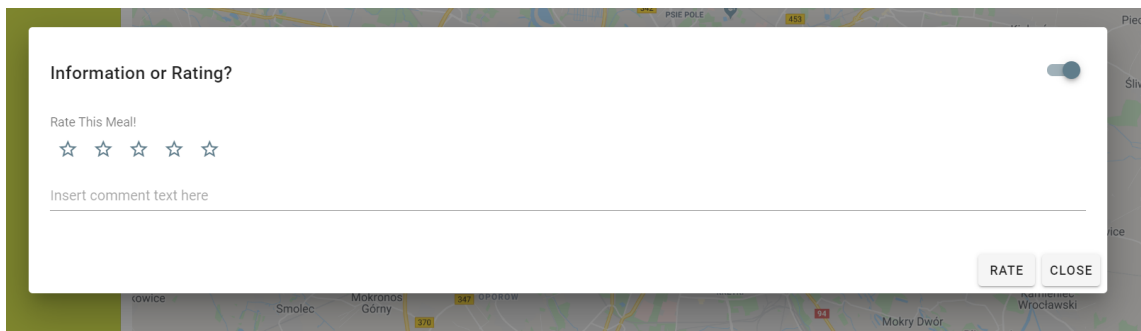
5.6. Dodanie oceny i słownego komentarza do wybranego dania przez użytkownika

W celu dodania oceny i komentarza użytkownik musi być zalogowany w aplikacji. Wtedy odblokuje się przycisk, przedstawiony na rysunku 16, pozwalający wywołać formularz dodawania oceny.



Rysunek 16 Przycisk przekierowujący na widok dodawania komentarza

Po przełączeniu okna prezentuje się ono tak jak na rysunku 17.



Rysunek 17 Widok dodawania komentarza

Po zaznaczeniu ilości gwiazdek oraz wpisaniu komentarza należy nacisnąć przycisk Rate, aby zapisać ocenę.

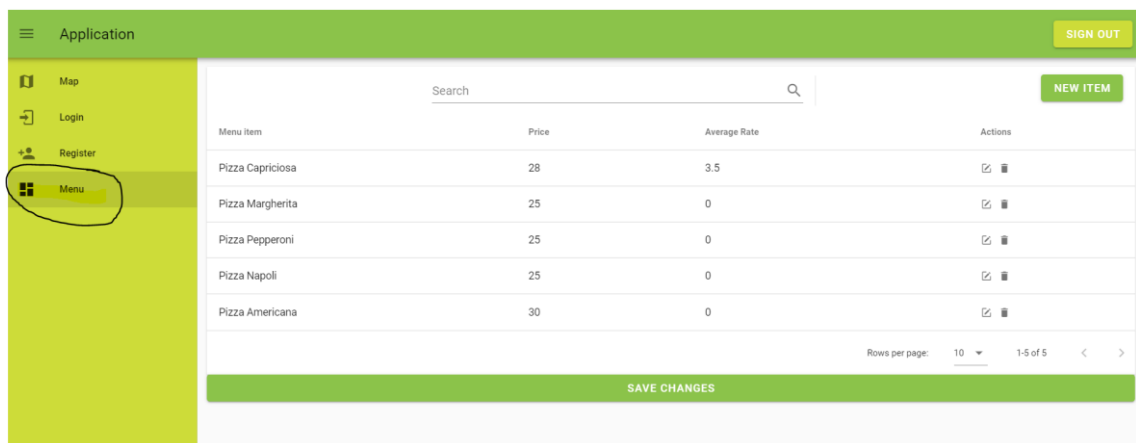
5.7. Rejestracja restauracji przez użytkownika

W celu rejestracji restauracji użytkownik korzysta z tego samego formularza jak w przypadku zwykłej rejestracji użytkownika, jednak tym razem zaznacza, że jest właścicielem restauracji. Skutkuje to pojawieniem się dodatkowych pól, w których należy podać informację o restauracji. Umieszczenie przycisku zmiany formularza prezentuje rysunek 18.

