الجمهورية الجزائرية الديمقراطية الشعبية
People's Democratic Republic of Algeria
وزارة التعليم العالي والبحث العلمي
Ministry of Higher Education and Scientific Research
المدرسة الوطنية العليا للتكنولوجيات المتقدمة
National Higher School of Advanced Technologies

Department of Electrical Engineering and Industrial Computing

Final Year Project for the Master's Degree

- Field -
Telecommunications

- Specialty -
Telecommunications and Networking Systems

- Subject -

KPI Analytics Platform:
An Integrated Approach for Multi-Vendor Mobile Network Performance Monitoring and Analysis

Realized by:

ALAOUCHICHE Abderrahmane Yaakoub
KESSOUM Mohamed Walid

Members of the Jury:

Name	Establishment	Grade	Quality
Mr. Islem BOUCHACHI	ENSTA	MCA	President
Mrs. Souhila BOUTARFA	ENSTA	MCB	Examiner
Mrs. Imane CHIALI	ENSTA	MCB	Examiner
Mrs. Kheira LAKHDARI	ENSTA	MAB	Supervisor
Mr. Abdelkader BELAHCENE	ENSTA	MCA	Co-Supervisor
Mr. Sifeddine ALREME	OTA (djezzy)	TBM	External Co-Supervisor

Algiers, 25/06/2025

2024–2025

Dedication

ALAOUCHICHE Abderrahmane Yaakoub

To my parents, grandparents, siblings and close ones for their constant support.
To my mother especially — your strength and sacrifices have been my greatest motivation. This achievement is as much yours as it is mine.

KESSOUM Mohamed Walid

To my parents, siblings, family, and close ones — for their invaluable support and constant encouragement throughout my academic journey. This accomplishment is a testament to their unwavering belief in me.

Acknowledgements

We would like to sincerely thank our academic supervisors, Dr. LAKHDARI Khiera and Dr. BELAHCENE Abdelkader, Assistant Professors in the GEII department at ENSTA, for their support and guidance during this project.

We also express our appreciation to Mr. ALREME Sifeddine, Traffic and Budget Manager at Djezzy, for his role as our industrial supervisor

We extend our gratitude to the members of the jury for their time, valuable comments, and thoughtful evaluation of our thesis.

Finally, we acknowledge the teaching staff at the National Higher School of Advanced Technologies, as well as the professional instructors involved in our training, whose expertise contributed to our academic development.

الملخّص

تُعد إدارة شبكات LTE متعددة الموردين مثل شبكة Djezzy التي تضم Huawei و Nokia و ZTE تحديًا بسبب تشتت البيانات واختلاف تنسيقات مؤشرات الأداء KPI . يقدم هذا البحث منصة مركزية وموحدة، محايدة من حيث المورد، لعرض مؤشرات الأداء عبر مختلف الأنظمة.

استنادًا إلى مشروع التخرج النهائي، الذي ركز على تصميم مستودع بيانات يحتوي على مؤشرات أداء موحدة، يقوم هذا العمل بتوسيع تلك القاعدة لتوفير مراقبة شاملة ومتناسقة على مستوى الشبكة. توفر المنصة لوحات تحكم تفاعلية وواجهة بحث مستقلة عن المورد لتحليل الأداء والمقارنة.

يمثل هذا العمل حلاً هندسيًا عمليًا لتحسين رؤية شبكة LTE وتعزيز الكفاءة التشغيلية في شركة Djezzy .

الكلمات المفتاحية: LTE ، متعدد البائعين، توحيد المؤشرات، المراقبة، مستودع البيانات، عرض بياني.

Abstract

Managing multi-vendor LTE networks like Djezzy's, which features Huawei, Nokia, and ZTE, poses challenges due to data fragmentation and incompatible KPI formats. This thesis presents a centralized, vendor-agnostic web platform that unifies KPI visualization across systems.

Building on our final engineering project, which focused on designing a Data Warehouse with normalized KPIs, this work extends that foundation to enable consistent, network-wide monitoring. The developed platform features interactive dashboards and a vendor-independent search interface for performance comparison and analysis.

This thesis provides a practical engineering solution aimed at improving LTE network visibility and operational efficiency at Djezzy.

Keywords: LTE, Multi-Vendor, KPI Normalization, Monitoring, Data Warehouse, Visualization.

Résumé

La gestion des réseaux LTE multi-fournisseurs, comme celui de Djezzy intégrant Huawei, Nokia et ZTE, présente des défis en raison de la fragmentation des données et des formats de KPI incompatibles. Ce mémoire présente une plateforme web centralisée et indépendante des fournisseurs, qui unifie la visualisation des KPI à travers les différents systèmes.

S'appuyant sur notre projet de fin d'études d'ingénieur, qui portait sur la conception d'un entrepôt de données avec des KPI normalisés, ce travail prolonge cette base pour permettre une surveillance cohérente à l'échelle du réseau. La plateforme développée propose des tableaux de bord interactifs et une interface de recherche indépendante du fournisseur pour la comparaison et l'analyse des performances.

Ce mémoire constitue une solution d'ingénierie concrète visant à améliorer la visibilité du réseau LTE et l'efficacité opérationnelle chez Djezzy.

Mots-clés : LTE, Multi-Fournisseur, Normalisation KPI, Monitoring, Entrepôt de Données, Visualisation, Flask.

Contents

List of Figures

Introduction

The effective management of modern LTE (Long Term Evolution) networks is often complicated by the use of equipment from multiple vendors, such as Huawei, Nokia, and ZTE. Each of these vendors uses its own way of organizing data and calculating Key Performance Indicators (KPIs), making it difficult to evaluate the overall performance of the network in a unified way. This becomes even more challenging when trying to assess performance at a geographic level, where different vendors may be responsible for different parts of the area.

The main issue is that traditional methods of aggregating KPIs across vendors can lead to inaccurate or misleading results, as they often assume that all vendors use the same formulas and definitions. This creates the illusion of a unified view, while in reality the data is inconsistent. As a result, network engineers struggle to make reliable, data-driven decisions.

To address this challenge, we developed a web-based solution using Flask that allows for unified visualization of KPIs, regardless of the vendor. The key idea behind our approach is to introduce a new "merged vendor" concept. This involves defining a set of custom, normalized KPI formulas based on a combination of the original formulas used by Huawei, Nokia, and ZTE. These formulas are weighted according to each vendor's share in the geographic area, enabling accurate and automatic aggregation of performance indicators at any time — without the need for manual calculations.

The normalized KPIs are stored and calculated within a centralized Data Warehouse (DWH), which was designed in prior work [1]. The Flask application connects to this DWH to visualize the network's global performance in a consistent and meaningful way. By doing so, it hides the vendor-specific differences and offers engineers a reliable tool to monitor and compare network behavior across different regions.

Project Objectives

The main goals of this work are:

- To develop a web application that provides a unified interface for visualizing LTE KPIs from different vendors.
- To use normalized KPI formulas that ensure data consistency across all vendors.
- To support network engineers with an efficient tool for evaluating and comparing network performance across geographic areas.

Project Structure

- **Chapter 1** describes the context and challenges of aggregating KPIs from multiple vendors and outlines the overall solution.
- **Chapter 2** explains the methodology and system design, including the architecture of the Data Warehouse and the components of the Flask application.
- **Chapter 3** presents the main features of the platform, especially the KPI Dashboard and Explore pages, and shows how they enable unified performance visualization.
- **Chapter 4** evaluates the system, discusses its current limitations, and concludes with suggestions for future improvements.

Chapter 1

Introduction and Problem Statement

This chapter introduces the main problem of managing KPIs in multi-vendor LTE networks like Djezzy's. Each vendor uses different tools and data formats, making it hard to get a clear and unified view of network performance. To solve this, we propose building a web application using Flask that connects to a normalized Data Warehouse.

1.1 Context and Motivation

Modern LTE networks form the backbone of mobile connectivity, enabling services such as voice, video, and high-speed data. To balance performance needs with cost and flexibility, operators like Djezzy adopt a multi-vendor strategy, deploying infrastructure from vendors including Huawei, Nokia, and ZTE.

While this approach offers advantages like reduced vendor lock-in and access to diverse technologies, it also creates operational complexity. Each vendor provides its own Element Management System (EMS) or Operations Support System (OSS), such as Huawei's PRS, Nokia's NetAct, and ZTE's UMI. These systems define KPIs differently, follow distinct data models, and use separate interfaces, making unified performance monitoring difficult.

As illustrated in Figure 1.1, vendor-specific tools create isolated monitoring environments or "silos," limiting cross-vendor visibility and complicating efforts to analyze and optimize overall network performance.

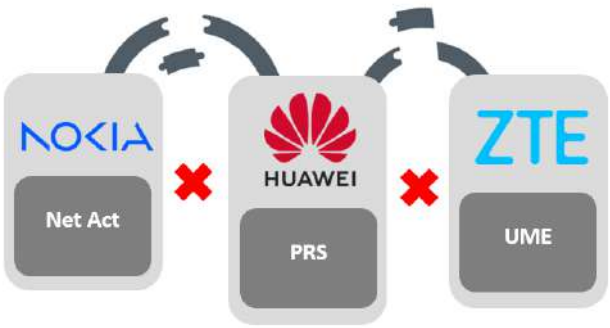


Figure 1.1: Isolated network monitoring silos caused by vendor-specific tools.

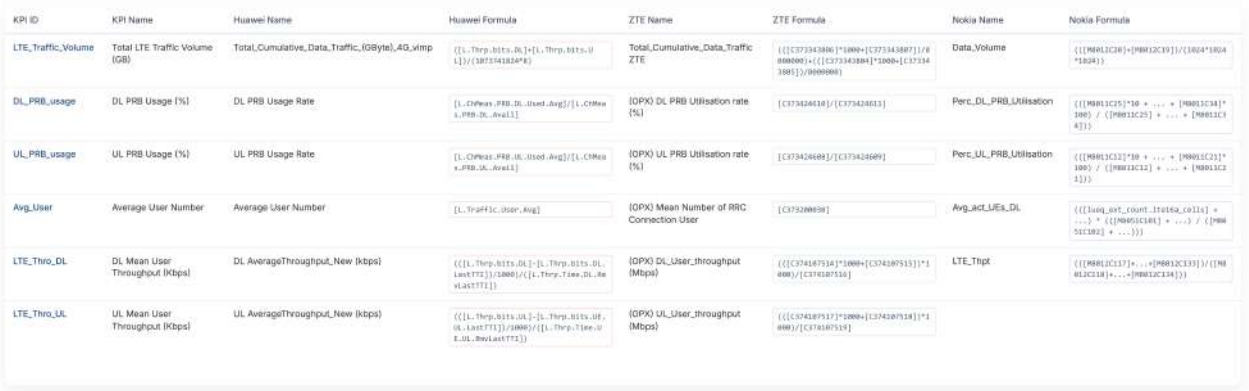
Currently, network engineers at Djezzy rely on three separate, vendor-specific platforms to monitor and analyze LTE performance. While each system-PRS, NetAct, and UMI—is feature-rich and well-integrated with its respective vendor's hardware, their specialization becomes a limitation: none provides a unified view of the network across all vendors simultaneously. Consequently, engineers must frequently switch between tools, manually align KPIs with inconsistent definitions, and reconcile fragmented insights to form a comprehensive picture of network health.

Moreover, the lack of standardized, unified KPI computation across platforms means that overall network performance indicators are not calculated using consistent or exact methodologies, leading to discrepancies in reported metrics. As a result, the aggregated performance view is not entirely reliable, introducing a margin of error that can obscure true network conditions.

This operational burden not only affects efficiency but also increases the risk of misinterpretation and compromises the accuracy of data-driven decision-making.

1.2 Problem Statement: The Illusion of Unified KPIs

When attempting to visualize LTE performance at the network level, it is tempting to aggregate KPI values across vendors. However, this approach is highly misleading. While KPI names may appear standardized (e.g., Call Setup Success Rate or Average DL PRB Usage), the underlying semantics, data sources, and calculation formulas differ significantly from one vendor to another.



KPI ID	KPI Name	Huawei Name	Huawei Formula	ZTE Name	ZTE Formula	Nokia Name	Nokia Formula
LTE_Traffic_Volume	Total LTE Traffic Volume (GB)	Total_Cumulative_Data_Traffic_(GByte)_4G_vimp	([L.Thrp.bits.DL]+[L.Thrp.bits.UL])/(1073741824*8)	Total_Cumulative_Data_Traffic ZTE	(([C373343806]*1000+[C373343807])/(8000000)+(([C373343804]*1000+[C373343805])/8000000)	Data_Volume	(([M8012C26]+[M8012C19])/(1024*1024*1024))
DL_PRB_usage	DL PRB Usage (%)	DL PRB Usage Rate	[L.ChMeas.PRB.DL.Used.Avg]/[L.ChMeas.PRB.DL.Avail]	(OPX) DL PRB Utilisation rate (%)	[C373424610]/[C373424611]	Perc_DL_PRB_Utilisation	(([M8011C25]*10 + ... + [M8011C34]*100) / ([M8011C25] + ... + [M8011C34]))
UL_PRB_usage	UL PRB Usage (%)	UL PRB Usage Rate	[L.ChMeas.PRB.UL.Used.Avg]/[L.ChMeas.PRB.UL.Avail]	(OPX) UL PRB Utilisation rate (%)	[C373424608]/[C373424609]	Perc_UL_PRB_Utilisation	(([M8011C12]*10 + ... + [M8011C21]*100) / ([M8011C12] + ... + [M8011C21]))
Avg_User	Average User Number	Average User Number	[L.Traffic.User.Avg]	(OPX) Mean Number of RRC Connection User	[C373200838]	Avg_act_UEs_DL	(([lueq_ext_count_ltetda_cells] + ...) * ([M8051C101] + ...) / ([M8051C102] + ...))
LTE_Thro_DL	DL Mean User Throughput (Kbps)	DL AverageThroughput_New (kbps)	([L.Thrp.bits.DL]-[L.Thrp.bits.DL.LastTTI])/1000)/([L.Thrp.Time.DL.NewLastTTI])	(OPX) DL_User_throughput (Mbps)	(([C374107514]*1000+[C374107515])*1000)/[C374107516]	LTE_Thpt	(([M8012C117]+...+[M8012C133]))/([M8012C118]+...+[M8012C134]))
LTE_Thro_UL	UL Mean User Throughput (Kbps)	UL AverageThroughput_New (kbps)	([L.Thrp.bits.UL]-[L.Thrp.bits.UL.LastTTI])/1000)/([L.Thrp.Time.UL.NewLastTTI])	(OPX) UL_User_throughput (Mbps)	(([C374107517]*1000+[C374107518])*1000)/[C374107519]		

Figure 1.2: Varying logic behind same KPI names across different vendors.[2]

As illustrated in Figure 1.2, a KPI reported by Huawei is not directly comparable to one reported by Nokia or ZTE. Each vendor relies on its own proprietary counters, data formats, and calculation logic. Therefore, blindly aggregating or comparing these values without normalization introduces **semantic inconsistency** that can lead to misleading visualizations, erroneous conclusions, and suboptimal optimization strategies.

