POLITECHNIKA WROCŁAWSKA

WYDZIAŁ ELEKTRONIKI

1. KIERUNEK: Teleinformatyka

SPECJALNOŚĆ: Projektowanie Sieci Teleinformatycznych

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

AUTOR:

Imię i nazwisko

PROWADZĄCY PRACĘ:

1. dr. inż. Wojciech Kmiecik, K2

OCENA PRACY:

WROCŁAW …….

Spis treści

[KIERUNEK: Teleinformatyka 1](#_Toc24541183)

[PRACA DYPLOMOWA 1](#_Toc24541184)

[INŻYNIERSKA 1](#_Toc24541185)

[dr. inż. Wojciech Kmiecik, K2 1](#_Toc24541186)

[WROCŁAW ……. 1](#_Toc24541187)

[1. Wstęp 3](#_Toc24541188)

[2. Przegląd i analiza istniejących rozwiązań 4](#_Toc24541189)

[1. Rozwiązania Front-endowe 4](#_Toc24541190)

[2. Rozwiązania Back-endowe 6](#_Toc24541191)

[3. Rozwiązania bazo-danowe 8](#_Toc24541192)

[3. Analiza potrzeb użytkowników 10](#_Toc24541193)

[1. Interesariusze 10](#_Toc24541194)

[2. Bezpośredni użytkownicy aplikacji 11](#_Toc24541195)

[3. Przewidziane środowisko implementacyjne produktu 13](#_Toc24541196)

[4. Przypadki użycia 14](#_Toc24541197)

[5. Wymagania funkcjonalne 14](#_Toc24541198)

[6. Wymagania niefunkcjonalne 19](#_Toc24541199)

[4. Projekt aplikacji 22](#_Toc24541200)

[5. Podsumowanie 22](#_Toc24541201)

[6. Bibliografia 22](#_Toc24541202)

# 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 oferuję. 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.

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

Moja praca będzie składać się z sześciu rozdziałów. W pierwszym rozdziale opisałem motywację do stworzenia pracy. W drugim rozdziale porównałem dostępne technologie pozwalające na implementacje tego typu aplikacji. Trzeci rozdział jest poświęcony analizie potrzeb użytkowników, co pozwoliło stworzyć listę wymagań funkcjonalnych oraz niefunkcjonalnych. Rozdział 4 zawiera projekt aplikacji. Budowę modułów, klas, komponentów, tabel w bazie danych oraz relacje między nimi, a także wykorzystane algorytmy. Rozdział 5 zawiera podsumowanie, które kończy pracę. Rozdział 6 zawiera bibliografię.

# Przegląd i analiza istniejących rozwiązań

Opierając się na podstawowych wymaganiach i trendach w projektowaniu aplikacji webowych powinna ona się składać 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:

## 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 Appliaction), 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 aplikacji.

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 prze 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.

## Rozwiązania Back-endowe

Najpopularniejszym typem aplikacji webowej pośredniczącej między interfesjem 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 wskazuję 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. [6]

Django

Django jest frameworkiem do tworzena 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 służące 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. Dependecy 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). [8]

ASP.NET Core

Jest to framework do tworzenia aplikacji webowych stworzony przez firmę Microsoft. Do działania wykorzystuję 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 [9]. 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.

## Rozwiązania bazo-danowe

W tym przypadku należy zacząć od podjęcia decyzji, czy baza danych ma być relacyjne, 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. [11]

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. [11]

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. [11]

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. [11]

SQLLite

SQLLite jest bardzo ciekawym, multiplatformowym rozwiązaniem implementacji relacyjnej bazy danych opartej na języku SQL. Czynnikiem odróżniających SQLLite 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. SQLLite 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 SQLLite 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. [11]

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.

