Przetwarzanie danych w chmurach obliczeniowych

Dokumentacja projektu

System rekomendacji restauracji

Patryk Śledź

303806

Informatyka Stosowana

Wydział Fizyki i Informatyki Stosowanej

Akademia Górniczo-Hutnicza w Krakowie

Spis treści

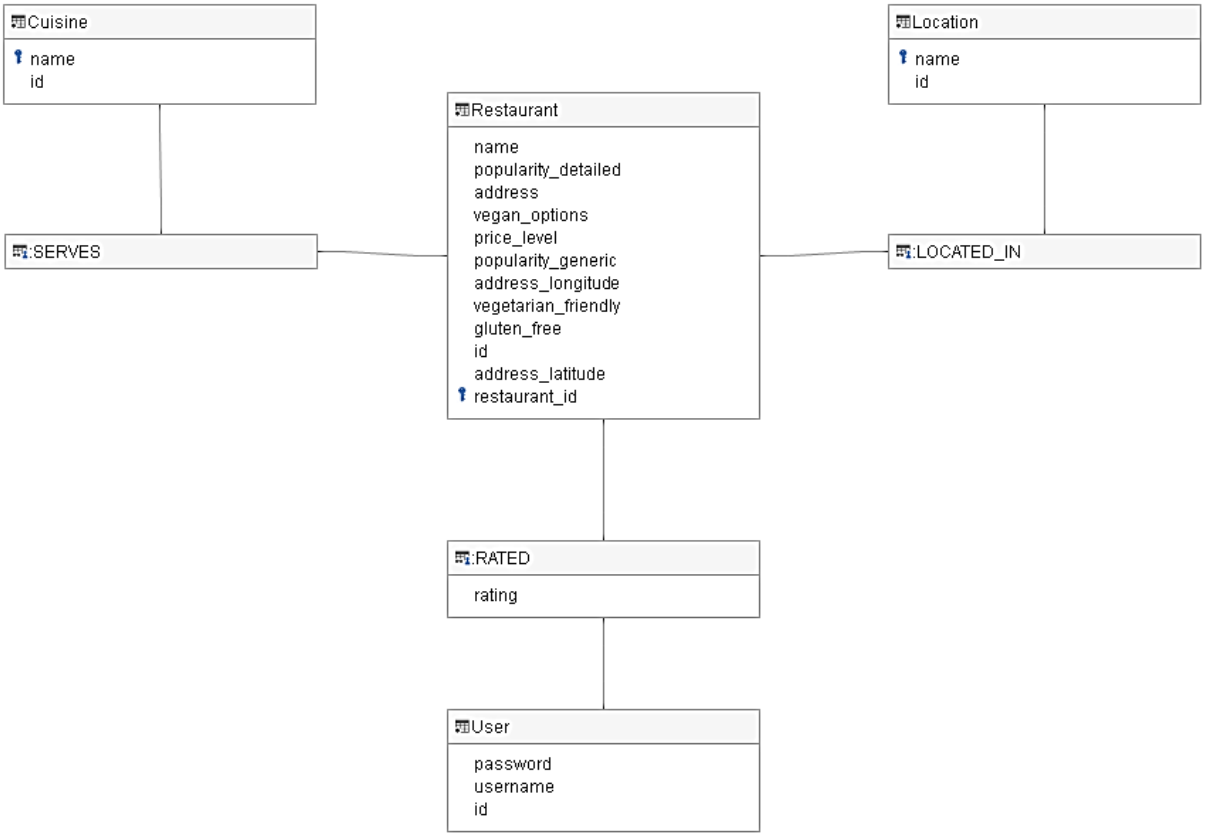
1. Opis problemu.

Celem projektu było przygotowanie projektu grafowej bazy danych wraz z interfejsem dostępu.

Wybranym tematem jest system rekomendacji restauracji wraz z implementacją aplikacji dostępu w postaci REST API oraz interfejsem użytkownika typu SPA.

2. Projekt bazy danych.

W projekcie wykorzystano bazę Neo4j wraz z usługą Neo4j AuraDB. Diagram UML przygotowanego schematu bazy jest widoczny na rys. 1..



Rysunek 1 Diagram UML modelu bazy danych

Zdefiniowane typy danych

Zdefiniowano 4 etykiety wierzchołków:

