Dokumentacja wstępna projektu z Big Data

Sebastian Trojan
01171271@pw.edu.pl
320664

Wiktor Woźniak
01171277@pw.edu.pl
320670

Mateusz Nizwantowski
01161932@pw.edu.pl
313839

10 stycznia 2025

Spis treści

1 Wstęp

1.1 Co planujemy zrobić

Chcemy stworzyć *proof of concept (POC)* systemu do przetwarzania danych opóźnień warszawskiej komunikacji miejskiej. Dlaczego tylko POC a nie pełne rozwiązanie?

Cel jest taki, aby można było wszystkie wymagane serwisy odpalić na jednym urządzeniu, co gryzie się z definicją systemów Big Data. Planujemy aby każdy serwis był wewnątrz *dockerowego* kontenera, a połączymy je za pomocą *docker-compose*. Dzięki temu łatwo będzie odpalić nasze rozwiązania na dowolnej maszynie, przetestować je, wejść w interakcję z systemem, a nawet używać to co przygotowujemy jako tymczasowe rozwiązanie produkcyjne. Jesteśmy jednak świadomi, że to co planujemy stworzyć nie będzie skalowalne. Oczywiście, wiele z komponentów które skonfigurujemy można by użyć w finalnym rozwiązaniu korzystającym z dobrodziejstw środowiska rozproszonego, co więcej wydaje nam się, że relatywnie niewiele zmian było by potrzebnych, aby takie finalne rozwiązanie otrzymać. Najważniejszą zmianą była by wymiana narzędzia *docker-compose* na *kubernetes* lub podobny odpowiednik dostarczany przez operatorów chmurowych. Pozwalało by to na komunikację pomiędzy maszynami oraz na automatyczne skalowanie komponentów najbardziej obciążonych. Według nas jest to jednak poza zakresem tego projektu, który naszym zdaniem i tak jest już rozbudowany.

1.2 Cele

Co do samego systemu przetwarzania danych, ma on umożliwić przetwarzanie w czasie rzeczywistym z małym opóźnieniem. Potencjalny użytkownik ma mieć możliwość wybrania numeru linii i sprawdzenia na mapce gdzie znajdują się autobusy tej linii. Co więcej może sprawdzić także jak sytuacja wyglądała na przykład wczoraj albo tydzień temu. Oprócz tego rozwiązanie ma zawierać klaster Spark (sztuczny bo na jednej maszynie) z 3 nodes, umożliwiający przetwarzanie zgromadzonych danych historycznych.

Jednym z celów (mniej oficjalnych) jest nasza chęć analizy danych opóźnień w połączeniu z pogodą i korkami. Nie znaleźliśmy nigdzie gotowej tabelki choćby z opóźnieniami, jest tylko API, więc stworzymy takie dane aby następnie móc poddać je naszym analizom. Dla naszej trójki komunikacja jest bliska sercu, a że lubimy pogrzebać w danych, to wykorzystamy ten projekt jako możliwość stworzenia odpowiedniej bazy, która będzie nam to umożliwiać.

Oczywistym celem tego projektu jest nauczenie się technologii Big Data. Co więcej chcemy stworzyć rozwiązanie, które da się odtworzyć na innych maszynach, nie tylko na naszych komputerach, nie spędzając nad tym zadaniem ogromu czasu. Daje to także potencjał do uruchomienia aplikacji w środowisku chmuro-

wym. Dzięki temu projektowi chcemy udoskonalić umiejętności pracy w grupie i zarządzanie projektami.

1.3 Oczekiwane benefity

Do tego podpunktu można podejść na wiele sposobów. Gdyby takie narzędzie, które planujemy stworzyć, istniało dało by to większą przejrzystość spółki ZTM. Istnieje szansa, że mogłoby to przykuć uwagę społeczeństwa do wyzwania jakim jest planowanie tras komunikacji miejskiej oraz na czym te wyzwania polegają. Użytkownicy komunikacji mogli by zobaczyć gdzie aktualnie znajdują się ich autobusy oraz lepiej zrozumieć czemu się spóźniają. My jako użytkownicy komunikacji miejskiej jesteśmy ciekawi, na przykład jaki odsetek autobusów kursujących po Warszawie jest spóźnionych o więcej niż 4 minuty. Dane które planujemy zbierać dają możliwość odpowiedzi na takie pytania. Posiadanie takich danych daje możliwość zbudowania jakiegoś modelu opóźnień, a w oparciu o tą wiedzę - predykcji opóźnień.