Example: DL PRB Usage

Consider the KPI "DL PRB Usage". Its name may be identical across vendors, but the formulas differ:

- **Huawei:**

$$\frac{\texttt{L.ChMeas.PRB.DL.Used.Avg}}{\texttt{L.ChMeas.PRB.DL.Avail}} \times 100$$

- **ZTE:**

$$\frac{\texttt{C373424610}}{\texttt{C373424611}} \times 100$$

- **Nokia:**

$$\frac{\sum_{i=25}^{34} \texttt{M8011C}i \times ((i-24) \times 10)}{\sum_{i=25}^{34} \texttt{M8011C}i} \times 100$$

Clearly, you cannot simply sum, average, or directly aggregate these KPIs across vendors. Doing so would result in distorted and unreliable indicators. Instead, a field study is required to deeply understand the structure and behavior of each KPI, in order to design a normalized formula that truly reflects the overall network's performance.

That is precisely the approach we adopted. For instance, in the case of DL PRB Usage, we worked closely with domain engineers at the company to define a unified abstraction model:

- **Unified abstraction:**

$$\frac{\text{Huawei used} + \text{ZTE used} + \text{Nokia weighted used}}{\text{Huawei avail} + \text{ZTE avail} + \text{Nokia total}} \times 100$$

This example illustrates just one KPI. In practice, similar vendor-specific discrepan- cies exist for all major KPIs—including Call Drop Rate, Accessibility, Availability, and Throughput. To enable reliable cross-vendor comparisons, each KPI must be redefined using an abstraction layer that harmonizes the formulas and normalizes values

1.3 Proposed Solution: A Vendor-Normalized Visualization Platform

The proposed solution is the development of a **centralized, web-based platform** that unifies and normalizes Key Performance Indicator (KPI) definitions across multiple vendors. It is important to emphasize that this platform is not intended to replace the established vendor-specific Element Management Systems (EMS) or Operations Support Systems (OSS), which remain **highly robust, feature-rich, and indispensable** for in-depth network management and troubleshooting within their respective vendor ecosystems.

Instead, our platform addresses a significant operational challenge: the fragmentation of KPI definitions, formats, and visualizations across heterogeneous vendor environments.

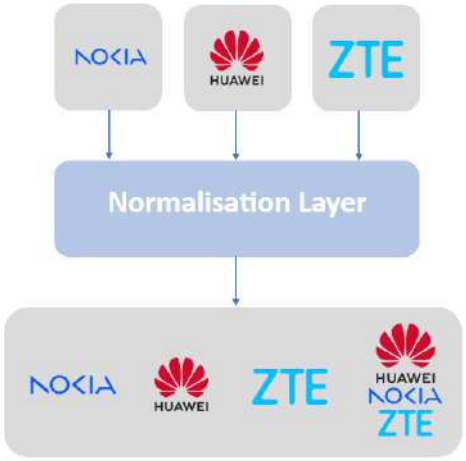


Figure 1.3: Cross-vendor analysis via unified platform with normalized KPIs.

As shown in Figure 1.3, the proposed platform introduces a normalization layer that harmonizes KPI definitions and semantics, enabling unified analysis and cross-vendor comparability. Our solution fills this critical gap by providing a **common, vendor-agnostic interface** that aggregates and normalizes KPI data, thus enabling consistent, high-level insight across the entire LTE network infrastructure.

The platform leverages a pre-existing, rigorously designed Data Warehouse (DWH) that implements **vendor normalization formulas** to abstract away underlying differences in KPI definitions and data semantics. This normalization layer is the core innovation enabling the platform to present coherent, comparable performance metrics regardless of the source vendor.

The web application, developed using the Flask [3] framework, selected for its lightweight nature and flexibility, provides:

- A **unified dashboard** (`/visualize`) presenting normalized KPIs from all vendors, offering network engineers a consolidated view of key performance trends.
- An **exploratory analysis module** (`/explore`) that allows for flexible filtering, segmentation, and drilling down into normalized KPIs to better understand network behavior under varying conditions.
- **Cross-vendor comparison tools** that highlight performance disparities and identify potential areas for optimization across the multi-vendor network.

This solution is designed to **complement**, rather than supplant, vendor-specific EMS/OSS tools. Those native platforms will continue to serve as the authoritative systems for detailed fault management, configuration, and vendor-specific feature utilization due to their depth and tight integration with vendor equipment.

Our platform, by contrast, empowers engineers with a **streamlined, vendor-neutral vantage point**, facilitating faster, more effective cross-network analysis and strategic decision-making. This high-level abstraction reduces operational complexity, enhances situational awareness, and ultimately contributes to more efficient network performance management.

Chapter 2

Methodology and System Design

This chapter explains how the system was designed and built. It starts with the technical requirements needed to hide vendor differences and support unified KPIs. Then, it describes the architecture, which includes a Data Warehouse and a Flask web app. We'll also explain how the backend and frontend work together, including how data is handled, displayed, and visualized.

2.1 Requirements Analysis: Toward a Unified Analytical Interface

Network operations and engineering teams identified the need for an analytical interface that overcomes the limitations of vendor-specific tools. The main requirements are:

- **Unified KPI View:** The platform must show KPIs from different vendors (Huawei, Nokia, ZTE) side-by-side on the same charts, and also provide an aggregated network-wide KPI view based on normalized formulas.
- **Vendor-Agnostic Exploration:** Users should be able to search and explore network entities (cells, sites) without needing to specify the vendor.
- **Accuracy and Consistency:** Aggregated KPIs must be calculated using reliable normalization methods to ensure accurate and comparable results.
- **Filtering and Interactivity:** The system should offer filtering by vendor (individual or all), technology (2G, 3G, 4G), geography (Wilaya, Commune, Cell), time granularity (daily, hourly, busy hour), and custom date ranges.
- **Responsiveness and Usability:** The user interface must be responsive, intuitive, and provide quick access to data, even with large datasets. Loading indicators and feedback are essential.
- **Data Export:** Users must be able to export chart data and lists for offline analysis or reporting.

- **Extensibility:** The platform should be designed to support future updates, including integration of new radio technologies or additional vendors.

To meet these requirements, the system uses a reliable backend Data Warehouse for data consolidation and normalization, combined with a web application that supports interactive visualization and user interaction.

2.2 Architectural Approach: Data Warehouse and Normalized KPI Abstraction

The adopted solution follows a multi-tiered architectural pattern, depicted in Figure 2.1. It comprises:

1. A backend Data Warehouse using PostgreSQL [4] responsible for ingesting, storing, processing, and normalizing raw performance data from multiple vendor systems. This DWH was designed and its ETL processes established as part of the Final Education Project [1].

2. A Flask-based web application (the focus of this thesis) that serves as the middleware and presentation layer, querying the DWH and rendering interactive visualizations to the end-user via a web browser.

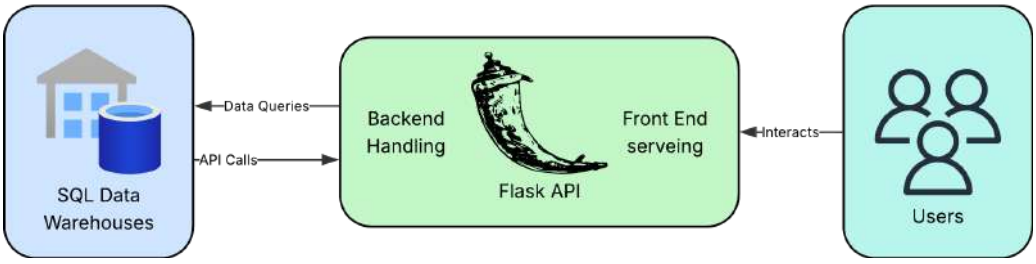


Figure 2.1: System Architecture: Flask Web Application and Data Warehouse.

2.2.1 Data Warehouse Design for Unified Views [1]

The Data Warehouse (DWH) for this telecom network performance monitoring system employs a dimensional modeling approach, manifesting as a star schema. This design is inherently optimized for analytical queries and business intelligence, which are the bedrock of effective data visualization.

A diagram of the star schema is presented in Figure 2.2, illustrating the central fact table and its relationships with the surrounding dimension tables. This schema supports efficient slicing, dicing, filtering, and drill-down operations, which are essential for responsive and insightful web-based dashboards.

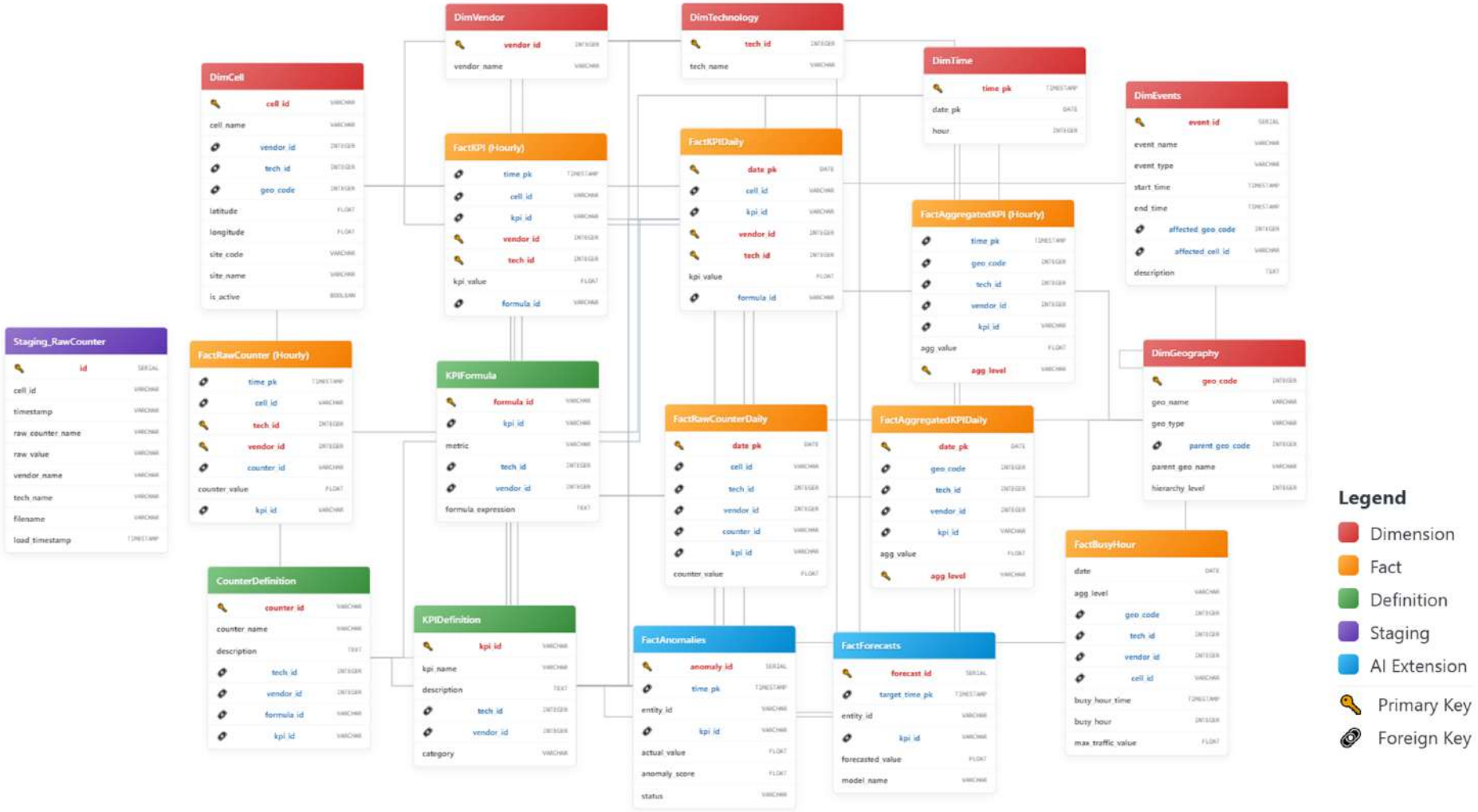


Figure 2.2: Data Warehouse Entity Relationship Diagram [1]

The following DWH elements are particularly crucial for enabling the Flask web application to deliver rich, performant, and semantically consistent visualizations :

1. **Foundation of Semantic Consistency for Unified Views:** A primary challenge in multi-vendor telecom networks (Huawei, Nokia, ZTE) is presenting a unified and comparable view of network performance. Raw counters often differ in naming, granularity, or definition. The DWH addresses this through:

   - `DimVendor`: This dimension table is fundamental. It allows the Flask application to present data either for a *specific vendor* or for the *entire network* through a synthetic "ALL" vendor (e.g., `vendor_id = 3`). This "ALL" vendor concept is key for aggregated, network-wide visualizations.
   - `KPIDefinition`: Provides user-friendly names and descriptions for KPIs, ensuring visualizations are clearly labeled. It defines KPIs for individual vendors and, importantly, for the "ALL" synthetic vendor.
   - `KPIFormula`: This table is central to achieving data harmonization. For the "ALL" vendor (`vendor_id = 3`), the `formula_expression` column stores the *normalized formulas*. These reconcile differences in raw counters and KPI definitions from disparate vendors, producing KPI values that are consistent and comparable across the entire network. This normalization is paramount for any meaningful network-wide visualization.