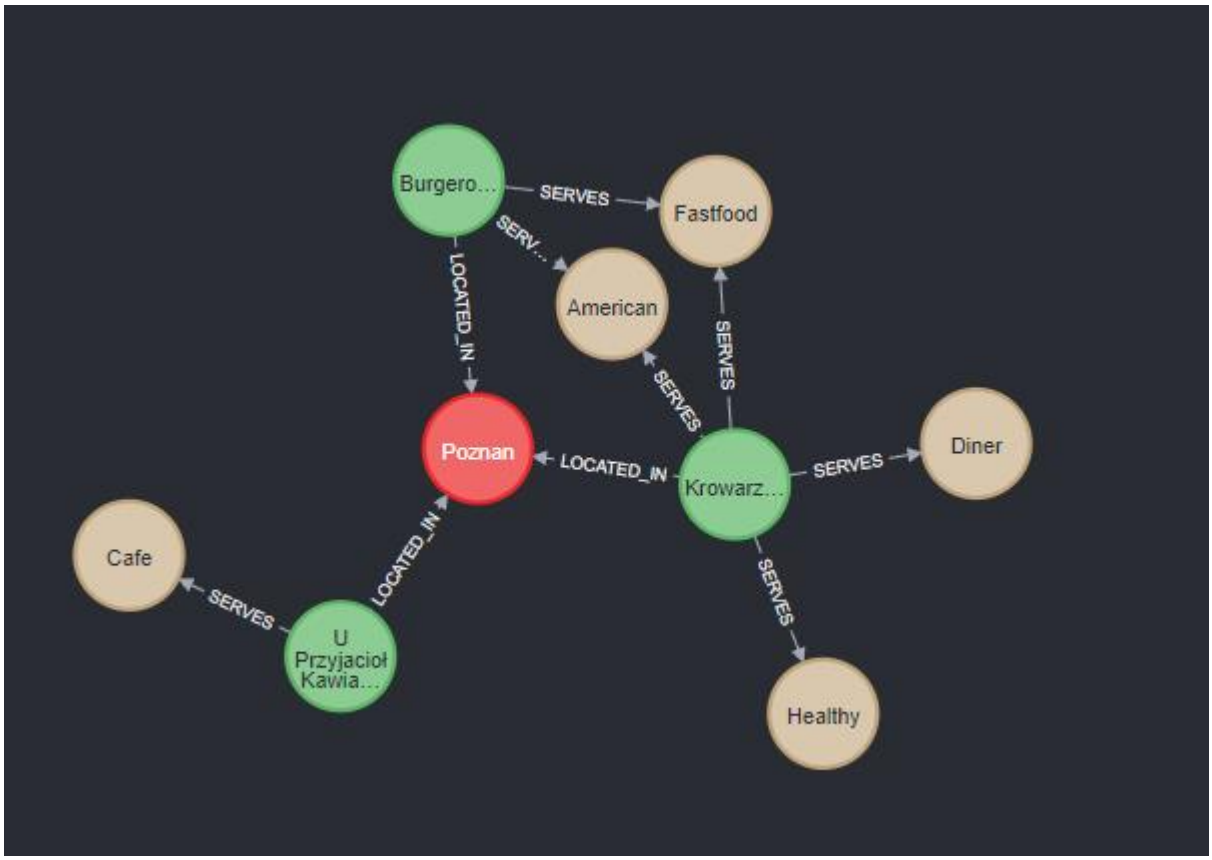
- **Restaurant** – reprezentuje restauracje,
- **User** – reprezentuje użytkowników,
- **Cuisine** – reprezentuje typy kuchni,
- **Location** – reprezentuje lokacje restauracji.

Zdefiniowano 3 relacje między wierzchołkami:

- **SERVES** – krawędź skierowana łączy wierzchołki Restaurant oraz Cuisine,
- **LOCATED_IN** – krawędź skierowana łączy wierzchołki Restaurant oraz Location,
- **RATED** – krawędź skierowana łączy wierzchołki User oraz Restaurant.

Pola zawarte w wierzchołkach oraz relacjach:

- **Restaurant**
  - id – domyślny identyfikator restauracji,
  - restaurant_id – unikalny identyfikator restauracji, wykorzystywany podczas importu danych z pliku CSV,
  - name – nazwa restauracji,
  - address – szczegółowy adres restauracji,
  - address_longitude – długość geograficzna,
  - address_latitude – szerokość geograficzna,
  - price_level – rozpiętość cen w restauracji,
  - popularity_generic – informacja na temat restauracji,
  - popularity_detailed – informacja szczegółowa na temat restauracji,
  - vegan_options – czy restauracja posiada opcje wegańskie w menu,
  - vegetarian_friendly – czy restauracja posiada opcje wegetariańskie w menu,
  - gluten_free – czy restauracja posiada opcje bez glutenu w menu,
- **User**
  - id – domyślny identyfikator użytkownika,
  - username – nazwa użytkownika,
  - password – hasło użytkownika,
- **Cuisine**
  - id – domyślny identyfikator rodzaju kuchni,
  - name – unikalna nazwa rodzaju kuchni,
- **Location**
  - id – domyślny identyfikator lokacji,
  - name – unikalna nazwa miasta,
- **:RATED**
  - rating – ocena restauracji.



Rysunek 2 Przykład relacji restauracji w Poznaniu

Na rys. 2. widoczny jest diagram przedstawiający uzyskane relacje dla 3 restauracji w Poznaniu.

Skrypty umożliwiające stworzenie i import potrzebnych danych zostały umieszczone w katalogu **cypher_scripts**. Dane potrzebne do stworzenia wierzchołków i relacji są dostępne w folderze **database_data**.

Wybrane polecenia tworzące wierzchołki i relacje

```
LOAD CSV WITH HEADERS FROM
'https://github.com/patryk0504/restaurants_recomendations_neo4j/raw/main
/city.csv'
AS row FIELDTERMINATOR ';'
CREATE (:Location {name : row.city})
```

Listing 1 Polecenie tworzące wierzchołki wraz z importem danych CSV

```
CREATE CONSTRAINT UniqueLocationNameConstraint ON (l:Location) ASSERT
l.name IS UNIQUE;
```

Listing 2 Polecenie tworzące ograniczenie na danym parametrze wierzchołka

```
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/patryk0504/restaurants_recomendations
_neo4j/main/restauracje_polska_cities_city.csv?token=AP2Q6BK6K57UKSGNTFR
Y4Q3BV2HKU' as row FIELDTERMINATOR ';'
MATCH (r:Restaurant {restaurant_id : row.restaurant_link})
MATCH (l:Location {name : row.city})
WITH r,l
MERGE (r) -[:LOCATED_IN]->(l);
```

Listing 3 Polecenie tworzące relacje :LOCATED_IN pomiędzy restauracją a miejscowością

Na listingu 1., listingu 2. oraz listingu 3. przedstawiono przykładowe polecenia języka Cypher.