Z drugiej strony dla osób planujących trasy, takie dane sprawiły by, że byli by oni w stanie lepiej optymalizować istniejące trasy, minimalizować opóźnienia i lokalizować tak zwane wąskie gardła, czyli odcinki tras, na których dochodzi powstawania znaczących opóźnień.

Finalnie liczymy, że jako studenci nauczymy się skutecznie konfigurować i wykorzystywać, narzędzia Big Data. Nie ma wątpliwości, że jako ludzie z roku na rok tworzymy coraz więcej danych, tak więc używane przez nas technologie będą tylko zyskiwać na popularności. Umiejętności sprawnego poruszania się w rozwiązaniach klasy Big Data, będą bardzo cenne w naszych karierach.

1.4 Istniejące podobne rozwiązania

Co ciekawe, nie znaleźliśmy rozwiązań, które spełniają nałożone przez nas wymagania. Jesteśmy przekonani, że Zarząd Transportu Miejskiego (ZTM) posiada infrastrukturę do przechowywania danych i ich analizy, jednak (co nas nie dziwi) udostępnia tylko surowe dane.

Istnieją strony i aplikacje takie jak:

- MobileMPK
- JakDojadę
- CzyNaCzas
- Jedzie.pl
- RealBus

Jednak podobnie jak API miasta stołecznego Warszawa, pozwalają tylko na przeglądanie danych w czasie rzeczywistym. Zatem z wysokim prawdopodobieństwem nie przechowują danych historycznych, tylko co jakiś czas wysyłają zapytanie do API i otrzymane dane przechowują w *Redisie* lub innym podobnym narzędziu. Oczywiście, żadne z wyżej wymienionych narzędzi nie pozwala na analizę danych. Jeżeli chcę się takową wykonać trzeba dane uzyskać samemu.

Rysunek 1: Fine, we'll do it ourselves

2 Źródła danych

2.1 Dane pogodowe

Te dane są najbardziej powszechne. Korzystamy z MeteoSource. Strona pozwala na 400 darmowych zapytań dziennie. Posiada ona rozbudowane dane dla danych miejsc, więc bez problemu można uzyskać dane dla Warszawy. Posiada wiele informacji pogodowych, a także przewidywane prognozy pogody na kolejne dni. Więcej szczegółów dotyczących wybranych danych przedstawimy w dalszej części dokumentu.

2.2 Dane dotyczące korków

Bezkonkurencyjnie, Google jest najlepszym dostawcą danych dotyczących ruchu na drogach w Polsce. Tak się składa, że żaden z nas nie ma doświadczenia z API Googla. Nie powinno to być problemem, gdyż jest ono uznawane za przyjazne dla programistów. Już wstępnie zapoznaliśmy się z obszerną dokumentacją. Ponadto co miesiąc jest pula 300$ darmowych zapytań. Będziemy eksperymentować z częstotliwością zapytań, aby możliwie nie przekroczyć tego pułapu.

