DOKUMENTACJA TECHNICZNA PROJEKTU GEOKODOWANIA

### 18 stycznia 2025

**Spis treści**

1. [Opis techniczny projektu](#_bookmark0) 3
   1. [Wprowadzenie](#_bookmark1) 3
      1. [Cel aplikacji](#_bookmark2) 3
      2. [Zakres dokumentacji](#_bookmark3) 3
      3. [Technologie i wymagania](#_bookmark4) 3
   2. [Funkcjonalności aplikacji](#_bookmark5) 4
      1. [Opis działania programu na podstawie diagramu](#_bookmark6) 4
      2. [Wczytywanie danych wejściowych](#_bookmark8) 6
      3. [Geokodowanie adresów](#_bookmark9) 6
      4. [Przetwarzanie danych adresowych](#_bookmark10) 6
      5. [Diagram przepływu przetwarzania danych](#_bookmark11) 7
      6. [Przetwarzanie wielowątkowe](#_bookmark13) 8
      7. [Zapis wyników](#_bookmark14) 10
      8. [Statystyki i analiza](#_bookmark15) 11
   3. [Struktura kodu](#_bookmark16) 14
      1. [Główne komponenty](#_bookmark17) 14
      2. [Przebieg programu](#_bookmark18) 15
      3. [Kluczowe fragmenty kodu](#_bookmark19) 15
      4. [Efektywność i skalowalność](#_bookmark20) 17
      5. [Główne klasy i struktury](#_bookmark21) 17
   4. [Obsługa sieci](#_bookmark22) 19
      1. [Tworzenie asynchronicznego gniazda](#_bookmark23) 19
      2. [Obsługa zapytań HTTP](#_bookmark24) 20
      3. [Kodowanie adresów URL](#_bookmark25) 21
      4. [Efektywność i zalety](#_bookmark26) 21
   5. [Format danych](#_bookmark27) 22
      1. [Dane wejściowe](#_bookmark28) 22
      2. [Dane wyjściowe](#_bookmark29) 22
      3. [Flagi przetwarzania](#_bookmark30) 23
      4. [Numer województwa](#_bookmark31) 23

2

**Rozdział 1**

**Opis techniczny projektu**

# Wprowadzenie

## Cel aplikacji

Celem aplikacji jest przetwarzanie, geokodowanie i analiza danych adresowych. Program wczytuje dane z pliku CSV, przetwarza informacje o adresach, wzbogaca je o współrzędne geograficzne (szerokość i długość geograficzną) oraz dodatkowe dane, takie jak nazwa miejsca, miasto, kod pocztowy i ulica. Wyniki są zapisywane w pliku CSV, który może być wykorzystany do dalszej analizy.

## Zakres dokumentacji

Dokumentacja opisuje:

Funkcjonalności programu,

Struktury danych i algorytmy użyte w aplikacji,

Szczegóły implementacyjne, takie jak wielowątkowość, obsługa sieci i bazy danych,

Format danych wejściowych i wyjściowych.

## Technologie i wymagania

Program został napisany w języku C++ i korzysta z następujących technologii:

**PostgreSQL z PostGIS** – do obsługi danych geograficznych,

**Nominatim API** – do geokodowania adresów,

**Wielowątkowość** – do równoległego przetwarzania danych,

**Asynchroniczne połączenia sieciowe** – do obsługi zapytań HTTP. Wymagania systemowe:

System operacyjny Linux,

Kompilator obsługujący standard C++17 lub nowszy,

Zainstalowana baza danych PostgreSQL z PostGIS,

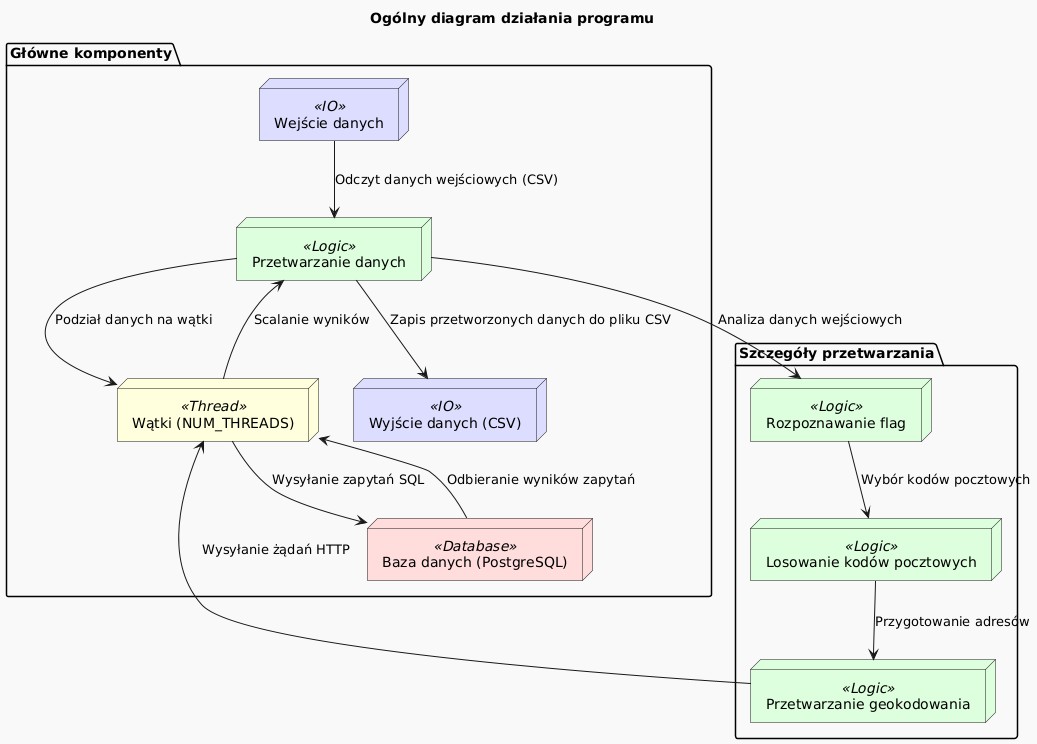
Biblioteki: pqxx, nlohmann::json, csvstream.

3

# Funkcjonalności aplikacji

## Opis działania programu na podstawie diagramu

Diagram przedstawia strukturę i przepływ danych w aplikacji. Składa się z dwóch głównych sekcji: komponentów głównych, które odpowiadają za ogólny przepływ danych w systemie, oraz szczegółowych etapów przetwarzania, które opisują wewnętrzne operacje wykonywane na danych.



Rysunek 1.1: Diagram przedstawiający przepływ danych w aplikacji.

Jak pokazano na rysunku [1.1,](#_bookmark7) aplikacja składa się z kilku głównych komponentów, które współpracują w celu przetwarzania danych adresowych. Poniżej opisano każdy z tych komponentów.