2. **Pre-Aggregated Data for Responsive Dashboards:** Visualizations, especially dashboards displaying trends over time or across broad geographical areas, can suffer from poor performance if they require on-the-fly aggregation of vast, granular datasets. The DWH mitigates this with:

   - `FactAggregatedKPI` (hourly) and `FactAggregatedKPIDaily` (daily): These fact tables are workhorses for the Flask application's high-level visualizations. They store KPI values that have already been calculated and aggregated to various geographical levels (e.g., `agg_level` such as 'COMMUNE', 'WILAYA', 'ALLNET'). When the Flask app requests a network-wide daily trend for a specific KPI, it queries `FactAggregatedKPIDaily` (e.g., for `agg_level = 'ALLNET'`, `vendor_id = 3`). This query is exceptionally fast due to pre-computation, leading to snappy and responsive dashboards.

3. **Granular Data for Drill-Down and Detailed Analysis:** While high-level views are essential, users often need to drill down into specifics to diagnose issues.

   - `FactKPI` (hourly) and `FactKPIDaily` (daily): These tables store KPI values at a granular level (per cell, per hour/day). If an anomaly is observed at a higher aggregation level (e.g., Wilaya), the Flask application can query these tables (filtered

for cells within that Wilaya and for `vendor_id = 3` for normalized values) to enable detailed investigation at the cell level.

4. **Rich Context through Dimension Tables for Filtering and Slicing:** Effective visualization requires context. Dimension tables provide this essential contextual information.

   - `DimGeography`: Enables the Flask application to present KPIs on maps, and to offer filters and aggregations by 'COMMUNE', 'WILAYA', or the entire network. The hierarchical structure (`parent_geo_code`, `hierarchy_level`) supports interactive drill-down and roll-up capabilities in visualizations.
   - `DimCell`: Provides detailed attributes for cell-specific visualizations, including `latitude` and `longitude` for map plotting, and `site_code`/`site_name` for grouping or labeling.
   - `DimTime`: Indispensable for all time-series visualizations (e.g., line charts, trend analyses). Columns like `date_pk` and `hour` allow precise filtering and grouping by various time periods.
   - `DimTechnology`: Allows users to filter visualizations to view performance specific to "2G", "3G", or "4G" networks.

5. **Performance Enhancement via Materialized Views:** For frequently accessed, complex aggregations, materialized views offer a significant performance boost.

   - `MV_DailyNetworkKPI` and `MV_HourlySiteKPI`: These pre-computed tables store the results of common, potentially expensive queries (e.g., daily network-wide KPI averages, hourly site-level KPI aggregations). The Flask application can query these MVs directly, leading to faster data retrieval for these predefined views and an improved user experience.

6. **Specialized Insights Ready for Visualization:** Certain complex metrics can be precalculated and stored for direct visualization.

   - `FactBusyHour`: This table stores pre-determined busy hour information for various aggregation levels and entities. The Flask application can directly query this table to visualize busy hours (e.g., for each Wilaya or for specific cells) without needing to perform complex traffic analysis computations in real-time.

In essence, the DWH is meticulously designed to empower the Flask application's visualization component by:

- Delivering **trustworthy and comparable KPIs** across a multi-vendor environment, primarily through the normalized formulas in `KPIFormula` and the "ALL" vendor concept.

- Ensuring **fast dashboard loading and high interactivity** by providing pre-aggregated data in tables like `FactAggregatedKPI(Daily)` and further optimized views in Materialized Views.
- Enabling **powerful filtering, drill-down capabilities, and contextual understanding** through well-structured dimension tables.
- Providing access to **granular data** via `FactKPI(Daily)` when detailed investigation is necessary.

The Flask application can thus rely on the DWH to provide data that is largely pre-processed or readily combinable, minimizing the computational load on the application server and ensuring a responsive and insightful user experience for network performance visualization.

2.2.2 Application Design: Flask as the Presentation Layer

The Flask web application is architected to provide a clear separation of concerns, with distinct backend and frontend components. Figure 2.3 illustrates the typical data flow for rendering a unified KPI view.

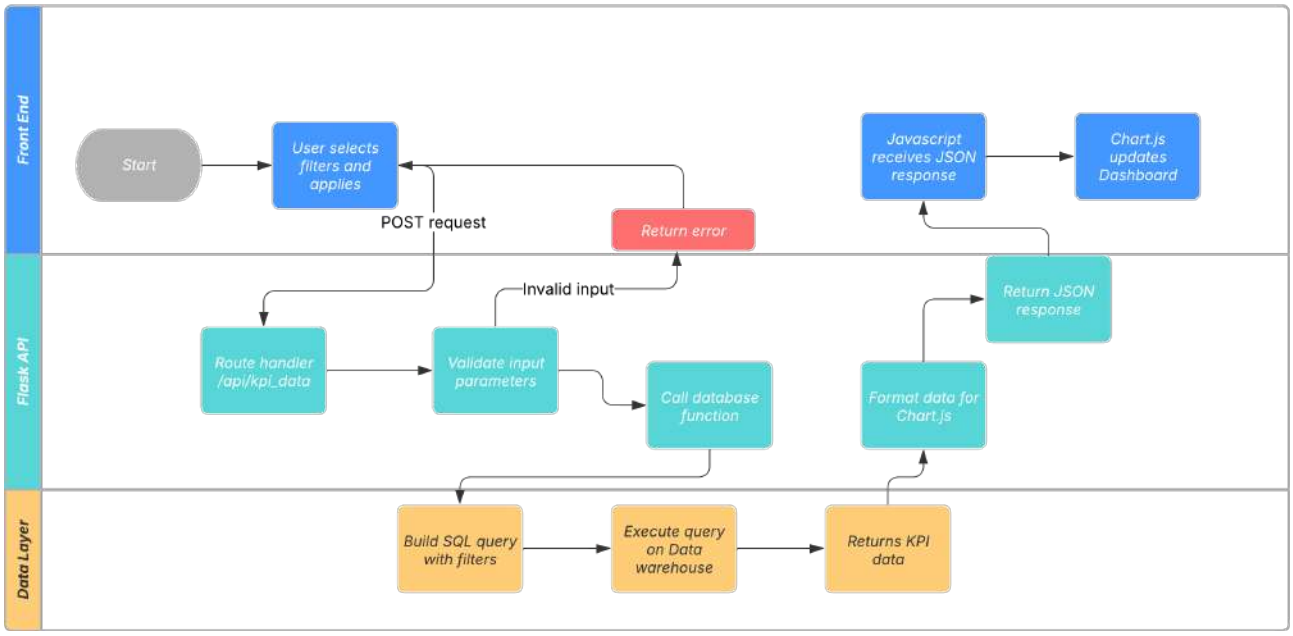


Figure 2.3: Detailed Application Data Flow for a Unified KPI View Request

This diagram highlights the modularity and clarity of the application's architecture. It emphasizes the distinct responsibilities of each component—from request handling and database access to data formatting and frontend rendering. The design ensures scalability and maintainability by cleanly separating concerns between business logic, data processing, and user interface updates.

2.2.3 Back-End Components (Flask)

The backend of the application is developed using the Flask microframework, which enables a lightweight and modular architecture. As illustrated in Figure 2.4, the project is structured to promote maintainability and separation of concerns, with clearly defined modules for configuration, database access, routing, and utilities.

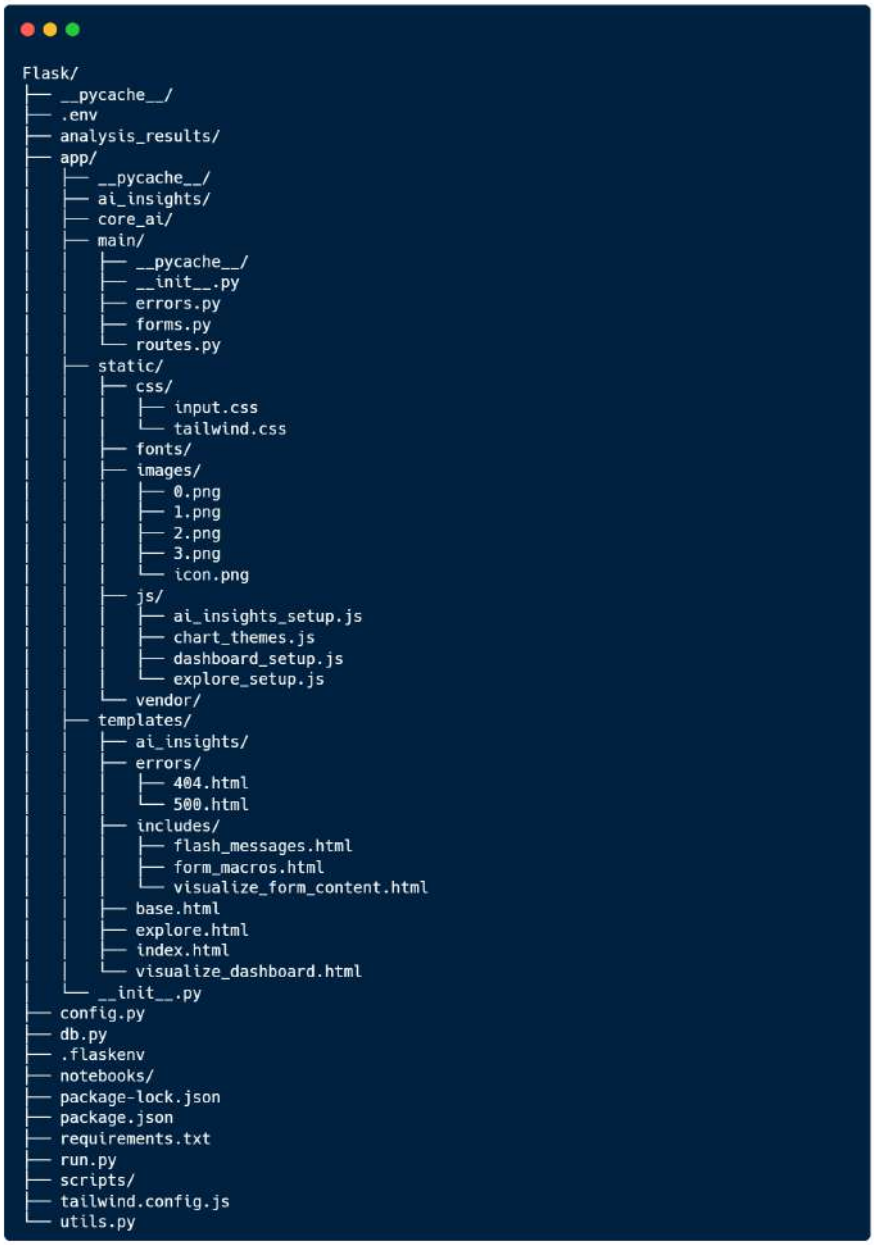


Figure 2.4: Flask project directory structure in the development environment.

Backend modules include:

- **`config.py`:** Manages application configurations (Development, Production) including database connection parameters (host, port, name, user, password from environment variables), secret key, cache settings, and logging levels. These variables are loaded from the `.flaskenv` file, which sets key environment parameters such as `FLASK_APP`, `FLASK_ENV`,

SECRET_KEY, and PostgreSQL connection details (PGHOST, PGPORT, etc.), allowing secure and environment-specific configuration without hardcoding credentials.

- **db.py:** This module acts as the data access layer, abstracting database interactions.
  - **Connection Pooling (init_pool, get_db, close_db):** Implements a PostgreSQL connection pool (psycopg2.pool.SimpleConnectionPool) to efficiently manage database connections. Connections are managed within Flask's application context (g).[5]
  - **Cursor Management (get_cursor):** Provides a context manager for obtaining database cursors (using psycopg2.extras.DictCursor), handling transactions (commit/rollback), and ensuring cursors are closed.
  - **Generic Query Functions (fetch_all, fetch_one, execute_commit):** Utility functions for executing SELECT queries and DML statements.
  - **get_dropdown_data:** Fetches distinct KPIs, cell IDs, technologies, vendors (including an "ALL" vendor representation), and geographical entities (Wilayas, Communes). This data is cached using Flask-Caching (@cache.memoize) for performance.
  - **get_cells_for_geo:** Dynamically fetches cell IDs based on selected technology, vendor, and geographical filters. Handles vendor_id=3 ("ALL") by querying for vendors 0, 1, and 2.
  - **get_multiple_kpi_results:** Core function for the KPI Dashboard.
    1. For comparative visualizations: fetches per-vendor data from raw KPI tables (FactKPI, FactKPIDaily) using vendor_id IN (0,1,2).
    2. For aggregated views: queries FactAggregatedKPI or FactAggregatedKPIDaily for vendor_id=3.

    The function adjusts granularity (hourly, daily, busyhour) and dynamically builds SQL JOINs based on selected KPIs.
  - **get_detailed_cell_statistics:** Computes active vs. created cell counts and segmentation (Good, Bad, Critical) using thresholds (e.g., throughput) from FactKPI.
  - **search_entities_v3:** Vendor-agnostic search across Wilayas, Communes, Sites, and Cells using pattern matching (ILIKE). Returns detailed metadata and Google Maps links.
  - **get_total_counts:** Retrieves global summary stats (e.g., number of active cells, sites) for the Explore page.
- **main/routes.py:** Defines application routes and associated logic.
  - **Helper Routes (/get_cells, /get_communes, /get_kpis, etc.):** Serve dynamic UI content via AJAX (e.g., updating dropdowns).

  - **`/visualize` (KPI Dashboard):**
    * **GET:** Populates default view (e.g., "ALL" vendor, latest dates).
    * **POST:** Validates form data, determines vendor/tech context, selects aggregation levels (Cell, Commune, Wilaya, ALLNET), fetches KPIs using `db.get_multiple_kpi_results`, and renders charts.
  - **`/explore` (Exploratory Analysis Interface Page):**
    * **GET:** Loads network and segmentation stats.
    * **POST:** Executes AJAX search via `db.search_entities_v3`.
  - **Export Routes:** Serve CSV downloads for KPI data and segmentation results.

