TECHNOLOGIES AND ENGINEERING

Journal homepage: https://technologies-engineering.com.ua/en
Vol. 26, No. 4, 2025

Received: 17.04.2025
Revised: 15.08.2025
Accepted: 17.09.2025

UDC 004.9

DOI: 10.30857/2786-5371.2025.4.5

Conceptual model of a web-based traffic flow management system in an urban environment based on microservice architecture

Andrii Roskladka[*]

Doctor of Economic Sciences, Professor
State University of Trade and Economics
02156, 19 Kyoto Str., Kyiv, Ukraine
https://orcid.org/0000-0002-1297-377X

Yevhenii Postrelko

Postgraduate Student
State University of Trade and Economics
02156, 19 Kyoto Str., Kyiv, Ukraine
https://orcid.org/0000-0001-9730-450X

Abstract. The relevance of the research arises from the growing load on urban transport infrastructure, the need for digital transformation of mobility management, and the implementation of intelligent data analytics technologies operating in real time. Such a system enables prompt responses to changes in traffic conditions, improves the efficiency of the route network, and enhances the overall quality of urban transportation services. The purpose of the study was to design the architecture of a web system that ensures the integration of data from city application programming interfaces, their automated processing, analytical interpretation, visualisation of results, and centralised configuration management. The methodological basis of the research included approaches of systems analysis, data flow modelling, the construction of entity-relationship diagrams, and the use of architectural design patterns to represent the structural and functional interaction of components. The article presented the development of a conceptual model of a web-based system for managing traffic flows in an urban environment, designed according to the principles of microservice architecture. The research outcome was the creation of a structural model of the system comprising five interrelated microservices: integration with city application programming interfaces (Node.js), data processing and aggregation (Python, Redis Streams), an analytical module with machine learning algorithms (Scikit-learn), a visualisation module (React, Mapbox, Chart.js), and a service for access control and configuration management. The paper provided a detailed description of their interaction logic, scaling mechanisms, and reliability assurance. The practical value of the study lied in the development of a universal architectural framework for implementing systems of analytics and monitoring of urban transport, which may serve as a prototype for intelligent mobility management platforms within the Smart City concept

Keywords: distributed information services; digital modelling of urban mobility; real-time data flow analysis; web analytics of transport systems; spatio-temporal data visualisation; Smart City

Introduction

Rapid urbanisation and the saturation of transport infrastructure require cities to adopt flexible digital solutions capable of handling large-scale data streams in near real time. Traditional monolithic approaches do not provide the necessary scalability, fault tolerance, or speed of deployment. By contrast, a microservice architecture, combined with web technologies and contemporary analytics, enables the development of manageable, transparent, and reproducible systems for traffic flow management at the urban scale. In this context, articulating a concept that links data acquisition, processing, analytics, and presentation within a coherent architectural frame becomes essential for reproducibility and future scaling.

Ukrainian and international scholars are actively advancing research on web architecture, microservices, traffic flow management, visual analytics, and the Smart City

Suggested Citation:
Roskladka, A., & Postrelko, Ye. (2025). Conceptual model of a web-based traffic flow management system in an urban environment based on microservice architecture. *Technologies and Engineering,* 26(4), 54-67. doi: 10.30857/2786-5371.2025.4.5.

[*]Corresponding author

paradigm. Ukrainian researchers M. Kotenko *et al*. (2024) have systematised security practices for microservice architecture – including service mesh, mutual Transport Layer Security, Zero Trust, and Development, Security, and Operations – demonstrating their practical value for building secure, distributed web systems. Their work emphasised layered defences (isolation, Application Programming Interface (API) gateways, policy-based access) and showed how service mesh patterns reduce the attack surface and centralise traffic governance in distributed environments, which is crucial for city-scale web platforms handling personal mobility data. D. Volkov & V. Liubchenko (2024) provided a review of threats and protection strategies for microservices and identified gaps in their application; their study underscored the importance of comprehensive risk-management approaches within microservice architecture and established a methodological foundation for the secure integration of urban transport services. In particular, they highlighted the absence of universal threat models for microservices and advocate STRIDE-oriented (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege) analysed coupled with container hardening and secure inter-service communication, aligning well with transport systems that must assure integrity and availability during peak loads or incidents. Y. Matseliukh & V. Lytvyn (2024) proposed a model for analysing passenger flows in low-carbon transport within a Smart City, and outlined a sequence of data sources and open datasets that are particularly useful for urban transport visualisations. Their correlation-regression analysis connected passenger throughput with CO_2 emissions, motivating analytics modules that not only forecast flows but also assess ecological impact; this directly informs key performance indicators sets for dashboards in a traffic management web system (e.g., trip time, occupancy, emission proxies). Y. Ohonovskyi *et al*. (2023) presented a prototype of an information system for monitoring and content analysis of citizens' appeals in a Smart City context, with an emphasis on web-oriented integration of municipal services. The proposed ideas proved suitable for composing a web platform for flow management with modular integration of urban data. This citizen-feedback channel complements sensor data: incident reports, complaints, and requests create additional signals for congestion detection and service prioritisation, which a microservice platform can ingest via dedicated ingestion and natural language processing services. A. Danyliuk & O. Muliarevych (2024) examined approaches to traffic control, including Cyber-Physical Systems/Internet of Things (CPS/IoT) solutions and elements of artificial intelligence; these approaches helped to establish the algorithmic context for the analytical microservice of the web system. Their surveys of adaptive signal control and parameter sets (speed, density, volume) provided a catalogue of measurable features that can be operationalised in forecasting pipelines, while the CPS perspective clarifies integration patterns with roadside devices through secure APIs. Building on systems analysis, O. Barabash *et al*. (2021) decomposed the urban transport system, developed a road-safety profile, and outlined criteria for flow efficiency, while separately highlighting the role of, and risks posed by, telecommunication threats. Their "traffic safety profile" concept suggested a morphology of factors (network design, intersection typology, pedestrian streams) that can be encoded as domain ontologies and used for rule-based diagnostics in tandem with predictive models. Y. Fornalchyk & V. Hilevych (2023) demonstrated that increasing motorisation correlates with higher accident rates, noise, and emissions, and substantiated a transition towards an artificial-intelligence-enabled transport network with centralised monitoring as a viable direction for mitigation. This evidence strengthened the case for city-wide coordination layers in the architecture (control plane plus monitoring plane) to mitigate externalities of motorisation through proactive forecasting and targeted control policies.

Y. Yu *et al*. (2025) systematised visual analytics techniques for heterogeneous mobility data, highlighting, among the principal visual objects, origin-destination maps, a range of progressive analytics methods, and inclusive interface designs. Their InclusiViz system combined deep models (Deep Gravity-style) with explainable analytics and what-if tooling, indicating how a traffic platform may evolve from descriptive dashboards to prescriptive, simulation-backed decision support with multi-level views (city-wide to neighbourhood). A. Wibowo *et al*. (2024) proposed the MxT model, which integrates streams from the X social network to detect obstructions and enable rapid route adjustments; their results reported over 91.6% detection accuracy and approximately a 15% reduction in travel time. This illustrated a viable "social sensing" microservice that enriches sensor feeds during disruptions (floods, protests), with grounding via ground-truth validation – an approach transferrable to European cities with similar incident typologies.

Building on the outlined problematisation, the study aimed to propose a coherent conceptual model of a web-based traffic flow management system for the urban environment, grounded in microservice architecture, which aligns data acquisition and integration with their analytical interpretation, visualisation, and the administrative management of configurations.

Materials and Methods

This article presented an original conceptual model developed within a design-science approach. The research methodology comprised several consecutive stages. Based on a review of publications and practical cases in urban mobility management, the functional zones of the system were identified: data acquisition and integration, data processing and aggregation, analytics, visualisation, and access and configuration. These zones were then used as criteria for the architectural design. The system was decomposed into five microservices that correspond to these zones, and their structure and interactions are described in the sections devoted to individual microservices. For each functional zone, a set of entity-relationship (ER), data-flow diagrams (DFD) and context-container-component-code (C4) diagrams was prepared to capture the domain structure, information flows and boundaries of microservice responsibility. The final stage was a qualitative comparison of candidate analytical tool stacks (Python, R, Excel/Power BI) using criteria such as performance, scalability and suitability for integration into a microservice architecture.

