Warsaw Public Transport Delays

Big Data Project

Team: QuestionMark

Team members:

- Marceli Korbin,
- Adam Narożniak,
- Bartosz Rożek,
- Szymon Szmajdziński

Project deliverables

High-level description of project deliverables from the user perspective, including benefits for the users from finishing the project.

- Providing user with the estimated tram commute time for the travel from point A to point B in Warsaw (raw text)
    - Point A and Point B are tram stops
- Providing user with the estimated delay
    - (optional) giving more probabilistic outlook of the delay and visualizing it
- Access to the historical searches for the commutes (no logging, it’s a shared history search for all users)
- Warning user against the most delayed routes this day (e.g., tram 4 is delayed by x on average)

Data sources

The results of the preliminary investigation of data sources to be used in the project, including the content and volume of available data resources

Traffic Data

URL: https://api.um.warszawa.pl/

Type: Stream + we will collect from the stream data to make them historical

Granularity: 10 seconds

Description: This data source provides live data on the location of the trams, the line and brigade number and the time

Timetable, Tram Stops, Tram Lines Data

URL: https://api.um.warszawa.pl/

Type: Static (changing only when the timetable changes)

Graduality: day (no need to fetch it more frequently)

Weather Data

URL: https://www.weatherapi.com/, https://open-meteo.com/

Type: Stream and historical (Weather API: back to 2010, Open Meteo: back to 1940)

Granularity:

- Weather API: 15 minutes for stream data, daily/hourly for historical data
- Open Meteo: hourly

Description: different types of data, including real-time weather, historical weather and future forecasts

Stream/Batch Train Location Data

We will store the raw outputs from which are: Lat, Lon, Time, Lines, Brigade. At this point we haven't conducted the experiments and decided on how the data needs to be processed to create the meaningful model. Here are some ideas on the potentially needed preprocessing: determining the distance between the expected tram stop and the current location (Euclidean or Manhattan distance).

The batch training should require the same preprocessing (unless the data in the preprocessed format is stored).

Timetable, Tram Stops, Tram Lines, Weather Data

A separate database will be kept for storing such data; an additional one might be set up to keep the historical data.

Model

Difference in time vs schedule ~ weather conditions + time of the day + day of the week + repairs + line number + tram stop

Architecture & Data procession

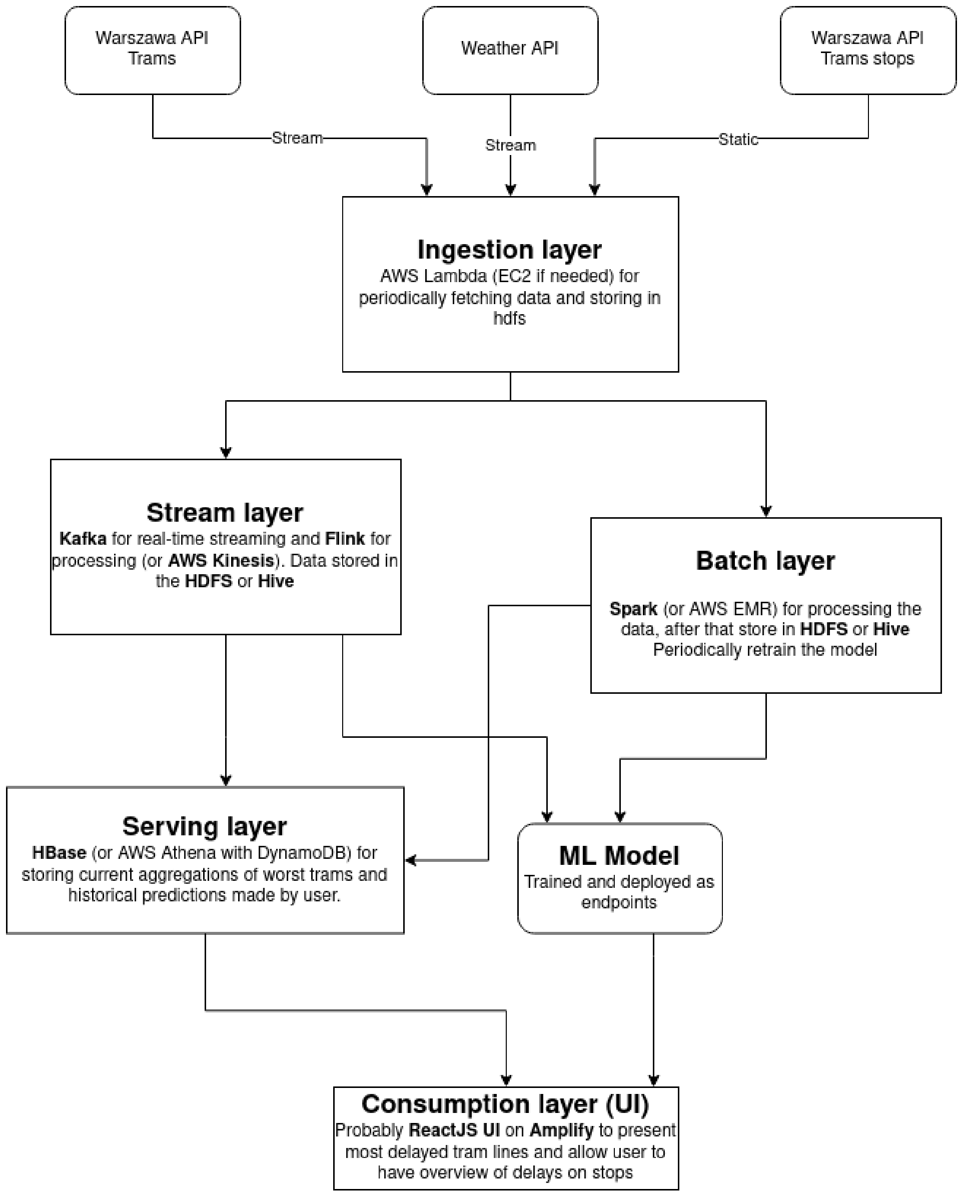
The proposed big data architecture integrates data from two sources: Warszawa API for tram data, a Weather API, along with static tram stops information. In the ingestion layer, data is periodically fetched and stored in HDFS using AWS Lambda or EC2 instances.