Pierwszy z nich demonstruje utworzenie wierzchołka Location. Do tego celu wykorzystywane jest polecenie wczytujące dane z pliku CSV, który jest zlokalizowany pod danym identyfikatorem URI. Jako, że plik zawiera separator będący znakiem „;” należało skorzystać z opcji FIELDTERMINATOR.

Drugi pokazuje metodę tworzenia ograniczenie na parametrze name wierzchołka Location. Ograniczenie jest potrzebne, aby przeciwdziałać możliwym powtórzeniom nazw miejscowości.

Trzeci listing zawiera definicję tworzącą relację LOCATED_IN pomiędzy restauracją a miejscowością.

3. Opis funkcjonalności udostępnionej przez API.

Aplikacja REST API, pełniąca rolę interfejsu dostępu do bazy Neo4j została napisana w języku Python z wykorzystaniem frameworka Flask.

Wykorzystane rozszerzenia i biblioteki:

- Flask-RESTful,
- Neo4j,
- Flask-Jwt-Extended,
- Flask-Bcrypt,
- Flask-Cors.

Zaimplementowane API spełnia zasady REST m.in.:

- odseparowano interfejs użytkownika od operacji na serwerze,
- aplikacja jest bezstanowa – nie przechowuje stanu sesji, do autoryzacji wykorzystywany jest JWT (JSON Web Token),
- ujednolicono interfejs – żądania do API dotyczące danego zasobu są ustalone, bez względu na źródło żądania.

Zdefiniowane endpointy API

- **[POST]** *api/auth/register* – umożliwia rejestrację użytkownika,
- **[POST]** *api/auth/login* – umożliwia logowanie użytkownika,
- **[GET]** *api/cuisine/list* – umożliwia pobranie listy rodzajów kuchni,
- **[GET]** *api/restaurant/list* – umożliwia pobranie listy restauracji,
- **[POST]** *api/restaurant****/<int:restaurant_id>/****rate* – umożliwia ocenę restauracji,
- **[GET]** *api/restaurant****/<int:restaurant_id>/****rates* – umożliwia pobranie oceny restauracji,
- **[GET]** *api/restaurant/recommendations/rating* – umożliwia pobranie rekomendacji ze względu na ocenione restauracje,
- **[GET]** *api/restaurant****/<int:restaurant_id>/****recommendations/country* – umożliwia pobranie rekomendacji restauracji podobnych do wybranej (w obrębie całego kraju),
- **[GET]** *api/restaurant****/<int:restaurant_id>/****recommendations/city* – umożliwia pobranie rekomendacji restauracji podobnych do wybranej (w obrębie danego miasta),
- **[GET]** *api/users/similar* – umożlwia pobranie informacji o użytkownikach, którzy mają zbliżone zainteresowania.

Wszystkie endpointy z wyjątkiem *api/auth/register* oraz *api/auth/login* wymagają autoryzacji tokenowej.

Kod aplikacji serwerowej został umieszczony w katalogu **api**.

Rekomendacje

W ramach tematu projektu, zaimplementowane 4 typy rekomendacji:

1. rekomendacja użytkowników o podobnych zainteresowaniach kulinarnych,
2. rekomendacja restauracji biorąc pod uwagę ocenione restauracje,
3. rekomendacje restauracji w obrębie miasta, biorąc pod uwagę wybraną restaurację,
4. rekomendacje restauracji w obrębie kraju, biorąc pod uwagę wybraną restaurację.

Punkt 3. oraz 4. zrealizowano wykorzystując tzw. Indeks Jaccarda. Mierzy on podobieństwo pomiędzy dwoma zbiorami danych i jest dany wzorem:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Indeks ten definiujemy jako iloraz mocy części wspólnej zbiorów oraz mocy sumy zbiorów. Jeżeli indeks jest równy 1 – zbiory są identyczne, a jeśli 0 – zbiory są różne i nie mają wspólnych elementów.

Punkt 1. zrealizowano wykorzystując podobieństwo cosinusowe. Zdecydowano się na takie rozwiązanie ponieważ użytkownicy są powiązani z restauracjami relacją wagową RATED. Podobnie jak współczynnik Jaccarda, gdy wartość podobieństwa jest równa 0 – brak podobieństwa, gdy 1 – pełne podobieństwo:

$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|}$$

Punkt 2. zrealizowano porównując ocenione restauracje zalogowanego użytkownika z ocenami innych użytkowników. Jeżeli inny użytkownik posiada przynajmniej jedną ocenę wspólną, a ta może się różnić o nie więcej niż 1pkt. To z pewnym prawdopodobieństwem, inne restauracje ocenione przez znalezionego użytkownika mogą być poprawnie zarekomendowane.

