# Konspekt Projektu: Analiza Wpływu Ruchu Drogowego na Jakość Powietrza w Warszawie

Michał Iwaniuk
Mateusz Jarosz
Bartosz Jezierski

9 grudnia 2025

## Spis treści

# 1 Wprowadzenie

## 1.1 Dane

- **API ZTM**: wykrywanie korków na podstawie niskiej prędkości autobusów w centrum.
- **API Open-Meteo**: analiza warunków pogodowych (deszcz, niska temperatura, wysokie stężenie PM2.5).
- **API Twitter**: analiza tweetów pod kątem słów kluczowych o negatywnym zabarwieniu (np. złość, zmęczenie, "korki", "smog").

## 1.2 Cel biznesowy

**System Dynamicznej Reklamy**
Automatyczne uruchamianie agresywnych kampanii reklamowych "na pocieszenie" na ekranach w autobusach lub w aplikacjach użytkowników znajdujących się w dotkniętych strefach.
Sprzedaż emocjonalna: użytkownik jest zmęczony, marznie i stoi w korku — reklama oferuje natychmiastowe rozwiązanie.

## 1.3 Przykład: Eskapizm Natychmiastowy ("Zabierz mnie stąd")

**Reklama biura podróży, tanich linii lotniczych:**
*„W Warszawie szaro i 4°C. Na Teneryfie teraz 26°C i słońce. Loty od 200 zł."*

# 2 Wymagania

| Wymaganie (Definicja) | Implementacja | Referencja w Raporcie |
|---|---|---|
| Pozyskanie danych z co najmniej dwóch źródeł | Wybrane źródła:<br>• 1. API ZTM Warszawa<br>• 2. API Open-Meteo<br>• 3. API Twitter | Sekcja 3 (Źródła Danych) |
| Zaprojektowanie i wykonanie sposobu składowania danych (np. format, tabele Apache Hive) | | |
| Zaprojektowanie i oprogramowanie komponentów do przetwarzania danych (konwersja, filtrowanie, analiza) | | |
| Automatyzacja przepływu danych (co najmniej Apache NiFi) | | |
| Składowanie danych (co najmniej Apache Hadoop i/lub Apache Hive) | | |
| Składowanie danych (co najmniej jedna wybrana platforma NoSQL, np. HBase) | | |
| Wsadowa analiza danych (co najmniej Apache Spark) | | |
| Utrzymanie kodu w repozytorium GIT (np. GitHub) | | |
| Przygotowanie testów funkcjonalnych | | |

Tabela 1: Tabela wymagań projektu (do uzupełniania).

# 3 Źródła Danych

## 3.1 API ZTM Warszawa

**Dostępność Danych:** Publiczne API Gminy Miejskiej Warszawa, dostępne pod adresem https://api.um.warszawa.pl/. Wymagany jest klucz API.

**Format:** JSON.

**Zawartość:** Dane w czasie rzeczywistym (aktualizacja 1 na minute) o lokalizacji pojazdów publicznych (autobusy, tramwaje) - szerokość i długość geograficzna, numer linii, numer taborowy, numer brygady pojazdu, czas zarejestrowania pozycji.

**Przykład Pozyskania:** (Przykład dla lokalizacji wszystkich autobusów)

```
# Zapytanie o pozycje wszystkich autobusów (type=1)
curl -X GET "https://api.um.warszawa.pl/api/action/busestrams_get/ \
?resource_id=f2e5503e-927d-4ad3-9500-4ab9e55deb59 \
&type=1&apikey=TWOJ_KLUCZ_API"
```

### 3.2 API Open-Meteo Weather Forecast

**Dostępność Danych:** Publiczne API, darmowe do użytku niekomercyjnego. Dostępne pod adresem https://open-meteo.com. Nie wymaga klucza API.

**Format:** JSON.

**Zawartość:** Dane meteorologiczne (prognozy oraz dane historyczne/bieżące) niezbędne do określenia "komfortu"przebywania na zewnątrz. Kluczowe atrybuty to: temperatura powietrza (2 m nad ziemią), opady deszczu (mm), zachmurzenie (%) oraz kod pogody (WMO code).

**Przykład Pozyskania:** (Przykład pobrania danych godzinowych dla Warszawy obejmujących temperaturę, deszcz i zachmurzenie)

```
# Zapytanie o temperaturę, deszcz i zachmurzenie (historia + bieżące)
curl -X GET "https://api.open-meteo.com/v1/forecast?latitude=52.2297&longitude=21.0122 \
&hourly=temperature_2m,rain,cloud_cover,weather_code \
&start_date=2025-11-14&end_date=2025-11-17"
```

### 3.3 API Twitter (twitterapi.io)

**Dostępność Danych:** Komercyjne API (wrapper dla serwisu X/Twitter), dostępne pod adresem https://twitterapi.io/. Wymagany jest klucz API.