Real-time streaming and processing are handled in the stream layer, where Kafka enables real-time data ingestion, and Flink facilitates real-time processing. In the preprocessing step, data types can be assigned to the elements of the JSON. Data can be stored in HDFS or Hive for future analysis.

The batch layer utilizes Spark or AWS EMR for processing batch data, storing it in HDFS or Hive. Here, all the useful information about the tram stops can be combined. For example, the name of the tram stops and the distance in meters from the beginning of the route to this tram stop can be merged. Additionally in this layer we will regularly retrain the machine learning model, which will be deployed as an endpoint.

The serving layer employs HBase or AWS Athena with DynamoDB for storing current aggregations of the worst tram delays and historical predictions made by users.

Finally, the processed information is presented in the consumption layer through a ReactJS UI hosted on AWS Amplify, allowing users to access real-time data about tram delays and providing an overview of delays at specific tram stops. This architecture combines real-time and batch processing, data storage, and machine learning to deliver valuable insights and interactive user experiences.


Warszawa API Trams
Weather API
Warszawa API Trams stops
Stream
Stream
Static
Ingestion layer
AWS Lambda (EC2 if needed) for periodically fetching data and storing in hdfs
Stream layer
Kafka for real-time streaming and Flink for processing (or AWS Kinesis). Data stored in the HDFS or Hive
Batch layer
Spark (or AWS EMR) for processing the data, after that store in HDFS or Hive
Periodically retrain the model
Serving layer
HBase (or AWS Athena with DynamoDB) for storing current aggregations of worst trams and historical predictions made by user.
ML Model
Trained and deployed as endpoints
Consumption layer (UI)
Probably ReactJS UI on Amplify to present most delayed tram lines and allow user to have overview of delays on stops

Team members

The list of team members and preliminary allocation of tasks to team members

- Marceli Korbin
  - Managing data source's APIs
  - Ingestion layer – setting AWS Lambda for fetching data
  - Batch Layer – setting Spark for processing the data and HDFS/Hive for storing it
- Adam Narożniak
  - Batch Layer – setting Spark for processing the data and HDFS/Hive for storing it
  - Stream layer – setting Apache Kafka for real-time streaming

    - Serving layer – setting HBase for storing data
- Bartosz Rożek
    - ML Model – designing the module with the model
    - Consumption layer – UI
    - Serving layer – setting HBase for storing data
- Szymon Szmajdziński
    - Batch Layer – setting Spark for processing the data and HDFS/Hive for storing it
    - ML Model – designing the module with the model
    - Stream layer – setting Apache Kafka for real-time streaming