Interreg Euro-MED

Co-funded by the European Union

LIBECCIO

sustainabiLIty of tourism By Enhancing Cooperation and dIgital transfOrmation

D2.1.1 Web enabled semantically enriched application software platform

Deliverable

30/09/2025

Project	LIBECCIO – 101093016
Work Package	WP2 – Pilot testing of up-scaled Destination Management Support System for tourism sustainability
Activity	Act. 2.1 - Enhancement of the capitalized tools to equip the up-scaled DMSS with all defined functionalities
Deliverable	D2.1.1 – Web enabled semantically enriched application software platform
Editor(s)	ATHENA
Contributor(s)	RER, CNA ABRUZZO, ADT, CCIT, BRTA, RDA Green Karst, KOTOR, RAIS, RWG
Version	V1.0
Date	30/9/2025

Executive Summary

The Deliverable D.2.1.1 presents the implementation of the Web enabled semantically enriched application software platform of the up-scaled Libeccio DMSS. The Libeccio DMSS is a comprehensive data processing and analytics platform specifically engineered to revolutionize tourism policy development through evidence-based decision making and real-time intelligence capabilities. This integrated system addresses the critical challenge of fragmented tourism data by seamlessly combining multiple data collection mechanisms, including web scraping from major platforms like TripAdvisor, Booking.com, and Airbnb, API integrations with services such as Google APIs, and offline data input capabilities for traditional sources like spreadsheets and manual entries. The DMSS architecture features a sophisticated multi-component design encompassing a Grafana-powered Visual Analytics GUI for intuitive data exploration, a robust Data Analytics module for machine learning and statistical processing, a centralized Database layer utilizing Elasticsearch and MongoDB for efficient raw and processed data storage, and a Process and Data Orchestrator that manages the entire data workflow from collection through analysis to storage.

This technological foundation enables tourism authorities, destination managers, and policymakers to transform diverse, real-time tourism data streams into actionable insights. These insights support proactive policy responses, strategic planning, and sustainable destination management practices. Ultimately, this facilitates a paradigm shift from intuition-based to data-driven governance in the rapidly evolving global tourism sector

Table of Contents

List of figures

List of tables

1. Introduction

The Libeccio DMSS represents a comprehensive technological solution specifically designed to enhance tourism policy development and implementation through data-driven insights and analytics. This sophisticated platform addresses the growing need for evidence-based decision making in the tourism sector by providing policymakers, destination managers, and stakeholders with robust tools for collecting, analysing, and visualizing diverse tourism-related data. The DMSS integrates multiple data sources, advanced analytics capabilities, and intuitive visualization interfaces to create a unified ecosystem that transforms raw tourism data into actionable intelligence for strategic planning and operational decision making.

At its core, the DMSS functions as an integrated data processing and analytics pipeline that seamlessly combines data ingestion, transformation, storage, and presentation capabilities. The system is engineered to handle the complexity and diversity of tourism data, from traditional statistical sources and government databases to real-time web data from booking platforms, review sites, and social media channels. By leveraging modern technologies including machine learning algorithms, distributed databases, and cloud-based analytics, the DMSS provides a scalable foundation for comprehensive tourism intelligence that can adapt to the evolving needs of destination management and policy formulation.

The contemporary tourism landscape presents unprecedented challenges for policymakers who must navigate complex interactions between visitor flows, local communities, economic impacts, and environmental sustainability. Traditional approaches to tourism policy development often rely on limited data sources, periodic surveys, and retrospective analysis that may not capture the dynamic nature of modern tourism patterns. The DMSS addresses these limitations by providing continuous data integration capabilities, predictive analytics, and comprehensive monitoring tools that enable proactive policy responses and evidence-based strategic planning.

Modern tourism destinations generate vast amounts of data through digital platforms, online booking systems, social media interactions, and mobile applications. However, this wealth of information often remains fragmented across multiple sources and formats, making it difficult for policymakers to gain comprehensive insights into tourism dynamics. The DMSS bridges this gap by creating a unified data ecosystem that aggregates information from diverse sources, applies sophisticated analytical techniques, and presents findings through accessible visualization tools that support informed decision making at all levels of tourism governance.

This report provides a comprehensive examination of the DMSS architecture, functionality, and implementation approach. The following sections present the conceptual design that underlies the system's capabilities, detailing how various components work together to create a cohesive platform for tourism analytics. The system architecture section explores the technical infrastructure, database design, and integration frameworks that enable the DMSS to handle complex data processing workflows while maintaining performance and reliability standards essential for policy support applications.