**Format:** JSON.

**Zawartość:** Dane tekstowe (tweety) wyszukiwane na podstawie słów kluczowych o negatywnym zabarwieniu emocjonalnym (np. "zmęczony", "korek", "smog", "zimno") w określonej lokalizacji. Odpowiedź API zawiera treść wpisu, identyfikator autora, czas utworzenia oraz metadane geolokalizacyjne. Dane te posłużą do oceny nastroju społecznego w danym oknie czasowym.

**Przykład Pozyskania:** (Przykład wyszukiwania tweetów o "zmęczeniu", "korkach"lub "smogu"w promieniu 20 km od Warszawy)

```
import requests

url = "https://api.twitterapi.io/twitter/tweet/advanced_search"

querystring = {"queryType":"Latest","query":"(zmeczony OR korek OR smog OR zimno) near:Warsaw

headers = {"X-API-Key": "API_KEY"}

response = requests.get(url, headers=headers, params=querystring)

print(response.json())
```

# 4 Hadoop Batch Layer

Poniżej zdefiniowano docelowy format składowania danych w Batch Layer (HDFS) w konwencji CSV, po wstępnym przetworzeniu.

## 4.1 Dane ZTM (Lokalizacje)

**Atrybuty (Kolumny):**

- `timestamp_utc` (String: Czas zdarzenia z pojazdu)
- `linia` (String: Numer linii)
- `pojazd_id` (String: Numer taborowy)
- `lat` (Double: Szerokość geograficzna)
- `lon` (Double: Długość geograficzna)
- `data_pozyskania` (String: Data w formacie YYYY-MM-DD)

**Przykładowy plik CSV (2 wpisy):**

```
timestamp_utc,linia,pojazd_id,lat,lon,data_pozyskania
"2025-11-17T14:30:05Z","180","3451",52.2497,21.0122,"2025-11-17"
"2025-11-17T14:30:07Z","503","2211",52.2230,21.0080,"2025-11-17"
```

## 4.2 Dane Open-Meteo (Warunki Pogodowe)

**Atrybuty (Kolumny):**

- `timestamp_godzinowy` (String: Czas pomiaru ISO 8601)
- `lat` (Double: Szerokość geograficzna)
- `lon` (Double: Długość geograficzna)
- `temperatura_2m` (Double: Temperatura w °C)
- `deszcz` (Double: Opady w mm)
- `zachmurzenie` (Integer: Procent pokrycia nieba chmurami)
- `data_pozyskania` (String: Data w formacie YYYY-MM-DD)

**Przykładowy plik CSV (2 wpisy):**

```
timestamp_godzinowy,lat,lon,temperatura_2m,deszcz,zachmurzenie,data_pozyskania
"2025-11-17T14:00",52.2297,21.0122,4.5,0.0,85,"2025-11-17"
"2025-11-17T15:00",52.2297,21.0122,3.8,1.2,100,"2025-11-17"
```

### 4.3 Dane Twitter (Sentyment i Opinie)

**Atrybuty (Kolumny):**

- `tweet_id` (String: Unikalny identyfikator wpisu)
- `created_at` (String: Czas utworzenia tweeta w formacie ISO 8601)
- `author_name` (String: Nazwa użytkownika)
- `text` (String: Pełna treść tweeta - kluczowa dla analizy NLP/sentymentu)
- `sentiment` (Int: Binarna klasyfikacja sentymentu na podstawie tekstu)
- `hashtags` (String: Lista hashtagów, oddzielona przecinkami)
- `data_pozyskania` (String: Data w formacie YYYY-MM-DD, partycja HDFS)

**Przykładowy plik CSV (2 wpisy):**

```
tweet_id,created_at,author_name,text,sentiment,hashtags,data_pozyskania
"1858123456789","2025-11-17T08:15:00Z","JanKowalski","Stoję w korku już godzinę, \
nienawidzę tego miasta. Zimno i szaro.", 1, "#korek #warszawa","2025-11-17"
"1858987654321","2025-11-17T08:20:45Z","AnnaNowak","Smog taki, że nie da się oddychać. \
Mam dość, chcę słońca!", 1, "#smog #depresja","2025-11-17"
```

## 5 Stos Technologiczny i Architektura

W projekcie wykorzystana zostanie architektura Lambda, zapewniająca zarówno archiwizację danych historycznych, jak i podejmowanie decyzji w czasie zbliżonym do rzeczywistego. Poniżej przedstawiono wybrane technologie oraz uzasadnienie ich użycia.

### 5.1 Apache NiFi (Ingestion Layer)

Apache NiFi posłuży do automatyzacji pobierania danych z trzech niezależnych źródeł API (ZTM, Open-Meteo, Twitter). Jest to narzędzie, które pozwoli na cykliczne odpytywanie endpointów HTTP, wstępną walidację poprawności JSON-ów oraz ich wstępne transformacje.

