1. Wstęp

Tematem naszej prezentacji jest projekt dotyczący „Monitorowania jakości powietrza z wykorzystaniem przetwarzania i analizy dużych zbiorów danych”.

W dobie rosnącej świadomości ekologicznej oraz potrzeby monitorowania środowiska, stworzyliśmy zaawansowany system do monitorowania jakości powietrza. System ten integruje dane z publicznych źródeł, takich jak Główny Inspektorat Ochrony Środowiska (GIOŚ), oraz pomiary dostarczane przez użytkowników. Celem jest dostarczenie narzędzia umożliwiającego analizę i wizualizację danych dotyczących zanieczyszczeń atmosferycznych w czasie rzeczywistym.

Realizacja projektu została podzielona na trzy kluczowe elementy: backend, baza danych oraz frontend, których omówienie nastąpi za chwilę.

2. Cel projektu i zakres

Głównym celem projektu było stworzenie systemu umożliwiającego efektywne gromadzenie, przetwarzanie oraz analizę danych o jakości powietrza z wielu źródeł w czasie rzeczywistym. Dane pochodzą z publicznych źródeł, takich jak GIOŚ (Główny Inspektorat Ochrony Środowiska), ale również od indywidualnych użytkowników. Kluczowe elementy projektu to:

- Zbieranie danych z API GIOŚ oraz od użytkowników.
- Backend wykorzystujący FastAPI.
- Baza danych InfluxDB zoptymalizowana dla szeregów czasowych.
- Wizualizacja danych w aplikacji frontendowej wykorzystującej Streamlit.

3. Architektura systemu

Nasz system został oparty na modelu klient-serwer:

- **Backend (FastAPI)** - aplikacja napisana w Pythonie, asynchroniczne API RESTful do obsługi danych.
- **Baza danych (InfluxDB)** - optymalna do przechowywania danych czasowych.
- **Frontend (Streamlit)** - interaktywne wizualizacje i intuicyjny interfejs użytkownika.

4. Backend - FastAPI

Backend odpowiada za pobieranie i przechowywanie danych:

- Zaimplementowany w Pythonie, podzielony na moduły zapewniające czytelność oraz możliwość łatwego rozwoju.
- Asynchroniczne pobieranie danych z API GIOŚ znacząco zwiększa wydajność i szybkość działania.
- Automatyczne harmonogramowanie pozwala na aktualizację danych co godzinę.

Backend został zaimplementowany z wykorzystaniem FastAPI, nowoczesnego frameworka do tworzenia interfejsów API w języku Python. FastAPI jest znane z wysokiej wydajności oraz wsparcia dla asynchronicznych operacji, co jest kluczowe przy obsłudze dużych zbiorów danych w czasie rzeczywistym.

4.1. Struktura Backend

Struktura backendu została podzielona na moduły, co ułatwia zarządzanie kodem oraz jego skalowalność:

- **main.py**: główny plik aplikacji, inicjalizuje aplikację FastAPI i rejestruje wszystkie trasy (endpoints),
- **database.py** obsługuje połączenie z bazą danych InfluxDB, zawiera funkcje do zapisywania i pobierania danych,
- **gios_api.py** odpowiada za komunikację z API GIOŚ, pobieranie danych o jakości powietrza,
- **scheduler.py** zarządza harmonogramem zadań, takich jak cykliczne pobieranie danych z GIOŚ,
- **models.py** definiuje modele danych za pomocą Pydantic, co umożliwia walidację i serializację danych.,
- **utils.py** zawiera funkcje pomocnicze, np. do formatowania dat.

4.2. Moduł database.py – Integracja z InfluxDB

Moduł database.py obsługuje połączenie z InfluxDB, bazą danych zoptymalizowaną pod kątem przechowywania i analizy szeregów czasowych. InfluxDB została wybrana ze względu na jej zdolność do obsługi dużych ilości danych czasowych oraz wbudowane funkcje agregacji i analizy. W porównaniu do tradycyjnych relacyjnych baz danych, takich jak PostgreSQL czy MySQL, InfluxDB oferuje lepszą wydajność przy operacjach na danych czasowych. Relacyjne bazy danych często wymagają skomplikowanych

indeksów i zapytań do efektywnego przetwarzania takich danych, podczas gdy InfluxDB jest zoptymalizowana specjalnie pod kątem takich operacji.