Wybrane polecenia rekomendacji

```
MATCH (l:Location)<-[:LOCATED_IN]-(n:Restaurant)-[:SERVES]->(c:Cuisine)
SET n.id = id(n)
RETURN n, collect(c.name) as cuisines, l.name as location
```

Listing 4 Polecenie wyszukujące wszystkie restauracje wraz z lokalizacją i rodzajem kuchni

```
MATCH (m:Restaurant)-[:SERVES|LOCATED_IN]-(t)-[:SERVES|LOCATED_IN]-(other:Restaurant)
WHERE id(m) = toInteger($restaurant_id)
WITH m, other, COUNT(t) AS intersection, COLLECT(t.name) AS i

MATCH (m)-[:SERVES|LOCATED_IN]-(mt)
WITH m, other, intersection,i, COLLECT(mt.name) AS s1

MATCH (other)-[:SERVES|LOCATED_IN]-(ot)
WITH m,other,intersection,i, s1, COLLECT(ot.name) AS s2

WITH m,other,intersection,s1,s2
WITH m,other,intersection,s1+[x IN s2 WHERE NOT x IN s1] AS union, s1, s2
RETURN m.name, other, s1,s2, round(((1.0*intersection)/SIZE(union)), 2) AS jaccard
ORDER BY jaccard DESC LIMIT 100
```

Listing 5 Polecenie zwracające rekomendację dla wybranej restauracji w całym kraju

```
MATCH (p1:User)-[x:RATED]->(r:Restaurant)<-[y:RATED]-(p2:User)
WHERE id(p1) = toInteger($user_id)
WITH COUNT(r) AS numberrated,SUM(x.rating * y.rating) AS xyDotProduct,
SQRT(REDUCE(xDot = 0.0, a IN COLLECT(x.rating) | xDot + a^2)) AS xLength,
SQRT(REDUCE(yDot = 0.0, b IN COLLECT(y.rating) | yDot + b^2)) AS yLength,
collect(r.name) as restaurants,
p1, p2 WHERE numberrated > 0
RETURN restaurants, p1.username as currentUser, p2.username as similarUser,
numberrated as numberSimilarRestaurants,
       round(xyDotProduct / (xLength * yLength), 2) AS sim
ORDER BY sim DESC
LIMIT 100;
```

Listing 6 Polecenie zwracające listę podobny użytkowników ze względu na ocenione restauracje

```
MATCH(u : User)
MATCH(r : Restaurant)
WHERE id(u) = toInteger($user_id) and id(r) = toInteger($restaurant_id)
WITH u,r
MERGE (u)-[r2:RATED]->(r)
SET r2.rating = toInteger($rate)
RETURN r
```

Listing 7 Polecenie dodające ocenę dla restauracji

```
MATCH (me:User)-[my:RATED]->(r:Restaurant)
MATCH (other:User)-[their:RATED]->(r)
WHERE id(me) = toInteger($user_id)
AND me <> other
AND abs(my.rating - their.rating) < 2
WITH other,r
MATCH (other)-[otherRating:RATED]->(r2:Restaurant)
WHERE r2 <> r
WITH avg(otherRating.rating) AS avgRating, r2, collect(other.username) as
other_users
MATCH (r2) -[:LOCATED_IN]-> (l:Location)
RETURN r2 as restaurant, avgRating, l.name as location, other_users
ORDER BY avgRating desc
LIMIT 25
```

Listing 8 Polecenie zwracające rekomendacje ze względu na dotychczas ocenione restauracje

Polecenia języka Cypher przedstawione na listingu 4. zwraca wszystkie restauracje wraz z informacjami na temat typu kuchni oraz lokalizacji. Polecenia na listingach od 5, 6 oraz 8 mają za zadanie zwrócić rekomendacje restauracji. Natomiast listing 7 zawiera sposób dodawania oceny restauracji.

Polecenie na listingu 5. wykorzystuje przytoczony w poprzednim dziale współczynnik Jaccarda. Sposób działania polecenia wygląda następująco:

- wyszukujemy wszystkie restauracje, które są powiązane w dowolnym stopniu z wybraną restauracją (zdefiniowaną przez indeks),
- zapamiętujemy znalezione restauracje wraz z ilością znalezionych cech wspólnych,
- pobieramy wszystkie dane opisujące wybraną restaurację oraz znalezione restauracje rekomendowane,
- finalnie liczymy iloraz mocy części wspólnej i sumy zbiorów. W celach wydajnościowych ograniczamy ilość zwracanych danych do 100 restauracji z największym współczynnikiem Jaccarda.

4. Aplikacja UI – frontend.

Aplikacja implementująca interfejs użytkownika jest aplikacją typu SPA. Do jej implementacji wykorzystano bibliotekę REACT.js.

Aplikacja wykorzystuje dodatkowo biblioteki:

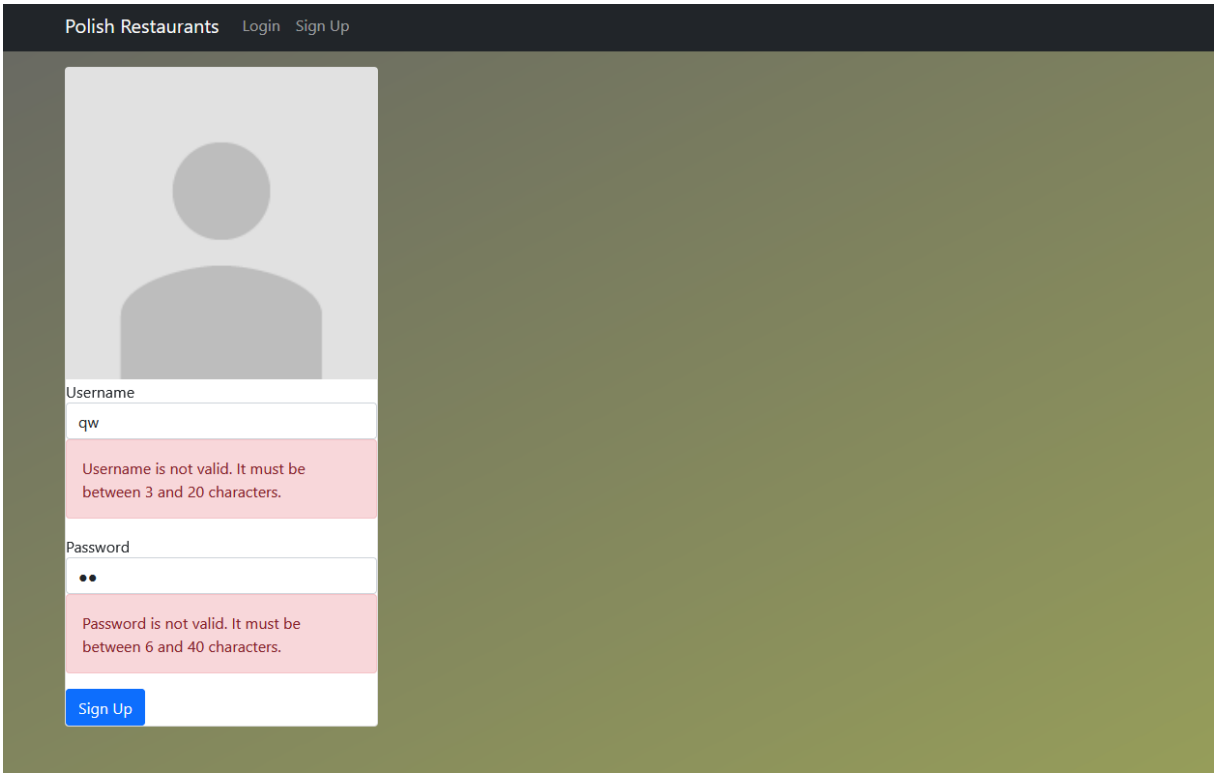
- React-Redux,
- React-Bootstrap,
- MUI,
- Leaflet

Aplikacja została podzielona na niezależne komponenty, gdzie każdy odpowiada za ściśle określoną rolę. Ponadto pozwalają na ponowne ich użycie co zwiększa czytelność kodu i minimalizuję jego powtarzalność. Kod aplikacji został umieszczony w katalogu **frontend.**

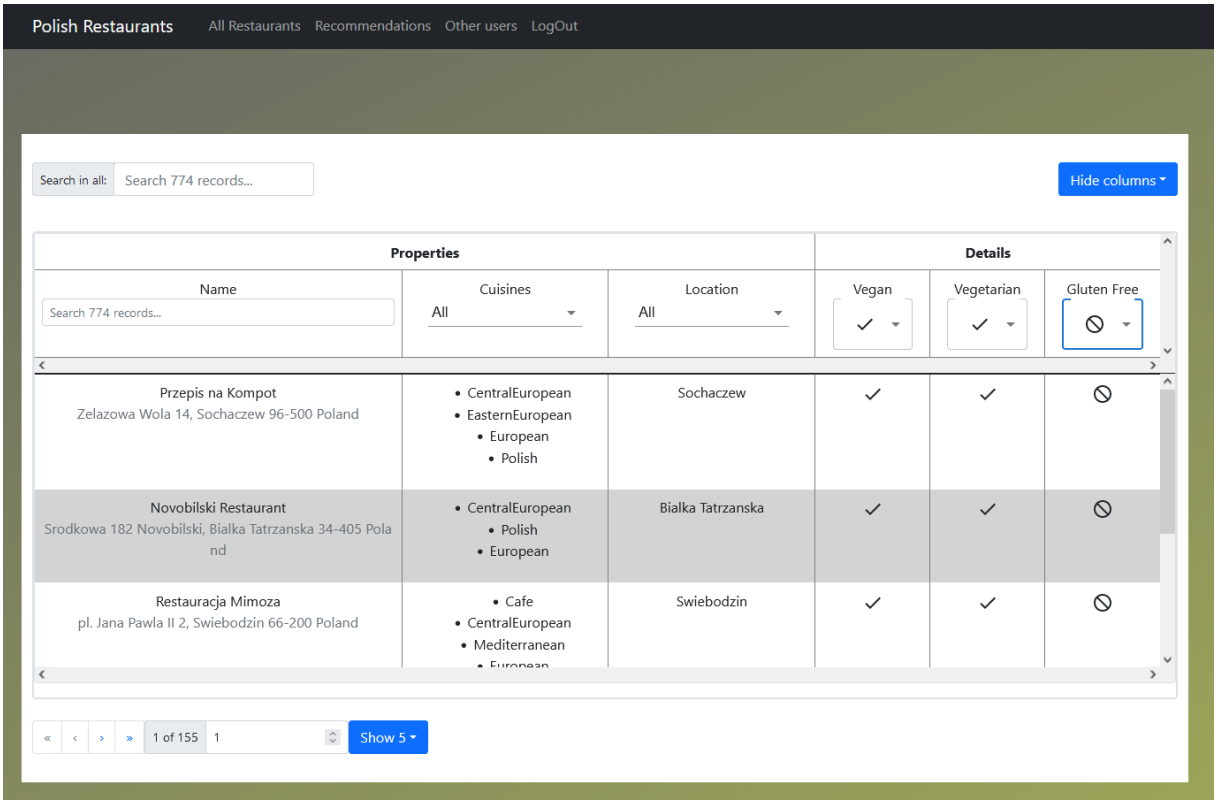
Zdefiniowany panel sterowania aplikacji

Serwis podzielony jest na strony, gdzie każda odpowiada za określoną funkcjonalność.