The conceptual model was developed as a sequential architectural design process with the key artefacts explicitly documented. The architectural solution followed a "context → containers → components" approach. Microservices were distinguished for integration with municipal APIs (data ingestion and dissemination), data processing and aggregation (validation, normalisation, and consolidation of indicators), analytics (calculation of indicators and support for forecasting algorithms), visualisation (maps and charts for presenting results), and access and configuration management (centralised settings and access rights). For each microservice, public interfaces and message exchanges were specified to ensure independent evolution of components and their scalability.

In constructing the conceptual model, the focus was on defining data schemas, interfaces and information flows at the architectural level. No concrete test datasets or quantitative experiments with real or synthetic records were designed within this study. In the conceptual model, a separate functional zone is formed by the results visualisation layer, which is responsible for presenting aggregated and analytical data in a clear form for different groups of users. To provide a coherent representation of the structure and information flows, a domain ER diagram was created (routes, stops, vehicles, traffic events, indicators/forecasts, users/roles, configurations), as well as DFD for levels 0-2 (the main channels of data ingestion, processing, and delivery) and a C4 diagram that shows microservice responsibility boundaries and their interactions. Together with interface specifications and data format descriptions, these diagrams enable other researchers to reproduce the proposed model under comparable conditions. The reproducibility of the solution was supported by documenting interface schemas, canonical definitions of indicators, and typical configurations, so that equivalent deployments can be implemented in different urban contexts without changing the underlying architectural principles.

Results and Discussion

Municipal API integration microservice. The system comprises distinct microservices, each responsible for a specific set of functions: integration with municipal APIs, data processing and aggregation, analytical computation, results visualisation, and configuration management. The municipal API integration microservice is intended to ingest, process, and preliminarily validate data on intra-urban traffic flows. Its primary task is envisaged to be the establishment of stable communication with external services that are expected to provide up-to-date information on public transport movements, routes, timetables, and potential delays. In the conceptual model, the microservice is designed to perform regular or event-driven requests to designated APIs, as discussed by A. Bokolo (2025), who emphasised the role of RESTful integration patterns for Smart City data exchange, ensuring the processing of received data and their transformation into an internal format suitable for further analytical interpretation.

Introducing a dedicated microservice for API integration is meant to enable high flexibility when data sources or response formats change, while minimising the impact on other components of the web system. Moreover, isolating data acquisition into an independent service is intended to enhance the solution's scalability and to simplify its maintenance; G. Şahin *et al.* (2024) outlined similar benefits of decoupled ingestion layers for maintainability and scale in municipal web systems.

In implementing the project, it is assumed that the developers will have access to open APIs of municipal transport services that provide the necessary information in a specified format. In particular, it is envisaged that the APIs support standard HTTP (HyperText Transfer Protocol) requests (GET method) and return responses in JSON (JavaScript Object Notation) containing the following key attributes: a unique vehicle identifier; geographic coordinates (latitude, longitude); current speed; route identifier; scheduled arrival time at the next stop; actual timestamp of the latest update. The Municipal API Integration microservice is positioned to play a critical role in ensuring reliable acquisition of traffic-flow data for subsequent analysis. Its principal functional components are proposed to include:

1) Initiation of requests to open municipal APIs. The microservice is expected to periodically issue HTTP requests (predominantly via the GET method) to designated municipal API endpoints to retrieve vehicle locations, routes, and timetables. A standardised REST architecture is intended to underpin this integration, consistent with contemporary Smart City practice; A. Bokolo (2025) underscored that consistent REST conventions improve interoperability and reduce coupling across civic data services.

2) Validation and verification of received data. Responses are to be checked against the expected structure, including the presence of mandatory fields (vehicle identifier, coordinates, speed, update time). In the event of malformed or incomplete records, the microservice is expected to filter them out to maintain a high level of information quality; I.M. Leghemo *et al.* (2025) drew attention to systematic quality gates for API-delivered mobility streams.

3) Transformation into an internal unified format. To ensure interoperability, data obtained from heterogeneous APIs are intended to be normalised to a predefined schema. This is meant to prevent errors during subsequent processing and visualisation.

4) Error handling and resilience. When an external API is unavailable or returns an invalid response, the microservice applies retry mechanisms with exponential backoff and emits error notifications for downstream auditing.

5) Real-time or scheduled data refresh. The microservice is designed to support both event-driven processing (e.g., upon receipt of an update signal) and periodic data collection at a configured interval, which is expected to enable optimisation of system load.

Thus, the integration microservice is expected to ensure reliable ingestion, validation, and preparation of transport data for subsequent analytical and visualisation processes within the web-based traffic flow analysis system. Implementing the municipal API integration microservice is assumed to require technologies capable of sustaining high throughput under large volumes of concurrent requests, offering flexibility in API handling, and ensuring robust data processing. Following a comparison of several technology stacks, the development would adopt Node.js with the Axios library for issuing HTTP requests. This choice is justified by Node.js's efficiency in handling asynchronous calls through its non-blocking I/O model, as highlighted by B.S. Beeraka (2025), who associated event-driven runtimes with high-frequency HTTP workloads in urban analytics, which is considered critical for tasks characterised by high-frequency data exchange.

The chosen Node.js technology is expected to enable the microservice requirements to be met effectively in terms of request processing speed, ease of API integration, and scalability under increasing load. A conceptual container diagram (Fig. 1) is intended to illustrate the architecture of the municipal API integration microservice implemented in the proposed stack. In this diagram, the microservice establishes connections to external APIs, handles requests, transforms data into the internal format, validates the retrieved information, and persists it for subsequent processing within the traffic flow analysis system.

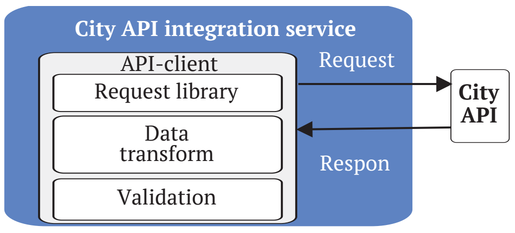

Figure 1*. Container diagram of the Municipal API Integration microservice*
Source: *designed by the authors*

In summary, the municipal API integration microservice is intended to provide a stable, configurable entry point for mobility data, decoupling external API variability from the system's internal contracts. Its design is expected to prioritise ingest throughput, schema normalisation, and explicit error-handling paths so that downstream services receive consistent, high-quality records. Operationally, the service is envisaged to support both scheduled and event-driven refresh modes, enabling cities to balance latency against load. By isolating acquisition concerns and exposing a narrow, well-typed interface, the microservice is intended to simplify maintenance, support independent scaling, and reduce the blast radius of upstream changes. Collectively, these properties are expected to improve reliability and to create a robust foundation for subsequent aggregation, analytics, and visualisation.

Data processing and aggregation microservice. The data processing and aggregation microservice is a key component that is intended to perform the initial processing of data received from the Municipal API Integration microservice. E. Puzio *et al.* (2025) described analogous staging of pre-analytics pipelines for transport feeds, which motivates the separation of ingestion and cleaning as a distinct service layer. Its primary purpose is to render the data into a structured form suitable for subsequent analysis, storage, and visualisation. This is envisaged to be achieved by filtering out malformed or incomplete records, normalising value formats (e.g., time and coordinates), consolidating data by routes, vehicles, or time intervals, and computing basic aggregates (such as average speed and mean delay). Thus, the microservice is positioned to serve as a buffer between raw, unprocessed data and the analytics modules, providing a reliable, high-quality foundation for all subsequent computations within the system.

Within this microservice, it is assumed that the input arrives as structured JSON objects from the preceding integration microservice and contains the following attributes: a unique vehicle identifier; a route identifier; the data capture time in Coordinated Universal Time (UTC); location coordinates; current speed; delay, in minutes, relative to the timetable. Based on the input data, aggregation is intended to be performed by routes; by time intervals; by geographical zones. As a result, analytical slices are expected to be produced that include, inter alia: average speed on a route over the selected period; frequency of stops or delays; number of active vehicles at a given moment. After aggregation, the data are to be stored in a standardised internal format (e.g., as JSON objects or database tables), facilitating seamless integration with subsequent system modules (analytics and visualisation). The Power BI Community (2021) outlined practical constraints of spreadsheet-centric pipelines for streaming data, and N.O. Quesado Filho *et al.* (2025) contrasted dashboard tooling with code-first stacks, which together support the decision to separate Extract-Transform-Load (ETL)-like routines from end-user visualisation layers. A comparison of these options according to the criteria of performance, flexibility, scalability and support for machine learning tools is provided in Table 1.