- **`utils.py`:** Contains helper functions.
  - **`format_chart_data_multiple_kpis`:** Converts raw KPI data into JSON format for Chart.js [6], separating datasets by KPI/Vendor and assigning labels/colors.
  - **`calculate_summary_stats`:** Computes min, max, avg, and count for KPI sets.
  - **`decimal_to_dms`:** Converts decimal GPS coordinates to DMS format.

- **`app/__init__.py`:** Application factory (`create_app`): Initializes Flask extensions (e.g., Flask-Caching), registers blueprints, sets up logging, error handlers, and Jinja [7] filters (e.g., `file_exists_filter`).

2.2.4 Front-End Design

The frontend is designed to be responsive, dynamic, and maintainable, relying on modern web technologies for optimal user experience:

- **HTML5 (in `templates/`):** The Flask application uses the Jinja2 templating engine to render dynamic HTML pages. Key templates include:
  - `base.html`: Defines the main layout, including the navigation bar, footer, and shared assets (CSS/JS). It also implements light/dark mode toggling via Alpine.js [8] and localStorage.
  - `visualize_dashboard.html`: Hosts the form for KPI visualization filters and the output containers for charts and statistics tables.
  - `explore.html`: Structures the Explore page with tabbed interfaces for "Search Entities" and "Statistics Health," including related forms and result displays.
  - `errors/404.html` and `errors/500.html`: Custom templates for handling client and server errors.
  - `includes/_flash_messages.html`: Renders transient messages (flashed messages) from the Flask backend.

- **CSS (TailwindCSS):** A utility-first framework compiled from `input.css` into `tailwind.css` [9] is used for styling. It enables fast UI prototyping and clean, consistent component theming, inspired by the Shadcn UI design system.
- **JavaScript (in `static/js/`):** The frontend logic is handled using vanilla JavaScript and Alpine.js for lightweight reactivity.
  - `dashboard_setup.js`: Manages the '/visualize' page, including:
    - Initializing Flatpickr [10] for date/time inputs.
    - Dynamically updating dependent dropdowns (e.g., Wilaya → Commune → Cell suggestions) via AJAX.
    - Managing vendor and technology selections and corresponding hidden inputs.
    - Performing client-side form validation.
    - Rendering and updating Chart.js visualizations based on server responses.
    - Handling presets (save/load/clear filter configurations).
  - `explore_setup.js`: Powers the '/explore' page, handling:
    - Chart initialization for segmentation and distribution visualizations.
    - AJAX-driven form submissions for entity searches.
    - Autocomplete suggestions for input fields.
    - Dynamic filter logic for charts on the statistics tab.
  - `chart_themes.js`: Centralizes Chart.js theming, ensuring consistent styling (colors, tooltips, scales). It supports reading CSS variables to adapt to theme changes dynamically.
  - **Alpine.js:** Provides reactive UI behavior, such as toggling menus, managing loading states, and controlling tab visibility.
- **Chart.js:** A JavaScript library used for rendering all KPI visualizations (line, bar, and pie charts), providing interactivity such as tooltips, legends, and theme-aware rendering.

The system design presented here solves the problem of vendor diversity by combining a normalized Data Warehouse with a modular Flask application. Key elements like the database module and the user-friendly frontend help display accurate KPI data. This design sets the stage for the features covered in the next chapter.

Chapter 3

Application Features and Unified Visualization

This chapter details the functionalities of the developed Flask web application, emphasizing how it achieves unified and comparative KPI visualization.

3.1 General User Interface and Navigation

The application provides a clean, responsive, and intuitive user interface built with HTML, TailwindCSS, and Alpine.js for consistent look and interactivity.

- **Layout:** Consistent layout with sticky navbar, footer, and dynamic main content.
- **Navigation:** Links to Home, Dashboard, Explore; highlights active page.
- **Theme:** Toggle between light and dark modes via navbar button.
- **Feedback:** Loading indicators and Toast notifications for user feedback.

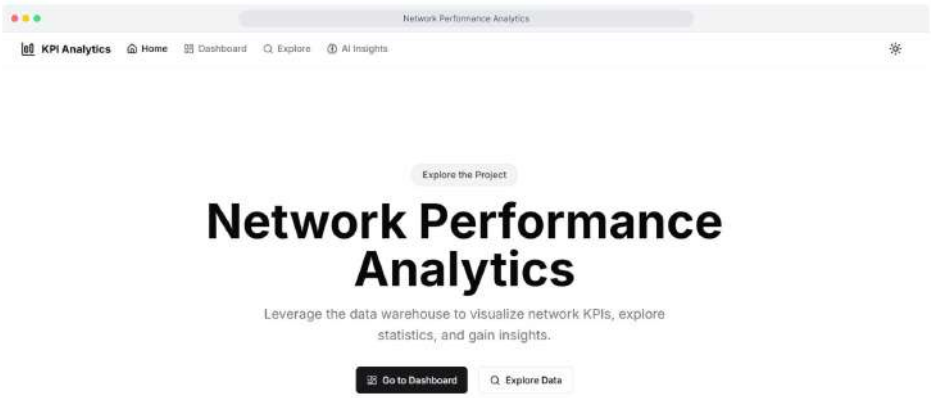


Figure 3.1: Main application Home page

3.2 KPI Dashboard : Interactive Performance Analysis

The KPI Dashboard serves as the primary interface for conducting in-depth time-series analysis of critical network performance indicators (KPIs). Accessible via the `/visualize` web route, its client-side interactivity and dynamic updates are predominantly managed by the JavaScript file located at `static/js/dashboard_setup.js`. This dashboard empowers users to explore network data through a granular and customizable lens.

3.2.1 Comprehensive Filter Panel

To facilitate precise and targeted analysis, the dashboard provides users with a comprehensive filter panel. This panel allows for the meticulous definition of the scope of investigation, ensuring that the visualized data aligns perfectly with the analytical requirements. The various filtering options are detailed below.

- **Vendor Selection:** Users can select network equipment vendors using a dedicated row of buttons. Options include individual vendors such as Huawei, Nokia, and ZTE, or a consolidated "ALL" vendors option. The "ALL" option, internally represented by `vendor_id=3`, is particularly crucial for generating unified views that encompass data from all available vendors, providing a holistic network performance overview. As illustrated in Figure 3.2, the button corresponding to the currently selected vendor (or "ALL") is visually highlighted for immediate user feedback. Figures 3.3 and 3.4 demonstrate example KPI charts generated when "ALL" vendors are selected under default settings, showcasing traffic and PRB/throughput metrics respectively.

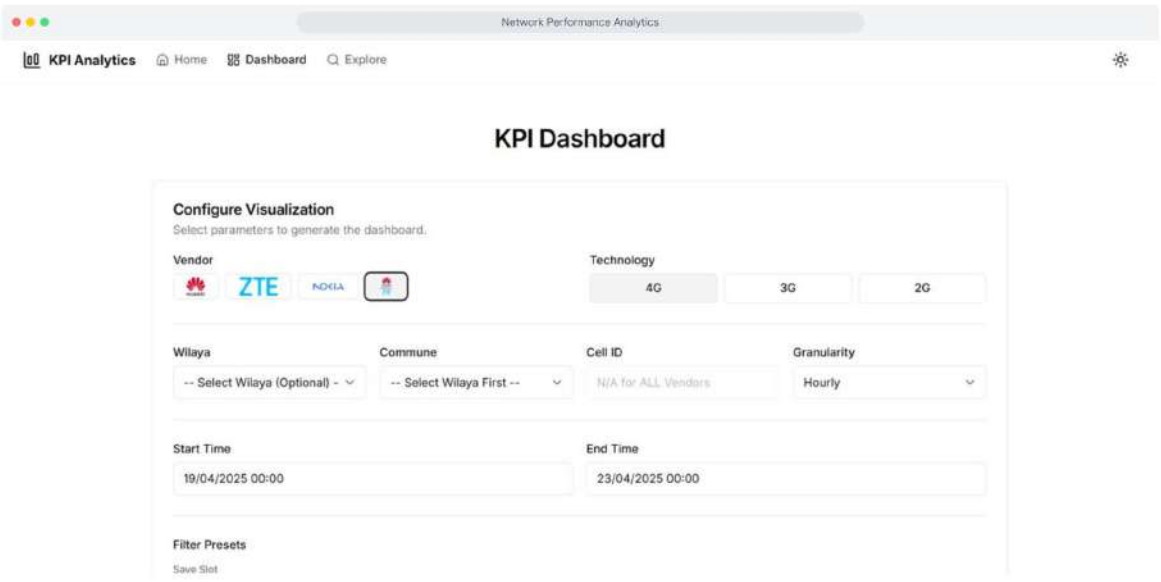


Figure 3.2: Dashboard page : Vendor selection

Figure 3.3: Example of Traffic KPIs displayed for the default “ALLNET” vendor selection.

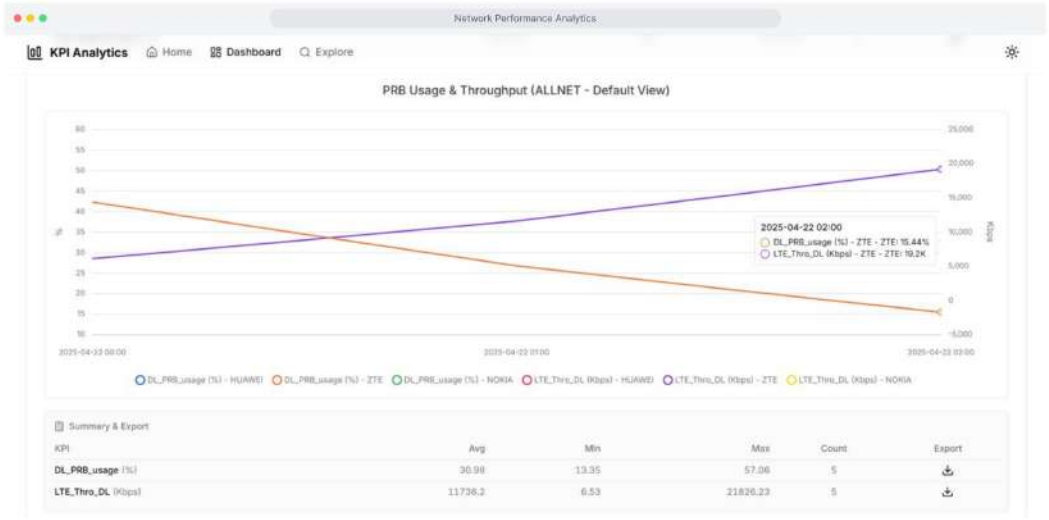


Figure 3.4: Example PRB Usage and Throughput KPIs for "ALLNET" selection.

- **Technology Selection:** A set of radio buttons enables users to filter the displayed KPIs based on network technology. Common selections include 4G (LTE) and 3G (UMTS), with 4G being the default technology selected upon loading the dashboard. This allows analysts to focus on performance metrics pertinent to specific radio access technologies.
- **Geographical Filters:** A sophisticated cascading system of dropdown menus allows for precise geographical filtering of network elements. This hierarchical selection process ensures data relevance to specific regions or sites:
  1. **Wilaya (Region/Province):** The first level of geographical filtering is by Wilaya. This dropdown (Figure 3.7) is populated with data sourced from the `DimGeography`

dimension table in the data warehouse. Selecting a Wilaya dynamically triggers an update of the subsequent Commune dropdown.

2. **Commune (Municipality):** Upon selection of a Wilaya, the Commune dropdown (Figure 3.8) is populated. This is achieved via an asynchronous JavaScript and XML (AJAX) call to the `/get_communes` server endpoint, which returns a list of communes belonging to the selected Wilaya.

3. **Cell ID:** For granular analysis at the cell level, users can utilize a text input field for Cell ID, shown in Figure 3.9. This input is enhanced with an associated datalist (`<datalist id="cell-list">`), providing auto-suggestions. These suggestions are dynamically populated based on the currently selected Technology, Vendor, and upstream Geographical filters (Wilaya, Commune) through AJAX calls to the `/get_cells` endpoint. Users also have the flexibility to manually enter a specific Cell ID. A key feature is that upon selecting a valid Cell ID (either from suggestions or manual input), the system automatically locks and updates the Vendor, Technology, Wilaya, and Commune filters. This ensures consistency by reflecting the true attributes of the selected cell, whose details are fetched via AJAX calls to `/get_cell_details` and geographical context from `/get_geo_for_cell`. This behavior is demonstrated in Figure 3.5, where Wilaya and Commune are auto-populated. Figure 3.6 shows the vendor selection dropdown which can also be part of this auto-update mechanism.

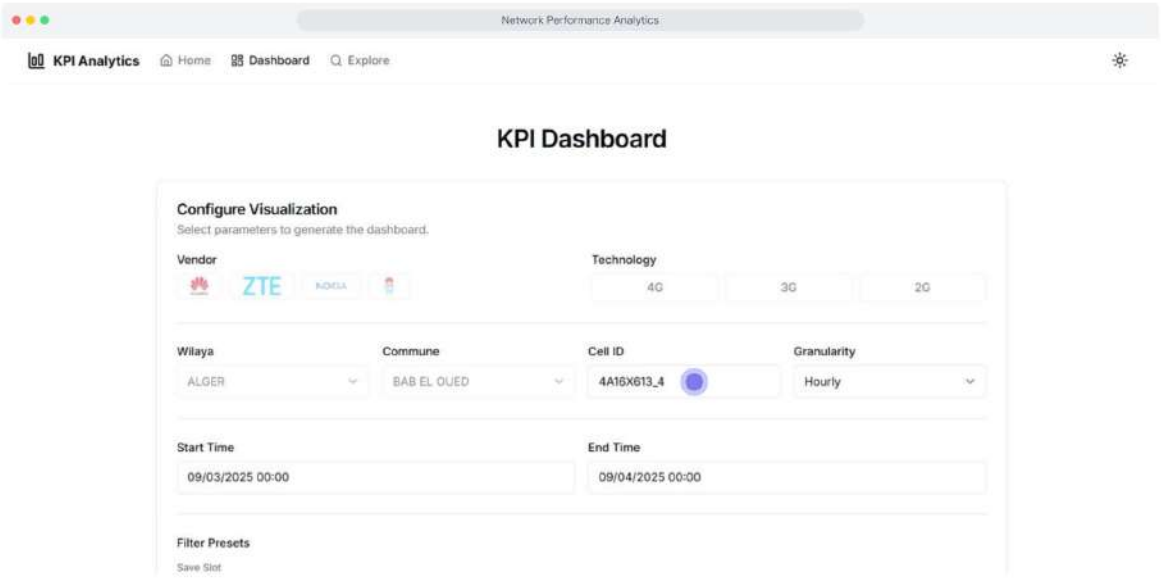


Figure 3.5: Cascading geographical filters based on selected Cell ID.