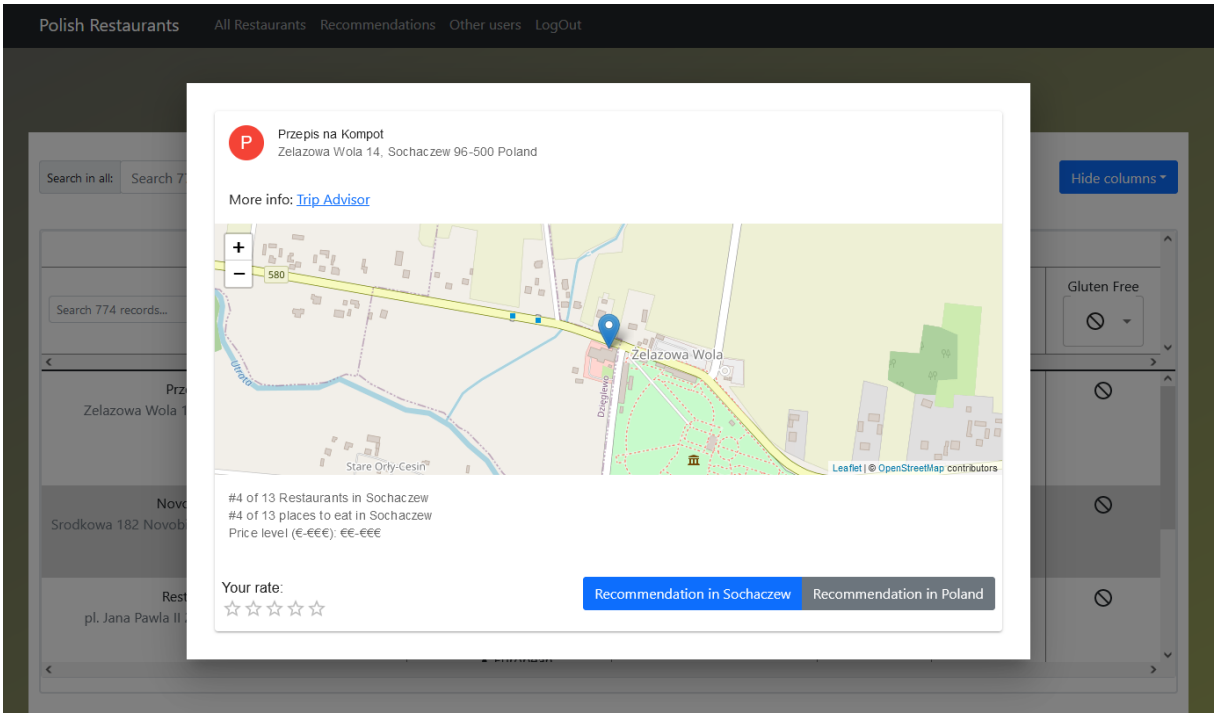
- Strona logowania/rejestracji



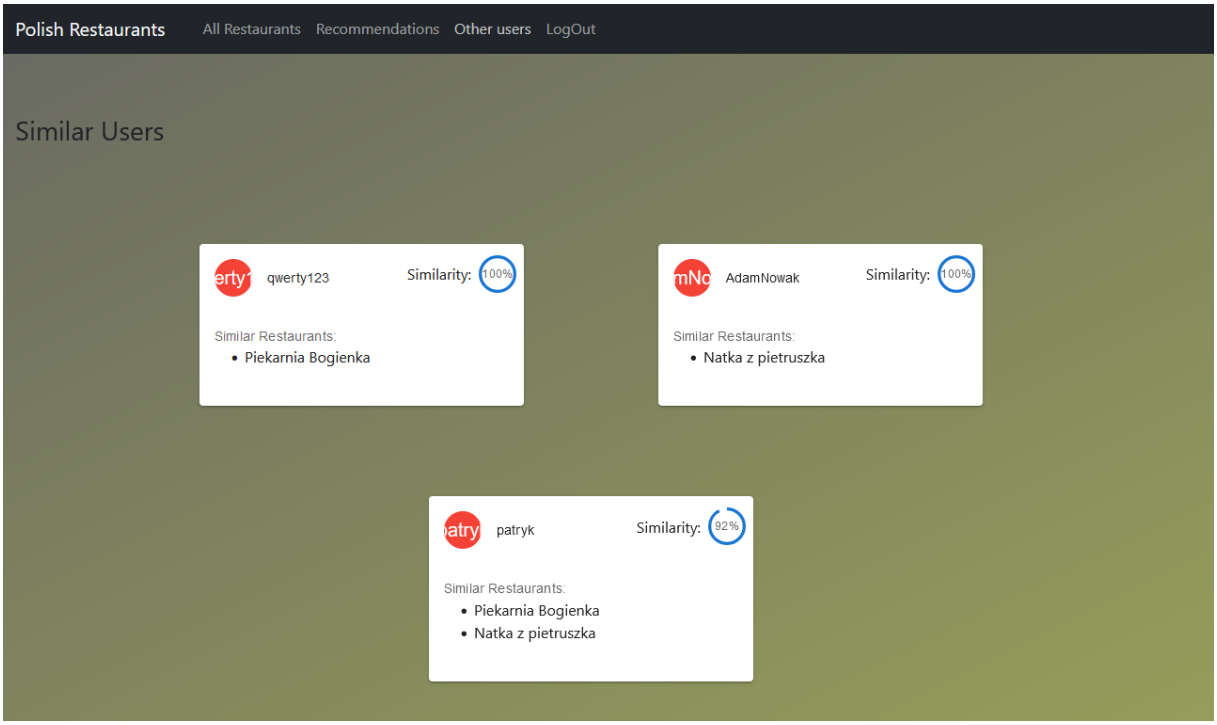
- Strona umożliwiająca przeglądanie dostępnych restauracji