2.3 Dane opóźnień komunikacji publicznej

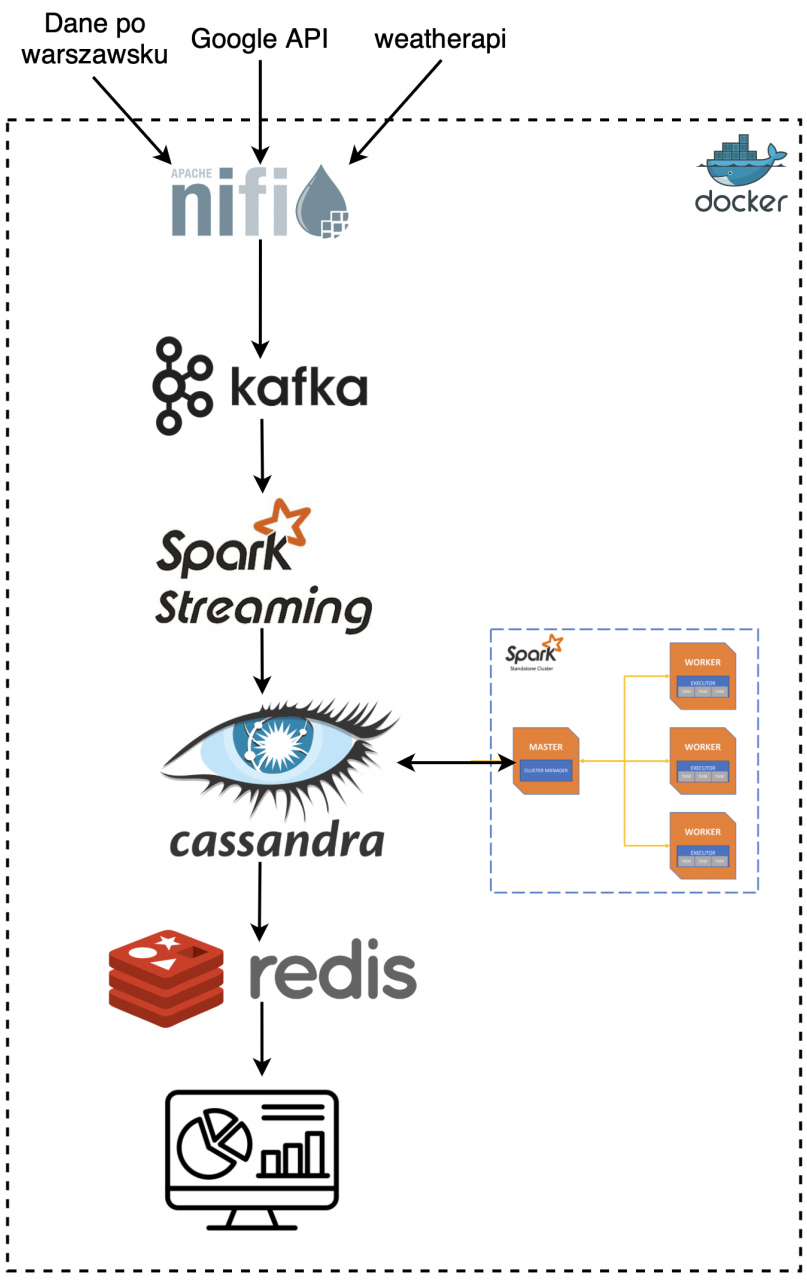
Dane o opóźnieniach będziemy brać z zbiorów udostępnianych w ramach inicjatywy "Dane po warszawsku". Jest to platforma, która działa od 2015. Jej celem jest umożliwienie mieszkańcom tworzenia rozwiązań, które przyczynią się do poprawy ich codziennego życia oraz funkcjonowania samorządu.

“Zgodnie ze światowymi trendami udostępniamy dane w sposób otwarty, nie pobieramy za nie opłat i nie pytamy, kto je pobiera. Staramy się, żeby były możliwie pełne, aktualne i uporządkowane, aby korzystanie z nich było jak najwygodniejsze, ale przede wszystkim aby publikowane dane mogły być paliwem i inspiracją do tworzenia rzeczywistych i innowacyjnych rozwiązań i aplikacji ułatwiających życie mieszkańców naszego miasta“ – wyjaśnia Tadeusz Osowski, dyr. Biura Cyfryzacji w stołecznym Ratuszu.

3 Używane technologie

- Git
- GitHub
- LaTeX
- Docker
- Docker Compose
- NiFi
- Kafka
- Spark
- Cassandra

4 Architektura



Rysunek 2: Diagram architektury naszego rozwiązania

4.1 NiFi

Do łączenia się z naszymi źródłami danych wykorzystamy platformę Apache NiFi. Jest ona świetnym narzędziem do "ciągnięcia" danych z uwagi na zaimplementowane funkcjonalności. Nie planujemy na tym etapie wykonywać przetwarzania danych, a jeżeli, to będą to bardzo lekkie transformacje. Dodatkowo będziemy wykorzystywać *Nifi Registry* do przechowywania, jako kod w naszym repozytorium, stworzonych przepływów danych. Jak zaznaczone na diagramie NiFi będzie wpychać dane do Kafki.