#### Główne komponenty systemu

Diagram zawiera sekcję zatytułowaną ”Główne komponenty”, w której przedstawiono podstawowe ele- menty programu oraz sposób, w jaki dane przepływają między nimi:

**Wejście danych**: Komponent ”Wejście danych”reprezentuje proces wczytywania danych wejścio- wych z pliku CSV. Dane te zawierają informacje o adresach, które zostaną przetworzone przez system. Proces odczytu danych jest pierwszym krokiem w przepływie, który dostarcza informacje do kolejnego komponentu, czyli ”Przetwarzania danych”.

**Przetwarzanie danych**: Komponent ”Przetwarzanie danych”jest odpowiedzialny za główną lo- gikę programu. Dane wczytane z pliku CSV są analizowane, przetwarzane i dzielone na mniejsze podzbiory, które następnie są przekazywane do wątków w celu równoległego przetwarzania. Ten etap obejmuje operacje takie jak analiza poprawności danych, przypisywanie flag oraz przygotowa- nie danych do dalszych operacji.

**Wątki (NUM THREADS)**: Komponent ”Wątki”reprezentuje wielowątkowe przetwarzanie da- nych, które pozwala na równoległe wykonywanie operacji na różnych fragmentach zbioru danych. Każdy wątek komunikuje się z bazą danych PostgreSQL, wysyłając zapytania SQL w celu uzyskania brakujących informacji lub weryfikacji istniejących danych.

**Baza danych (PostgreSQL)**: Komponent ”Baza danych”odpowiada za przechowywanie infor- macji geograficznych, takich jak współrzędne geograficzne (szerokość i długość geograficzna) dla adresów. Wątki wysyłają zapytania SQL do bazy danych, a wyniki tych zapytań są zwracane do odpowiednich wątków w celu dalszego przetwarzania.

**Wyjście danych (CSV)**: Komponent ”Wyjście danych”reprezentuje proces zapisywania prze- tworzonych danych do pliku CSV. Po zakończeniu wszystkich operacji dane są scalane i zapisane w uporządkowanym formacie, który zawiera zarówno oryginalne informacje, jak i uzupełnione dane, takie jak współrzędne geograficzne.

#### Szczegóły przetwarzania danych

W diagramie znajduje się również sekcja ”Szczegóły przetwarzania”, która opisuje bardziej szczegółowe etapy przetwarzania danych, wykonywane wewnątrz komponentu ”Przetwarzanie danych”:

**Rozpoznawanie flag**: W pierwszym kroku szczegółowego przetwarzania dane wejściowe są ana- lizowane w celu określenia, które informacje są obecne, a które brakuje. Na podstawie tej analizy każdemu rekordowi przypisywana jest flaga, która wskazuje, jakie operacje należy wykonać. Na przykład, jeśli brakuje kodu pocztowego, ale miasto i ulica są dostępne, rekord otrzymuje flagę oznaczającą konieczność uzupełnienia brakującego kodu pocztowego.

**Losowanie kodów pocztowych**: W przypadku brakujących kodów pocztowych program prze- prowadza losowanie na podstawie rozkładu prawdopodobieństwa obliczonego z dostępnych danych. Rozkład ten jest tworzony na podstawie liczby wystąpień poszczególnych kodów pocztowych w da- nych wejściowych. Wynik losowania jest przypisywany do odpowiednich rekordów, aby uzupełnić brakujące informacje.

**Przetwarzanie geokodowania**: Kolejnym krokiem jest przygotowanie adresów do wysyłania żą- dań HTTP do serwera geokodowania. Adresy są kodowane w odpowiednim formacie, a następnie wysyłane w celu uzyskania współrzędnych geograficznych, takich jak szerokość i długość geogra- ficzna. Wyniki są odbierane przez wątki i przypisywane do odpowiednich rekordów.

**Komunikacja z wątkami**: Po zakończeniu przetwarzania geokodowania dane są przekazywane do wątków, które realizują równoległe operacje, takie jak wysyłanie zapytań SQL do bazy danych PostgreSQL. Wątki odbierają wyniki zapytań i integrują je z danymi przetwarzanymi w systemie.

#### Podsumowanie

Diagram przedstawia kompleksowy przepływ danych w programie, który łączy w sobie operacje wej- ścia/wyjścia, wielowątkowość, komunikację z bazą danych oraz obsługę żądań HTTP. Dzięki logicznemu podziałowi na główne komponenty oraz szczegółowe etapy przetwarzania, program jest w stanie efektyw- nie przetwarzać duże zbiory danych adresowych, uzupełniać brakujące informacje i zapisywać wyniki w uporządkowanej formie. Kluczowym elementem jest wykorzystanie flag do klasyfikacji danych oraz loso- wanie kodów pocztowych w przypadku brakujących informacji, co pozwala na automatyczne uzupełnianie danych wejściowych.

## Wczytywanie danych wejściowych

Program wczytuje dane z pliku CSV. Każdy rekord zawiera informacje o adresie (ulica, kod pocztowy, miasto, województwo, kraj) oraz dodatkowe dane, takie jak numer umowy, data rozpoczęcia i zakończenia.

Wczytane dane są następnie przetwarzane w celu przygotowania ich do geokodowania:

Adresy są analizowane i klasyfikowane na podstawie dostępnych danych (np. brak ulicy, brak kodu pocztowego).

Rekordy są oznaczane flagami (flaga1, flaga2), które określają, jakie dane są dostępne i jakie przetwarzanie będzie wymagane.

W przypadku brakujących danych, rekordy są dodawane do wektora rozklad, gdzie później zostaną uzupełnione.

## Geokodowanie adresów

Aplikacja przetwarza dane adresowe i wzbogaca je o:

Szerokość i długość geograficzną,

Nazwę miejsca (np. pełny adres w formacie tekstowym).

W przypadku brakujących danych program korzysta z losowego uzupełniania na podstawie rozkładu prawdopodobieństwa.