# 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 definiuję 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]

## 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”.

## Bezpośredni użytkownicy aplikacji

W projektowanej aplikacji można wyróżnić następujące typy użytkowników, którzy będą z nią mieli bezpośredni kontakt:

|  |  |
| --- | --- |
| 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. |

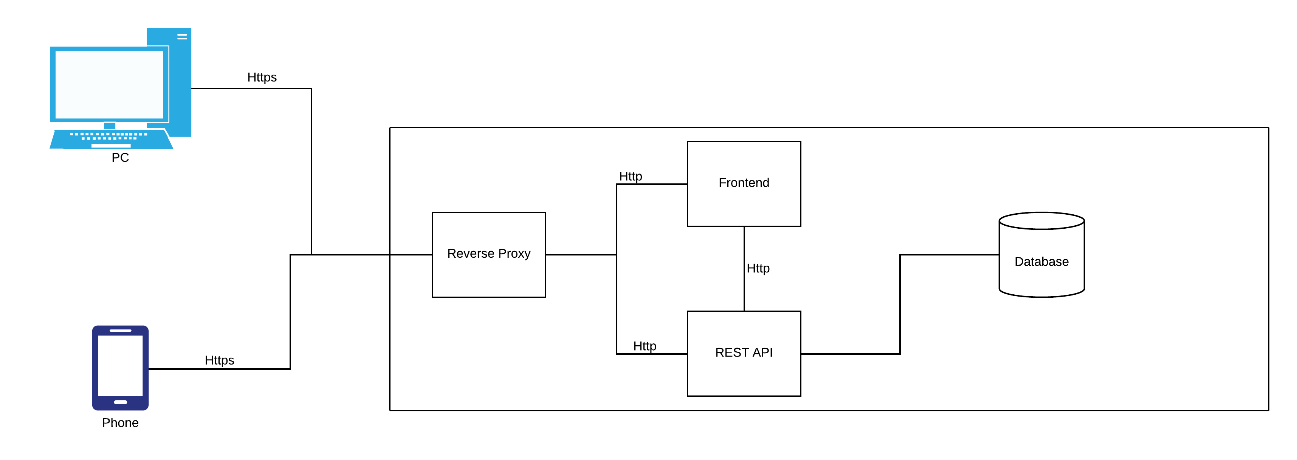
|  |  |
| --- | --- |
| Nazwa użytkownika | Osoba zarządzająca restauracją |
| Rola użytkownika | Tworzenie i edycja menu restauracji, przeglądanie ocen klientów |
| Znajomość dziedzinowa przedmiotu sprawy | Ekspert |
| Doświadczenie w sferze technologii | Nowicjusz |
| Inne cechy charakteryzujące użytkownika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