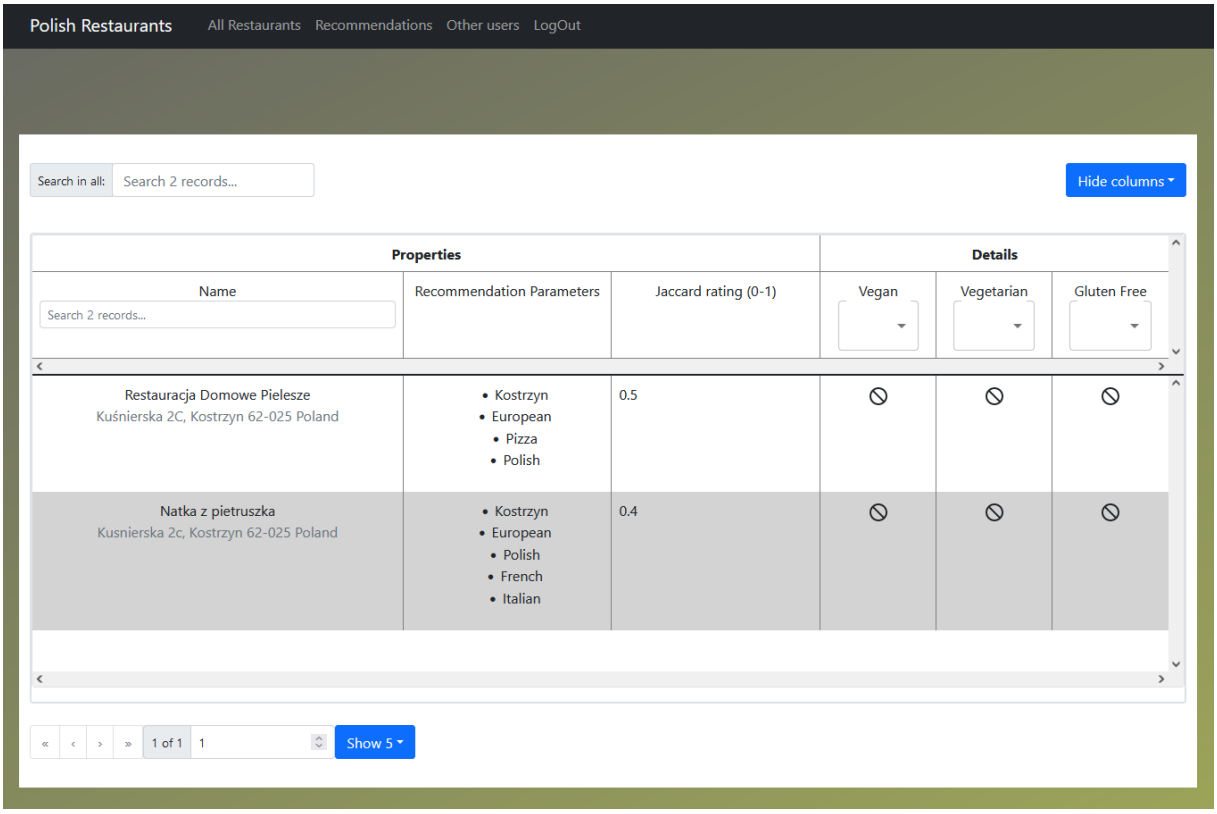
- Strona wyświetlająca szczegóły restauracji