Rysunek 18 Formularz rejestracji restauracji

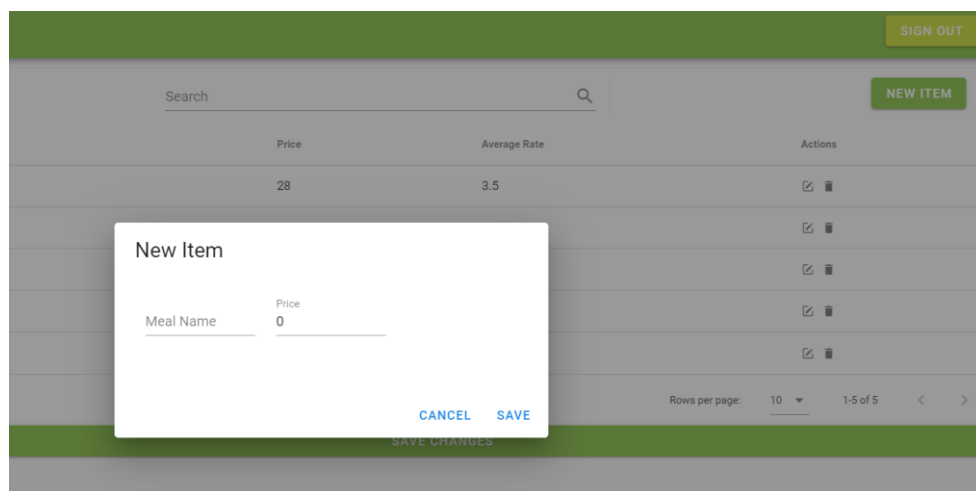
5.8. Użytkownik dodaje nowe danie do menu swojej restauracji

Po zalogowaniu się jako właściciel restauracji użytkownik zostaje przekierowany na panel edycji menu. Dodatkowo pojawia się opcja dostępu do tego panelu z bocznego menu. Proces przedstawiają rysunki 19 i 20.



Rysunek 19 Widok edycji menu

W celu dodania nowego dania należy użyć guzika New Item, następnie podać informacje o daniu, a potem w celu zapisania danych w aplikacji użyć przycisku Save Changes.



Rysunek 20 Widok edycji menu: dodawanie nowego dania

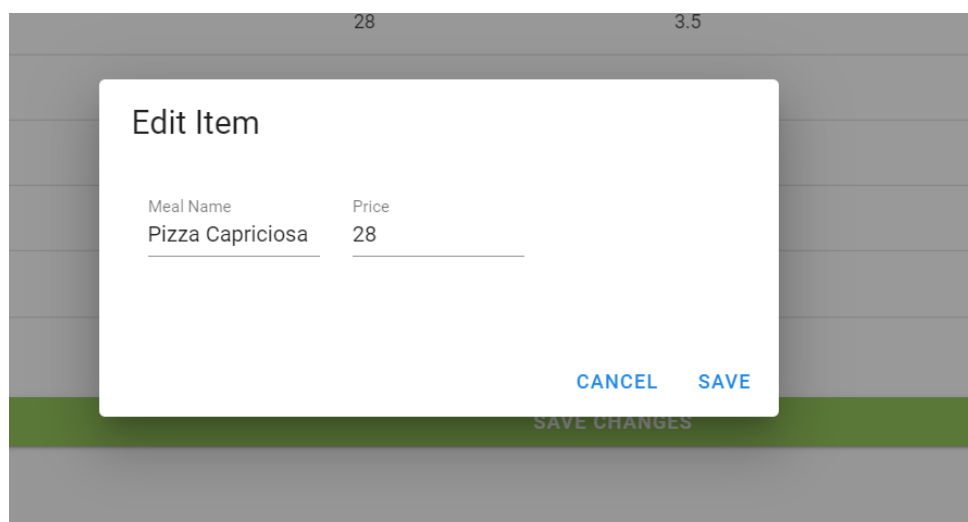
5.9. Użytkownik zmienia nazwę dania w menu swojej restauracji

W celu edycji dania należy wybrać je z listy, a następnie użyć pierwszego przycisku w kolumnie Actions. Proces przedstawiony jest na rysunkach 21 i 22.

Menu Item	Price	Average Rate	Actions
Pizza Capriciosa	28	3.5	 

Rysunek 21 Widok edycji menu: przycisk wywołujący okno edycji dania

Następnie wyświetli się ekran edycji, gdzie należy dokonać zmian. Następnie przycisk Save zapisuje zmiany lokalnie, a w celu zapisania ich w bazie danych należy użyć przycisku Save Changes.