|  |  |
| --- | --- |
| 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. |

## Przewidziane środowisko implementacyjne produktu

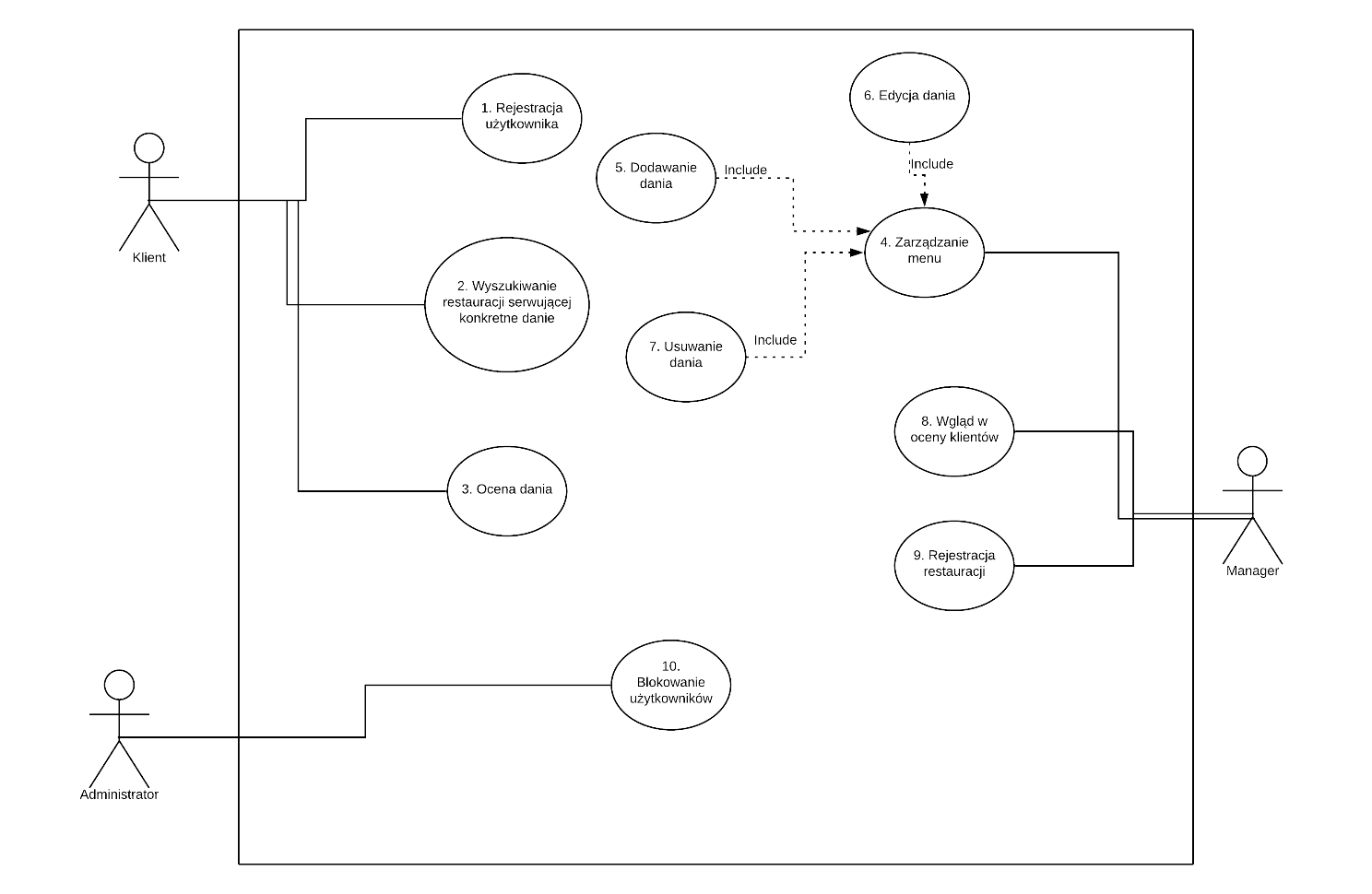
Poniższy diagram przedstawia środowisko implementacyjne systemu. Oprócz wcześniej wspomnianych elementów, czyli aplikacji front-endowej, REST API oraz bazy danych występuje tutaj aplikacja spełniająca zadanie reverse proxy. Jest to aplikacja pośrednicząca w wymianie zapytań pomiędzy klientem, a resztą 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 pochądzącego 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 architekutry 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 kompresje gzip zmniejsza ilość przesyłanych danych
* „SSL Termination” – aplikacja stanowiąc styk z siecią zewnętrzną pozwala na niestosowanie szyfrowanego połączanie 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 przechowywanie 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 wyślę reverse proxy, tym samym odciążając pozostałe aplikacje z wykonywania tych samych operacji. [14]



## Przypadki użycia

Poniższy diagram prezentuje 3 typy użytkowników jako aktorzy, a także przypisane do nich przypadki użycia. Definiuję on kto w aplikacji może wykonywać określone akcje.