The deployment information section outlines the implementation strategy, including infrastructure requirements, scalability considerations, and operational procedures necessary for successful system deployment in various organizational contexts. This technical foundation enables tourism authorities and destination management organizations to implement the DMSS according to their specific requirements and technical capabilities while ensuring optimal performance and user experience.

Subsequently, the report provides detailed explanations of each system module, examining how individual components contribute to the overall functionality of the DMSS. These sections explore the data collection mechanisms, processing algorithms, storage strategies, and visualization capabilities that collectively enable the system to transform diverse tourism data into meaningful insights. Through this comprehensive analysis, readers will gain a thorough understanding of how the DMSS can be leveraged to enhance tourism policy development, monitor destination performance, and support strategic decision making in the complex and dynamic tourism environment.

The integration of these components creates a powerful platform that empowers tourism stakeholders to move beyond intuition-based decision making toward evidence-driven policy development, ultimately leading to more effective tourism management strategies that balance economic development, visitor satisfaction, and sustainable destination practices.

2. DMSS Architecture

2.1. Conceptual Architecture

The DMSS system delivers a complete data processing and analytics pipeline that seamlessly integrates multiple components for data collection, transformation, storage, and visualization. Figure 1 presents the conceptual architecture of this proposed system. Users engage with the platform through a Visual Analytics GUI built on Grafana, which delivers an intuitive graphical interface for data exploration, analysis, and visualization.

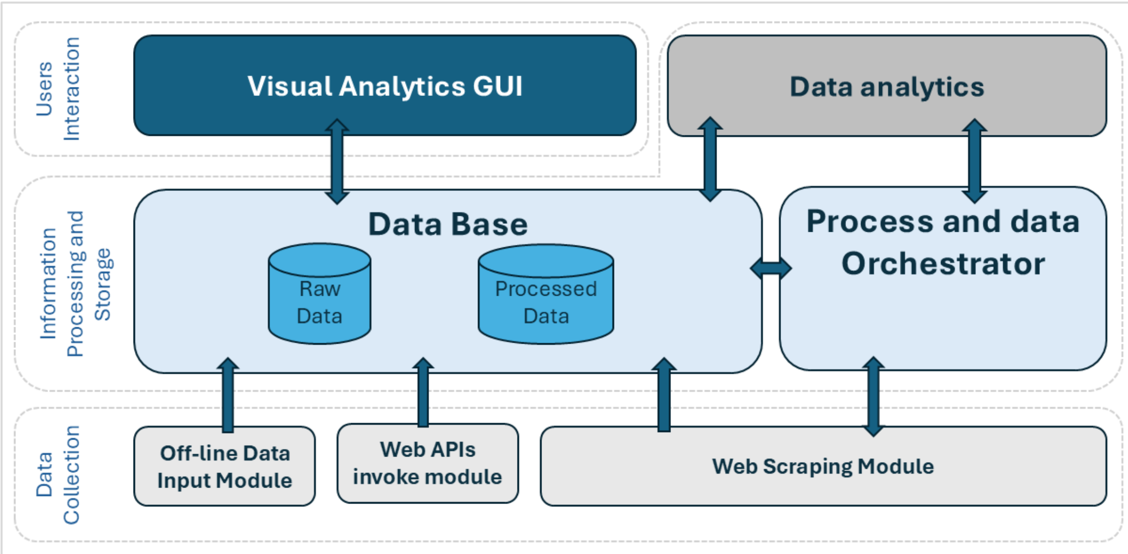

Figure 1 Libeccio DMSS Conceptual Architecture

The system architecture consists of several interconnected modules. The **Visual Analytics GUI** functions as the main user interface, presenting data-driven insights and findings. Working alongside this interface is the **Data Analytics** module, which executes machine learning algorithms, statistical computations, and other analytical processes.

Data storage is managed through a centralized Database layer organized into two distinct sections: raw data repositories and processed data repositories. This structure maintains clear boundaries between original input data and refined, transformed, or analysed datasets. The database leverages both NoSQL and Relational Data Bases to ensure efficient indexing and data management capabilities.

The **Process and Data Orchestrator** operates as the system's central hub, providing internal mechanisms for coordinating the entire data processing workflow, encompassing data gathering, preprocessing, analysis, and storage operations.