Rysunek 22 Widok edycji menu: okno edycji dania

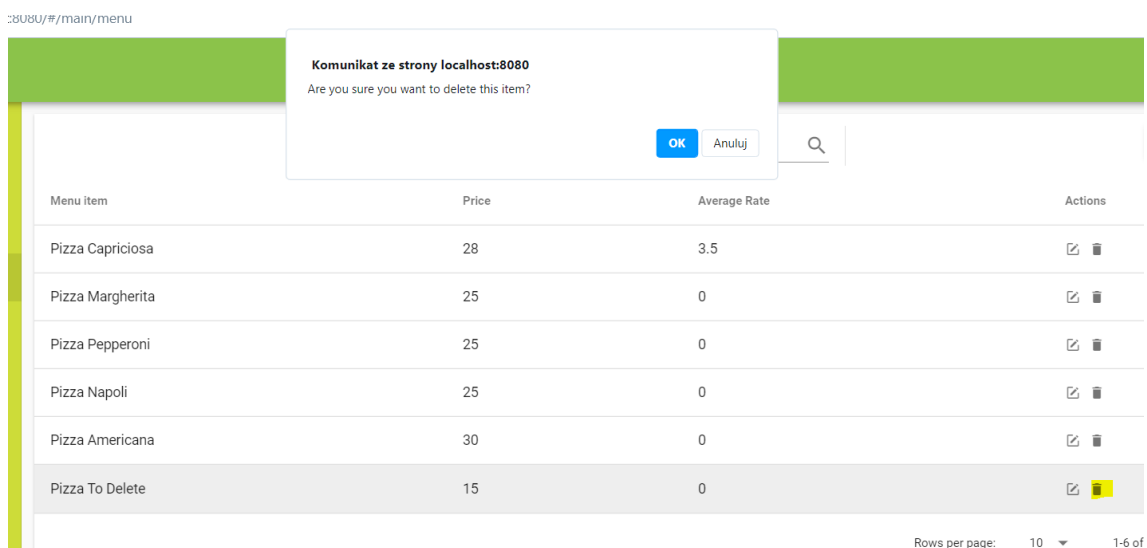
5.10. Użytkownik zmienia cenę dania w menu swojej restauracji

W celu zmiany ceny dania należy postępować tak jak w przypadku zmiany nazwy (5.9).

5.11. Użytkownik usuwa danie z menu swojej restauracji

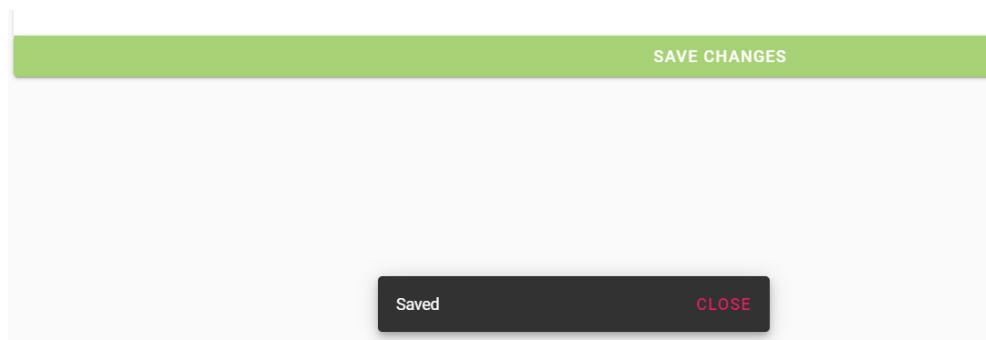
W celu usunięcia dania z menu należy w kolumnie Actions wybrać drugi przycisk, a następnie przyciskiem Save Changes zapisać zmiany w bazie danych.

Aplikacja prosi o potwierdzenie przed usunięciem wpisu z listy. Proces usuwania dania z restauracji oraz komunikaty przedstawione są na rysunkach 23 i 24.



Rysunek 23 Widok edycji menu: Usuwanie dania z listy

W przypadku poprawnego zapisania zmian pojawia się potwierdzenie.



Rysunek 24 Widok edycji menu: komunikat o poprawnym zapisaniu danych w bazie

5.12. Użytkownik ma wgląd w informację o ocenach dań w swojej restauracji

Na ekranie edycji menu użytkownik widzi średnią ocenę dania w kolumnie Average Rate. Sposób prezentacji ocen przedstawiony jest na rysunku 25.