Table 1*. Comparison of analytical tools for implementing a web system*

Criterion	Python	R	Excel / Power BI
Computing performance	High (vectorisation, NumPy)	Average, good for statistics	Low, not suitable for large volumes
Libraries for machine learning/AI	Scikit-learn, XGBoost, Prophet	Caret, GTFSwizard, Forecast	Minimum (statistics only)

Table 1. Continued

Criterion	Python	R	Excel / Power BI
Working with spatial data	GeoPandas, Folium	SF, Leaflet, GTFSwizard	Only through external plugins
Integration into a microservice system	High (FastAPI, Flask, Celery)	Limited, harder to scale	Not intended for server systems
Learning curve	Moderate	Moderate	Low
Open code	Yes	Yes	Partially (closed formulas)

Source: *compiled by the authors*

Although R is noted to have a few quality libraries for analysing GTFS transport data (e.g., GTFSwizard), its weaker representation in microservices and web integration ecosystems is assumed to limit its use for real-world deployment of transport analytics. Using Excel/Power BI is considered justified in cases of visual data analysis for non-specialists, however, this approach does not scale, offers limited automation, and does not support the flexible application of machine learning algorithms. Python is considered to have an advantage in all key aspects: computational speed, support for machine learning libraries; integration with infrastructure components (FastAPI, Celery); flexibility of spatial analysis (GeoPandas, Shapely).

For implementing the data processing and aggregation microservice, the Python language was selected conceptually, as its libraries are intended to provide high efficiency when working with data arrays – specifically, pandas for tabular processing, NumPy for numerical operations, and datetime for time-interval manipulation. Python is chosen for its optimal balance between syntactic simplicity, a broad toolkit for data analysis, and straightforward integration with streaming solutions (e.g., Redis Streams or Apache Kafka). Compared with other languages (such as Java or Scala), Python is considered more suitable for prototyping and for flexible handling of semi-structured data, which are typical of transport analytics. K.Ntouros *et al.* (2025) emphasised the practical advantages of Python-centric data engineering for municipal platforms, a consideration reflected in the prioritisation of a lightweight but extensible stack. As the data-streaming system, Redis Streams is considered due to its lightweight nature, rapid deployment, and minimal configuration requirements. This is particularly advantageous in urban settings, where the system is expected to operate with reduced latencies and respond swiftly to incoming records. F. Yang (2025) highlighted the utility of stream abstractions that combine event buffering with time-windowed aggregation, a capability mirrored by Redis Streams for transport-data grouping.

The data processing and aggregation microservice is positioned to play a critical role in transforming raw information on traffic flows into aggregated, clean, and structured data that can be used directly for subsequent analysis and visualisation. Its architecture is intended to enable the efficient detection of anomalous values, the grouping of data by routes, vehicles, or time intervals, and the construction of core metrics – average speed, delays, and the number of active vehicles. Taken together, these design choices are expected to provide an optimal balance between performance, implementation simplicity, and adaptability to data volumes, thereby delivering a scalable foundation for further integration with the system's analytics and visualisation modules (Horokhovskyi & Oshovska 2015). The diagram (Fig. 2) was intended to depict the principal stages of transport data processing within the microservice implemented in Python, using Redis Streams as the streaming mechanism. Input data are assumed to enter Redis Streams, where they are buffered and forwarded to the processor. The workflow is expected to perform filtering, normalisation, and aggregation, followed by grouping by routes, time intervals, or geozones. The aggregated results are to be stored in the internal repository (Redis JSON) and prepared for subsequent use by other system modules.

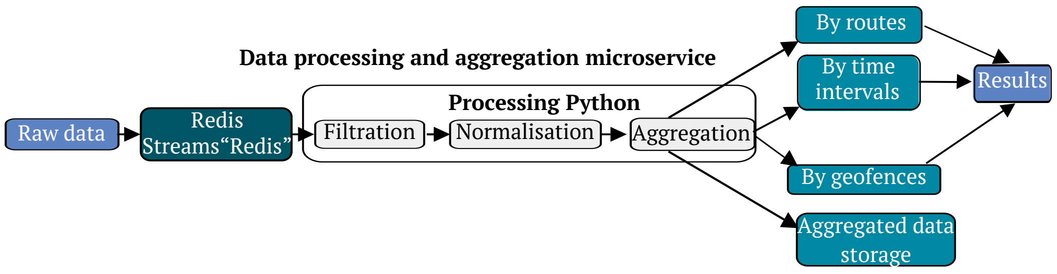

Figure 2. *Microservice architecture for data processing and aggregation*
Source: *designed by the authors*

In summary, the data processing and aggregation microservice is intended to act as the system's stabilising layer, converting heterogeneous, event-level inputs into consistent, analysis-ready datasets. Its workflow prioritises deterministic cleaning rules, schema normalisation, and configurable aggregation windows so that downstream modules receive comparable indicators across routes, times, and zones. By separating operational (near-real-time) and periodic (batch) modes, the service is expected to balance latency with completeness and to prevent contention under peak loads. The explicit generation of core metrics – such as average speed, delay distributions, and active-vehicle counts – provides a minimal yet sufficient feature set for forecasting and visualisation. Collectively, these properties are intended to reduce error propagation, simplify debugging, and create a scalable foundation for the analytical and presentation layers.

Analytics microservice for urban transport analytics and forecasting. The analytics microservice is a key component of the web-based traffic flow analysis system, as it is intended to enable the transition from aggregated data to practically meaningful insights. This component is designed to process cleaned and structured data to identify patterns, forecast problematic segments, and formulate hypotheses for improving the city's route network. In contrast to the Data Processing and Aggregation microservice, the analytics module is oriented not only toward handling current values but also toward generating conclusions that may have strategic significance for urban mobility planning. The analytics microservice is assumed to perform intelligent processing of aggregated transport data to detect anomalies, derive performance indicators, and generate forecasts of load across the city's route network. It operates conceptually on data prepared by the preceding microservices, which are expected to supply cleaned, structured time-series information on vehicle movements.

The microservice is designed to perform in-depth analytics of transport data, specifically: detecting congestion and zones of heightened traffic intensity; analysing deviations from timetables and delays in public transport; computing route efficiency metrics (e.g., load factor, average travel time); producing baseline forecasts of delays or route load for specified periods; generating a set of recommendations (hypotheses) for the optimisation of the transport network. Thus, the analytics microservice is intended to provide the intelligent data processing required for well-founded managerial decisions in urban transport. Within the analytics microservice, it is assumed that the inputs are the aggregated and normalised datasets produced by the preceding Processing and Aggregation microservice. These data are expected to be error-cleaned, structured by key parameters, and ready for analytical interpretation. A basic input structure may be represented as a table or as JSON objects with the following records: route identifier; time interval; average speed on the route for the corresponding interval; average deviation from the timetable; number of vehicles on the route; route load factor; territorial zone or city sector. All records are to be generated with a uniform temporal step, which is intended to enable the construction of time series, the application of cluster or correlation analysis, and the assessment of indicator dynamics over time.

One of the module's principal functions is envisaged to be the detection zones of heightened traffic load – that is, segments characterised by persistently low average speeds combined with a high number of vehicles. To this end, filters are intended to be applied to identify critical speed thresholds, or cluster analysis is to be employed to delineate typical traffic situations across the city. A.O. Oyenuga *et al.* (2025) described comparable urban-mobility analyses that translate such clusterings into resource-allocation signals for city operations, which informs the choice to keep the feature space compatible with route-level interventions. Another function of the microservice is proposed to be the analysis of deviations from public transport timetables. For this purpose, the microservice is expected to compare the actual and scheduled times at stops. The results are intended to enable an assessment of delay levels by day of the week and by time interval, to identify peak periods of instability, and to localise zones where timetable violations are systematic. Where additional data become available, such as weather or event calendars, these factors are intended to be incorporated as covariates in regression-style models, following practice outlined by E. Erişkin (2024) for transport-policy evaluation. A dedicated functional block is assumed to be responsible for computing route efficiency, including the calculation of average travel time, a regularity coefficient (the ratio of actual to scheduled arrivals), speed variance, and route load levels across different times of day. These indicators are intended to allow for a quantitative assessment not only of the stability but also of the predictability of public transport operations.