Data acquisition and input functions are positioned at the foundation of the architecture. An **Offline data input module** enables external file imports, including XLSX spreadsheets and manual data entry through web-based forms. The **Web APIs invoke module** incorporates software clients that connect to Open Data APIs, such as Google's API services, for supplementary data collection. Concurrently, the system features a **Web scraping module** utilizing Apify technology, which extracts data from multiple online sources including TripAdvisor, Booking.com, Airbnb, Google, Windy, and Wikipedia. The system also supports API integration capabilities for streamlined data exchange and enhancement.

The data processing workflow initiates with collection from original sources through web scraping, API calls, or offline inputs. Data then passes through the Process and Data Orchestrator, which transforms and directs information to the designated database. Subsequently, the processed data becomes available through the Visual Analytics GUI while simultaneously supporting sophisticated analytics within the Data Analytics module. This comprehensive system provides a scalable, user-centric pipeline for gathering, processing, and analysing varied datasets.

2.2. Modular Architecture

The DMSS was developed with a combination of technologies, tools, languages, libraries, and development software. For the development, open-source tools and frameworks that allow fast and easy integration to the developing tools were used. Thus, DMSS is composed of multi-modular architecture, where each module is independent and the communication of the modules is based in well-defined protocols, ensuring the scalability and reliability of the system, as depicted in Figure 2. DMSS's modules are categorized into three basic modules groups, according to their main functionality:

- **Data collection Module Group** provides functionalities of automatic data collection through web APIs or manual dataset uploads.
- **Data Management Module Group** provides data processing orchestration and data storage functionalities.
- **Data Analysis and Visualization Module Group** provides state-of-the-art Visual Analytics approaches for extracting insights from the data interactive presentation.

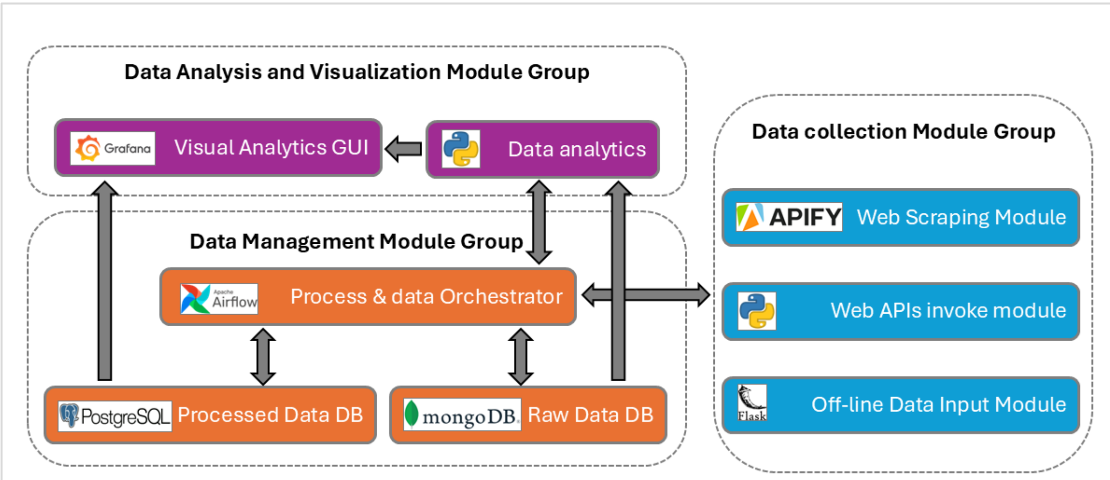

Figure 2 Libeccio DMSS multimodule system architecture

The **Data collection Module Group** handles a variety of data sources. The application tasked with data retrieval from information available public on internet, is APIFY[1], which is a robust cloud-based solution for web automation and scraping, to collect structured and semi-structured information from targeted websites. By implementing APIFY actors,

[1] https://apify.com/

the system automates the extraction of data such as accommodation listings, prices, user reviews, etc. across platforms including Airbnb, Booking.com, Google Places and TripAdvisor. This application offers scraping mechanisms that ensure data freshness and consistency. Each run is parameterized based on the specific query context (e.g., location, date range, or number of items), enabling flexible and targeted data retrieval. The output of this layer is a set of cleaned JSON datasets, which are then passed to the subsequent processing and storing components of the system.