## Wymagania funkcjonalne

Wymagania wobec aplikacji dzielą się na dwa typy: funkcjonalne i niefunkcjonalne. 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:

|  |  |  |
| --- | --- | --- |
| **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 | | |
| **Priorytet:** Wysoki | **Konflikty:** | |

|  |  |  |
| --- | --- | --- |
| **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 | | |
| **Priorytet:** Wysoki | **Konflikty:** | |

|  |  |  |
| --- | --- | --- |
| **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 | | |
| **Priorytet**: Wysoki | **Konflikty**: | |

|  |  |  |
| --- | --- | --- |
| **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 | | |
| **Priorytet**: Wysoki | **Konflikty**: | |

|  |  |  |
| --- | --- | --- |
| **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 | | |
| **Priorytet**: Średni | **Konflikty**: | |

|  |  |  |
| --- | --- | --- |
| **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 | | |
| **Priorytet**: Średni | **Konflikty**: | |

|  |  |  |
| --- | --- | --- |
| **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. | | |
| **Priorytet**: Wysoki | **Konflikty**: | |

|  |  |  |
| --- | --- | --- |
| **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. | | |
| **Priorytet**: Wysoki | **Konflikty**: | |

|  |  |  |
| --- | --- | --- |
| **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. | | |
| **Priorytet**: Niski | **Konflikty**: | |

|  |  |  |
| --- | --- | --- |
| **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. | | |
| **Priorytet**: Niski | **Konflikty**: | |

|  |  |  |
| --- | --- | --- |
| **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. | | |
| **Priorytet**: Wysoki | **Konflikty**: | |

|  |  |  |
| --- | --- | --- |
| **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. | | |
| **Priorytet**: Średni | **Konflikty**: | |

|  |  |  |
| --- | --- | --- |
| **Nr wymagania**: 13 | **Typ wymagania**:  Funkcjonalne | **Nr. przypadku użycia**: 8 |
| **Opis**: Użytkownik ma wgląd w komentarze tekstowe na temat dań w swojej restauracji | | |
| **Przesłanka**: W celach analizy biznesowej użytkownik powinien mieć możliwość przeglądania informacji na temat komentarzy klientów. | | |
| **Kryterium spełnienia**: Użytkownik widzi informacje na temat komentarzy konkretnych dań. | | |
| **Priorytet**: Średni | **Konflikty**: | |

|  |  |  |
| --- | --- | --- |
| **Nr wymagania**: 14 | **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. | | |
| **Priorytet**: Wysoki | **Konflikty**: | |

|  |  |  |
| --- | --- | --- |
| **Nr wymagania**: 15 | **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 | | |
| **Priorytet**: Wysoki | **Konflikty**: | |

## Wymagania niefunkcjonalne

Drugim typem wymagań wobec projektowanej aplikacji są wymagania niefunkcjonalne. 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:

|  |  |  |
| --- | --- | --- |
| **Nr wymagania**: 16 | **Typ wymagania**:  Niefunkcjonalne | **Nr. przypadku użycia**: brak |
| **Opis**: Aplikacja kliencka uruchamia się na większości dostępnych przeglądarek | | |
| **Przesłanka**: Aplikacja powinna docierać do jak największej liczby użytkowników | | |
| **Kryterium spełnienia**: Aplikacja uruchamia się na trzech najczęściej wybieranych przeglądarkach (Chrome, Firefox, Microsoft Edge) | | |
| **Priorytet**: Wysoki | **Konflikty**: | |

|  |  |  |
| --- | --- | --- |
| **Nr wymagania**: 17 | **Typ wymagania**:  Niefunkcjonalne | **Nr. przypadku użycia**: brak |
| **Opis**: Aplikacja serwerowa nie wymaga więcej niż xxx pamięci RAM oraz yyy przestrzeni dyskowej | | |
| **Przesłanka**: Aplikacja nie powinna zabierać zbyt dużo zasobów maszyny, na której jest zainstalowana, aby zredukować koszty | | |
| **Kryterium spełnienia**: Aplikacja nie przekracza xxx pamięci RAM oraz yyy przestrzeni dyskowej podczas normalnej pracy | | |
| **Priorytet**: Średni | **Konflikty**: | |