## Przetwarzanie danych adresowych

Proces przetwarzania danych adresowych składa się z następujących kroków:

* + - 1. **Inicjalizacja zmiennych**: Przygotowanie struktur danych i wektorów na potrzeby przetwarzania.
      2. **Wczytanie pliku CSV**: Odczyt danych wejściowych z pliku.
      3. **Iteracja przez rekordy**: Pobranie rekordu z pliku i jego przetwarzanie krok po kroku.
      4. **Przypisywanie flag**: Każdy rekord jest analizowany na podstawie dostępności danych adresowych (ulica, kod pocztowy, miasto), a następnie przypisywana jest odpowiednia wartość flagi flaga1. Opis poszczególnych wartości flag:

flaga1 = 0: Brak wszystkich kluczowych danych adresowych (ulica, kod pocztowy, miasto są puste).

flaga1 = 1: Dane są częściowo dostępne, ale nie pasują do żadnego z pozostałych warunków.

flaga1 = 2: Ulica i miasto są dostępne, ale kod pocztowy jest pusty, a ulica nie jest taka sama jak miasto.

flaga1 = 3: Wszystkie kluczowe dane są dostępne, ale ulica zawiera nazwę miasta.

flaga1 = 4: Wszystkie kluczowe dane są dostępne, ale ulica jest równa nazwie miasta.

flaga1 = 5: Kod pocztowy i miasto są dostępne, ale ulica jest pusta.

flaga1 = 6: Kod pocztowy jest dostępny, ale zarówno ulica, jak i miasto są puste.

flaga1 = 7: Wszystkie dane adresowe są puste (inny przypadek niż flaga1 = 0).

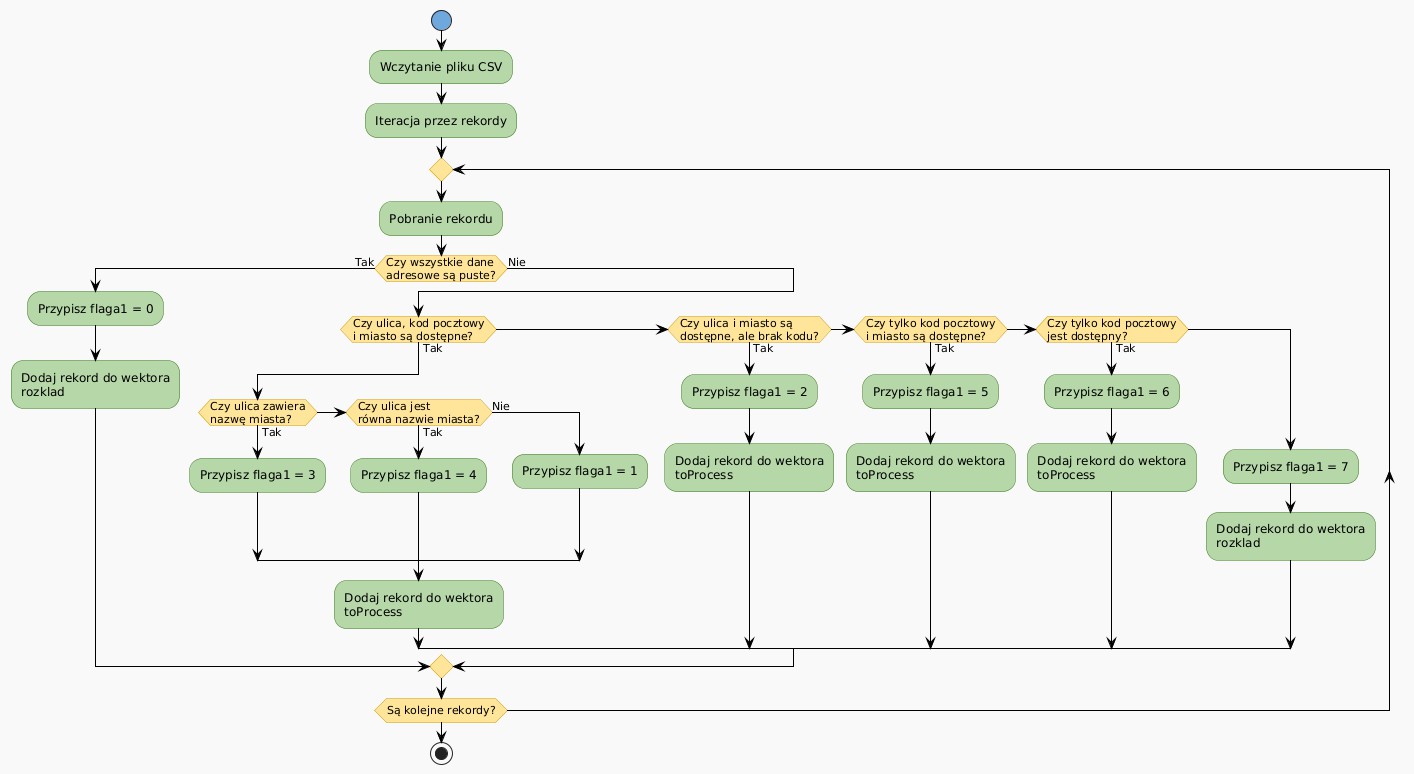
* + - 1. **Przypisywanie flagi** flaga2: Flaga flaga2 jest domyślnie ustawiana na wartość 0. Jest używana do oznaczania rekordów gotowych do dalszego przetwarzania.
      2. **Klasyfikacja rekordów**: Na podstawie wartości flagi flaga1, rekordy są klasyfikowane i dodawane do odpowiednich struktur danych:

Rekordy o wartościach flaga1 = 0 lub flaga1 = 7 są dodawane do wektora rozklad.

Rekordy z innymi wartościami flaga1 są dodawane do wektora toProcess.

## Diagram przepływu przetwarzania danych

Na Rysunku [1.2](#_bookmark12) przedstawiono diagram przepływu przetwarzania danych wejściowych. Diagram ilustruje proces wczytywania danych z pliku CSV, klasyfikacji rekordów na podstawie dostępnych danych oraz przygotowania ich do dalszego przetwarzania.



Rysunek 1.2: Diagram przepływu przetwarzania danych wejściowych.

**Opis diagramu** Diagram przedstawia proces przetwarzania rekordów adresowych z pliku CSV. Kroki są następujące:

* + - 1. **Inicjalizacja zmiennych**: Przygotowanie struktur danych i wektorów (rozklad oraz toProcess) na potrzeby przetwarzania.
      2. **Wczytanie pliku CSV**: Odczytanie danych wejściowych z pliku CSV.
      3. **Iteracja przez rekordy**: Przeglądanie każdego rekordu w pliku CSV w pętli.

#### Sprawdzenie, czy wszystkie dane adresowe są puste:

Jeśli tak, rekord jest klasyfikowany jako niekompletny (flaga1 = 0) i dodawany do wektora

rozklad.

W przeciwnym razie, przechodzi do dalszego przetwarzania.

#### Sprawdzenie, czy dostępne są wszystkie dane adresowe (ulica, kod pocztowy, miasto):

Jeśli tak, następuje dodatkowa analiza:

* + - * + Jeśli ulica zawiera nazwę miasta, rekord otrzymuje flaga1 = 3.
        + Jeśli ulica jest równa nazwie miasta, rekord otrzymuje flaga1 = 4.
        + W przeciwnym razie, rekord otrzymuje flaga1 = 1. Rekord jest następnie dodawany do wektora toProcess.