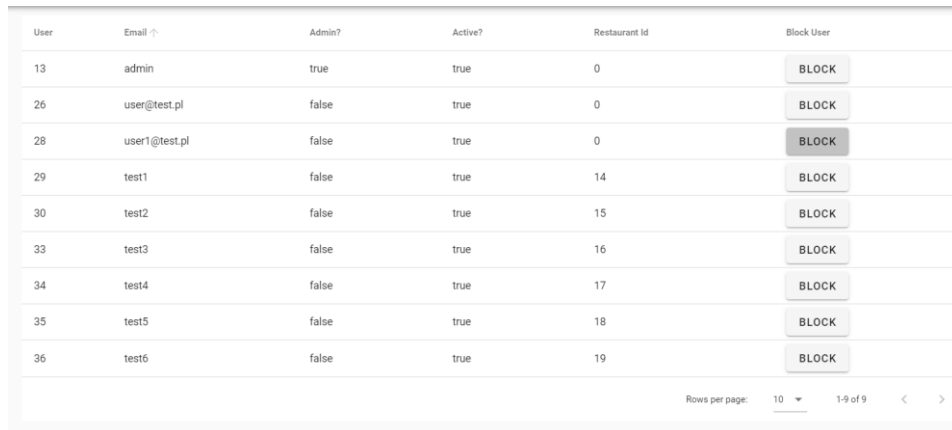
Average Rate	
	3.5
	0
	0
	0
	0

Rysunek 25 Widok edycji menu: przedstawienie ocen właścicielowi restauracji

Tak jak widać, gdy nie ma jeszcze żadnych ocen wyświetlane jest 0.

5.13. Administrator blokuje użytkownika

W celu blokady użytkownika należy być zalogować się jako administrator. Po zalogowaniu użytkownik zostanie przeniesiony do panelu administratorskiego. Po użyciu przycisku z ostatniej kolumny użytkownik zostanie zablokowany. Przycisk zaznaczony jest na rysunku 26.



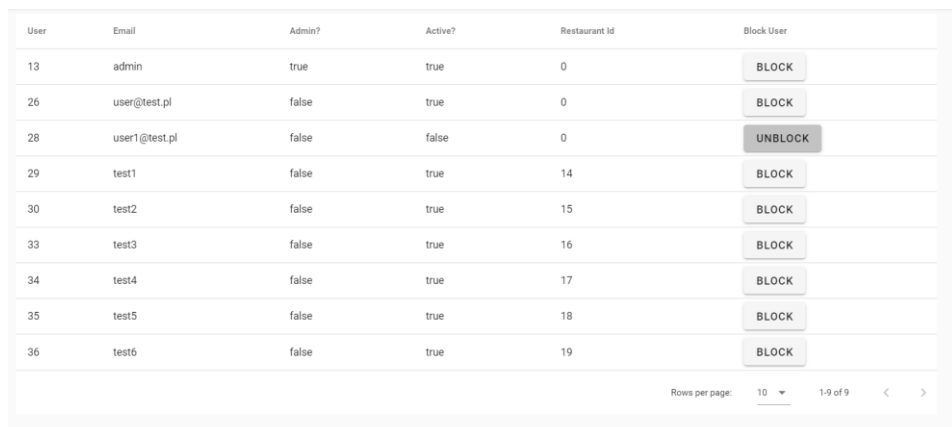
User	Email	Admin?	Active?	Restaurant Id	Block User
13	admin	true	true	0	BLOCK
26	user@test.pl	false	true	0	BLOCK
28	user1@test.pl	false	true	0	BLOCK
29	test1	false	true	14	BLOCK
30	test2	false	true	15	BLOCK
33	test3	false	true	16	BLOCK
34	test4	false	true	17	BLOCK
35	test5	false	true	18	BLOCK
36	test6	false	true	19	BLOCK

Rows per page: 10 1-9 of 9 < >

Rysunek 26 Panel administratorski: blokowanie użytkownika

5.14. Administrator odblokowuje użytkownika

W celu odblokowania użytkownika administrator musi użyć przycisku z ostatniej kolumny na panelu administratorskim. Przycisk zaznaczony jest na rysunku 27.



User	Email	Admin?	Active?	Restaurant Id	Block User
13	admin	true	true	0	BLOCK
26	user@test.pl	false	true	0	BLOCK
28	user1@test.pl	false	false	0	UNBLOCK
29	test1	false	true	14	BLOCK
30	test2	false	true	15	BLOCK
33	test3	false	true	16	BLOCK
34	test4	false	true	17	BLOCK
35	test5	false	true	18	BLOCK
36	test6	false	true	19	BLOCK

Rows per page: 10 1-9 of 9 < >

Rysunek 27 Panel administratorski: odblokowywanie użytkownika

5.15. Hasła w bazie danych są przechowywane w formie niejawnej

W bazie danych przechowywane są wyniki funkcji skrótu SHA-256. Rysunek 28 przedstawia część danych w bazie, pokazując sposób przechowywania informacji o hasłach.

	id	email	password
▶	13	admin	8c6976e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a81f6f2ab448a918
	26	user@test.pl	9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08
	28	user1@test.pl	9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08
	29	test1	9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08
	30	test2	9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08
	33	test3	9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08
	34	test4	9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08
	35	test5	9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08
	36	test6	9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08