In subsequent iterations, the microservice is intended to incorporate a forecasting module for delays or congestion. This can be achieved by building time-series models (e.g., Seasonal AutoRegressive Integrated Moving Average – SARIMA) or by applying machine-learning methods (such as linear regression or eXtreme Gradient Boosting (XGBoost)) to predict metric values for the subsequent hours or days. N. Schetakis *et al.* (2025) discussed short-term transport forecasting pipelines where classical time-series baselines remain competitive under constrained data, which motivates the decision to stage SARIMA alongside tree-based learners. S.A. Inamdar & S.S. Kulkarni (2025) explored quantum-inspired variants for near-term improvements; in the case of this article, such methods are regarded as future options once a stable benchmark is established. The final stage of the analytics microservice is envisaged to involve generating hypotheses for the optimisation of the transport network. Drawing on analyses of congestion, delays, and related metrics, the microservice is expected to propose adjustments to the route network – for example, relocating or adding stops, modifying headways, or introducing additional services during peak hours. All hypotheses are intended to be logged and prioritised by criticality, with forwarding to an administrative console for expert review. Thus, the analytics microservice enables the transition from aggregated data to concrete managerial conclusions that may be applied both to the tactical regulation of traffic and to the strategic planning of urban mobility.

The diagram (Fig. 3) was intended to depict the structure for processing aggregated transport data within the analytics microservice. Input data in JSON or comma-separated values (CSV) format are assumed to be received from the preceding Processing and Aggregation module. In the first stage, key metrics are expected to be computed (average speed, delays, traffic intensity), after which congestion and unstable routes are intended to be identified. Where required, the forecasting module is planned to be activated, employing machine-learning or time-series algorithms. The final stage is intended to generate hypotheses for improving the route network. The results are to be passed, as structured JSON objects, to the subsequent components of the system.

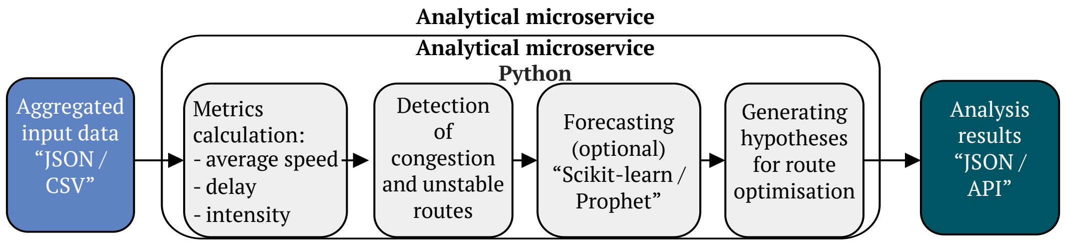
Analytical microservice
Analytical microservice
Python
Aggregated input data "JSON / CSV"
Metrics calculation:
- average speed
- delay
- intensity
Detection of congestion and unstable routes
Forecasting (optional) "Scikit-learn / Prophet"
Generating hypotheses for route optimisation
Analysis results "JSON / API"

Figure 3*. Analytical microservice architecture*

Source: *designed by the authors*

For the implementation of the analytics microservice, a technology stack is proposed that is oriented towards efficient handling of tabular and time-series data, mathematical modelling, and machine learning. Python is positioned to serve as the primary implementation language, being widely adopted in urban and transport analytics and allowing a rapid transition from prototypes to reproducible pipelines. Y. Yu *et al.* (2025) illustrated how Python ecosystems – pandas for feature engineering and scikit-learn for modelling – can underpin multi-level mobility visualisation with explainable components, which aligns with design of this study for a modular analytics layer. The suite of specialised Python libraries is intended to enable both basic statistical processing and more advanced models for analysing temporal trends and forecasting. In particular, the InclusiViz line of work by the same authors demonstrated that code-first stacks facilitate consistent metric definitions across maps and charts, an approach authors of this study planed to mirror to maintain indicator reproducibility.

In the domain of policy-facing forecasting and optimisation, E. Erişkin (2024) showed that Python pipelines can blend behavioural covariates (e.g., weather, event calendars) with machine-learning baselines for transport strategy appraisal; the feature space is therefore planned to remain extensible so that such covariates can be incorporated without altering service boundaries. For real-time or near-real-time processing, the analytics microservice is intended to interface with streaming sources (e.g., Redis Streams), or to operate on periodically updated JSON/CSV files or tables supplied by the preceding microservice. F. Yang (2025) emphasised the advantage of stream abstractions that combine event buffering with windowed aggregation; this insight motivates authors' choice to expose time-window parameters at the configuration level so that cities can tune latency versus completeness. The selected stack is expected to enable the processing of large volumes of transport data, the scaling of computations as load increases, and the seamless integration of the results with the system's visualisation and administration microservices.

To support spatial analysis, the stack is intended to include GeoPandas for topology-aware joins and Shapely for geometric operations, enabling, for instance, corridor-level congestion detection and zone-based aggregation without bespoke Geographic Information System (GIS) servers. K. Ntouros *et al.* (2025) argued for lightweight, Python-centric data engineering in municipal platforms, noting that such stacks reduce operational overhead while preserving extensibility; this perspective underlies authors' preference for a lean but composable toolchain. Where higher-throughput ingestion becomes necessary, the microservice boundary is planned to admit Kafka-style brokers, yet the baseline remains deliberately minimal to keep deployment attainable for smaller cities.

Model selection for short-term forecasting is intended to balance classical time-series methods (e.g., SARIMA) and non-linear learners (e.g., gradient-boosted trees such as XGBoost) so that the system can operate under both scarce-data and richer-feature scenarios. N. Schetakis *et al.* (2025) reported that well-tuned classical baselines remain competitive when observation windows are short or sensors are uneven, which informs authors' decision to stage SARIMA alongside machine-learning alternatives rather than replacing them outright. Exploratory tracks, including quantum-inspired variants discussed by S.A. Inamdar & S.S. Kulkarni (2025), were considered future options once deterministic benchmarks are established and validated against municipal ground truth. Taken together, the proposed Python-first stack is intended to provide a pragmatic compromise: rapid prototyping for research iterations, explicit contracts for service integration, and a clear path to scaling analytics as cities expand data coverage and latency expectations.

In summary, the analytics microservice is intended to convert cleaned, aggregated streams into decision-ready insights by combining descriptive indicators, anomaly detection, and short-term forecasting within one configurable pipeline. Its design is expected to preserve reproducibility through uniform time steps, stable metric definitions, and explicit feature schemas, while accommodating both near-real-time updates and periodic batch analyses. By emitting compact outputs – congestion flags, timetable-stability scores, route-efficiency metrics, and ranked optimisation hypotheses – the service is positioned to support both operational interventions and longer-horizon planning. Clear JSON contracts and stateless execution paths are envisaged to ease integration with visualisation and administrative layers, enabling independent scaling as data coverage grows. Collectively, these properties are intended to improve interpretability, shorten feedback loops for transport managers, and provide a robust analytical core for the overall system.

Visualisation microservice for interactive urban mobility dashboards. The visualisation microservice is intended to serve as the terminal presentation layer for all processed information in the system, transforming analytical and aggregated data into clear, interactive visual components. Its principal aim is to provide convenient, intuitive access to complex transport information for a broad range of users – from municipal analysts to ordinary citizens and transport operators. D.M. Kozachenko *et al.* (2025) argued that user-centred map interfaces improve decision latency for operational staff, which underpins the choice to emphasise rapid situational awareness over purely exploratory views. The scope of responsibility is proposed to include building interactive maps with public transport routes, congestion, and stops; displaying charts of headways, average speed, delays, and passenger volumes; generating heat maps of transport load across different times of day; presenting graphical analytics outputs in the form of diagrams, histograms, and time series.