However, the system must also support manual data entry from various sources, such as municipalities and relevant tourism organizations. To facilitate this, a mechanism is provided for uploading well-structured Excel files, which are then parsed and stored into the database. Furthermore, many data sources offer APIs to access stored information. Therefore, the system invokes custom designed API clients to automatically retrieve, validate, and ingest data from these services into the processing pipeline.

The **Data Management Module Group** contains data processing orchestrator and storage. The DMSS supports two different storage solutions. A noSQL database, set up on MongoDB server[2], maintains all gathered data, which later and after processing only useful information is accumulated in a PostgreSQL[3] database. The MongoDB document-based structure is suitable for handling unstructured data gathered from various sources. Such data, however, needs to be mapped in a structured format and then handled by a robust relational database system that provides strong support for structured queries. To enhance fault tolerance and availability, the system is designed with storage-level redundancy in mind. This architecture not only ensures resilience and data durability but also separates the raw data intake from the structured information, allowing for modular scalability and maintenance.

The software responsible for data sharing is the Apache Airflow[4], which serves as the central orchestration tool for managing and scheduling data workflows. Airflow introduces Directed Acyclic Graphs (DAGs), which define in a formal way the data processing pipelines used in the system. Further, it automates the sequence of data extraction, transformation and distribution of tasks. Within the DMSS the Airflow handles the scheduling of data gathering tasks, as well as the processing and accumulation of data in the ideal database. The processing functionality extends beyond conventional data transformation tasks, orchestrating machine learning techniques to tackle various problems such as sentiment analysis or descriptive analytics essential for informed decision-making.

The **Data Analysis and Visualization Module Group** is dedicated to visual and analytical information, providing stakeholders with actionable insights through intuitive dashboards and advanced data exploration tools. For real-time visual analytics, the system employs Grafana, an open-source platform that enables the creation of interactive dashboards by connecting to the PostgreSQL and if necessary to the MongoDB backend. Grafana facilitates the monitoring of key performance indicators (KPIs), temporal trends, and geospatial data. Accompanying the visual interface, the system implements a data analytics and machine learning environment built in Python. Leveraging powerful libraries

[2] https://www.mongodb.com/
[3] https://www.postgresql.org/
[4] https://airflow.apache.org/

such as Pandas for data manipulation, PyTorch for deep learning, and Transformers (Hugging Face[5]) for natural language processing, the system supports advanced data processing techniques.

2.3. DMSS Deployment Environment

The development and deployment of DMSS is based on the microservices architectural approach. The reason behind this selection is that microservices are more suitable for independent development of each tool, for easy deployment process and for building a high scalable system. The major guidelines that were followed are:

- Each tool is implemented as a microservice. Exception on this are the GUIs of each tool which are also separate microservices.
- All data storage Data Bases will be realized as microservices, with static volumes for data storage.
- The microservices will communicate and exchange information using messages via specific web-enable APIs interfaces.

The Figure 3 depicts the deployment architecture of the DMSS platform.

