DOKUMENTACJA APLIKACJI

Air Quality App

Aleksandra Przybyło

Kornelia Lis

Informatyka i Ekonometria

Spis treści

1. Słownik aktorów

1. **Użytkownik (User)**
   Osoba odwiedzająca aplikację w przeglądarce internetowej. Wpisuje nazwę miasta i oczekuje zwrócenia pomiarów jakości powietrza.
2. **Aplikacja Flask (System)**
   Kod serwera odpowiedzialny za obsługę żądań HTTP i logikę biznesową.
3. **Zewnętrzne API (GIOŚ, Google Geocoding)**
   - **API GIOŚ** dostarcza informacji o stacjach pomiarowych, czujnikach i wartościach pomiarów.
   - **Google Geocoding API** służy do pozyskiwania współrzędnych (szerokości i długości geograficznej) na podstawie nazwy miejscowości.

---

2. Przypadki użycia

1. **UC1: Podanie nazwy miasta i wyświetlenie wyników pomiarów**
   - Aktorzy: Użytkownik
   - Opis skrócony: Użytkownik wprowadza nazwę miasta w formularzu, system geokoduje je poprzez Google API i wyszukuje najbliższą stację pomiarową (w promieniu 20 km). Następnie wyświetla podstawowe informacje (aktualne wartości parametrów, barwne wskaźniki jakości powietrza, brak stacji lub błędy, jeśli proces wyszukiwania zawiedzie).
2. **UC2: Wyświetlenie bieżących i godzinowych danych pomiarowych**
   - Aktorzy: Użytkownik
   - Opis skrócony: Po wybraniu stacji pomiarowej (w ramach UC1), system pobiera z API GIOŚ informacje o poszczególnych czujnikach, pobiera najnowsze wartości oraz przygotowuje dane do wykresu godzinowego (od 00:00 do ostatniej pełnej godziny).
3. **UC3: Obliczenie średnich dla województw**
   - Aktorzy: Użytkownik
   - Opis skrócony: Na podstawie wybranej stacji system sprawdza, w jakim województwie się ona znajduje i oblicza uśrednione pomiary dla tego województwa. Dodatkowo system oblicza średnie dla wszystkich województw, aby wyświetlić je w widoku w formie tabeli.

---

3. Specyfikacja przypadków użycia

3.1. UC1: Podanie nazwy miasta i wyświetlenie wyników pomiarów

Identyfikator: UC1
Aktorzy: Użytkownik
Opis:
Użytkownik wprowadza nazwę miasta w formularzu na stronie głównej (`index.html`) i oczekuje informacji o jakości powietrza. Kod odpowiada za geokodowanie miasta (Google

API), wyszukanie najbliższej stacji GIOŚ i wyświetlenie wyników lub stosownych komunikatów o błędzie.

Warunki początkowe:

- Aplikacja jest dostępna pod adresem (np. `http://localhost:5000/`).
- Użytkownik otwiera stronę główną.
- Ewentualna zawartość cache (STATIONS_CACHE) może, ale nie musi, być wypełniona.

Warunki końcowe:

- Użytkownik widzi:
    1. Aktualny stan powietrza z najbliższej stacji (jeśli stacja została znaleziona w promieniu 20 km).
    2. Informację o błędzie, jeśli system nie znalazł stacji w żądanym promieniu, lub nie udało się zgeokodować miasta.

Główny przepływ:

1. Użytkownik wchodzi na stronę główną – metoda `GET` w `index()`.
2. System wyświetla formularz z polem `city` i przyciskiem.
3. Użytkownik wpisuje w polu `city` nazwę miasta – metoda `POST` w `index()`.
4. System odbiera nazwę miasta i wywołuje funkcję `geocode_city(city)`.
    1. `geocode_city` wysyła zapytanie do **Google Geocoding API** z kluczem `API_KEY`.
    2. Jeśli geokodowanie powiedzie się (status 200) i istnieją wyniki, system pobiera współrzędne `lat, lng`.
    3. Jeśli wystąpi błąd lub nie ma wyników, zwraca `(None, None)`.
5. Jeśli współrzędne nie zostały zwrócone (tzn. `(None, None)`), system renderuje `index.html` z komunikatem błędu: „Nie udało się znaleźć współrzędnych miasta.”
6. Jeżeli współrzędne zostały zwrócone, system wywołuje `get_all_stations_cached()` w celu pobrania (lub zcache'owanych) stacji GIOŚ.
    1. Funkcja `get_all_stations_cached()` sprawdza, czy w `STATIONS_CACHE` są już dane i czy nie minął ich czas ważności (TTL).
    2. Jeśli cache jest ważny, zwraca dane z pamięci podręcznej.
    3. Jeśli nie, wywołuje `get_all_stations()` i zapisuje wynik w cache.
7. System wywołuje `find_nearest_stations(lat, lon, stations)` i szuka najbliższej stacji w promieniu 20 km:
    1. `find_nearest_stations` oblicza odległości metodą `haversine` i sortuje stacje od najbliższej do najdalszej.
    2. System wybiera pierwszą stację, która spełnia warunek `dist <= 20`.
8. Jeśli brak stacji w promieniu 20 km, system renderuje `index.html` z informacją: „Nie znaleziono stacji w promieniu 20 km.”
9. W przeciwnym wypadku system zapamiętuje znalezioną stację i jej odległość.
10. (Przekazanie sterowania do UC2 i UC3 w celu dalszych kalkulacji).
11. System w efekcie końcowym renderuje `index.html` z:
    - Nazwą miasta.
    - Najbliższą stacją i odległością.
    - Aktualnymi danymi o zanieczyszczeniach (wyliczonymi w UC2).