Thanks to this microservice, users are expected to be able to review the current state of the transport situation, analyse its dynamics, compare data across periods, detect anomalies, and support well-founded decisions. Within the visualisation microservice, inputs are assumed to arrive from the analytics microservice and from the store of aggregated data. These inputs adhere to a standardised format suitable for subsequent rendering in the web interface. The data are intended to be provided as Geospatial JSON (GeoJSON) objects for mapping routes, stops, and vehicles; JSON tables or arrays for time charts, delays, and traffic intensity; heat-map arrays for constructing density layers based on coordinates and event counts; and timestamps for building dynamic visualisations. Y. Yu *et al.* (2025) highlighted that mobility dashboards benefit from multi-level drill-downs and explanations attached to visual primitives; indicator definitions and tooltips are therefore planned to be exposed that tie back to analytical computations. Y. Cao *et al.* (2025) demonstrated that heat-map overlays are effective for temporal crowding patterns, which motivates the inclusion of configurable time windows and colour scales for peak-hour exploration.

To align with digital-twin practices, B.P. Rafamatanantsoa *et al.* (2024) described City2Twin's separation of static 3D context from dynamic streams; the visual layer is planned to respect this separation by treating basemap and dynamic indicators as distinct sources, enabling smooth playback and reduced redraw overhead. H.A. Adrianto *et al.* (2024) showed that analyst-facing dashboards benefit from scenario toggles and multi-chart coordination; accordingly, the microservice is intended to provide filter panels (route, time, zone) that trigger coordinated updates across maps and charts. A. Wibowo *et al.* (2024) reported that disruption-aware overlays sourced from social streams shorten route-adjustment time in critical events; the design retains a pluggable "incident overlay" so that cities can add such feeds without altering the core renderer.

On the implementation side, the visualisation microservice is intended to be deployed as a separate front-end application that interacts with other system modules via a REST API. React is planned to be used as the primary technology, providing a component architecture with incremental UI updates. For web mapping, Leaflet or Mapbox GL JS are envisaged to serve as the cartographic engines; Leaflet offers lightweight vector overlays, while Mapbox GL enables GPU-accelerated styles and animated state transitions for moving objects. M. Laituri *et al.* (2025) drew attention to the value of real-time public dashboards during crisis response, and this is mirrored by reserving a "live mode" that prioritises update cadence and progressive rendering. For charts and heatmaps, Chart.js, D3.js, or Recharts are intended to be employed, with D3 reserved for bespoke interactions and Recharts for rapid composition in React. Where 3D context becomes a requirement, the microservice boundary is planned to admit a lightweight 3D scene (e.g., deck.gl) layered over the same data contracts to preserve consistency with 2D views.

In summary, the visualisation microservice is intended to provide a coherent, low-latency interface that turns complex spatio-temporal data into actionable views for diverse user groups. By standardising input contracts (GeoJSON, JSON tables, heat-map arrays) and separating map, chart, and filter concerns, the layer is expected to remain responsive under peak loads while preserving consistency of indicators across widgets. A focus on accessibility, progressive rendering, and clear affordances is intended to shorten time-to-insight for operational staff and to support reproducible analysis for experts. Through stateless rendering, cache-friendly assets, and explicit API boundaries, the microservice is positioned to scale independently of upstream analytics. Collectively, these properties are intended to ensure that the presentation tier strengthens decision-making without constraining future extensions such as 3D context or incident overlays.

Access and configuration management microservice. The access and configuration management microservice is intended to function as the administrative centre of the web-based traffic flow analysis system. Its primary aim is intended to be the provision of centralised control over key system parameters and the offering of basic user authentication and authorisation capabilities. This is expected to enable the delineation of access levels to analytics, the configuration of data refresh frequency, and the adaptation of the system to the needs of a specific city or transport model. The functions of this microservice are proposed to include: creating and managing API keys required for integration with municipal or third-party data sources; setting integration parameters: city, route network, update timings, filters, and time intervals; defining user roles and managing access rights; maintaining an audit log to ensure transparency of administrative actions.

Implementing this microservice is intended to ensure the scalability and flexibility of the entire system without requiring modifications to the code of the core functional microservices. This is particularly important when deploying the system across multiple cities or under varying transport parameters and integration settings. The access and configuration management microservice is envisaged to perform a crucial systemic role – providing flexible administration of all key parameters of the web system and enforcing access restrictions for different user categories. Its functionality is divided into several core blocks:

1) API key management. The system is expected to provide an interface for creating, viewing, and deactivating API keys used to integrate municipal data sources. Each key can be associated with a specific subsystem (e.g., integration with GPS trackers or municipal portals) and constrained by expiry and request-rate limits.

2) Integration parameter configuration. An administrator is expected to be able to set core configuration parameters: city, set of transport routes, source polling cadence (e.g., every five minutes), time zone, and filters by transport type or city districts. These settings are intended to be stored centrally and propagated to other microservices via a configuration API.

3) Users and access roles. A basic authentication mechanism (e.g., tokens or JSON Web Token (JWT)) and role-based authorisation are provided. Typical roles are envisaged to be provided: administrator, analyst, and guest viewer. Each role is expected to grant access to a defined set of functions; for instance, only administrators are to be allowed to alter configuration or create API keys.

4) Audit log. All configurations are intended to be recorded with timestamp, author, and action type. This is expected to enable activity monitoring and to ensure transparency of system governance.

This microservice is not intended to process analytical or cartographic data; however, it is considered critical to the continuous operation of the entire system and to its adaptation to new conditions without code changes. Its presence is expected to render the system scalable, multi-scenario, and centrally manageable from a single point. The Access and Configuration Management microservice presupposes an administrative console through which authorised users are intended to be able to configure the system, create API keys, manage data refresh frequency, and define role-based access policies. Accordingly, the primary technological requirement is the rapid implementation of CRUD functions (Create, Read, Update, and Delete) with minimal front-end development effort.

Description of the overall architecture of the web-based traffic flow management system. The adoption of a microservice architecture is intended to enable the creation of a flexible and scalable transport system in which each component is expected to fulfil a narrowly specialised function. Z. Wang *et al.* (2025) reported that microservice decompositions support city-scale forecasting around metro stations, which aligns with the intention to keep compute hotspots independently scalable. C. Campos *et al*. (2025) argued that modelling data flows at the architectural level – within oneAPI-style pipelines – improves performance and manageability in multi-component systems; this view underpins the decision to foreground flow contracts between services. Integrating the diagrams and models within a single subsection is intended to ensure logical coherence and improves the perception of the architecture as a unified information space in which components are expected to operate in an interrelated and sequential manner.

At the architectural level, the system is envisaged to comprise the following principal containers:

- Frontend (client tier) – intended to provide users with access to transport-data visualisation, interactive charts, heatmaps, and configuration interfaces.
- API Gateway/Router – intended to coordinate user requests and to route them to the appropriate microservices.
- Microservices – Municipal API Integration, Data Processing and Aggregation, Analytics, Visualisation, and Access & Configuration Management – intended to encapsulate domain-specific responsibilities.
- Database – intended to serve as a central repository for aggregated and historical data.
- Configuration/Logging Store – intended to hold parameters, API keys, and audit logs.

Each microservice is intended to be deployed independently, allowing only those components under load to be scaled. The architectural style is intended to accord with contemporary approaches to mobility management systems – particularly in smart cities – where isolated logic, distributed data, and a high degree of modularity are preferred. Z. Wang *et al.* (2025) emphasised such modular scaling in urban mobility platforms, while C. Campos *et al.* (2025) highlighted coordination of heterogeneous compute stages via explicit flow specifications. Figure 4 presented the C4 container-level diagram of the system. The diagram is intended to present the overall architecture of the web-based traffic flow management system built on a microservice model. The system is described as consisting of several independent containers (microservices), each intended to perform a clearly defined function. The user (analyst, dispatcher, or operator) is expected to interact with the system via a React-based web interface. The web interface is intended to send requests to an API Gateway, which is intended to act as a router between the client and the internal microservices. All processed information is intended to be stored in a central database (PostgreSQL), while configuration parameters – including change logs and tokens – are intended to be maintained in a separate configuration store (Redis or JSON-based storage). Beyond the structural architecture and user scenarios, an important aspect of conceptual modelling is intended to be the description of how data are transmitted and transformed within the system. For this purpose, a DFD is planned to be employed to depict, in a clear and intuitive manner, the sources, processors, stores, and directions of information flow.