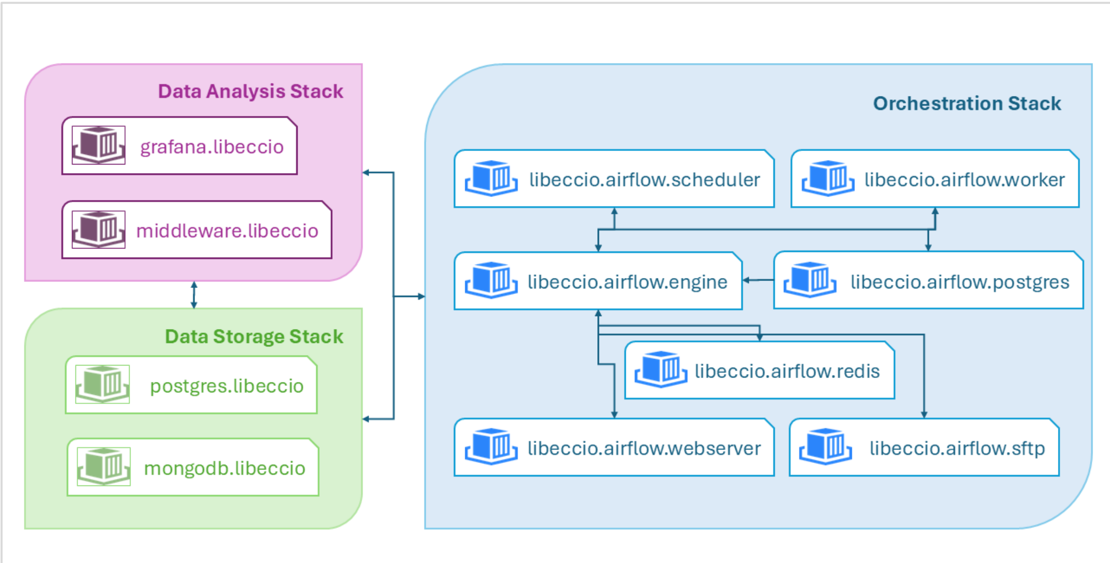

Figure 3 Libeccio DMSS microservices deployment diagram

The microservices architecture is implemented usinga virtualization infrastructure based on Docker[6] containers. Each microservice in the Figure 3 has been implemented as a Docker container and deployed over a Docker Engine. The Docker Engine is installed in a virtual machine operating over a ProxMox Virtual Environment[7] supporting KVM hypervisor[8] installed in a bare metal server system. For the management of the containers' environment, the Portainer.io tool[9] (Figure 4) was used for fast and reliable deployment and re-deployment of the DMSS in a production environment.

[5] https://huggingface.co/
[6] https://www.docker.com/
[7] https://www.proxmox.com/en/
[8] https://linux-kvm.org/page/Main_Page
[9] https://www.portainer.io/

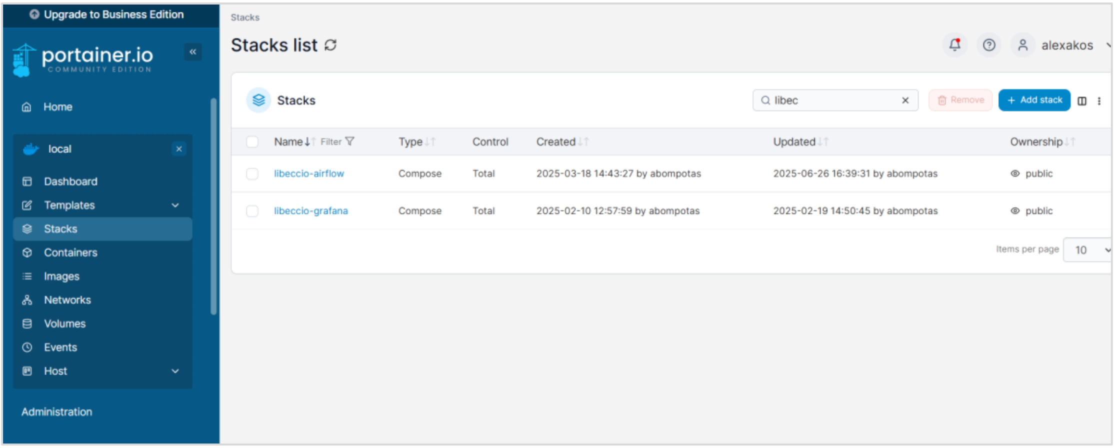

Figure 4 DMSS deployment infrastructure from Portainer.io platform.

3. DMSS Modules

3.1. Process orchestration

This section provides a comprehensive overview of the orchestration component at the core of our data gathering and processing system. This pivotal component, implemented using the open-source platform Apache Airflow, is responsible for the automated execution, scheduling, and monitoring of all data-related workflows. The system is designed to ingest data from a variety of sources, including programmatic access via APIs, manual uploads of CSV files, and automated web crawling and scraping. In the context of DMSSs, the ability to harness information from disparate sources is a significant competitive advantage. Our system is engineered to aggregate data from a multitude of origins, employing a range of acquisition techniques. This raw data, once collected, undergoes a series of transformations and processing steps to produce meaningful insights.