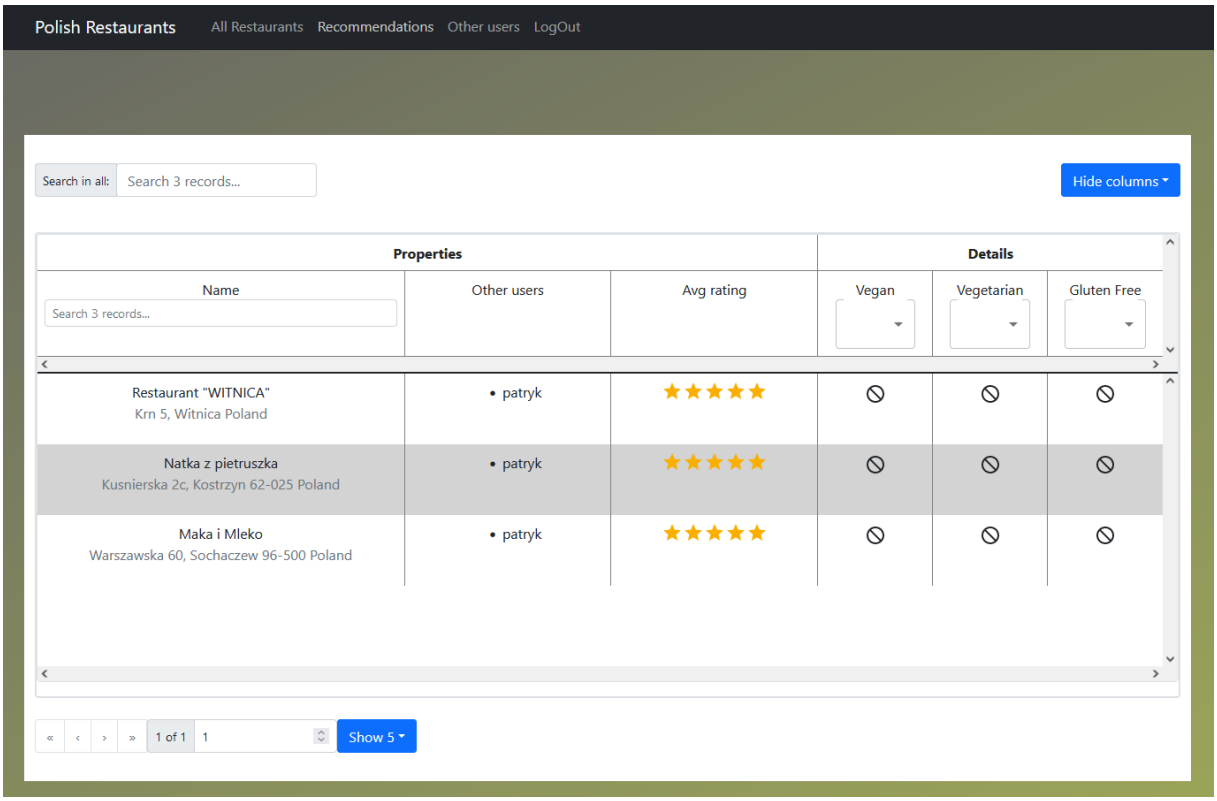
- Strona wyświetlająca podobnych użytkowników ze względu na ocenione restauracje



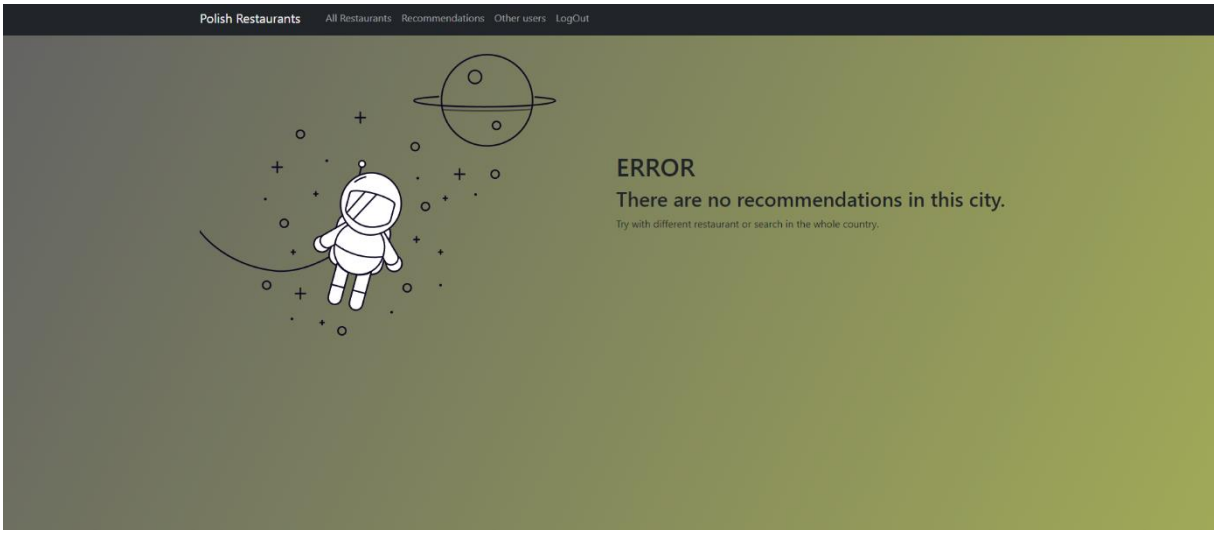
- Strona wyświetlająca rekomendacje restauracji dla danej restauracji w mieście/kraju



- Strona wyświetlająca rekomendacje restauracji ze względu na dotychczas ocenione restauracje



- Strona informująca o braku rekomendacji dla wybranej restauracji



5. Podsumowanie.

Podsumowując, założenia projektu zostały spełnione. Aplikacja została udostępniona w serwisie Heroku pod adresem https://polish-restaurants.herokuapp.com.

W ramach testów aplikacji udostępniono przygotowane konto pilotażowe:

- **Login**: qwerty
- **Hasło**: qwerty

Aplikacja została zaimplementowana w sposób umożliwiający łatwe rozbudowywanie o kolejne funkcjonalności. Wykorzystane narzędzia Flask/React umożliwiają dużą skalowalność.

Możliwe usprawnienia to m.in. poprawa interfejsu użytkownika oraz usprawnienie wyszukiwania restauracji. Ponadto połączenie zaimplementowanego API z oficjalnym API (np. TripAdvisor) udostępniającym dane na temat restauracji, znacznie zwiększyłoby ilość aktualnych informacji na ich temat m.in. oceny czy zdjęcia.

6. Wykaz literatury.

- Wykład,
- https://www.kaggle.com/stefanoleone992/tripadvisor-european-restaurants,
- https://neo4j.com/docs/
- https://flask.palletsprojects.com/en/2.0.x/
- https://pl.reactjs.org/docs/getting-started.html