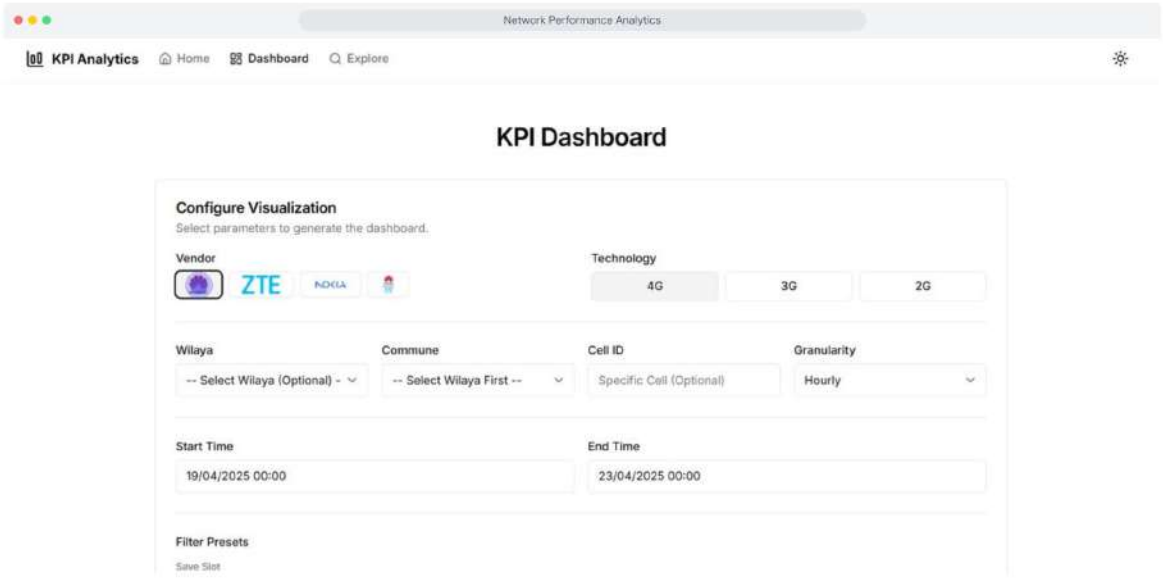


Figure 3.6: Vendor selection dropdown automatically updated by Cell ID.

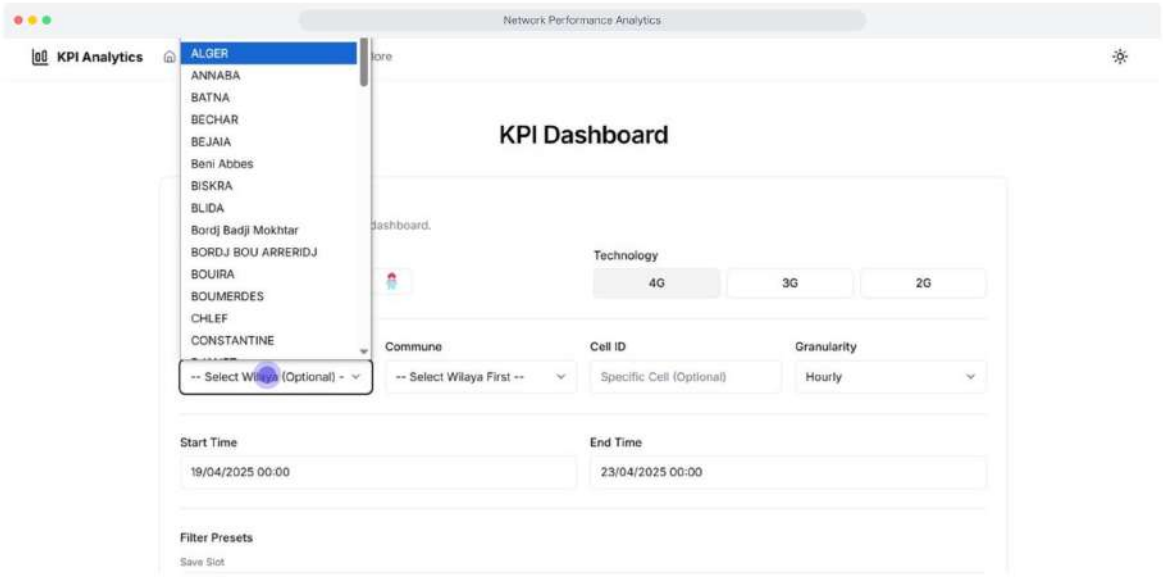


Figure 3.7: Wilaya dropdown list, populated from the `DimGeography` dimension table.

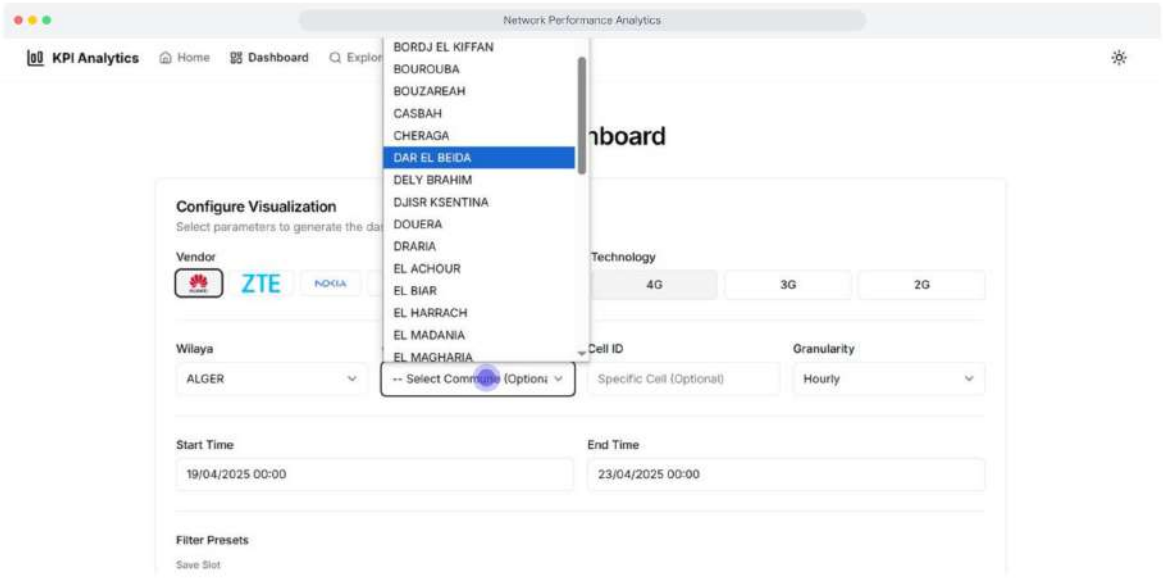


Figure 3.8: Commune dropdown dynamically populated post-Wilaya selection.

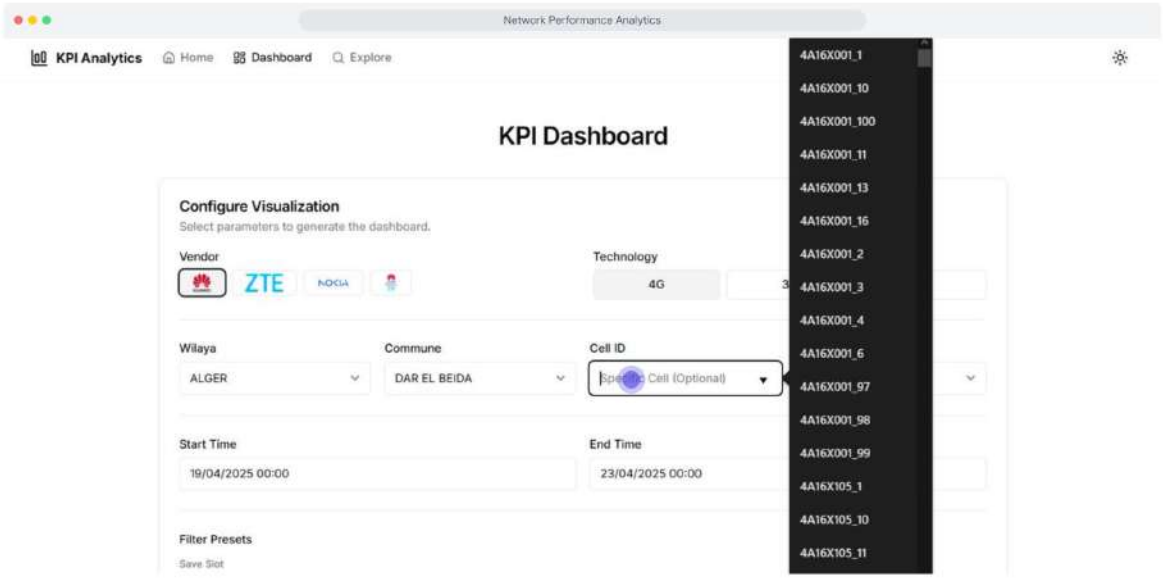


Figure 3.9: Cell ID input with auto-complete and contextual filter retrieval.

- **Dashboard Tabs & Visualization Scope:** The dashboard interface is thoughtfully organized into dedicated tabs to support both high-level KPI aggregation and detailed root cause analysis. This structure enhances user experience and analytical efficiency:
  - When a user selects a specific Cell ID within the filter form, all relevant charts and visualizations pertaining to that cell are automatically rendered within the appropriate tab (e.g., the "Cell" tab, as shown in Figure 3.10).
  - Crucially, the broader geographical context of the selected entity—encompassing its Commune (Figure 3.11) and Wilaya—is also clearly displayed. This provides a

comprehensive understanding of the cell's performance relative to its locality and its potential impact on the wider network.

– Depending on the level of detail specified in the filter form, users can opt to view aggregated data at varying granularities: **Wilaya** level, **Commune** level, or individual **Cell** level.

This modular and hierarchical design allows network engineers and analysts to seamlessly navigate between different levels of detail, facilitating both routine performance monitoring and in-depth troubleshooting tasks.

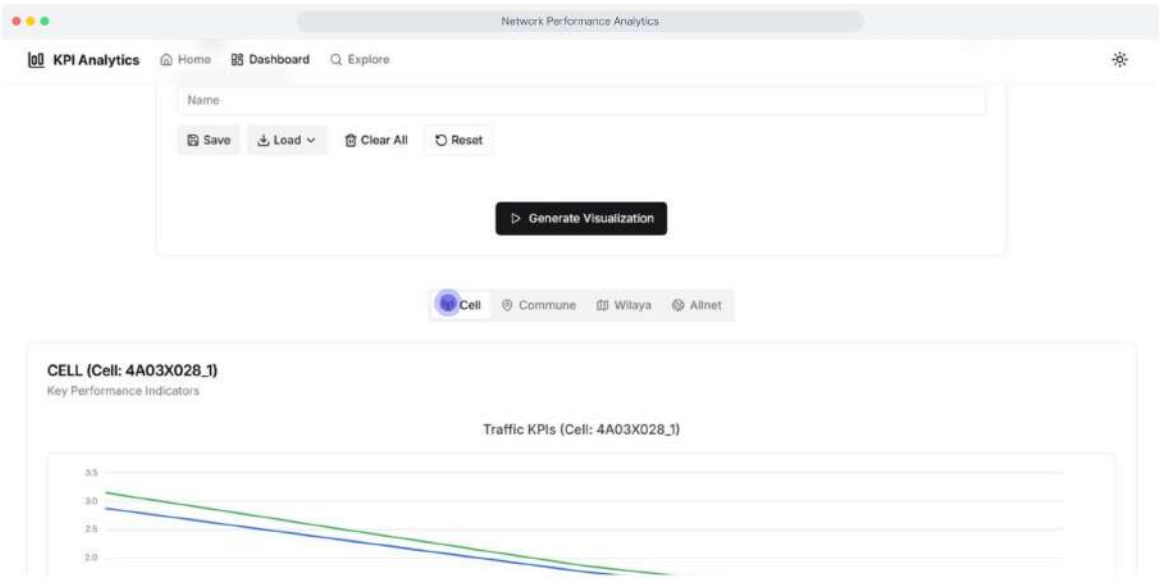


Figure 3.10: "Cell" tab visualizations showing KPIs for a selected cell.

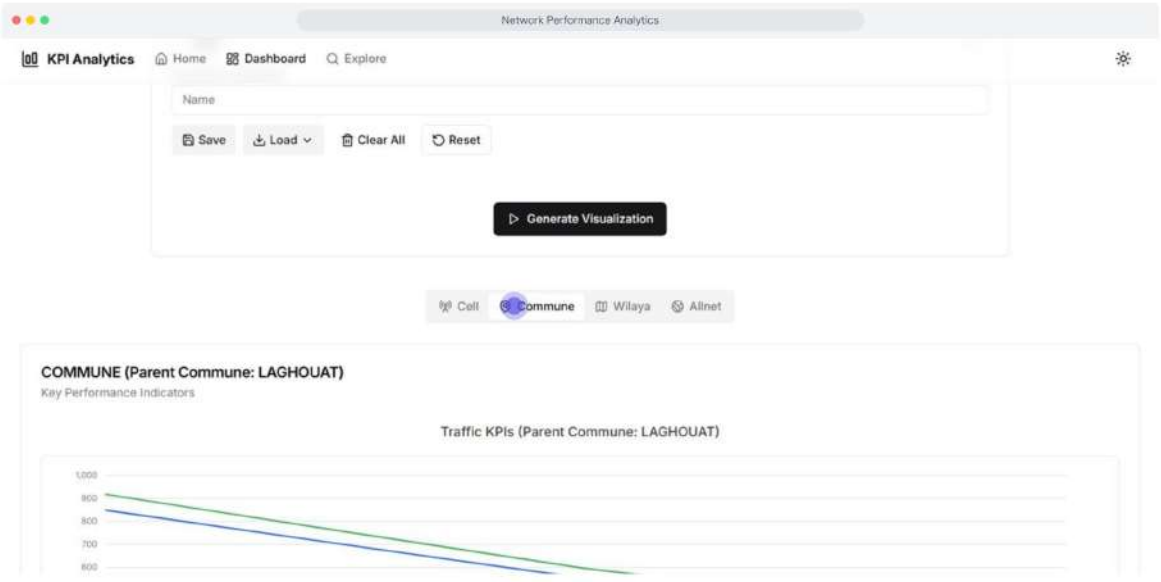


Figure 3.11: "Commune" tab visualizations for aggregated KPIs of cell's commune.