#### Sprawdzenie, czy dostępne są ulica i miasto, ale brak kodu pocztowego:

Jeśli tak, rekord otrzymuje flaga1 = 2 i jest dodawany do wektora toProcess.

#### Sprawdzenie, czy dostępne są tylko kod pocztowy i miasto:

Jeśli tak, rekord otrzymuje flaga1 = 5 i jest dodawany do wektora toProcess.

#### Sprawdzenie, czy dostępny jest tylko kod pocztowy:

Jeśli tak, rekord otrzymuje flaga1 = 6 i jest dodawany do wektora toProcess.

* + - 1. **Pozostałe przypadki**: Jeśli żadne z powyższych kryteriów nie zostało spełnione, rekord otrzymuje

flaga1 = 7 i jest dodawany do wektora rozklad.

* + - 1. **Zakończenie iteracji**: Proces powtarza się dla każdego rekordu w pliku CSV, aż wszystkie zostaną przetworzone.

## Przetwarzanie wielowątkowe

Program wykorzystuje przetwarzanie wielowątkowe w celu zwiększenia wydajności operacji na dużych zbiorach danych. Domyślnie uruchamiane są 32 wątki, które równolegle wykonują różne zadania, takie jak przetwarzanie danych wejściowych, wysyłanie żądań HTTP czy zapis wyników.

#### Główne elementy przetwarzania wielowątkowego:

#### Inicjalizacja danych wejściowych:

Dane są wczytywane z pliku CSV, a następnie klasyfikowane na podstawie ich zawartości (np. brakujące dane adresowe, kompletne dane, itp.).

Każdy rekord jest przypisywany do odpowiedniego wektora (toProcess, rozklad, flaga1, flaga2, itd.) w zależności od jego klasyfikacji.

Wektory są także podzielone na części odpowiadające liczbie wątków (all, all2, all3, all4) w celu równoległego przetwarzania.

#### Podział pracy:

Dane do przetwarzania są dzielone na fragmenty odpowiadające liczbie wątków (NUM THREADS

= 32).

Każdy wątek otrzymuje zakres rekordów do przetworzenia, co pozwala na równoległe operacje.

#### Równoległe przetwarzanie danych:

Każdy wątek wykonuje funkcję perform requests, która obsługuje przetwarzanie rekordów w zakresie przypisanym do danego wątku.

Wątki komunikują się z serwerem za pomocą asynchronicznych żądań HTTP, wykorzystując gniazda w trybie nieblokującym oraz mechanizm epoll.

Odpowiedzi serwera są analizowane, a wyniki (np. współrzędne geograficzne, nazwa miejsca) są przypisywane do odpowiednich rekordów.

#### Obsługa brakujących danych:

Rekordy z brakującymi danymi są przetwarzane w dodatkowych iteracjach, aż do uzyskania wymaganych informacji (np. kodu pocztowego lub współrzędnych geograficznych).

W przypadku braku danych, rekordy są klasyfikowane i przypisywane do odpowiednich wek- torów (np. flaga2, flaga3).

#### Zapis wyników:

Po zakończeniu przetwarzania dane są zapisywane w pliku CSV za pomocą funkcji saveToCSV.

Plik wynikowy zawiera dodatkowe kolumny, takie jak współrzędne geograficzne (Szerokosc,

Dlugosc), flaga przetwarzania (Flaga 1, Flaga 2) oraz numer województwa (Nr Wojewodztwa).

#### Szczegóły implementacyjne:

Program korzysta z bibliotek standardowych C++ (<thread>, <mutex>, <vector>, <atomic>) do obsługi wątków i synchronizacji.

Mechanizm epoll jest wykorzystywany do obsługi asynchronicznych żądań HTTP w trybie nie- blokującym.

Dane wejściowe są wczytywane z pliku CSV za pomocą biblioteki csvstream.hpp.

Wyniki są analizowane za pomocą biblioteki nlohmann::json do parsowania odpowiedzi w formacie JSON.

Synchronizacja dostępu do współdzielonych struktur danych (np. occurrences, workers data) jest zapewniona za pomocą mutexów (std::mutex).

**Podział pracy na wątki:** Dane są dzielone na fragmenty odpowiadające liczbie wątków (NUM THREADS

= 32). Zakresy danych są obliczane na podstawie liczby rekordów i liczby wątków:

int M = NUM\_THREADS; // liczba wątków

int N = toProcess.size(); // liczba rekordów do przetworzenia std::vector<std::pair<int, int>> ranges;

int chunkSize = N / M;

for (int i = 0; i < M; ++i) { int start = i \* chunkSize;

int end = (i == M - 1) ? N : start + chunkSize; ranges.push\_back({start, end});

}

Każdy wątek przetwarza dane w zakresie [start, end).

**Mechanizm przetwarzania żądań HTTP:** Wątki wysyłają żądania HTTP do serwera za pomocą funkcji create nonblocking socket i epoll. Żądania są wysyłane w formacie:

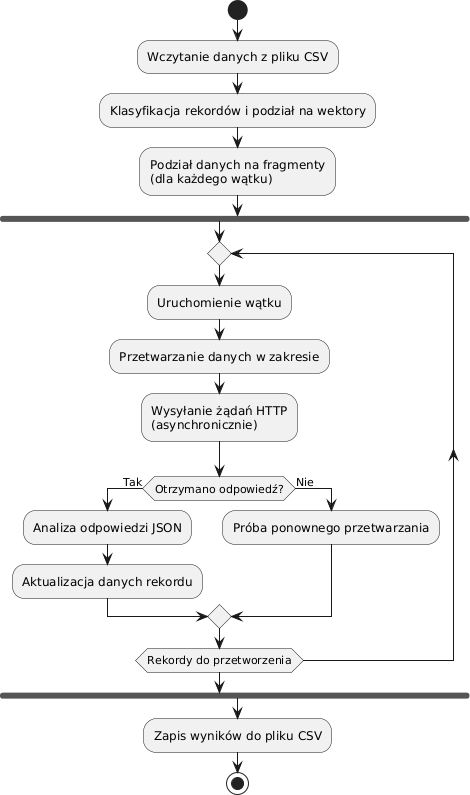
GET /search.php?q=<adres>&format=json&limit=1 HTTP/1.1 Host: 127.0.0.1:8080

Connection: keep-alive

Odpowiedzi w formacie JSON są parsowane za pomocą biblioteki nlohmann::json, a wyniki (np. współrzędne geograficzne) są przypisywane do odpowiednich rekordów.