- Średnimi wojewódzkimi (wyliczonymi w UC3).

Alternatywny przepływ:

- **A1**: Błąd geokodowania (krok 4) – system wraca do widoku głównego z komunikatem „Nie udało się znaleźć współrzędnych miasta.”
- **A2**: Stacja nieznaleziona w promieniu 20 km (krok 8) – system wyświetla komunikat „Nie znaleziono stacji w promieniu 20 km.”

3.2. UC2: Obliczenie bieżących i godzinowych danych pomiarowych

Identyfikator: UC2
Aktorzy: Użytkownik (pośrednio, w ramach UC1)
Opis:
System po zidentyfikowaniu najbliższej stacji (w UC1) pobiera informacje o czujnikach i najnowsze oraz godzinowe wartości pomiarowe.

Warunki początkowe:

- Została określona najbliższa stacja pomiarowa.
- Mamy jej `station_id`.

Warunki końcowe:

- Aplikacja wyświetla dla każdego parametru (np. PM10, PM2.5, NO2, itp.):
  - Bieżącą wartość i klasę koloru (np. `level-very-good`, `level-good`, ...).
  - Dane godzinowe w celu rysowania wykresu (jeśli występują).

Główny przepływ:

1. Na podstawie identyfikatora stacji wywołana zostaje funkcja `get_sensors(station_id)`, pobierająca z **API GIOŚ** czujniki dostępne dla tej stacji.
2. Dla każdego czujnika:
   1. System pobiera najnowszą wartość wywołaniem `get_latest_value(sensor_id)`.
      - Wywołanie trafia pod endpoint `DATA_URL + sensor_id`.
      - Jeśli nie ma danych, zwraca `None`.
   2. System oblicza klasę koloru przy pomocy `get_color_class(param_name, value)`.
      - Funkcja uwzględnia wartość przedziałową dla danego parametru (PM10, PM2.5, NO2, O3, SO2).
   3. Następnie system pobiera godziny i wartości metodą `get_hourly_data(sensor_id)` (w pętli lub osobno):
      - Wywołanie trafia pod endpoint `DATA_URL + sensor_id`.
      - Dla każdego rekordu w polu `values`:
        - Parsuje datę i wartość pomiarową.
        - Odfiltrowuje dane z dzisiejszej doby (od 00:00 do ostatniej pełnej godziny).

        - Sortuje rosnąco według godziny.
    4. Gotowe dane są składowane w `hourly_plot_data[param_name]` w formie listy `(godzina, wartość)`.
3. Dla brakujących wartości system ustawia `"Brak danych"`.
4. Aplikacja (kontroler `index()`) przekazuje `pollutant_data` i `hourly_plot_data` do szablonu `index.html`, co pozwala na wyświetlenie aktualnych odczytów i wykresów godzinowych.

3.3. UC3: Obliczenie średnich dla województw

Identyfikator: UC3
Aktorzy: Użytkownik (pośrednio, w ramach UC1)
Opis:
System, znając województwo wybranej stacji, oblicza średnią wartość pomiarów w tym województwie oraz dla wszystkich pozostałych województw.

Warunki początkowe:

- Została określona najbliższa stacja pomiarowa i pobrano z jej obiektu pole `addressVoivodeship` (lub `provinceName`).
- Jest dostępna lista wszystkich stacji GIOŚ (z cache lub ze świeżego zapytania).

Warunki końcowe:

- Aplikacja wyświetla średnie wartości zanieczyszczeń dla wybranego województwa (pole `voivodeship_averages`) oraz tabelę średnich dla wszystkich województw (`all_voiv_averages`).

Główny przepływ:

1. System odczytuje atrybut `chosen_voiv` ze stacji (`addressVoivodeship` lub `city.commune.provinceName`).
2. Aplikacja wywołuje `calculate_voivodeship_averages(chosen_voiv)`, aby obliczyć średnie w województwie:
    1. Funkcja pobiera (z cache) listę wszystkich stacji GIOŚ.
    2. Filtruje je tak, by zostały tylko te ze wskazanego województwa.
    3. Dla każdej stacji pobiera czujniki (`get_sensors`), a następnie wszystkie wartości (`get_all_values`).
    4. Sumuje te wartości i liczy ich średnią (`avg = sum(vals) / len(vals)`), a następnie przypisuje klasę koloru przez `get_color_class`.
    5. Zwraca słownik wyników dla każdego parametru.
3. Funkcja `calculate_all_voivodeships_averages_cached()` jest wywoływana w celu uzyskania tabeli zbiorczej dla wszystkich województw.
    1. Sprawdza, czy `ALL_VOIV_CACHE` jest aktualny.
    2. Jeśli nie, wywołuje `calculate_all_voivodeships_averages()`.
    - Analogicznie sumuje wszystkie dane w podziale na województwa.