4.2 Kafka

Żaden z nas nie pracował wcześniej z Kafką, więc ciężko jest nam powiedzieć coś więcej. Wiemy, że Kafka dobrze współpracuje z NiFi i Sparkiem. Będzie ona stanowić buffer w procesowaniu (który raczej nie będzie potrzebny). Chcemy jednak przetestować czy Kafka doda lub odejmie opóźnienie w procesowaniu i jak wpłynie na niezawodność rozwiązania. Nie jesteśmy jeszcze pewni czy każde źródło danych będzie miało swój osobny *topic* czy wszystko będziemy wrzucać do jednego potoku.

4.3 Spark

4.3.1 Część analityczna

Planujemy, aby ta część była klastrem złożonym z 3 kontenerów: 1 master i 2 slave. Oczywiście, nie ma to dużo sensu w przypadku tego jak będziemy uruchamiać całą infrastrukturę (na jednej maszynie) jednak chodzi tu o nauczenie się, a niekoniecznie o sensowność całego procesu. W tym module będzie możliwość analizowania danych z bazy danych i wykorzystywania całego potencjału Sparka, w celu znalezienia ciekawych zależności w danych.

4.3.2 Część przetwarzająca dane w czasie rzeczywistym

Podobnie jak powyżej, ten krok nie ma sensu w przypadku uruchamiania wszystkich serwisów na jednej maszynie. W rozproszonym przypadku, logika jest taka, że nie chcemy, aby część analityczna i przetwarzająca dane wchodziły w inferencje ze sobą. W przypadku wykonywania obciążających obliczeń, część analityczna mogłaby znacząco spowolnić obliczenia wykonywane w czasie rzeczywistym i pogorszyć doświadczenie użytkowników. W tym module będziemy przetwarzać dane i zapisywać do bazy danych.

4.4 Cassandra

Wiele debatowaliśmy na temat wyboru właściwej bazy danych. Finalnie wybór padł na Cassandrę. Innymi opcjami było trzymanie danych w HDFS, Hbase lub nawet w Kafce. Cassandra nie powinna mieć problemu z taką ilością danych. Obsłuży relatywnie dużą liczbę zapytań jednocześnie. Cassandra nie potrzebuje

innych usług poza podstawowymi, takimi jak Java. Natomiast Hbase korzysta z HDFS. Dodatkowo Cassandra posiada język CQL (odpowiednik SQL) który pozwala na interakcje z bazą. Tymczasem używając Hbase musielibyśmy korzystać z dodatkowej biblioteki w Pythonie.

4.5 Redis (opcjonalne)

Ten krok zdecydowanie nie będzie potrzebny, jednak lubimy tworzyć systemy wydajne, więc się zdecydowaliśmy na dodanie go. Zakładając, że nasze rozwiązania stało by się popularne, obciążenie na części od NiFi do Cassandry nie zmieniło by się. Użytkownicy obciążaliby bazę danych. Cassandra ma możliwości przechowywania najnowszych danych w cache, jednak nie są one tak rozbudowane jak w platformie Redis. Dlatego, gdy nasz sprzęt nie dawałby rady na obsługę użytkowników, na przykład w godzinach szczytu, pewnym rozwiązaniem byłoby umieszczenie pomiędzy bazą, a użytkownikiem tak zwanej *in-memory database*. Wówczas przechowywalibyśmy w Redisie odpowiedzi na najczęstsze zapytania, co znacząco odciążyłoby główną bazę danych.

4.6 Platforma do wyświetlania danych (opcjonalne)

Podczas gdy, według nas ten krok nie wnosi nic do procesowania danych o dużym wolumenie, to jeżeli pozostanie nam trochę czasu, to poszukamy i zintegrujemy jakieś narzędzie do wyświetlania danych, na przykład jako stronę webową. Nie będzie to nic wyrafinowanego, najprawdopodobniej jakaś mapka w której będzie można wybrać datę, godzinę i numer linii autobusu, a następnie zobaczyć gdzie on jest, gdzie być powinien i ile ma opóźnienia.