Rysunek 28 Sposób przechowywania haseł w bazie danych

5.16. Baza danych jest zabezpieczona hasłem

W celu połączenia się z bazą danych aplikacja serwerowa korzysta z następującego ciągu połączenia (ang. Connection String), który przechowywane jest przez aplikację w pliku konfiguracyjnym i może być w każdej chwili zmieniony. Przykład przedstawiony jest na rysunku 29.

```
"ConnectionStrings": {
  "DefaultConnection": "server=localhost;port=3306;database=bachelor;user=root;password=admin"
```

Rysunek 29 Ciąg połączenia do bazy danych (ang. Connection string) wykorzystywany przez aplikację

Na środowisku implementacyjnym zostanie zastosowane silniejsze hasło, składające się z co najmniej 12 znaków, zawierające znaki specjalne, również takie spoza klawiatury oraz nie składające się ze słów. Ma to zabezpieczyć hasło przed złamaniem metodą siłową (ang. brute force) oraz metodą z wykorzystaniem tęczowych tablic (ang. rainbow tables).

5.17. Połączenia poza siecią lokalną są szyfrowane

Po wejściu na adres aplikacji zaprezentowana jest informacja o niezaufanym certyfikacie. Oznacza, to że połączenie jest szyfrowane, ale certyfikat nie pochodzi z zaufanego źródła. Przykład takiego komunikatu przedstawia rysunek 30. Przyczyną tego jest fakt, że został on wygenerowany automatycznie podczas uruchomienia w środowisku lokalnym. W środowisku implementacyjnym zostanie zastosowany komplet klucz i certyfikat udostępniony przez zaufaną organizację. Wymaga to jednak podania publicznego adresu aplikacji, co jest niemożliwe w momencie testowania na środowisku lokalnym. [16]



Połączenie nie jest poufne

Ten serwer nie mógł udowodnić, że należy do **localhost**. Jego certyfikat bezpieczeństwa nie jest zaufany w systemie operacyjnym tego komputera. Może to być spowodowane błędną konfiguracją lub przechwyceniem połączenia przez atakującego.

NET::ERR_CERT_AUTHORITY_INVALID

[Wróć do bezpiecznej witryny](#)

[> Więcej informacji](#)

Rysunek 30 Prośba o zaufanie certyfikatowi wygenerowanemu podczas uruchamiania aplikacji na środowisku lokalnym (ang. Self-Signed Certificate)

5.18. Aplikacja tworzy logi z wykonywanych operacji.

Aplikacja tworzy logi w następującym formacie:

[INFO]/[ERROR] <Data i godzina> <Informacja o wykonanej akcji>

Zostają one zapisane w pliku logs.log w folderze, z którego uruchamiana jest aplikacja.