**Zapis wyników:** Dane są zapisywane w pliku CSV w formacie:

Lp;DataPoczatku;DataKonca;SumaUbezpieczenia;Odnowienia; Ulica;KodPocztowy;Miasto;Wojewodztwo;Kraj;ReasekuracjaO; ReasekuracjaF;Szerokosc;Dlugosc;Flaga\_1;Flaga\_2;Nr\_Wojewodztwa



Rysunek 1.3: Proces przetwarzania wielowątkowego

## Zapis wyników

Wyniki przetwarzania są zapisywane w pliku CSV z dodatkowymi kolumnami, takimi jak współrzędne geograficzne, nazwa miejsca oraz flaga przetwarzania. Funkcja saveToCSV odpowiada za iterację przez przetworzone dane i zapisanie ich w odpowiednim formacie do pliku.

#### Opis działania funkcji:

* + - 1. **Otwarcie pliku**: Funkcja próbuje otworzyć plik o nazwie podanej w parametrze filename. Jeśli plik nie może zostać otwarty, zwracany jest komunikat o błędzie i funkcja kończy swoje działanie.
      2. **Zapis nagłówka**: Do pliku zapisywany jest nagłówek zawierający nazwy kolumn, takie jak: Lp, DataPoczatku, DataKonca, Ulica, KodPocztowy, Miasto, Wojewodztwo, Kraj, Szerokosc, Dlugosc, Flaga 1, Flaga 2, Nr Wojewodztwa.
      3. **Iteracja przez dane**: Funkcja iteruje przez wektor dataToCSV, który zawiera przetworzone rekordy adresowe.
      4. **Przypisanie województwa i numeru województwa**: Dla każdego rekordu wywoływana jest funkcja getWojewodztwoMapa, która na podstawie kodu pocztowego przypisuje nazwę województwa oraz jego numer.
      5. **Zapis danych do pliku**: Każdy rekord jest zapisywany w formacie CSV, gdzie poszczególne pola są oddzielone średnikami. Zawartość rekordu obejmuje dane adresowe, współrzędne geograficzne, flagi przetwarzania oraz numer województwa.
      6. **Zamknięcie pliku**: Po zapisaniu wszystkich rekordów plik zostaje zamknięty.

#### Nagłówek pliku CSV:

Lp;DataPoczatku;DataKonca;SumaUbezpieczenia;Odnowienia;Ulica;KodPocztowy;Miasto; Wojewodztwo;Kraj;ReasekuracjaO;ReasekuracjaF;Szerokosc;Dlugosc;Flaga\_1;Flaga\_2;Nr\_Wojewodztwa

**Pola danych:** Każdy zapisany rekord zawiera następujące pola:

Lp – liczba porządkowa rekordu.

DataPoczatku, DataKonca – daty związane z okresem obowiązywania danych.

SumaUbezpieczenia, Odnowienia – dane finansowe.

Ulica, KodPocztowy, Miasto, Wojewodztwo, Kraj – dane adresowe.

ReasekuracjaO, ReasekuracjaF – informacje o reasekuracji.

Szerokosc, Dlugosc – współrzędne geograficzne.

Flaga 1, Flaga 2 – flagi przetwarzania.

Nr Wojewodztwa – numer województwa.

## Statystyki i analiza

Program analizuje częstość występowania kodów pocztowych w danych wejściowych i wykorzystuje te statystyki do losowego uzupełniania brakujących informacji. Proces ten jest realizowany w następujących krokach:

#### Analiza kodów pocztowych:

Na podstawie danych wejściowych zliczana jest liczba wystąpień każdego kodu pocztowego.

Częstości występowania są normalizowane, aby stworzyć rozkład prawdopodobieństwa dla każdego kodu pocztowego. Rozkład ten jest przechowywany w wektorze probabilities, a odpowiadające mu kody w wektorze codes.

Rozkład prawdopodobieństwa jest wykorzystywany do uzupełniania brakujących kodów poczto- wych w rekordach.

#### Losowe uzupełnianie brakujących informacji:

Dla każdego rekordu z brakującymi danymi (np. kodem pocztowym lub miastem) program analizuje jego flagę (flaga1) i podejmuje odpowiednie działania:

* Jeśli rekord ma brakujący kod pocztowy (flaga1 == 0), jest on losowany na podstawie roz- kładu prawdopodobieństwa.
* Jeśli rekord ma brakujące miasto (flaga1 == 5) lub inne brakujące dane, podejmowane są dodatkowe próby uzupełnienia danych.

Program śledzi liczbę prób (proby) dla każdego rekordu, aby uniknąć nieskończonych iteracji.

W przypadku braku możliwości uzupełnienia danych rekord jest oznaczany odpowiednią flagą i przechowywany w specjalnym wektorze.

#### Przetwarzanie danych w pętli:

Program działa w pętli, w której rekordy są przetwarzane iteracyjnie, aż do wyczerpania danych do przetworzenia.

Dla każdego rekordu:

* + - 1. Wyciągane są numery pozycji (vecPosNumbers) z zapytania SQL.
      2. Jeśli rekordy zawierają brakujące dane, są one przetwarzane za pomocą funkcji losującej kod pocztowy. Losowanie odbywa się z wykorzystaniem klasy std::discrete distribution, która korzysta z wektora probabilities.
      3. Wyniki są zapisywane w strukturze toProcess, a dodatkowe wystąpienia są rejestrowane w mapie temp occurrences.

Jeśli dane nadal są niekompletne, rekord jest ponownie przetwarzany w kolejnych iteracjach.

**Przykład działania algorytmu:** Przetwarzanie rekordu z brakującym kodem pocztowym (flaga1

== 0):

if (toProcess[num].flaga1 == 0) {

std::discrete\_distribution<> dist(probabilities.begin(), probabilities.end()); std::string pCode = codes[dist(gen)];

toProcess[num].kodPocztowy = pCode; addOccurrenceTemp(temp\_occurrences, "", pCode, 0, num);

}

Przetwarzanie rekordu z brakującym miastem (flaga1 == 5):

if (toProcess[num].flaga1 == 5 && toProcess[num].proby == 0) { toProcess[num].proby += 1;

addOccurrenceTemp(temp\_occurrences, toProcess[num].miasto, "", 5, num);

}

#### Zalety podejścia:

Wykorzystanie statystyk kodów pocztowych pozwala na bardziej realistyczne uzupełnianie braku- jących danych.

Iteracyjne przetwarzanie i śledzenie liczby prób dla każdego rekordu minimalizuje ryzyko błędów i nieskończonych pętli.

Program jest skalowalny i może obsługiwać duże zbiory danych dzięki wykorzystaniu wielowątko- wości i mechanizmu std::discrete distribution.

# Struktura kodu

## Główne komponenty

Kod aplikacji jest podzielony na kilka głównych komponentów:

#### Struktury danych:

* Address – struktura przechowująca dane adresowe, takie jak ulica, kod pocztowy, miasto, województwo, współrzędne geograficzne, flagi przetwarzania oraz inne szczegóły,
* ThreadSafeDeque – bezpieczna wątkowo kolejka FIFO, używana do przechowywania i prze- twarzania zapytań SQL w środowisku wielowątkowym.