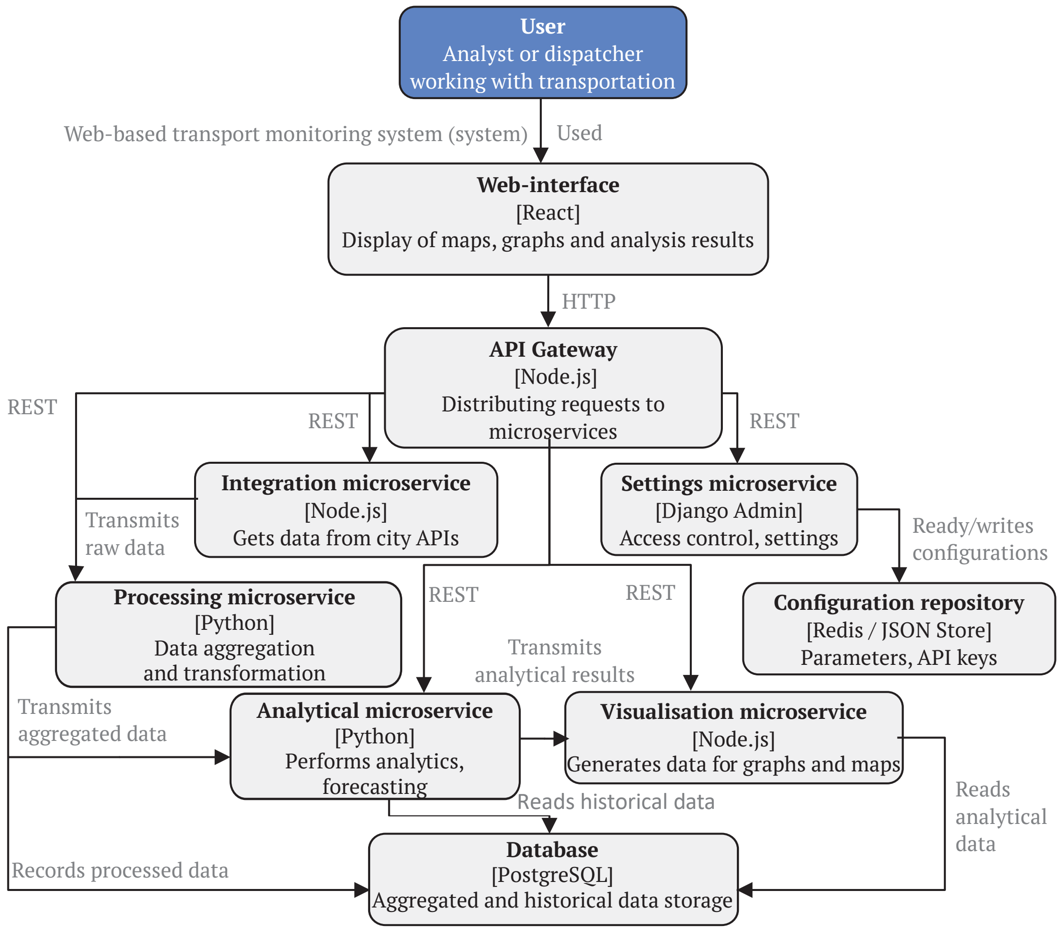

***Figure 4**. C4 container-level diagram*

***Source:** designed by the authors*

Such diagrams were particularly valuable in transport information systems where large volumes of real-time streams require structured processing; this observation supports the choice to make data-flow boundaries explicit. B.P. Rafamatanantsoa *et al.* (2024) showed, in the City2Twin context, that microservice-oriented architectures facilitate the tracing and scaling of data flows across dynamic streams and static context; the DFD is intended to mirror this separation of concerns. The components captured by a DFD are intended to include: external data sources/receivers (users, APIs); processing processes (microservices); intermediate and final data stores (database, cache); data flows between components. The DFD for the traffic flow management system is intended to be shown in Figure 5. The data-flow diagram is intended to depict the key information transformations within the system. Data are expected to arrive from external sources (municipal APIs), to be processed by the integration and aggregation microservices, to be stored in the database, to be subjected to analytical processing, and ultimately to be presented to the user. All processes are intended to be configured via a dedicated configuration microservice, which is expected to ensure flexibility, centralised parameter control, and efficient data movement. For the storage, processing, and subsequent analysis of transport data, the web system is intended to employ a centralised relational database. Its structure is intended to be modelled using an ER diagram, which is intended to visualise the core entities, their attributes, and the relationships between them. S. Batita *et al.* (2024) reported that relational, domain-oriented schemas remain effective in urban transport systems for scalability and analytical integration, while A. Mansurova *et al.* (2025) described schema designs that handle high-volume GPS events under city workloads; these findings motivate the choice of a relational backbone with explicit keys and constraints. The list of database entities is intended to be presented in Table 2.

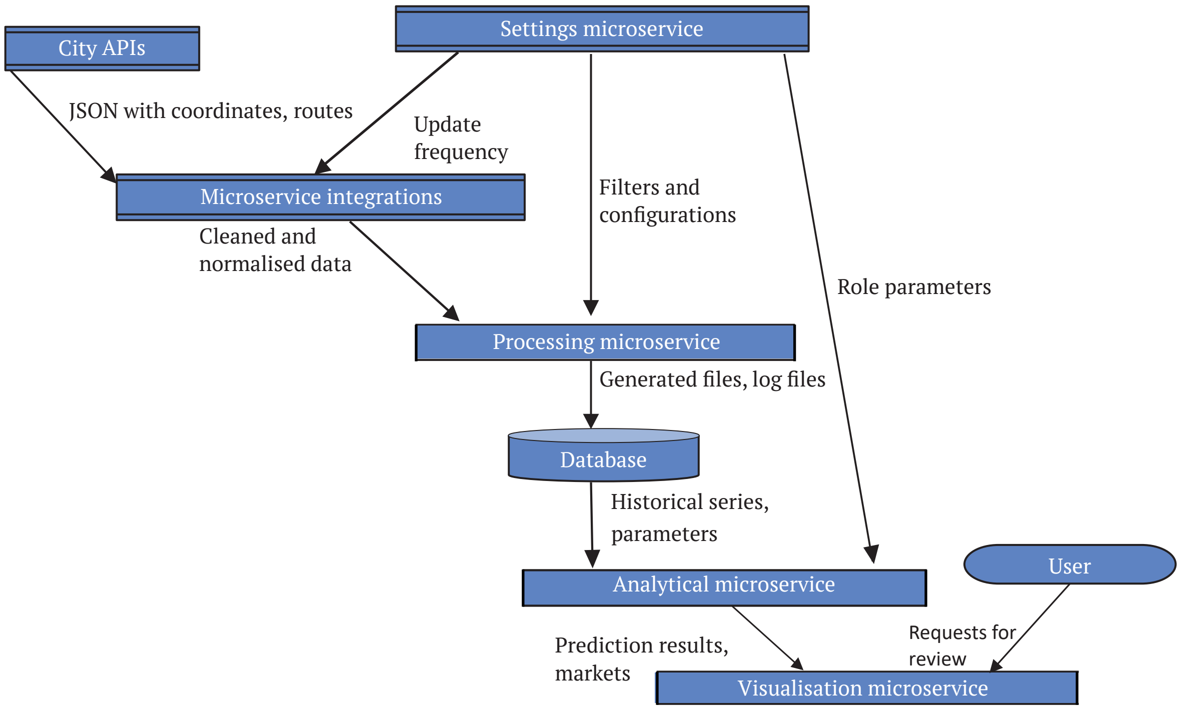
City APIs
Settings microservice
JSON with coordinates, routes
Update frequency
Microservice integrations
Filters and configurations
Cleaned and normalised data
Role parameters
Processing microservice
Generated files, log files
Database
Historical series, parameters
Analytical microservice
User
Prediction results, markets
Requests for review
Visualisation microservice

***Figure 5**. Service DFD diagram*

***Source:** designed by the authors*

***Table 2**. Principal database entities*

Entity	Description
Routes	Route number, mode of transport, origin and terminus stops.
Vehicles	Identifier, type, associated route number, GPS coordinates.
Stops	Stop name, coordinates, list of routes serving the stop.
Traffic events	Timestamp, location, delay, vehicle ID, distance to the nearest stop.
Analytics/Forecast	Model type, route ID, date, forecast output.
Users	Name, role (analyst, administrator), access token.