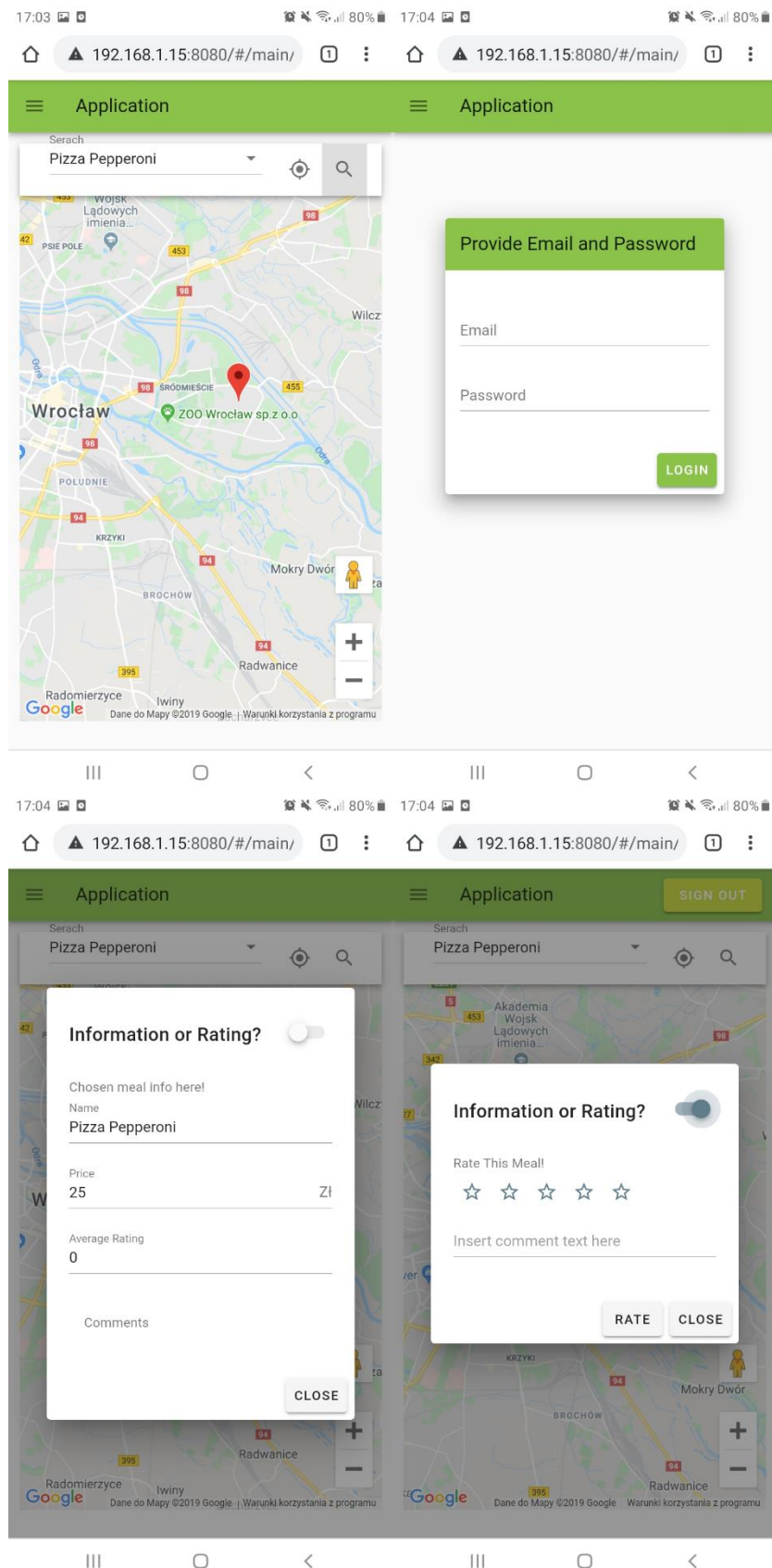
Przykładowe logi przedstawione są na rysunku 31.

```
[INFO] 22-11-2019 18:31:56 Message: Establishing connection to dabatase
[INFO] 22-11-2019 18:31:56 Message: Getting restaurant data from database
[INFO] 22-11-2019 18:31:56 Message: Establishing connection to dabatase
[INFO] 22-11-2019 18:31:56 Message: Getting restaurant data from database
[INFO] 22-11-2019 18:31:56 Message: Establishing connection to dabatase
[INFO] 22-11-2019 18:31:56 Message: Getting restaurant data from database
[INFO] 22-11-2019 18:31:56 Message: Establishing connection to dabatase
[INFO] 22-11-2019 18:31:56 Message: Getting restaurant data from database
[INFO] 22-11-2019 18:31:56 Message: Establishing connection to dabatase
[INFO] 22-11-2019 18:31:56 Message: Getting restaurant data from database
[INFO] 22-11-2019 18:31:56 Message: Establishing connection to dabatase
[INFO] 22-11-2019 18:31:56 Message: Getting restaurant data from database
[INFO] 22-11-2019 18:31:56 Message: Establishing connection to dabatase
[INFO] 22-11-2019 18:31:56 Message: Getting restaurant data from database
[INFO] 22-11-2019 18:31:56 Message: Establishing connection to dabatase
[INFO] 22-11-2019 18:31:56 Message: Getting restaurant data from database
```

Rysunek 31 Przykład logów tworzonych przez aplikację

5.19. Aplikacja jest kompatybilna z urządzeniami mobilnymi

Aplikacja mobilna dostępna jest pod tym samym adresem co aplikacja na komputery. Odpowiednio się dostosowuje do mniejszego ekranu, zapewnia wsparcie dla dotyku oraz pozwala na korzystanie z tych samych funkcjonalności. Widok aplikacji z urządzenia mobilnego przedstawiony jest na rysunku 32.



Rysunek 32 Widok aplikacji mobilnej

6. Podsumowanie

Uważam, że cel pracy, czyli opracowanie projektu i zaimplementowanie aplikacji internetowej i mobilnej, która katalogując dane o daniach w restauracjach, pozwoli odnaleźć jej użytkownikom restaurację, w której zjedzą konkretne danie został zrealizowany. Dokonałem analizy dostępnych frameworków, które mogłyby posłużyć do wykonania takiej aplikacji, a następnie wybrałem, te które uznałem za najlepsze, w tym przypadku. Następnie przeanalizowałem interesariuszy aplikacji, na podstawie których stworzyłem 3 typy użytkowników, którzy mogą korzystać z aplikacji. Dzięki temu mogłem przygotować diagram przypadku użycia, który z kolei posłużył do stworzenia listy wymagań funkcjonalnych i нефункциональных wobec aplikacji. Przygotowałem również schematy ideowe, a następnie udokumentowałem w jaki sposób zostały zrealizowane wymagania w aplikacji. Tak jak widać w rozdziale 6 przygotowana aplikacja spełnia każde z wymagań z rozdziału 3.5 oraz 3.6.