    3. Zwraca dane z pamięci podręcznej bądź świeżo przeliczone.

2. Aplikacja (kontroler `index()`) przekazuje `voivodeship_averages` i `all_voiv_averages` do szablonu `index.html`, co umożliwia wyświetlenie tabeli ze średnimi na warstwie frontendu.

Alternatywny przepływ:

- **A1**: Jeśli stacja nie ma zdefiniowanego województwa, system nie liczy indywidualnych danych dla wybranego województwa. Zwracana jest pusta informacja lub komunikat o braku danych.
- **A2**: Jeśli brak stacji w danym województwie (np. błąd w danych), wynikiem będzie pusty zbiór.

---

4. Licencje

1. **Apache License 2.0** – permisywna licencja open-source

---

5. Technologie

5.1. Backend

5.1.1. Python 3.x

- **Wersja**: Aplikacja działa w środowisku Python 3
- **Oficjalna dokumentacja**: https://docs.python.org/3/

5.1.2. Flask

- **Właściwości używane w projekcie**:
  - **Routing** (`@app.route('/', methods=['GET','POST'])`)
  - **Renderowanie szablonów** (`render_template('index.html', ...)`)
  - **Obsługa żądań typu `POST`** (formularz z polem na nazwę miasta)
  - **Tryb debugowania** (`app.run(debug=True)`) dla łatwiejszego wykrywania błędów w trakcie developmentu.
- **Oficjalna dokumentacja**: https://flask.palletsprojects.com/

5.1.3. requests

- **Opis**: Biblioteka służąca do wykonywania zapytań HTTP (GET, POST, itp.) w Pythonie
- **W projekcie** używana do komunikacji z zewnętrznymi API:
  - **Google Geocoding API** (pobieranie współrzędnych na podstawie nazwy miasta).
  - **API GIOŚ** (pobieranie informacji o stacjach, czujnikach i pomiarach jakości powietrza).

- **Oficjalna dokumentacja**:
  https://docs.python-requests.org/

5.1.4. math i datetime

- **math**: standardowa biblioteka Pythona, m.in. używana do obliczania odległości (funkcja `haversine`) i konwersji kątów (radians).
- **datetime**: standardowa biblioteka Pythona do operacji na datach i czasie. W projekcie wykorzystywana do filtrowania danych godzinowych (`get_hourly_data`).

5.1.5. time

- **Opis**: moduł standardowej biblioteki Pythona.
- **W projekcie**: stosowany do uzyskania aktualnego czasu (timestamp) i porównywania go z czasem przechowywania danych w pamięci podręcznej (cache).

5.1.6. Jinja2 (silnik szablonów w Flasku)

- **Opis**: renderowanie szablonów HTML.
- **Oficjalna dokumentacja**:
  https://jinja.palletsprojects.com/

5.1.7. Cache (pamięć podręczna)

- **Opis**: Aplikacja nie używa bazy danych, ale przechowuje część danych z zewnętrznych API w pamięci podręcznej (słowniki Python) z ograniczonym czasem ważności (TTL).
- **Implementacja**:
  - `STATIONS_CACHE` i `ALL_VOIV_CACHE` to globalne słowniki, przechowujące dane pobrane z API GIOŚ (np. listy stacji lub obliczone średnie).
  - Jeśli dane są przestarzałe (upłynął TTL – np. 10 minut), aplikacja wywołuje ponownie zapytania do API GIOŚ i odświeża cache.

5.1.8. Ngrok

- **Opis**: Narzędzie umożliwiające bezpieczne tunelowanie lokalnego serwera na publiczny URL.
- **W projekcie**: Używane do tymczasowego udostępnienia aplikacji działającej lokalnie w sieci publicznej na potrzeby testowania i prezentacji.
- **Oficjalna dokumentacja**:
  https://ngrok.com/docs

5.2. Frontend

5.2.1. HTML (HyperText Markup Language)

- **Użycie w projekcie**:
  - Plik `index.html` zawiera szkielet i strukturę strony.
  - Jinja2 wypełnia dynamiczne elementy danymi z backendu (np. nazwa stacji, wartości pomiarów, błędy).
- **Dokumentacja**:

- https://developer.mozilla.org/en-US/docs/Web/HTML (MDN)
- https://html.spec.whatwg.org/ (WHATWG – Living Standard)

5.2.2. CSS (Cascading Style Sheets)

- **Użycie w projekcie**:
  - Plik `styles.css` dołączany w `<head>`