#### Funkcje:

* addOccurrence – dodaje informacje o wystąpieniu adresu do mapy occurrences,
* getWojewodztwoMapa – określa województwo na podstawie kodu pocztowego,
* url encode – koduje ciąg znaków w formacie URL, aby przygotować go do wysyłania w zapytaniach HTTP,
* perform requests – obsługuje zapytania HTTP w celu pobrania brakujących danych adre- sowych,
* perform random – iteracyjnie przetwarza dane z kolejki workers data, uzupełniając brakujące informacje na podstawie statystyk i zapytań SQL,
* saveToCSV – zapisuje przetworzone dane do pliku CSV w określonym formacie,
* removeWord, removeWordsWithDot, removeAfterSlash – funkcje pomocnicze do czyszczenia i przetwarzania danych tekstowych,
* create nonblocking socket – tworzy nieblokujące gniazdo sieciowe dla zapytań HTTP,
* addOccurrenceTemp – dodaje tymczasowe wystąpienie adresu do lokalnej mapy w trakcie przetwarzania.

#### Zmienne globalne:

* toProcess – wektor przechowujący dane do przetworzenia,
* postalCodeCount – mapa przechowująca liczbę wystąpień kodów pocztowych w danych wej- ściowych,
* occurrences – mapa przechowująca dane o wystąpieniach adresów do dalszego przetwarzania,
* codes – wektor przechowujący unikalne kody pocztowe,
* probabilities – wektor przechowujący prawdopodobieństwa wystąpień kodów pocztowych, obliczone na podstawie danych wejściowych,
* workers data – wątkowo bezpieczna kolejka, zawierająca zapytania SQL do przetwarzania przez wątki,
* dataToCSV – wektor przechowujący dane gotowe do zapisu w pliku CSV.

**Wątki:**

* Aplikacja wykorzystuje NUM THREADS równoległych wątków do przetwarzania danych, co po- zwala na efektywne wykorzystanie zasobów obliczeniowych.

## Przebieg programu

#### Inicjalizacja:

Program rozpoczyna od zainicjalizowania zmiennych globalnych, takich jak postalCodeCount, occurrences, toProcess oraz innych struktur danych.

Dane wejściowe są wczytywane z pliku CSV (Szkody2.csv) za pomocą klasy csvstream.

Każdy rekord jest analizowany i klasyfikowany na podstawie kompletności danych (np. brak kodu pocztowego, brak miasta).

#### Analiza kodów pocztowych:

Program zlicza wystąpienia kodów pocztowych i oblicza ich prawdopodobieństwa, które są prze- chowywane w wektorze probabilities.

Na podstawie tych prawdopodobieństw brakujące kody pocztowe są losowane przy użyciu klasy

std::discrete distribution.

#### Przetwarzanie danych:

Dane są podzielone na zakresy i rozdzielane pomiędzy wątki.

Każdy wątek wykonuje zapytania HTTP lub SQL w celu uzupełnienia brakujących informacji, takich jak współrzędne geograficzne, kod pocztowy czy miasto.

W przypadku niepowodzenia dane są ponownie przetwarzane w kolejnych iteracjach.

#### Zapis wyników:

Po zakończeniu przetwarzania dane są zapisywane do pliku CSV w określonym formacie, zawiera- jącym m.in. kod pocztowy, miasto, województwo, współrzędne geograficzne i inne szczegóły.

## Kluczowe fragmenty kodu

**Struktura** Address**:** struct Address

{

std::string lp; std::string ulica; std::string kodPocztowy; std::string miasto; std::string wojewodztwo; std::string kraj; std::string lot; std::string lat;

int flaga1; int flaga2;

std::string sklejone; std::string numerUmowy; std::string dataPoczatku;

std::string dataKonca; std::string sumaUbezpieczenia; std::string odnowienia; std::string reasekuracjaO; std::string reasekuracjaF; std::string adresujedn;

int proby = 0;

bool dbProcess = false;

};

**Funkcja** addOccurrence**:**

void addOccurrence(const std::string &city, const std::string &postcode, int flag, int vecPos)

{

if (!city.empty() || !postcode.empty())

{

std::lock\_guard<std::mutex> lock(occurrencesMutex); auto &entry = occurrences[{city, postcode, flag}]; entry.first++;

entry.second += std::to\_string(vecPos) + " ";

}

}

**Funkcja** perform requests**:**

void perform\_requests(int thread\_id,

std::shared\_ptr<std::vector<std::string>> &data, std::pair<int, int> range)

{

int NUM\_REQUESTS = range.second - range.first; int epoll\_fd = epoll\_create1(0);

if (epoll\_fd == -1)

{

perror("epoll\_create1"); exit(1);

}

int sockfd = create\_nonblocking\_socket(HOST, PORT); if (sockfd == -1)

{

exit(1);

}

epoll\_event ev, events[NUM\_REQUESTS]; ev.events = EPOLLOUT | EPOLLET; ev.data.fd = sockfd;

if (epoll\_ctl(epoll\_fd, EPOLL\_CTL\_ADD, sockfd, &ev) == -1)

{

perror("epoll\_ctl: EPOLLOUT");

close(sockfd); exit(1);

}

// Obsługa zapytań HTTP i przetwarzania odpowiedzi...

}

## Efektywność i skalowalność

Program jest zoptymalizowany pod kątem wielowątkowości, co pozwala na równoległe przetwarzanie dużych zbiorów danych.

Wykorzystanie asynchronicznych zapytań HTTP i SQL minimalizuje czas oczekiwania na odpo- wiedzi.

Struktury takie jak ThreadSafeDeque oraz mechanizmy synchronizacji (np. std::mutex) zapew- niają bezpieczeństwo wątkowe.

## Główne klasy i struktury

**Struktura** Address

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | struct Address { | | |
| 2 |  | std :: string | lp; |
| 3 |  | std :: string | ulica; |
| 4 |  | std :: string | kod Pocztowy ; |
| 5 |  | std :: string | miasto ; |
| 6 |  | std :: string | wojewodztwo ; |
| 7 |  | std :: string | kraj; |
| 8 |  | std :: string | lot; |
| 9 |  | std :: string | lat; |
| 10 |  | int flaga1 ; |  |
| 11 |  | int flaga2 ; |  |
| 12 |  | std :: string | sklejone; |
| 13 |  | std :: string | numerUmowy ; |
| 14 |  | std :: string | dataPoczatku ; |
| 15 |  | std :: string | dataKonca; |
| 16 |  | std :: string | sumaUbezpieczenia ; |
| 17 |  | std :: string | odnowienia ; |
| 18 |  | std :: string | reasekuracjaO ; |
| 19 |  | std :: string | reasekuracjaF ; |
| 20 |  | std :: string | adresujedn ; |
| 21 |  | int proby = | 0; |
| 22 |  | bool db Proce | ss = false; |
| 23 | }; |  |  |

**Klasa** ThreadSafeDeque

Kolejka FIFO bezpieczna wątkowo, używana do przechowywania zapytań SQL.