Możliwym rozwojem aplikacji byłoby dodanie modułu, który uzupełniałby bazę restauracji korzystając z już udostępnionych informacji, na przykład w Google Maps. Dzięki temu użytkownicy szukający restauracji mogliby wybierać nie tylko z tych zarejestrowanych manualnie, ale także z tych umieszczonych automatycznie w bazie. Dzięki temu sytuacja, w której użytkownik zrazi się do portalu przez brak szukanego dania w bazie nie powinna mieć miejsca. Moduł mógłby działać w oparciu o sztuczną inteligencję i pobierać informację również z innych źródeł, chociażby ze zdjęć menu umieszczonych na portalach społecznościowych, a następnie konwertować pozyskane dane do wspólnego formatu i umieszczać je w bazie danych.

7. Bibliografia

[1]. „Most Popular Technologies”

<https://insights.stackoverflow.com/survey/2019#technology> (dostęp: 02.12.2019)

[2]. Shilpa Jain. „Ultimate DEATH Match: SPA Vs. MPA”

<https://medium.com/@jainshilpa1993/ultimate-death-match-spa-vs-mpa-82e0b79ae6b6> (dostęp: 02.12.2019)

- [3]. Afonso Barros. „Angular / React / Vue pros and cons. A question most developers will ask themselves in 2018.”
<https://medium.com/@afonsobarros/angular-react-vue-pros-and-cons-75e161311e86> (dostęp: 02.12.2019)
- [4]. „Representational state transfer”
https://en.wikipedia.org/wiki/Representational_state_transfer#Architectural_constraints (dostęp: 02.12.2019)
- [5]. „Express. Fast, unopinionated, minimalist web framework for Node.js”
<https://expressjs.com> (dostęp: 02.12.2019)
- [6]. Rolando Santamaria Maso. „NO, you most probably don't need Express in your Node.js REST API”
<https://medium.com/sharenowtech/there-are-expressjs-alternatives-590d14c58c1c> (dostęp: 02.12.2019)
- [7]. „Why Django?”
<https://www.djangoproject.com/start/overview/> (dostęp: 02.12.2019)
- [8]. „Spring Framework”
<https://spring.io/projects/spring-framework> (dostęp: 02.12.2019)
- [9]. „Web Framework Benchmarks. Round 18. 2019-07-09”
<https://www.techempower.com/benchmarks/#section=data-r18&hw=ph&test=plaintext> (dostęp: 02.12.2019)
- [10]. „What is ASP.NET Core?”
<https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet-core> (dostęp: 02.12.2019)
- [11]. Mark Smallcombe. „SQL vs. NoSQL: How Are They Different and What Are the Best SQL and NoSQL Database Systems?”
<https://www.xplenty.com/blog/the-sql-vs-nosql-difference/> (dostęp: 02.12.2019)
- [12]. „Appropriate Uses For SQLite”

<https://www.sqlite.org/whentouse.html> (dostęp: 02.12.2019)

- [13]. James & Suzanne Robertson. „Volere. Wzorzec specyfikacji wymagań. Wersja 14 Styczeń 2009”

<https://docplayer.pl/935258-Volere-wzorzec-specyfikacji-wymagan-wersja-14-styczen-2009.html> (dostęp: 02.12.2019)

- [14]. „What is a Reverse Proxy vs. Load Balancer?”

<https://www.nginx.com/resources/glossary/reverse-proxy-vs-load-balancer/>
(dostęp: 02.12.2019)

- [15]. Steve Smith, Scott Addie, and Luke Latham. „Dependency injection in ASP.NET Core”

<https://docs.microsoft.com/pl-pl/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-3.0> (dostęp: 02.12.2019)

- [16]. „SSL For Free. Free SSL Certificates & Free Wildcard SSL Certificates in Minutes”

<https://www.sslforfree.com> (dostęp: 02.12.2019)