- **Granularity Selection:** A dropdown menu (Figure 3.12) allows users to choose the time granularity for KPI aggregation. The available options are Daily, Hourly, or Busy Hour. This selection directly influences which fact tables from the data warehouse are queried by the backend: `FactKPIDaily` for daily granularity, `FactKPI` for hourly, and `FactBusyHour` for busy hour data (or their aggregated counterparts if applicable).

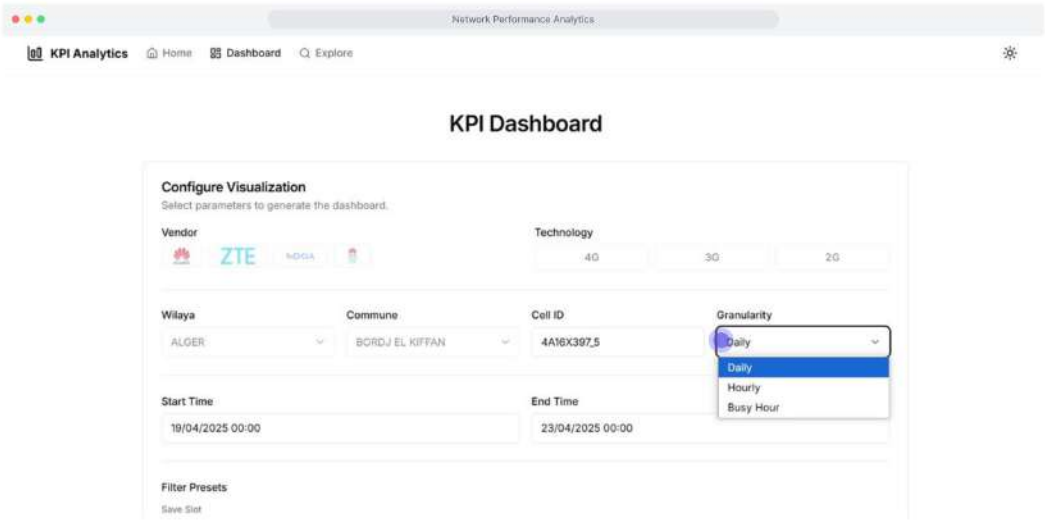


Figure 3.12: Granularity selection dropdown (Daily, Hourly, Busy Hour).

- **Date/Time Range Selection:** Two input fields, seamlessly integrated with the Flatpickr JavaScript library, provide an intuitive calendar and time picker interface (Figure 3.13). Users can select the start and end dates and times for their analysis period. The inputs are formatted as `DD/MM/YYYY HH:00`. By default, the range is typically set to the last 3 days relative to the latest available data timestamp in the `DimTime` dimension table, ensuring users initially see recent trends.

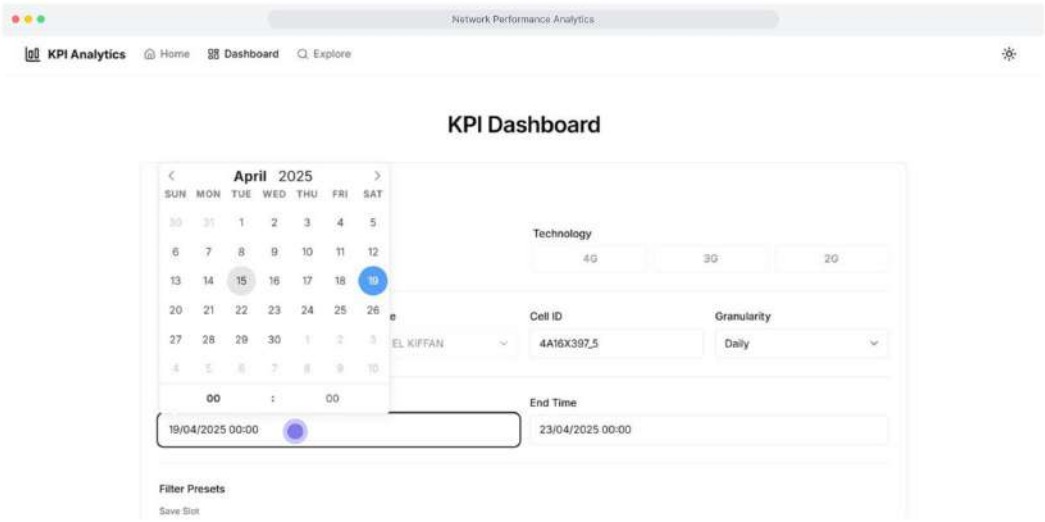


Figure 3.13: Date/Time range selection interface using Flatpickr.

- **Filter Preset Management:** To streamline repetitive analysis workflows, users can save their current filter configurations as presets and load them for future sessions. The system allows for up to three preset slots. As shown in Figure 3.14, controls include a dropdown to select a preset slot, a text input for naming the preset, and buttons to Save, Load, Clear All presets, and Reset to Defaults. These presets are conveniently stored in the browser's local storage, making them persistent across sessions for the same user on the same browser. Figure 3.15 shows an example visualization after loading a preset for a specific cell (`4A16X397_5`) with daily granularity. Figure 3.16 illustrates the act of selecting a Wilaya-level preset. After loading such a preset, visualizations like hourly traffic KPIs (Figure 3.17) and hourly PRB/throughput KPIs (Figure 3.18) for the selected Wilaya (e.g., Algiers) are generated.

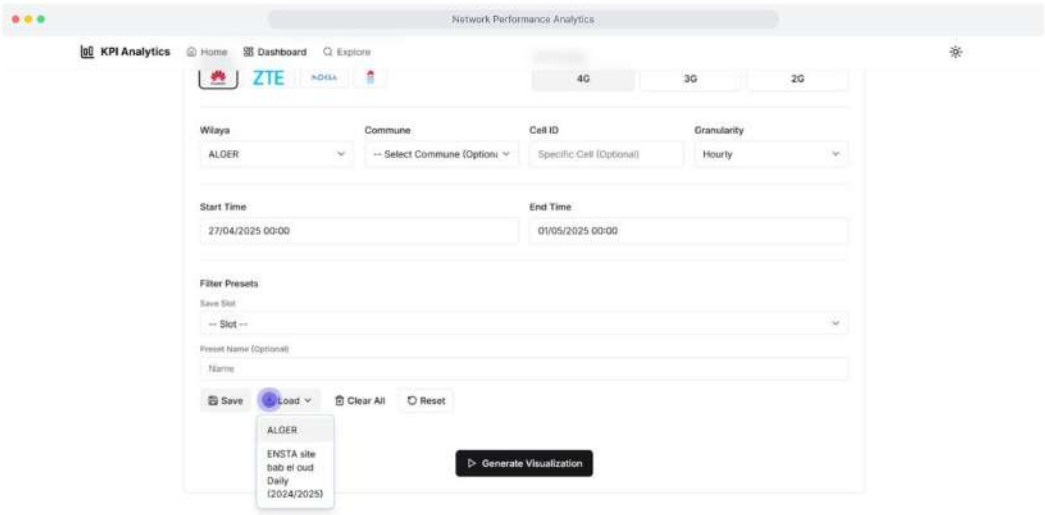


Figure 3.14: Filter preset management controls (Save, Load, Clear, Reset).

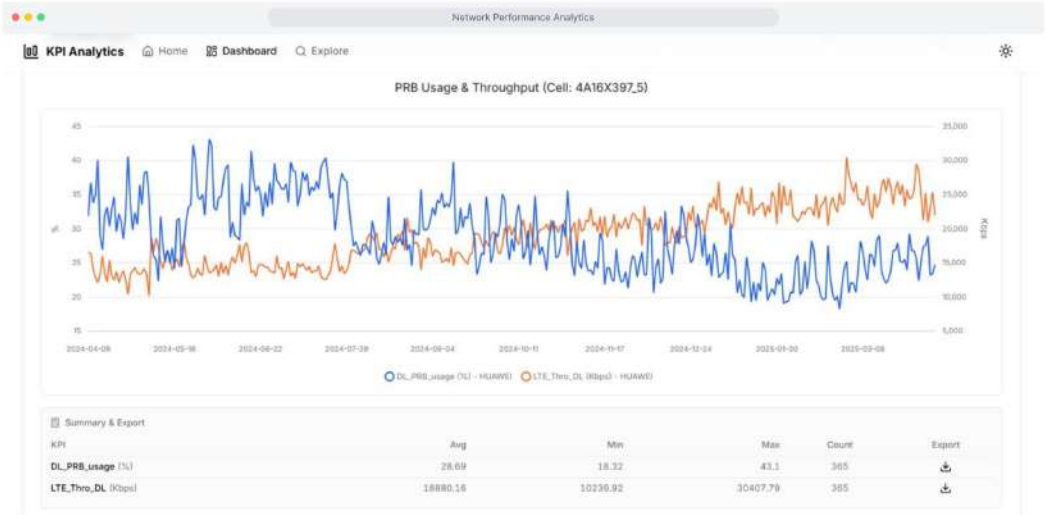


Figure 3.15: Daily PRB Usage/Throughput for cell `4A16X397_5` via preset.

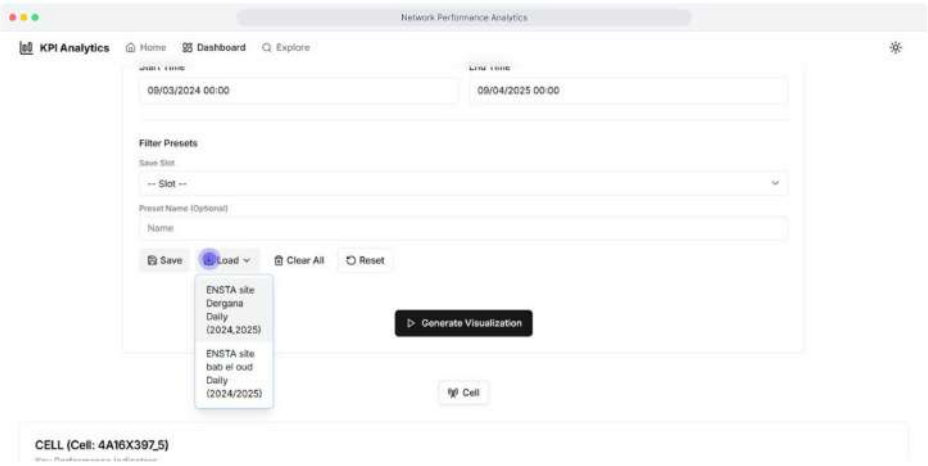


Figure 3.16: Loading a saved Wilaya-level filter preset.

Figure 3.17: Hourly Traffic KPIs for Algiers loaded from Wilaya preset.

Figure 3.18: Hourly PRB Usage/Throughput for Algiers from Wilaya preset.

- **Generate Visualization Button and Feedback Mechanisms:** The *Generate Visualization* button is a pivotal element in the user workflow. Clicking this button initiates the process of submitting the meticulously selected filter criteria to the server, which then queries the data warehouse and prepares the data for chart generation.

  Prior to submission, the client-side script (`dashboard_setup.js`) executes robust validation routines. These checks ensure data integrity and usability:

  – All mandatory fields (such as Vendor, Technology, and Date Range) must be completed.

  – The selected date interval must be logically consistent (i.e., the start date must precede or be the same as the end date).

  – If geographical or cell-level filters are applied, they must adhere to a valid hierarchical structure (e.g., a selected Commune must belong to the selected Wilaya).

  Upon successful validation and submission, a loading spinner is prominently displayed. This visual cue indicates to the user that the dashboard is actively processing the request and fetching the relevant data from the backend data warehouse, significantly improving the user experience by providing immediate feedback on system activity.

  In scenarios where the chosen combination of filters does not yield any corresponding data in the data warehouse (e.g., no recorded KPIs for a specific cell in the selected time range), the system handles this gracefully. A non-intrusive notification, such as the alert message shown in Figure 3.19, is displayed. This informs the user that no data was found for their selected criteria and encourages them to revise their filter inputs.

  This feedback mechanism is typically triggered by server-side checks that determine data availability, with the notification rendered client-side using JavaScript alert modals or inline message boxes, depending on the specific context. This approach ensures a responsive, transparent, and guided user experience, effectively bridging the visualization logic with the underlying data availability in the warehouse.

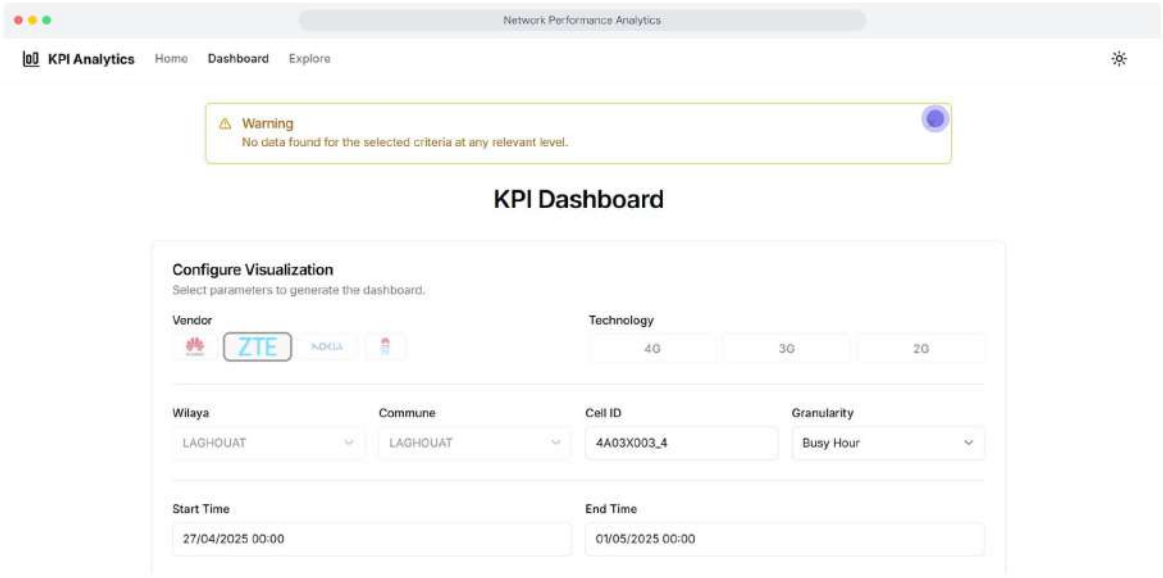


Figure 3.19: Alert message for no data available with current filters.