***Source:** compiled by the authors*

Figure 6 presented a visual representation of the database in the form of an ER diagram. It is intended to outline the core entity sets (such as Routes, Vehicles, Stops, MovementEvents, Indicators, Users/Roles, and Configurations) together with their primary-foreign key relationships and cardinalities. The diagram is expected to guide normalisation choices (for example, separating time-stamped movement events from relatively static vehicle or route metadata) and to make integrity constraints explicit for implementation. By fixing these invariants at the schema level, the model is positioned to support scalable ingestion, consistent historical queries, and reproducible analytics across different city deployments. The ER diagram is intended to depict the structure of the database that is expected to store information on transport routes, vehicles, stops, traffic events, and analytical results. Each route is intended to be linked to a set of vehicles and stops; each vehicle is intended to be associated with movement events; and each analytical record is intended to be generated by a system user. This model is intended to encode relationships between real-world objects and to support rapid access to the required information. In summary, the overall architecture is intended to provide a clear separation of concerns, where each microservice fulfils a narrowly scoped role and communicates through explicit, well-typed contracts. This decomposition is expected to simplify scaling and fault isolation, while the API Gateway, shared schemas, and configuration store maintain coherence across the system. The C4, DFD, and ER artefacts together are meant to anchor implementation and testing by fixing boundaries, data flows, and invariants before code. Operationally, the design aims to balance near-real-time updates with batch consolidation, so that cities can tune latency, load, and cost to local needs. Collectively, these properties are intended to deliver a robust foundation for deployment, evolution, and reproducible analysis in heterogeneous urban environments.

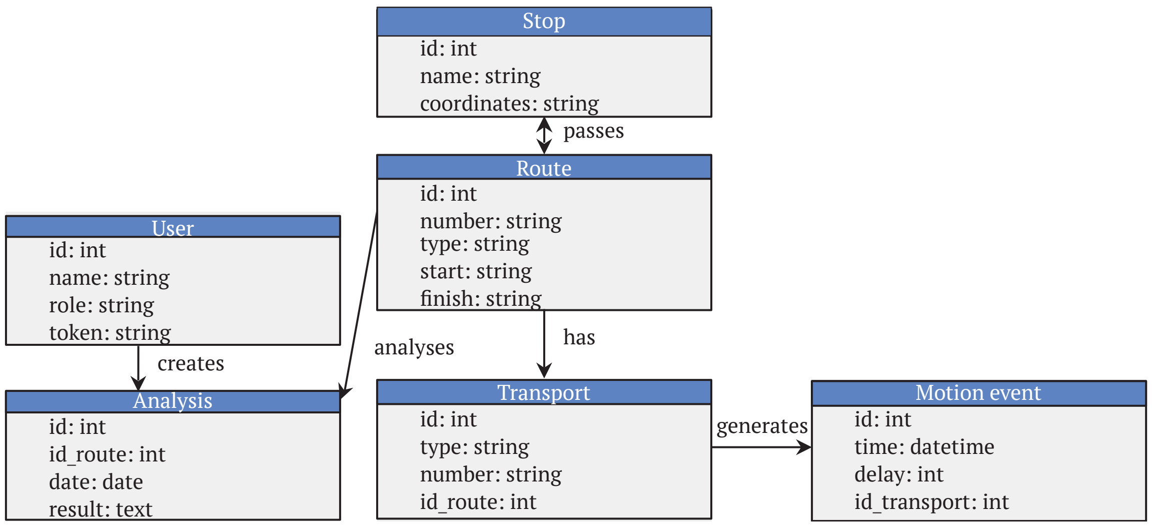

***Figure 6**. ER diagrams of the databases*

***Source:** designed by the authors*

Conclusions

The study has enabled the formulation of a coherent conceptual model of a web-based system for managing urban traffic flows grounded in a microservice architecture. It was found that delineating the system's functional zones at the microservice level ensures a clear separation of responsibilities and facilitates scaling without disrupting the overall data-processing logic. The analysis showed that the chosen sequence of modelling artefacts – an ER diagram for the domain, a multi-level DFD to trace information flows, and a C4 diagram to refine container boundaries and interactions – is sufficient for the unambiguous reproduction of the proposed architecture by other researchers and developers. It was demonstrated that integrating these artefacts within a single methodological framework minimises ambiguities in data interpretation and conflicts between components, which is particularly important in urban scenarios characterised by heterogeneous data sources.

It was found that the asynchronous interaction of the integration module with municipal application programming interfaces ensures stable message intake and simplifies the unification of data formats. The analysis showed that deploying a dedicated processing and aggregation module makes it possible to standardise core traffic metrics and prepare aligned temporal slices for subsequent interpretation. It was substantiated that the analytics module should be constructed as a superstructure over aggregated indicator sets, focusing on the computation of indicators and baseline forecasts, whereas the visualisation microservice provides cartographic and time-series renderings of results for the prompt interpretation of situations across the urban space.

The proposed separation of processing into operational and periodic modes proved methodologically sound. It was established that the operational loop is appropriate for obtaining current indicators of load and deviations from the timetable, whereas the periodic loop serves to reconcile historical data and to construct aggregate measures. It was demonstrated that unified metric definitions and deterministic calculation functions are critical to the reproducibility of results across diverse urban contexts and ensure consistency between visual representations and the outputs of the analytics module. Future research will focus on the stepwise validation of the model using real urban data, the deployment of a prototype connected to open application programming interfaces, the extension of the analytics module with methods for forecasting short-term delays and overloads, and the development of harmonised visualisation scenarios for different user categories, thereby enabling an assessment of the proposed architecture's scalability and reproducibility under practical conditions.

Acknowledgements

The authors express their gratitude to the State University of Trade and Economics, which provided comprehensive support for scientific research by university teachers and postgraduate students.

Funding

None.

Conflict of Interest

None.

References

[1] Adrianto, H.A., Sitanggang, I.S., Akbar, A., Neyman, S.N., & Azhim, M.F. (2024). Interactive dashboard for visualization and analysis of landcover change and land/forest fire. *IOP Conference Series: Earth and Environmental Science*, 1418, article number 012082. doi: 10.1088/1755-1315/1418/1/012082.

[2] Barabash, O., Weigang, G., & Komar, K. (2021). Formation of traffic safety profile in central parts of the city and its informational protection. *Transport Technologies*, 2(2), 42-51. doi: 10.23939/tt2021.02.042.

[3] Batita, S., Makni, A., & Amous, I. (2024). Intelligent transportation systems: A survey on data engineering. In *DATA 2024 proceedings of the 13th international conference on data science, technology and applications* (pp. 169-179). Setúbal: SCITEPRESS – Science and Technology Publications. doi: 10.5220/0012857300003756.

[4] Beeraka, B.S. (2025). Enterprise microservices: Revolutionizing telecom network architecture. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 11(1), 290-297. doi: 10.32628/cseit25111231.

[5] Bokolo, A. (2025). Enabling seamless interoperability of digital systems in smart cities using API: A systematic literature review. *Journal of Urban Technology*, 31(4-5), 123-156. doi: 10.1080/10630732.2024.2427543.

[6] Campos, C., Asenjo, R., & Navarro, Á. (2025). Exploring data flow design and vectorization with oneAPI for streaming applications on CPU+GPU. *The Journal of Supercomputing*, 81, article number 428. doi: 10.1007/s11227-024-06891-3.

[7] Cao, Y., Li, Q.-J., & Yang, Z. (2025). Identifying the spatial range of the Pearl River Delta urban agglomeration from a differentiated perspective of population distribution and population mobility. *Applied Sciences*, 15(2), article number 945. doi: 10.3390/app15020945.

[8] Danyliuk, A., & Muliarevych, O. (2024). Ant colony algorithm in traffic flow control. *Advances in Cyber-Physical Systems*, 9(2), 158-163. doi: 10.23939/acps2024.02.158.

[9] Erişkin, E. (2024). Collaborative game-theoretic optimization of public transport fare policies: A global framework for sustainable urban mobility. *Sustainability*, 16(24), article number 11199. doi: 10.3390/su162411199.

[10] Fornalchyk, Y., & Hilevych, V. (2023). Characteristics of motorization's impact on the urban population. *Transport Technologies*, 4(2), 68-75. doi: 10.23939/tt2023.02.068.

[11] Horokhovskyi, O., & Oshovska, K. (2015). Quality management and obtaining optimal results in the city's transport network management system. *Information Technologies and Computer Engineering*, 12(3), 29-32.

[12] Inamdar, S.A., & Kulkarni, S.S. (2025). Advancing traffic volume prediction and synthetic data generation with machine learning and deep learning. *International Journal for Multidisciplinary Research*, 7(1), 1-18. doi: 10.36948/ijfmr.2025.v07i01.35530.