### 5.2 Apache Kafka (Buffering Layer)

Apache Kafka zostanie wykorzystana jako bufor pośredniczący między NiFi a warstwą przetwarzania. Utworzone zostaną tematy dla każdego źródła danych (np. `ztm_traffic`, `weather_conditions`, `social_mood`). Użycie Kafki pozwala na odseparowanie procesu pobierania danych od ich analizy, gwarantując, że w przypadku spowolnienia przetwarzania żadne dane nie zostaną utracone.

### 5.3 Apache Spark (Processing Layer)

Apache Spark będzie stanowił główny silnik analityczny projektu. Jego zadaniem będzie łączenie strumieni danych w czasie rzeczywistym (np. łączenie informacji o deszczu z informacją o zatorze drogowym w tym samym oknie czasowym). Spark posłuży również do transformacji surowych danych do formatów wymaganych przez warstwy składowania.

### 5.4 Apache Hadoop HDFS & Apache Hive (Batch Layer)

HDFS wraz z nakładką Apache Hive będzie pełnił rolę magazynu danych historycznych.

- **Rola:** Składowanie wszystkich surowych danych pozyskanych z API w formacie ORC oraz udostępnianie ich w formie tabelarycznej (SQL).
- **Cel biznesowy:** Umożliwienie późniejszej analizy skuteczności algorytmów oraz generowanie raportów historycznych, np. zestawienia dni, w których warunki do wyświetlania reklam były spełnione najczęściej.

### 5.5 Apache HBase (Serving Layer)

Apache HBase zostanie wykorzystana jako warstwa serwująca wyniki.

- **Rola:** Przechowywanie aktualnego "stanu" poszczególnych dzielnic Warszawy (np. Klucz: `Centrum`, Wartości: `Poziom_Korka: Wysoki, Pogoda: Deszcz, Sentyment: Negatywny, Decyzja_Reklamowa: TAK`).
- **Cel biznesowy:** Zapewnienie możliwości natychmiastowego odczytu dla systemów wyświetlających reklamy w autobusach, które muszą w ułamku sekundy wiedzieć, jaką treść zaprezentować pasażerom.

## 6 Przykładowe Testy Funkcjonalne

Poniżej przedstawiono zestaw testów weryfikujących poprawność działania poszczególnych warstw systemu, ze szczególnym uwzględnieniem składowania danych w HDFS oraz serwowania decyzji reklamowych w HBase.

### 6.1 Test 1: Weryfikacja struktury katalogów i uprawnień w HDFS

**Cel:** Potwierdzenie, że Apache NiFi poprawnie tworzy strukturę katalogów dla poszczególnych źródeł danych (partitioning) i nadaje odpowiednie uprawnienia. **Komenda:**

```
hdfs dfs -ls -R /user/project_ads/data/
```

**Oczekiwany wynik:** Widoczna hierarchia katalogów z podziałem na źródła i daty, np.:

```
drwxr-xr-x ... /user/project_ads/data/weather/date=2025-11-17
drwxr-xr-x ... /user/project_ads/data/twitter/date=2025-11-17
drwxr-xr-x ... /user/project_ads/data/ztm/date=2025-11-17
```

### 6.2 Test 2: Weryfikacja zawartości i formatu danych (Weather API)

**Cel:** Sprawdzenie, czy dane pogodowe są poprawnie parsowane z JSON do formatu CSV i czy zawierają kluczowe atrybuty (temperatura, deszcz) niezbędne do podjęcia decyzji reklamowej. **Komenda:**

```
hdfs dfs -cat /user/project_ads/data/weather/date=2025-11-17/part-00001.csv | head -n 5
```

**Oczekiwany wynik:** Wyświetlenie rekordów zgodnych ze schematem (timestamp, geo, temp, deszcz, zachmurzenie), np.:

```
2025-11-17T14:00,52.2297,21.0122,4.5,0.0,85,"2025-11-17"
2025-11-17T15:00,52.2297,21.0122,3.8,1.2,100,"2025-11-17"
```

### 6.3 Test 3: Weryfikacja Logiki Biznesowej i Scoringu (HBase)

**Cel:** Weryfikacja, czy moduł analityczny (Apache Spark) poprawnie skorelował trzy strumienie danych: warunki pogodowe, sentyment tweetów oraz **płynność ruchu (ZTM)**. Test sprawdza obliczenie "Wskaźnika Dyskomfortu" (ang. *Discomfort Score*) – oczekujemy, że wykrycie korka w połączeniu z deszczem i negatywnymi wpisami da bardzo wysoki wynik, uruchamiając flagę reklamową. **Komenda (HBase Shell):**

```
get 'ad_decisions', 'Warszawa_Centrum'
```

**Oczekiwany wynik:** Rekord zawierający wyliczone miary, w tym status ruchu drogowego:

```
COLUMN                  CELL
 metrics:score          timestamp=1731945600000, value=95
 metrics:weather_cond   timestamp=1731945600000, value=Rain_Cold
 metrics:sentiment      timestamp=1731945600000, value=NEGATIVE
 metrics:traffic_status timestamp=1731945600000, value=CONGESTION
 status:show_ad         timestamp=1731945600000, value=TRUE
```