4.3. Moduł gios_api.py – Pobieranie Danych z API GIOŚ

Moduł ten wykorzystuje asynchroniczne żądania HTTP do pobierania danych z API GIOŚ. Asynchroniczność pozwala na równoczesne pobieranie danych z wielu czujników, co znacząco przyspiesza cały proces.

4.4. Moduł scheduler.py – Harmonogramowanie Zadań

Aby zapewnić aktualność danych, moduł scheduler.py wykorzystuje bibliotekę schedule do cyklicznego uruchamiania zadań, takich jak pobieranie nowych danych z GIOŚ co godzinę.

4.5. Moduł models.py – Walidacja Danych z Pydantic

Wykorzystanie Pydantic pozwala na definiowanie schematów danych oraz ich walidację. Dzięki temu zapewniamy, że dane przychodzące i wychodzące z naszego API są zgodne z oczekiwanym formatem.

5. Dlaczego wybraliśmy InfluxDB zamiast tradycyjnej bazy relacyjnej?

Wybór InfluxDB zamiast tradycyjnej relacyjnej bazy danych w naszym projekcie monitorowania jakości powietrza był podyktowany kilkoma kluczowymi czynnikami:

1. **Optymalizacja pod kątem danych szeregów czasowych**: InfluxDB jest specjalnie zaprojektowany do obsługi danych z sygnaturą czasową, co pozwala na efektywne przechowywanie i szybkie przetwarzanie dużych ilości takich danych.
2. **Wysoka wydajność zapisu i odczytu**: Dzięki architekturze zoptymalizowanej pod kątem wysokiej przepustowości zapisu, InfluxDB radzi sobie lepiej z dużymi wolumenami danych w porównaniu do tradycyjnych baz relacyjnych.
3. **Elastyczność schematu**: InfluxDB pozwala na dynamiczne dodawanie nowych pól i tagów bez konieczności przeprowadzania skomplikowanych migracji schematu, co jest korzystne w przypadku zmieniających się danych pomiarowych.

4. **Wbudowane funkcje analizy danych czasowych**: InfluxDB oferuje funkcje takie jak agregacja, downsampling czy polityki retencji, które ułatwiają analizę i zarządzanie danymi szeregów czasowych.
5. **Efektywna kompresja danych**: Dzięki sposobowi przechowywania danych, InfluxDB umożliwia lepszą kompresję, co przekłada się na oszczędność miejsca na dysku.

Podsumowując, InfluxDB jest lepiej dostosowany do potrzeb aplikacji opartych na danych szeregów czasowych, takich jak nasz system monitorowania jakości powietrza, oferując lepszą wydajność, skalowalność i elastyczność w porównaniu do tradycyjnych relacyjnych baz danych.

6. Frontend - Streamlit

Frontend umożliwia użytkownikowi łatwe przeglądanie i analizę danych:

- Wizualizacja danych na mapie oraz wykresach.
- Dynamiczny interfejs, intuicyjna obsługa oraz elastyczna agregacja danych.

7. Wyzwania projektowe Big Data

W projekcie napotkaliśmy na typowe problemy Big Data:

- Wolumen danych: konieczność przechowywania i efektywnego przetwarzania dużych ilości danych.
- Szybkość działania (Velocity): konieczność szybkiej aktualizacji i agregacji danych.
- Zróżnicowanie danych (Variety) oraz ich wiarygodność (Veracity).
- Generowanie wartości (Value) poprzez praktyczne wizualizacje.

8. Podsumowanie

Podsumowując, zastosowanie InfluxDB, FastAPI oraz Streamlit pozwoliło nam stworzyć efektywny, skalowalny oraz przyjazny użytkownikowi system analizy jakości powietrza. Rozwiązanie jest gotowe do dalszego rozwoju oraz adaptacji w kolejnych projektach Big Data.

Dziękuję za uwagę. Teraz odpowiem na ewentualne pytania.