[13] Kotenko, M., Moskalyk, D., Kovach, V., & Osadchyi, V. (2024). Navigating the challenges and best practices in securing microservices architecture. In *CPITS-II 2024: Workshop on cybersecurity providing in information and telecommunication systems II* (pp. 1-16). Kyiv: Borys Grinchenko Kyiv Metropolitan University.

[14] Kozachenko, D.M., Klyga, O.V., & Kharchenko, Ye.V. (2025). Modeling of train sets in tasks of railway stations technical and operational evaluation. *Science and Transport Progress*, 2(110), 88-97. doi: 10.15802/stp2025/331674.

[15] Laituri, M., Kalra, Y., & Yang, C. (2025). The disappearance of COVID-19 data dashboards: The case of ephemeral data. *COVID*, 5(1), article number 12. doi: 10.3390/covid5010012.

[16] Leghemo, I.M., Segun-Falade, O.D., Odionu, C.S., & Azubuike, C. (2025). Continuous data quality improvement in enterprise data governance: A model for best practices and implementation. *Journal of Engineering Research and Reports*, 27(2), 29-45. doi: 10.9734/jerr/2025/v27i21391.

[17] Mansurova, A., Mussina, A., Aubakirov, S., Nugumanova, A., & Yedilkhan, D. (2025). From raw GPS to GTFS: A real-world open dataset for bus travel time prediction. *Data*, 10(8), article number 119. doi: 10.3390/data10080119.

[18] Matseliukh, Y., & Lytvyn, V. (2024). Modeling of passenger flows analysis system of low-carbon transportation in a smart city. *Information Systems and Networks*, 15, 430-448. doi: 10.23939/sisn2024.15.430.

[19] Ntouros, K., Papatheodorou, K., Gkologkinas, G., & Drimzakas-Papadopoulos, V. (2025). A Python framework for crop yield estimation using Sentinel-2 satellite data. *Earth*, 6(1), article number 15. doi: 10.3390/earth6010015.

[20] Ohonovskyi, Y., Berko, A., Chyrun, L., & Chyrun, S. (2023). Information system prototype for monitoring and content analysis of complaints from smart city residents. *Information Systems and Networks*, 13, 24-45. doi: 10.23939/sisn2023.13.024.

[21] Oyenuga, A.O., Apeh, C.E., Odionu, C.S., & Austin-Gabriel, B. (2025). Advancing public safety and housing solutions: A comprehensive framework for machine learning and predictive analytics in urban policy optimization. *International Journal of Frontiers in Science and Technology Research*, 8(1), 24-43. doi: 10.53294/ijfstr.2025.8.1.0024.

[22] Power BI Community. (2021). Power BI challenge 12 wrap up: Transport & shipping data. *Microsoft Power BI Community Blog*. Retrieved from https://community.powerbi.com/t5/Community-Blog/Power-BI-Challenge-12-Wrap-Up-Transport-amp-Shipping-Data/ba-p/1788343.

[23] Puzio, E., Drozdz, W., & Kolon, M. (2025). The role of intelligent transport systems and smart technologies in urban traffic management in Polish smart cities. *Energies*, 18(10), article number 2580. doi: 10.3390/en18102580.

[24] Quesado Filho, N.O., Guimarães, C.G.C., & Oliveira-Neto, F.M. (2025). Gtfswizard: A set of tools for exploring and manipulating general transit feed specification in R language. *Contribuciones a Las Ciencias Sociales*, 18(1), article number e14620. doi: 10.55905/revconv.18n.1-197.

[25] Rafamatanantsoa, B.P., Jeddoub, I., Yarroudh, A., Hajji, R., & Billen, R. (2024). City2Twin: An open urban digital twin from data integration to visualization and analysis. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 48(2), 387-394 doi: 10.5194/isprs-archives-xlviii-2-w8-2024-387-2024.

[26] Şahin, G., Avcı, D.A., Karagenç, Ş., Pak, B.K., Seke, P., & Tunalı, A.Ç. (2024). Enhancing user experience in insurance applications: Adopting microservices architecture and innovative methods. *Orclever Proceedings of Research and Development*, 5(1), 274-284. doi: 10.56038/oprd.v5i1.548.

[27] Schetakis, N., Bonfini, P., Alisoltani, N., Blazakis, K., Tsintzos, S.I., Askitopoulos, A., Aghamalyan, D., Fafoutellis, P., & Vlahogianni, E.I. (2025). Data re-uploading in Quantum Machine Learning for time series: Application to traffic forecasting. *arXiv*. doi: 10.48550/arXiv.2501.12776.

[28] Volkov, D., & Liubchenko, V. (2024). Securing microservices: Challenges and best practices. In *ICST-2024 information control systems & technologies*. Odesa: Odesa Polytechnic National University.

[29] Wang, Z., Yu, D., Zheng, X., Meng, F., & Wu, X. (2025). A model-data dual-driven approach for predicting shared bike flow near metro stations. *Sustainability*, 17(3), article number 1032. doi: 10.3390/su17031032.

[30] Wibowo, A., Ruslanjari, D., Surahmat, A., Karyaningsih, D., & Vera, N. (2024). The MxT model: Leveraging social media data for real-time route optimization in disaster-prone urban transport networks. *International Journal of Transport Development and Integration*, 8(4), 587-594. doi: 10.18280/ijtdi.080410.

[31] Yang, F. (2025). Leveraging mobile interaction technologies for real-time decision making in enterprise management systems. *International Journal of Interactive Mobile Technologies*, 19(2), 65-78. doi: 10.3991/ijim.v19i02.53743.

[32] Yu, Y., Wang, Y., Zhang, Y., Qu, H., & Liu, D. (2025). InclusiViz: Visual analytics of human mobility data for understanding and mitigating urban segregation. *arXiv*. doi: 10.48550/arXiv.2501.03594.

Концептуальна модель веб-системи управління транспортними потоками у міському середовищі на основі мікросервісної архітектури

Андрій Роскладка

Доктор економічних наук, професор
Державний торговельно-економічний університет
02156, вул. Кіото, 19, м. Київ, Україна
https://orcid.org/0000-0002-1297-377X

Євгеній Пострелко

Аспірант
Державний торговельно-економічний університет
02156, вул. Кіото, 19, м. Київ, Україна
https://orcid.org/0000-0001-9730-450X

Анотація. Актуальність дослідження зумовлена зростанням навантаження на міську транспортну інфраструктуру, потребою у цифровій трансформації управління мобільністю та впровадженні інтелектуальних технологій аналізу даних у режимі реального часу. Така система дає змогу оперативно реагувати на зміни у русі транспорту, підвищувати ефективність маршрутної мережі та покращувати якість міських перевезень. Метою дослідження було формування архітектури веб-системи, яка забезпечує інтеграцію даних із міських інтерфейсів прикладного програмування, їх автоматизовану обробку, аналітичну інтерпретацію, візуалізацію результатів та централізоване керування конфігураціями. Методологічну основу становили підходи системного аналізу, моделювання потоків даних, побудова діаграм «сутність-зв'язок» та застосування архітектурних шаблонів для відображення структурної та функціональної взаємодії компонентів. У статті розроблено концептуальну модель веб-системи управління транспортними потоками у міському середовищі, побудовану за принципами мікросервісної архітектури. Результатом роботи було створення структурної моделі системи, що включила п'ять взаємопов'язаних мікросервісів: інтеграції з міськими інтерфейсами прикладного програмування (Node.js), обробки та агрегації даних (Python, Redis Streams), аналітичного модуля з підтримкою алгоритмів машинного навчання (Scikit-learn), модуля візуалізації (React, Mapbox, Chart.js) та сервісу керування доступом і конфігураціями. Детально описано логіку їхньої взаємодії, механізми масштабування та забезпечення відмовостійкості. Практична цінність дослідження полягала у розробленні універсальної архітектонічної основи для впровадження систем аналітики й моніторингу міського транспорту, що можуть стати прототипом інтелектуальних платформ управління мобільністю в концепції Smart City

Ключові слова: розподілені інформаційні сервіси; цифрове моделювання міської мобільності; аналіз потоків даних у реальному часі; веб-аналітика транспортних систем; візуалізація просторово-часових даних; Smart City