Spis tabel i rysunków

Tabela 1 Informacje o typie użytkownika – klient branży gastronomicznej.....	13
Tabela 2 Informacje o typie użytkownika – osoba zarządzająca restauracją	13
Tabela 3 Informacje o typie użytkownika – administrator aplikacji.....	14
Tabela 4 Karta wymagań funkcjonalnych: Utworzenie konta przez użytkownika	18
Tabela 5 Karta wymagań funkcjonalnych: Użytkownika loguje się do systemu	18
Tabela 6 Karta wymagań funkcjonalnych: Wyświetlanie mapy użytkownikowi	18
Tabela 7 Karta wymagań funkcjonalnych: Wyszukiwanie restauracji w systemie, która ma w swojej karcie podane przez użytkownika danie	19
Tabela 8 Karta wymagań funkcjonalnych: Zaprezentowanie użytkownikowi komentarzy i ocen innych użytkowników	19
Tabela 9 Karta wymagań funkcjonalnych: Dodanie oceny i słownego komentarza do wybranego dania przez użytkownika	19
Tabela 10 Karta wymagań funkcjonalnych: Rejestracja restauracji przez użytkownika	20

Tabela 11 Karta wymagań funkcjonalnych: Użytkownik dodaje nowe danie do menu swojej restauracji.....	20
Tabela 12 Karta wymagań funkcjonalnych: Użytkownik zmienia nazwę dania w menu swojej restauracji.....	20
Tabela 13 Karta wymagań funkcjonalnych: Użytkownik zmienia cenę dania w menu swojej restauracji.....	21
Tabela 14 Karta wymagań funkcjonalnych: Użytkownik usuwa danie z menu swojej restauracji.....	21
Tabela 15 Karta wymagań funkcjonalnych: Użytkownik ma wgląd w informacje o ocenach dań w swojej restauracji.	21
Tabela 16 Karta wymagań funkcjonalnych: Administrator blokuje użytkownika.	22
Tabela 17 Karta wymagań funkcjonalnych: Administrator odblokowuje użytkownika.	22
Tabela 18 Karta wymagań niefunkcjonalnych: Hasła w bazie danych są przechowywane w formie niejawnej.....	23
Tabela 19 Karta wymagań niefunkcjonalnych: Baza danych jest zabezpieczona hasłem	23
Tabela 20 Karta wymagań niefunkcjonalnych: Połączenia poza siecią lokalną są szyfrowane	23
Tabela 21 Karta wymagań niefunkcjonalnych: Aplikacja tworzy logi z wykonywanych operacji.	24
Tabela 22 Karta wymagań niefunkcjonalnych: Aplikacja jest kompatybilna z urządzeniami mobilnymi	24
 Rysunek 1 Schemat środowiska implementacyjnego	16
Rysunek 2 Diagram przypadków użycia	17
Rysunek 3 Schemat bazy danych.....	25
Rysunek 4 Schemat ideowy interfejsu użytkownika.....	27
Rysunek 5 Schemat ideowy klas w aplikacji serwerowej	29
Rysunek 6 Przykład klasy - modelu danych	30
Rysunek 7 Główny widok aplikacji	31
Rysunek 8 Widok formularza rejestracji użytkownika	32
Rysunek 9 Widok formularza logowania się użytkownika	32
Rysunek 10 Widok mapy z wyszukiwarką.....	33
Rysunek 11 Prośba o zgodę na wykorzystanie lokalizacji	33

Rysunek 12 Wyszukiwarka restauracji	34
Rysunek 13 Wyszukiwarka restauracji: lista odpowiedzi	34
Rysunek 14 Wyszukiwanie restauracji: Przedstawienie wyników wyszukiwania	35
Rysunek 15 Widok informacji o daniu.....	36
Rysunek 16 Przycisk przekierowujący na widok dodawania komentarza	36
Rysunek 17 Widok dodawania komentarza	37
Rysunek 18 Formularz rejestracji restauracji	37
Rysunek 19 Widok edycji menu	38
Rysunek 20 Widok edycji menu: dodawanie nowego dania	38
Rysunek 21 Widok edycji menu: przycisk wywołujący okno edycji dania	38
Rysunek 22 Widok edycji menu: okno edycji dania.....	39
Rysunek 23 Widok edycji menu: Usuwanie dania z listy	39
Rysunek 24 Widok edycji menu: komunikat o poprawnym zapisaniu danych w bazie .	40
Rysunek 25 Widok edycji menu: przedstawienie ocen właścicielowi restauracji.....	40
Rysunek 26 Panel administratorski: blokowanie użytkownika	41
Rysunek 27 Panel administratorski: odblokowywanie użytkownika.....	41
Rysunek 28 Sposób przechowywania haseł w bazie danych	42
Rysunek 29 Ciąg połączenia do bazy danych (ang. Connection string) wykorzystywany przez aplikację.....	42
Rysunek 30 Prośba o zaufanie certyfikatowi wygenerowanemu podczas uruchamiania aplikacji na środowisku lokalnym (ang. Self-Signed Certificate)	43
Rysunek 31 Przykład logów tworzonych przez aplikację	43
Rysunek 32 Widok aplikacji mobilnej	44