5 Planowane funkcjonalności

5.1 Oglądanie opóźnień w czasie rzeczywistym na mapce

Jest to funkcja, którą na ogół umożliwiają inne platformy zajmujące się tą tematyką. Aby móc to zrobić nie musimy przechowywać danych. Wystarczy tylko pobrać dane z API i przetworzyć je. Jednak, aby ujednolicić nasze rozwiązanie, najpierw będziemy je przetwarzać i zapisywać, a następnie odczytywać z bazy.

5.2 Przeglądanie opóźnień w historii

Jest to funkcjonalność, której nie udało nam się znaleźć w podobnych serwisach, a w naszej opinii byłaby użyteczna. Powiedzmy, że chcemy gdzieś dojechać w godzinach szczytu, jednak nie wiemy ile nam to zajmie. Jako próbę oszacowania można by spojrzeć, ile zajął podobny kurs do pokonania w zeszłym tygodniu. Kolejną zaletą takiej funkcji jest przejrzystość systemu komunikacji, która potencjalnie polepszy zrozumienie przez użytkowników jak on działa oraz dlaczego autobusy się spóźniają.

5.3 Możliwość wykonywania analiz

Szczególnie dla nas, studentów Inżynierii i Analizy Danych, ale pewnie i dla osób planujących trasy autobusów, ciekawa będzie funkcjonalność analizy opóźnień przy użyciu klastra, na którym postawiony jest Spark. Podczas gdy normalni użytkownicy będą mieli ograniczone możliwości interakcji z bazą danych, analitycy będą mieli dostęp do wszystkich gromadzonych danych. Pozwoli to na znajdywanie zależności pomiędzy zbieranymi parametrami.

Zamysł jest taki, aby po odkryciu interesujących zależności, na przykład pomiędzy pogodą, a opóźnieniami, stworzyć wizualizację i przygotować proces przetwarzania danych. Celem tego jest, aby zwykli użytkownicy mogli śledzić i obserwować dane zależności w czasie rzeczywistym (jeżeli to możliwe) a jeżeli nie to z akceptowalnym opóźnieniem, na przykład godzinowym albo dziennym. Przykładem takiego procesu może być analiza średniego opóźnienia komunikacji w ciągu dnia, które następnie po ustaleniu czy jest to wystarczająco ciekawe i odkrywcze, będzie udostępnione dla użytkowników i podliczane pod koniec dnia.