|  |  |  |
| --- | --- | --- |
| **Nr wymagania**: 18 | **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 | | |
| **Priorytet**: Wysoki | **Konflikty**: | |

|  |  |  |
| --- | --- | --- |
| **Nr wymagania**: 19 | **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 | | |
| **Priorytet**: Średni | **Konflikty**: | |

|  |  |  |
| --- | --- | --- |
| **Nr wymagania**: 20 | **Typ wymagania**:  Niefunkcjonalne | **Nr. przypadku użycia**: brak |
| **Opis**: Połączenia poza siecią lokalną są szyfrowane | | |
| **Przesłanka**: Aplikacja kliencka będzie wymieniać z API dane, które są narażone na sniffing | | |
| **Kryterium spełnienia**: Połączenia poza siecią lokalną są zabezpieczone protokołem HTTPS | | |
| **Priorytet**: Średni | **Konflikty**: | |

|  |  |  |
| --- | --- | --- |
| **Nr wymagania**: 21 | **Typ wymagania**:  Niefunkcjonalne | **Nr. przypadku użycia**: brak |
| **Opis**: Aplikacja powinna być responsywna na zadowalającym poziomie. Reakcja na polecenia szybsza niż 2 s. | | |
| **Przesłanka**: Wolno działająca aplikacja może wpływać na doświadczenie użytkownika | | |
| **Kryterium spełnienia**: Podczas normalnej pracy aplikacja jest responsywna na zadowalającym poziomie. | | |
| **Priorytet**: Średni | **Konflikty**: | |

|  |  |  |
| --- | --- | --- |
| **Nr wymagania**: 22 | **Typ wymagania**:  Niefunkcjonalne | **Nr. przypadku użycia**: brak |
| **Opis**: Aplikacja powinna być intuicyjna w obsłudze. Zwykły użytkownik może ją obsługiwać bez instrukcji, a manager restauracji i administrator bez instrukcji lub krótkim (<15 min) szkoleniu. | | |
| **Przesłanka**: Aplikacja powinno się dać obsługiwać bez instrukcji. Wpłynie to pozytywnie na doświadczenie użytkownika, a także ograniczy prace nad dokumentacją. | | |
| **Kryterium spełnienia**: Użytkownik nie ma wątpliwości podczas korzystania z aplikacji | | |
| **Priorytet**: Wysoki | **Konflikty**: | |

|  |  |  |
| --- | --- | --- |
| **Nr wymagania**: 23 | **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. | | |
| **Priorytet**: Średni | **Konflikty**: | |

# Projekt aplikacji

# Podsumowanie

# Bibliografia

1. <https://insights.stackoverflow.com/survey/2019#technology> dostęp: 27.10.2019
2. https://medium.com/@jainshilpa1993/ultimate-death-match-spa-vs-mpa-82e0b79ae6b6
3. <https://medium.com/@afonsobarros/angular-react-vue-pros-and-cons-75e161311e86>
4. https://en.wikipedia.org/wiki/Representational\_state\_transfer#Architectural\_constraints
5. https://expressjs.com
6. <https://medium.com/sharenowtech/there-are-expressjs-alternatives-590d14c58c1c>
7. <https://www.djangoproject.com/start/overview/>
8. <https://spring.io/projects/spring-framework>
9. https://www.techempower.com/benchmarks/#section=data-r18&hw=ph&test=plaintext
10. https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet-core
11. <https://www.xplenty.com/blog/the-sql-vs-nosql-difference/>

1. <https://www.sqlite.org/whentouse.html>

1. <https://docplayer.pl/935258-Volere-wzorzec-specyfikacji-wymagan-wersja-14-styczen-2009.html>
2. https://www.nginx.com/resources/glossary/reverse-proxy-vs-load-balancer/