3.2.2 Dynamic Chart Display Area

After submitting the filter form—or by default on initial load—the backend retrieves and processes the required KPI data. This is done via the `db.get_multiple_kpi_results` function, which queries the data warehouse based on selected filters. The resulting data is then formatted using `utils.format_chart_data_multiple_kpis` and sent to the frontend, where Chart.js renders dynamic, interactive charts.

The chart area is structured for readability and detailed analysis. Key features include:

- **KPI Grouping:** Charts are organized into logical categories for ease of interpretation, such as:
  - *Traffic KPIs:* e.g., `LTE_Traffic_Volume_DL`, `LTE_Traffic_Volume_UL`.
  - *PRB/Throughput KPIs:* e.g., `DL_PRB_usage`, `LTE_Thro_DL`.
  - *Access/User KPIs:* e.g., `RRC_SR(%)`, `E_RAB_SR(%)`, `Avg_User`.
- **Interactive Charts:** Users can hover to view tooltips with timestamp, KPI value, and vendor. Legends allow toggling specific data series for focused comparison.
- **Summary Statistics:** Below each chart, a table summarizes key metrics—average, min, max, and count—calculated by `utils.calculate_summary_stats`.
- **CSV Export:** Each KPI series can be exported via a "Download CSV" button, which triggers an HTTP request to the `/export/csv` endpoint.

Figure 3.21 shows an example chart with summary statistics and export options.

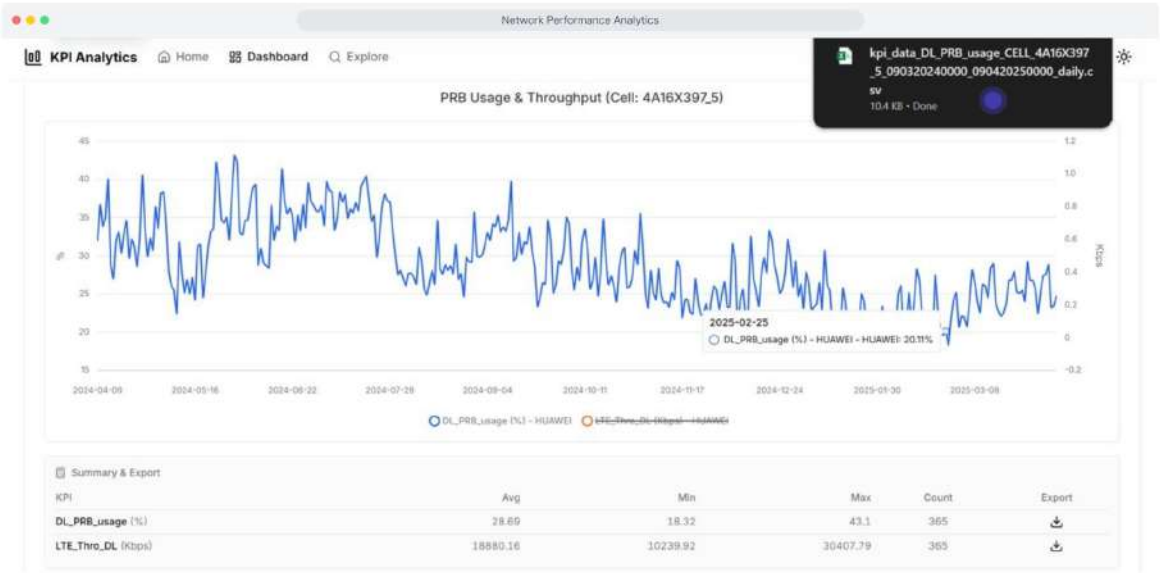


Figure 3.20: `LTE_Thro_DL` chart with summary stats and CSV export buttons.

time_point	kpi_value	metric	vendor_name
2024-04-09	31.9211	%	HUAWEI
2024-04-10	36.6657	%	HUAWEI
2024-04-11	33.8822	%	HUAWEI
2024-04-12	34.9491	%	HUAWEI
2024-04-13	40.0103	%	HUAWEI
2024-04-14	28.951	%	HUAWEI
2024-04-15	27.0379	%	HUAWEI
2024-04-16	31.9988	%	HUAWEI
2024-04-17	33.0977	%	HUAWEI
2024-04-18	30.3724	%	HUAWEI
2024-04-19	32.9217	%	HUAWEI
2024-04-20	34.6081	%	HUAWEI
2024-04-21	29.7085	%	HUAWEI
2024-04-22	32.1767	%	HUAWEI
2024-04-23	31.0893	%	HUAWEI
2024-04-24	28.5823	%	HUAWEI
2024-04-25	32.157	%	HUAWEI

Figure 3.21: CSV export.

3.3 Explore Page (`/explore`): Vendor-Agnostic Network Insights

The Explore page, accessible via the `/explore` route and primarily managed by the client-side script `static/js/explore_setup.js`, provides comprehensive tools for high-level network exploration and aggregated statistical summaries. Unlike the KPI Dashboard, this page is vendor-agnostic and supports cross-vendor comparative insights. It is organized into two main tabs: *Search Entities* and *Statistics & Health*.

3.3.1 Search Entities Tab

This tab provides a versatile interface to search for network entities across technologies and vendors using multiple filters.

- **Search Form:** Search criteria include:
  - Wilaya name
  - Commune name
  - Site code or name
  - Cell ID or name
  - Technology (dropdown)
  - Vendor (dropdown)

  Autocomplete suggestions are provided via dedicated API endpoints such as `/autocomplete/` an example of this autocomplete functionality can be seen for the "Cell (ID or Name)" field in Figure 3.22.

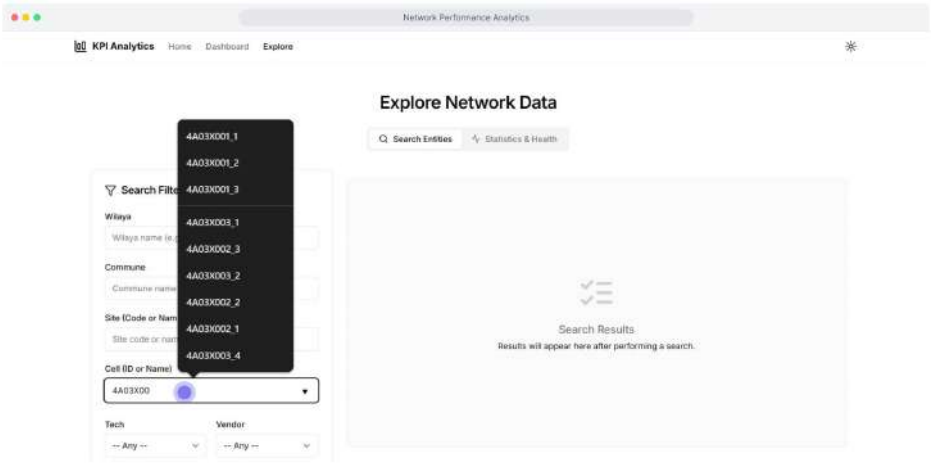


Figure 3.22: Explore Page: Entity search form with multi-criteria filtering and autocomplete support.

- **Unified Search Results:** Searches are handled by `db.search_entities_v3`, which returns cross-vendor results including:
  - Matched wilayas (name, code)
  - Matched communes (name, code, parent wilaya)
  - Matched sites (code, name)
  - Matched active cells (cell ID, name, technology, vendor, site, commune, wilaya, coordinates in DMS and decimal, and Google Maps links)

  Results are demonstrated in Figure 3.23.

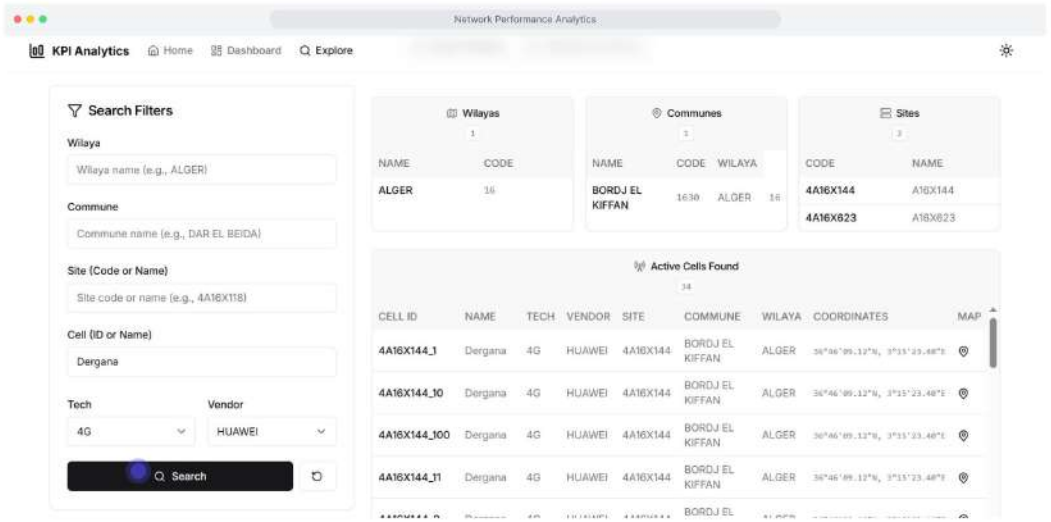


Figure 3.23: Explore Page: Unified search results with matched cells and detailed metadata.

- **Example: Locating a Specific Cell Near Our School**
  As a practical example, we searched for the cell with ID `4A16X397_5`, which is located near our school. Figure 3.24 shows the search form after inputting the cell ID, while Figure 3.25 shows the resulting location on Google Maps.

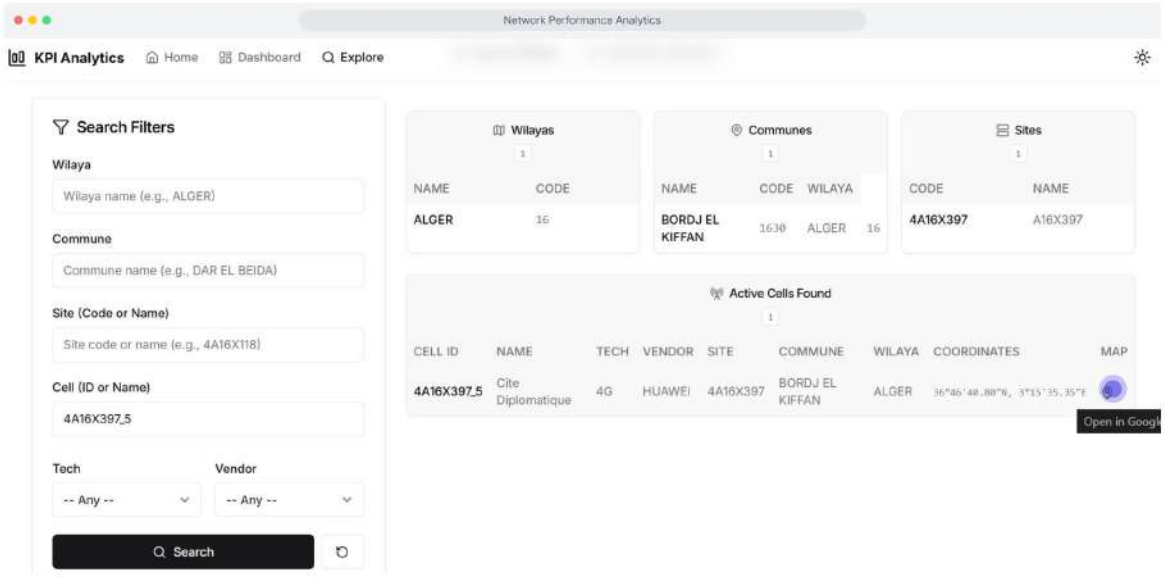


Figure 3.24: Explore Page: Search input for the cell `4A16X397_5`.