5.4 Przewidywanie opóźnień (opcjonalne)

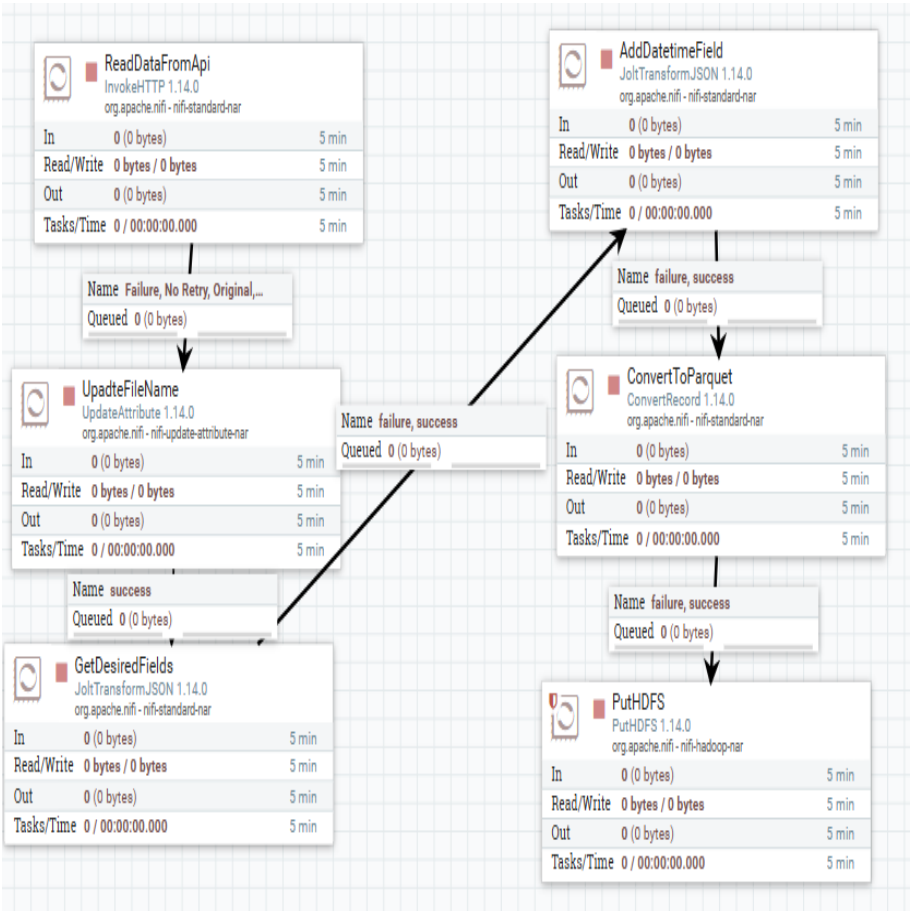
Spark umożliwia procesowanie nowych danych przy użyciu modeli uczenia maszynowego. Jeżeli starczy nam czasu, będziemy chcieli zbudować (choćby prymitywny) mechanizm przewidywania opóźnień. Nie chodzi nam o to, aby był on dokładny (choć było by to bonusem) lecz, aby nauczyć się tworzyć takie rozwiązania od strony technicznej oraz zbadać ich wydajność.

6 Dane pogodowe

6.1 Schemat w NiFi

Na rysunku (3) przedstawiamy schemat przetwarzania danych pogodowych za pomocą NiFi. W pierwszej kolejności, za pomocą InvokeHTTP, ładujemy surowe dane w postaci JSON z pogodowego API. Zapytania wykonujemy co 10 minut. Następnie zmieniamy nazwę pliku tak, aby wszystkie pliki zawierały wspólny prefiks, a następnie datę i czas z dokładnością do minut. Za pomocą dwóch procesorów JoltTransformJson wybieramy interesujące nas dane pogodowe, głównie odrzucając przewidywaną pogodę. Dodajemy także jedno dodatkowe pole, które wskazuje na datę oraz godzinę, wraz z minutami, przetwarzania. Jesteśmy świadomi, że obie te operacje można zrobić za pomocą jednego procesora, jednakże zdecydowaliśmy o rozbiciu ze względu na czytelność. Dodanie dwóch procesorów zamiast jednego niesie akceptowalny dla nas narzut czasowy, który jest minimalny. Ostatnim krokiem jest zmiana formatu na Parquet oraz zapisanie danych do HDFS. W wyniku otrzymujemy tabelę o następujących własnościach:

- **temperature** - temperatura z dokładnością do jednego miejsca po przecinku w stopniach Celsjusza. Mierzona jest ona na wysokości 2 metrów nad ziemią.
- **wind_speed** - prędkość wiatru z dokładnością do jednego miejsca po przecinku w $\frac{m}{s}$. Mierzony jest on 10 metrów nad ziemią.
- **wind_angle** - kąt wiania wiatru, gdzie 180° oznacza kierunek południowy.
- **wind_dir** - kierunek wiatru oznaczony za pomocą odpowiednich kierunków geograficznych.
- **cloud_cover** - procent nieba pokrytego chmurami.
- **processingTime** - moment w czasie otrzymania/przetworzenia danych.