1

class Thread SafeDeque { public:

std :: stringstream pop\_front ();

void push\_back ( const std :: stringstream & value); size\_t size () const;

private:

2

3

4

5

6

7

8

std :: deque < std :: string > data; std :: mutex mtx ;

};

9

10

# Obsługa sieci

## Tworzenie asynchronicznego gniazda

Funkcja create nonblocking socket tworzy gniazdo TCP/IP w trybie nieblokującym, używając bi- blioteki <sys/socket.h> oraz <fcntl.h>. Jest to kluczowe dla obsługi wielu połączeń jednocześnie, co pozwala na efektywne przetwarzanie zapytań HTTP w środowisku wielowątkowym.

Listing 1.1: Funkcja tworząca asynchroniczne gniazdo

int create\_nonblocking\_socket ( const char \* host , const char \* port) { struct addrinfo hints , \* res;

int sockfd ;

memset (& hints , 0 , sizeof( hints)); hints. ai\_family = AF\_UNSPEC ; hints. ai\_socktype = SOCK\_STREAM ;

if ( getaddrinfo ( host , port , & hints , & res) != 0) { perror(" getaddrinfo ");

return -1;

}

sockfd = socket( res - > ai\_family , res - > ai\_socktype , res - > ai\_protocol ); if ( sockfd == -1) {

perror(" socket"); freeaddrinfo ( res); return -1;

}

int flags = fcntl( sockfd , F\_GETFL , 0);

if ( fcntl( sockfd , F\_SETFL , flags | O\_NONBLOCK ) == -1) { perror(" fcntl");

close( sockfd ); freeaddrinfo ( res); return -1;

}

if ( connect( sockfd , res - > ai\_addr , res - > ai\_addrlen ) == -1) { if ( errno != EINPROGRESS ) {

perror(" connect"); close( sockfd ); freeaddrinfo ( res); return -1;

}

}

freeaddrinfo ( res); return sockfd ;

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

#### Opis działania:

Funkcja korzysta z getaddrinfo, aby uzyskać informacje o adresie serwera na podstawie parame- trów host i port.

Gniazdo jest tworzone za pomocą funkcji socket, a następnie ustawiane w tryb nieblokujący przy użyciu fcntl.

Połączenie jest inicjowane za pomocą funkcji connect. Jeśli połączenie nie może zostać ustanowione natychmiast, funkcja zwraca kod błędu EINPROGRESS, co jest zgodne z trybem nieblokującym.

## Obsługa zapytań HTTP

Funkcja perform requests obsługuje wysyłanie zapytań HTTP do serwera geokodowania przy użyciu mechanizmu epoll. Umożliwia to obsługę wielu zapytań jednocześnie w sposób asynchroniczny.

Listing 1.2: Funkcja obsługująca zapytania HTTP

|  |  |  |  |
| --- | --- | --- | --- |
| 1  2  3 | void perform\_requests ( int thread\_id ,  std :: shared\_ptr < std :: vector < std :: string >> & data , std :: pair <int , int > range) { | | |
| 4 |  |  |  |
| 5 |  | int NUM\_REQUESTS = range. second - range. first; |  |
| 6 |  | int epoll\_fd = epoll\_create1 (0); |  |
| 7 |  | if ( epoll\_fd == -1) { |  |
| 8 |  | perror(" epoll\_create1 "); |  |
| 9 |  | exit (1); |  |
| 10 |  | } |  |
| 11 |  |  |  |
| 12 |  | int sockfd = create\_nonblocking\_socket ( HOST , PORT ); |  |
| 13 |  | if ( sockfd == -1) { |  |
| 14 |  | exit (1); |  |
| 15 |  | } |  |
| 16 |  |  |  |
| 17 |  | epoll\_event ev , events[ NUM\_REQUESTS ]; |  |
| 18 |  | ev. events = EPOLLOUT | EPOLLET ; |  |
| 19 |  | ev. data. fd = sockfd ; |  |
| 20 |  |  |  |
| 21 |  | if ( epoll\_ctl( epoll\_fd , EPOLL\_CTL\_ADD , sockfd , & ev) == -1) { |  |
| 22 |  | perror(" epoll\_ctl: EPOLLOUT "); |  |
| 23 |  | close( sockfd ); |  |
| 24 |  | exit (1); |  |
| 25 |  | } |  |
| 26 |  |  |  |
| 27 |  | int requests\_sent = 0; |  |
| 28 |  | bool keep\_going = true; |  |
| 29 |  |  |  |
| 30 |  | while ( keep\_going ) { |  |
| 31 |  | int nfds = epoll\_wait ( epoll\_fd , events , NUM\_REQUESTS , -1); |  |
| 32 |  | if ( nfds == -1) { |  |
| 33 |  | perror(" epoll\_wait "); |  |
| 34 |  | exit (1); |  |
| 35 |  | } |  |
| 36 |  |  |  |
| 37 |  | for ( int n = 0; n < nfds; ++n) { |  |
| 38 |  | if ( events[ n]. events & EPOLLOUT && requests\_sent < NUM\_REQUESTS ) | { |
| 39 |  | // Wysy łanie zapytania HTTP |  |
| 40 |  | } else if ( events[ n]. events & EPOLLIN ) { |  |
| 41 |  | // Odbieranie odpowiedzi |  |
| 42 |  | } |  |
| 43 |  | } |  |
| 44 |  | } |  |
| 45 |  |  |  |
| 46 |  | close( sockfd ); |  |
| 47 |  | close( epoll\_fd ); |  |
| 48 | } |  |  |

#### Opis działania:

Funkcja korzysta z mechanizmu epoll, aby monitorować zdarzenia związane z gniazdem.

Zdarzenie EPOLLOUT oznacza gotowość do wysyłania danych (zapytania HTTP).

Zdarzenie EPOLLIN oznacza dostępność danych do odczytu (odpowiedź HTTP).

Mechanizm epoll pozwala na obsługę wielu zapytań jednocześnie w jednym wątku, co zwiększa wydajność.

## Kodowanie adresów URL

Funkcja url encode konwertuje ciąg znaków na format zgodny z URL, zamieniając niedozwolone znaki na ich reprezentację procentową (%XX).