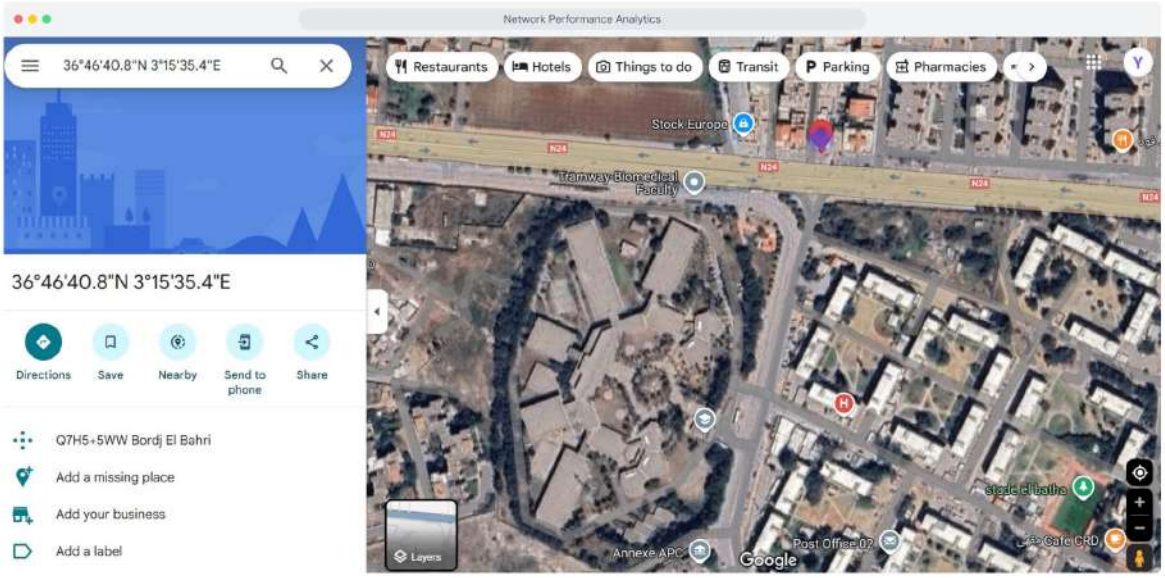


Figure 3.25: Google Maps: Location of the cell `4A16X397_5`

3.3.2 Statistics & Health Tab

This tab offers an overview of the network's composition and health status through multiple interactive and informative components:

- **Quick Stats Section:** Displays aggregate counts of core network elements such as total active cells, sites, wilayas (provinces), communes (municipalities), supported technologies, and distinct vendors (excluding the aggregated "ALL" option). These metrics are dynamically fetched from the backend using `db.get_total_counts`, providing a quick snapshot of network scale.

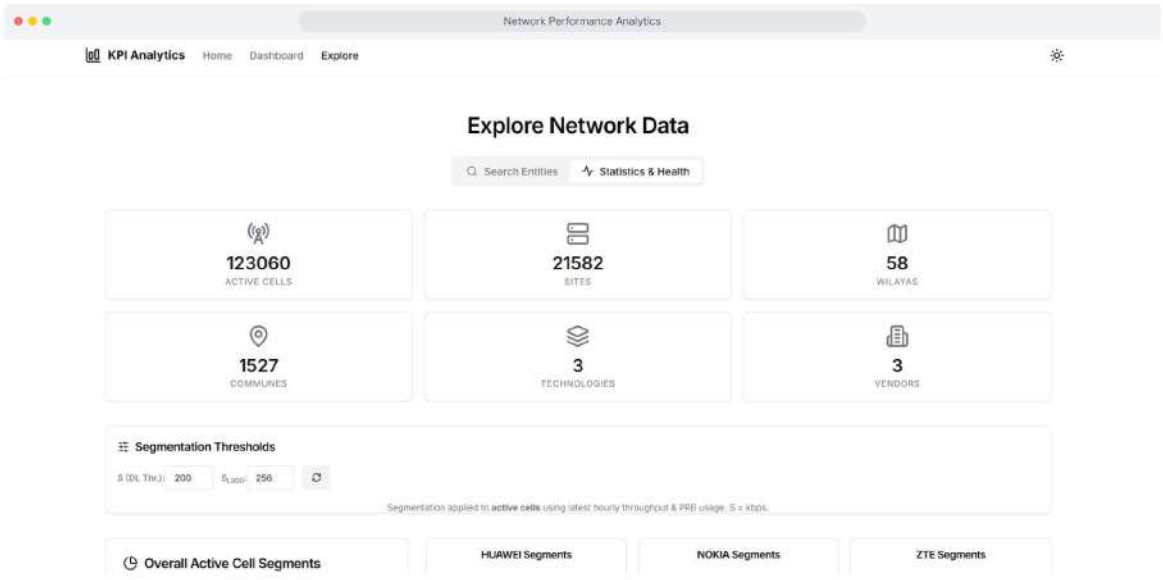


Figure 3.26: Explore Page: Quick Stats section with network entity counts.

- **Cell Segmentation Thresholds & Statistics:** Users can define throughput thresholds used to classify cells into health categories. Input fields include:
  - `S`: General downlink throughput threshold (in Kbps).
  - `S_L900`: Specialized threshold for L900 band cells.

  After submission, segmentation statistics are refreshed accordingly.

  This section includes:
  - *Overall Active Cell Segmentation Pie Chart:* Shows the distribution of active cells into "Good," "Bad," "Critical," and "Indeterminate" categories based on the latest hourly KPI data. Data is aggregated using `db.get_detailed_cell_statistics`.
  - *Vendor-Specific Segmentation Pie Charts:* These smaller pie charts show the same segmentation breakdown for each major vendor (Huawei, Nokia, ZTE), allowing for per-vendor health assessments.

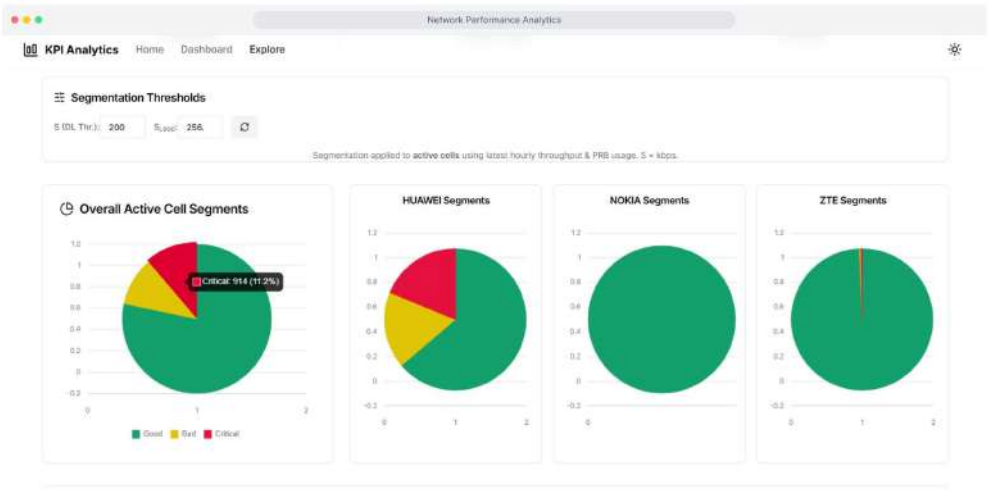


Figure 3.27: Explore Page: Overall and vendor-specific cell segmentation pie charts.

- **Detailed Counts Table:** A tabular summary displaying the number of created and active cells, as well as the distribution of cells by quality category—Good, Bad, and Critical—segmented by both technology and vendor. This data is retrieved from the function `db.get_detailed_cell_statistics`, and it provides a quick overview of the current network health across all domains. The visualization of this data is shown in Figure 3.28.

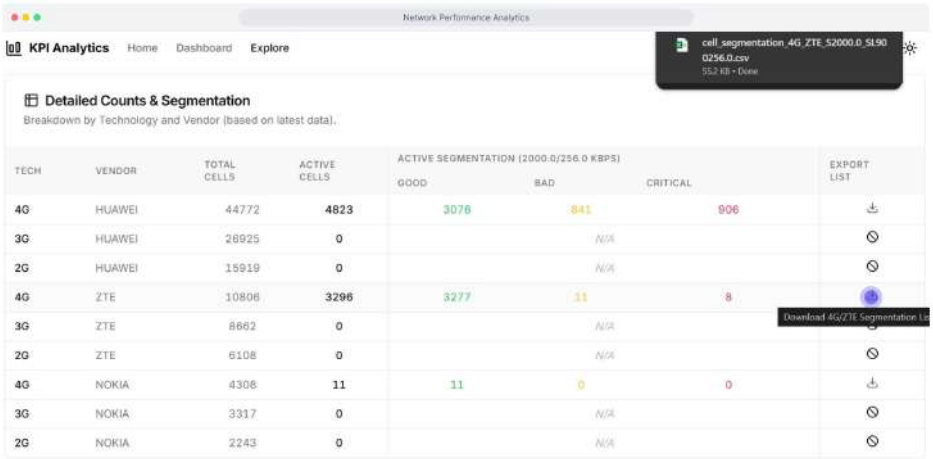


Figure 3.28: Explore Page: Detailed segmentation table by quality, tech, vendor.

- **Segmentation List Export:** Each row of the table includes a download button that allows exporting the list of cells for a given segmentation category (e.g., Good 4G Huawei cells) as a CSV file.
  This export is handled via the endpoint `/export/segmentation/<tech_id>/<vendor_id>`, making it convenient for external analysis or reporting purposes. An example of a resulting CSV file export is illustrated in Figure 3.29.

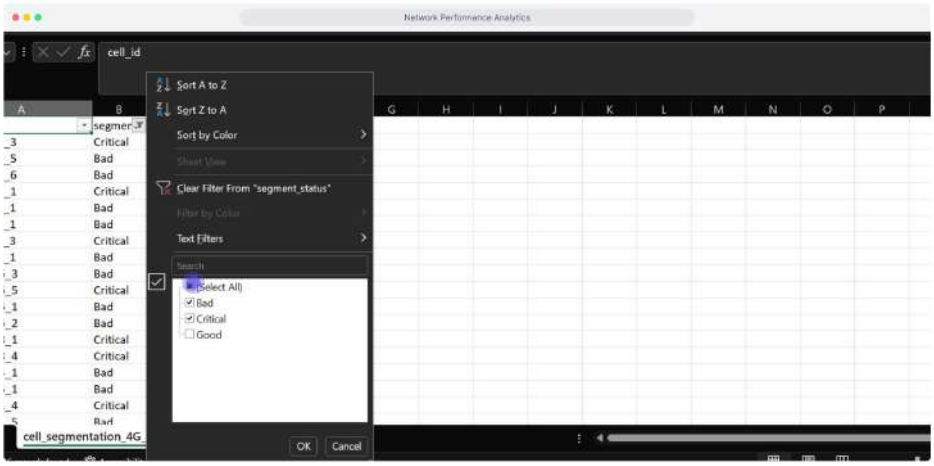


Figure 3.29: CSV export result: Example file generated for a specific segmentation category.

- **Interactive Cell Distribution Pie Chart:** Lets users visualize cell distribution based on:
  - *Vendor filters:* Huawei, Nokia, ZTE.
  - *Technology filters:* 2G, 3G, 4G.
  - *Cell status filters:* Toggle between "Total Cells" and "Active Cells."

  The chart updates dynamically using client-side processing via `explore_setup.js` and the data object `detailedStatsData.by_tech_vendor`.

Figure 3.30: Explore Page: Interactive pie chart for ZTE's total cell technology.

The features described in this chapter show how the application brings together KPI data from multiple vendors in a clear, interactive way. The KPI Dashboard allows detailed filtering and comparison, while the Explore page provides vendor-independent search and statistics. Together, they give engineers a practical tool for monitoring network performance.

Chapter 4

Discussion, Limitations, and Conclusion

4.1 Evaluation of the Proposed Framework

The web application successfully fulfills its primary objective of providing a unified, accurate, and interactive LTE performance monitoring solution. Through the abstraction of vendor-specific details via a semantic normalization layer in the Data Warehouse (DWH), the platform delivers consistent and actionable insights to network operators.

Originally, the scope of this project aimed to support all radio access technologies—2G, 3G, and 4G. However, due to time constraints, the current implementation focuses solely on LTE (4G) but covers all major vendors in the network: Huawei, Nokia, and ZTE. Despite this narrowing of scope, the underlying architecture of both the data warehouse and the web application was designed to be adaptable and scalable. It supports easy extension to additional technologies such as 2G, 3G, and even future deployments like 5G. Similarly, the framework is built to accommodate new vendors should the operator decide to onboard others in the future.

This design ensures that the solution is not limited to Djezzy's network but can be generalized across mobile operators who face similar challenges in harmonizing heterogeneous KPI definitions. The only significant bottleneck in scaling this solution is the time-intensive process of defining and validating normalization logic for each new KPI, vendor, or technology.

4.2 Limitations

- The system currently supports a fixed set of normalized KPIs. Expanding this catalog would significantly enhance the platform's analytical capabilities.
- Real-time data ingestion and processing are not yet implemented; the system operates in batch mode with periodic updates.
- The accuracy of unified KPIs is contingent on the correctness and completeness of the normalization formulas.

- The framework focuses on aggregated KPI visualization and does not delve into vendor-specific counter-level diagnostics.

4.3 Conclusion and Future Work

This thesis has introduced a scalable and extensible platform for vendor-agnostic LTE KPI visualization. The normalization framework, in conjunction with an interactive front-end, enables unified analysis across multiple vendors—resolving a key pain point in multi-vendor network environments.

Future work will focus on:

- Integrating AI modules—such as anomaly detection and KPI forecasting—developed as part of our final education project [1].
- Enabling real-time data ingestion and streaming visualizations to support near-instantaneous network monitoring.
- Applying machine learning to automate and refine the normalization process, thereby improving accuracy and scalability.
- Incorporating a notification and alert system based on dynamic threshold rules to assist in proactive network management.

Overall, the architecture and approach presented in this work demonstrate high potential for broader application across mobile operators, offering a future-proof solution for performance monitoring in complex, heterogeneous network environments.

Bibliography

[1] A. Y. Alaouchiche and M. W. Kessoum, "Machine Learning-Based Performance Optimization in LTE Networks," Final Education Project, ENSTA, 2025.

[2] D. O. T. Algérie, "Internal KPI Documentation and Formulas," Accessed during engineering internship at Djezzy, 2025, unpublished company resource.

[3] A. R. et al., *Flask: Web Development, One Drop at a Time*, Pallets Projects, 2024, https://flask.palletsprojects.com/.

[4] P. G. D. Group, *PostgreSQL Documentation*, 2024, https://www.postgresql.org/docs/.

[5] F. D. G. et al., *psycopg2: PostgreSQL database adapter for Python*, 2024, https://www.psycopg.org/docs/.

[6] C. Contributors, *Chart.js: Simple yet flexible JavaScript charting*, 2024, https://www.chartjs.org/.

[7] P. Projects, *Jinja2 Documentation*, 2024, https://jinja.palletsprojects.com/.

[8] C. Porzio, *Alpine.js: A rugged, minimal framework for JavaScript*, 2024, https://alpinejs.dev/.

[9] T. Labs, *Tailwind CSS: A Utility-First CSS Framework*, 2024, https://tailwindcss.com/.

[10] G. Schier, *flatpickr: Lightweight and powerful datetime picker*, 2024, https://flatpickr.js.org/.