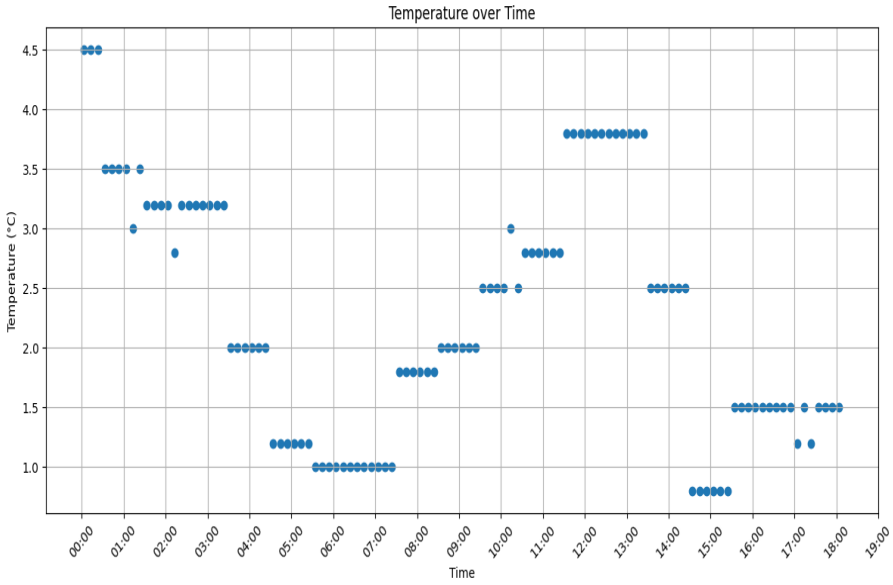


Rysunek 3: Schemat NiFi dla pogody

7 Analiza danych w spark

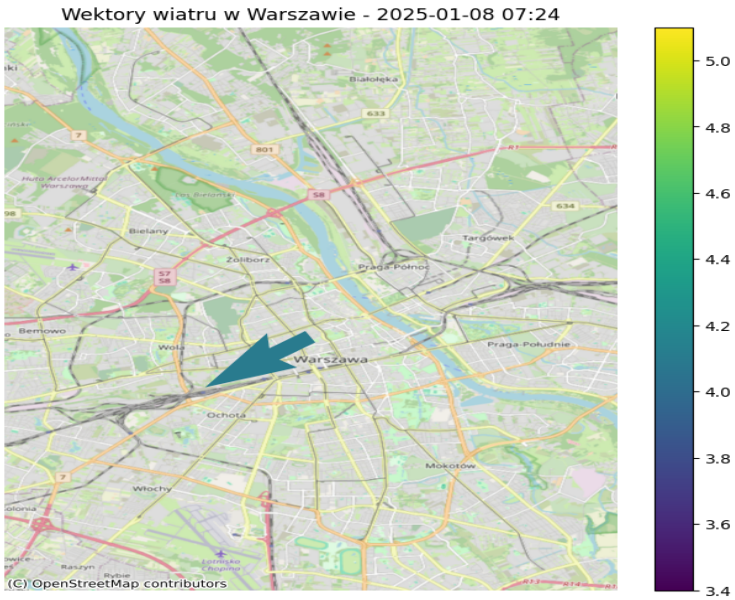
Na podstawie danych pogodowych stworzyliśmy dwie proste wizualizacje.

Pierwsza z wizualizacji widoczna na rysunku (4), przedstawia zmianę temperatury w czasie. Prezentujemy dane co 10 minut. Taka wizualizacja mogłaby zostać wykorzystywana do np. optymalizacji systemu grzania lub chłodzenia w komunikacji miejskiej.



Rysunek 4: Wykres temperatury od czasu w dniu 08.01.2025

Drugą wizualizacją jest mapa (5), pokazująca moc oraz kierunek wiatru. Za pomocą suwaka można ustawiać odpowiednią godzinę, z dokładnością do 10 minut. W raporcie przedstawiamy widok dla pojedynczego momentu w czasie. Strzałka wskazuje kierunek wiatru, a jej kolor wskazuje na jego prędkość. Taka wizualizacja, zrobiona w kontekście całego roku, mogłaby posłużyć do określenia struktury wiat przystankowych. Gdyby na przykład okazało się, że przez dużą część roku wiatr wieje z jednego kierunku, to można by było zabudować wiaty z przodu na odpowiednich przystankach, na których pasażerowie są najbardziej narażeni na wiatr.



Rysunek 5: Mapa wiatru w dniu 08.01.2025