Listing 1.3: Funkcja kodująca adres URL

std :: string url\_encode ( const std :: string & sSrc) { const char DEC 2HEX [16 + 1] = " 0123456789 ABCDEF ";

const unsigned char \* pSrc = ( const unsigned char \*) sSrc. c\_str (); const int SRC\_LEN = sSrc. length ();

unsigned char \* const p Start = new unsigned char[ SRC\_LEN \* 3]; unsigned char \* pEnd = p Start;

const unsigned char \* const SRC\_END = pSrc + SRC\_LEN ;

for (; pSrc < SRC\_END ; ++ pSrc) { if ( SAFE [\* pSrc ])

* pEnd ++ = \* pSrc; else {
* pEnd ++ = ’%’;
* pEnd ++ = DEC 2HEX [\* pSrc >> 4];
* pEnd ++ = DEC 2HEX [\* pSrc & 0 x0F ];

}

}

std :: string sResult (( char \*) pStart , ( char \*) pEnd ); delete [] p Start;

return sResult;

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

#### Opis działania:

Funkcja iteruje po znakach ciągu wejściowego sSrc.

Jeśli znak jest zgodny z formatem URL (np. litery, cyfry), jest kopiowany bez zmian.

Niedozwolone znaki są zamieniane na ich reprezentację procentową (%XX), gdzie XX to wartość szesnastkowa znaku.

## Efektywność i zalety

Mechanizm epoll pozwala na asynchroniczne przetwarzanie wielu zapytań HTTP w jednym wątku, co minimalizuje opóźnienia wynikające z oczekiwania na odpowiedzi sieciowe.

Kodowanie URL zapewnia poprawność zapytań HTTP, umożliwiając wysyłanie danych zawierają- cych znaki specjalne.

Ustawienie gniazda w trybie nieblokującym pozwala na wykonywanie innych operacji w trakcie oczekiwania na zdarzenia sieciowe.

# Format danych

## Dane wejściowe

Plik CSV wejściowy powinien zawierać dane adresowe oraz dodatkowe informacje, które są przetwarzane w aplikacji. Struktura pliku wejściowego powinna być zgodna z poniższymi kolumnami:

Lp – numer porządkowy rekordu,

NumerUmowy – numer identyfikujący umowę,

DataPoczatku – data rozpoczęcia umowy,

DataKonca – data zakończenia umowy,

SumaUbezpieczenia – wartość sumy ubezpieczenia,

Odnowienia – liczba odnowień umowy,

Ulica – nazwa ulicy w adresie,

KodPocztowy – kod pocztowy,

Miasto – nazwa miejscowości,

Wojewodztwo – województwo, w którym znajduje się adres,

Kraj – nazwa kraju,

ReasekuracjaO – informacje o reasekuracji obligatoryjnej,

ReasekuracjaF – informacje o reasekuracji fakultatywnej.

Listing 1.4: Przykładowy plik CSV wejściowy

Lp; NumerUmowy ; Data Poczatku ; DataKonca ; Suma Ubezpieczenia ; Odnowienia ; Ulica; Kod Pocztowy ;

*‹→* Miasto ; Wojewodztwo ; Kraj; Reasekuracja O ; Reasekuracja F

1;12345;2023 -01 -01;2023 -12 -31;100000;1; Kwiatowa 10;00 -001; Warszawa ; Mazowieckie ; Polska ;

*‹→* Tak ; Nie

2;12346;2023 -02 -01;2023 -12 -31;150000;0; Lipowa 5;30 -001; Krak ów; Mał opolskie ; Polska ; Nie; Tak

1

#### Przykład zawartości pliku wejściowego:

2

3

## Dane wyjściowe

Po przetworzeniu przez aplikację, dane wejściowe są wzbogacane o dodatkowe informacje uzyskane w wyniku geokodowania oraz przetwarzania. Plik wynikowy zawiera następujące kolumny:

Lp – numer porządkowy rekordu (zgodny z danymi wejściowymi),

DataPoczatku, DataKonca, SumaUbezpieczenia, Odnowienia, Ulica, KodPocztowy, Miasto, Wojewodztwo, Kraj, ReasekuracjaO, ReasekuracjaF – dane wejściowe bez zmian,

Szerokosc – szerokość geograficzna uzyskana w wyniku geokodowania,

Dlugosc – długość geograficzna uzyskana w wyniku geokodowania,

Adresujedn – pełny adres w formacie tekstowym (np. Kwiatowa 10, Warszawa, Polska),

Flaga 1 – flaga wskazująca typ przetwarzania (np. brak danych, brak kodu pocztowego, brak miasta itp.),

Flaga 2 – flaga wskazująca liczbę prób przetwarzania (np. ilość zapytań do serwera geokodowania),

Nr Wojewodztwa – numer województwa przypisany na podstawie kodu pocztowego.

Listing 1.5: Przykładowy plik CSV wynikowy

Lp; DataPoczatku ; DataKonca ; SumaUbezpieczenia ; Odnowienia ; Ulica; Kod Pocztowy ; Miasto ;

*‹→* Wojewodztwo ; Kraj; ReasekuracjaO ; ReasekuracjaF ; Szerokosc; Dlugosc; Flaga\_1 ; Flaga\_2 ;

*‹→* Nr\_Wojewodztwa

1;2023 -01 -01;2023 -12 -31;100000;1; Kwiatowa 10;00 -001; Warszawa; Mazowieckie ; Polska; Tak ; Nie

*‹→* ;52 .2297 ;21 .0122 ;1 ;0 ;7

2;2023 -02 -01;2023 -12 -31;150000;0; Lipowa 5;30 -001; Krak ów; Małopolskie; Polska; Nie; Tak

*‹→* ;50 .0647 ;19 .9450 ;1 ;0 ;6

1

2

3

## Flagi przetwarzania

W aplikacji wykorzystano dwa typy flag, które są zapisywane w danych wynikowych:

Flaga 1 – określa typ brakujących danych lub metodę przetwarzania:

* 0 – brak danych adresowych (ulica, kod pocztowy, miasto),
* 1 – pełne dane adresowe (ulica, kod pocztowy, miasto),
* 2 – brak kodu pocztowego, ale dostępne miasto i ulica,
* 3 – brak miasta, ale dostępne kod pocztowy i ulica,
* 4 – ulica i miasto są takie same,
* 5 – brak ulicy, ale dostępne miasto i kod pocztowy,
* 6 – brak miasta, dostępny tylko kod pocztowy.

Flaga 2 – liczba prób przetwarzania rekordu w przypadku błędów lub braków w danych.

## Numer województwa

Numer województwa (Nr Wojewodztwa) jest przypisywany na podstawie kodu pocztowego zgodnie z mapą województw:

Mazowieckie – kody od 00-001 do 09-999 (Nr Wojewodztwa = 1),

Małopolskie – kody od 30-001 do 34-999 (Nr Wojewodztwa = 2),

Śląskie – kody od 40-001 do 44-999 (Nr Wojewodztwa = 3),

...

Pełna mapa województw została zaimplementowana w funkcji getWojewodztwoMapa.