Central to this intricate process is the orchestrator, a component that ensures the seamless and efficient execution of these data pipelines. After a thorough evaluation of various workflow management systems, Apache Airflow was selected for its flexibility, scalability, and extensive community support. The diverse nature of our data acquisition methods presents unique orchestration challenges and Apache Airflow is exceptionally well-suited to address these complexities for several key reasons. Firstly, its flexibility and extensibility, being Python-based, allow for the creation of highly customized and dynamic pipelines. We can easily write custom operators and hooks to interact with any proprietary systems or specialized data sources. Secondly, its scalability, derived from a modular architecture and a choice of executors, enables our system to scale horizontally. As the volume of data and the number of pipelines grows, we can distribute the workload across multiple worker nodes to ensure timely processing. Thirdly, its built-in web interface provides comprehensive monitoring and alerting capabilities, offering visibility into the health and performance of our data pipelines. We can monitor task execution in real-time, view logs for debugging, and configure alerts to be notified of any failures, which is crucial for maintaining data integrity. Finally, Airflow's design encourages the development of idempotent tasks —tasks that can be re-run multiple times with the same result. This, combined with the ability to backfill (running a DAG for a past period), is invaluable for recovering from failures and reprocessing data when business logic changes.

To better comprehend its role in our system, it's important to understand how Apache Airflow works. It is a powerful platform used to programmatically author, schedule, and monitor workflows. Originally developed at Airbnb, it has become a de facto standard for data engineering pipelines. Workflows in Airflow are defined as Directed Acyclic Graphs (DAGs), which are collections of tasks with defined dependencies. In our system, these DAGs are designed to manage the full data lifecycle: they communicate with the data collection mechanisms that run at scheduled intervals, trigger the data transformation and preprocessing systems and orchestrate the ML infrastructure that annotates and extracts information from the gathered data. This structure ensures a clear and predictable path of execution. Its architecture is composed of several key components that work in concert: a Web Server for user interaction; a Scheduler that triggers workflows; an Executor that runs the tasks; a Metadata Database that stores the state of all operations; and a DAG Directory where the workflow definitions reside.

The process of orchestrating a data pipeline begins with defining a DAG in a Python script. Within this script, individual tasks are defined using Operators. Airflow provides a rich set of pre-built operators which we leverage to consume data from REST APIs, process manually uploaded CSV files and execute web scraping scripts. Dependencies between these tasks are explicitly defined, creating a clear execution path. For instance, a task that transforms data will only run after the initial data ingestion task has successfully completed. Furthermore, special operators called Sensors can be used to create event-driven workflows by waiting for a certain condition to be met, such as the arrival of a new file in a directory, before triggering subsequent tasks.

To illustrate the orchestration logic in practice, the figure below presents a representative Directed Acyclic Graph (DAG) designed for scraping user reviews from selected platforms. This DAG outlines the step-by-step workflow, beginning with a scheduled trigger, followed by review scraping tasks implemented via Apify actors or custom crawlers, data validation steps, and culminating in the structured storage of cleaned results. It visually demonstrates how Airflow coordinates asynchronous and dependent operations to ensure data quality, reliability, and seamless integration with the broader data pipeline.

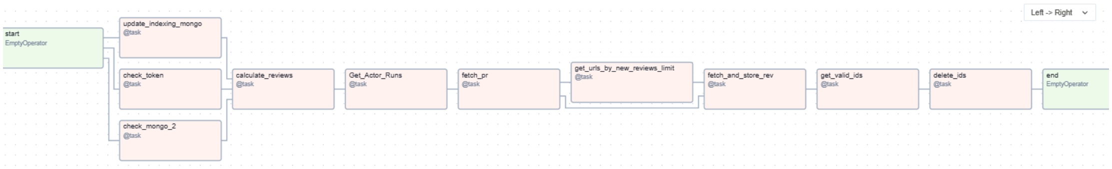

Figure 5 Example of Directed Acyclic Graph (DAG) in Apache Airflow

- **`update_indexing`**
  This task scans all available entries and checks whether each one already exists in the designated MongoDB collection used for review extraction. It ensures that no duplicate data is scraped, and that only new or unprocessed entries are considered in the pipeline.
- **`check_token_and_mongo_connection`**
  This step verifies that all necessary authentication tokens and database connections are valid and accessible. It ensures that the system can interact securely and reliably with both the MongoDB instance and any external APIs or services before continuing execution.
- **`calculate_reviews`**
  This task estimates the total number of reviews eligible for extraction, considering API usage limitations, time constraints, and data freshness. It helps manage resource allocation and optimize the crawling process within quota and time budget limits.
- **`get_actor_runs`**
  This component interacts with the Apify API to confirm connectivity and retrieve the current or previous actor runs. It validates the availability and status of the scraping infrastructure before launching new tasks.

- `fetch_pr`
  This step parses and processes incoming input (e.g., review requests or metadata) to ensure it is correctly formatted and compliant with the requirements of the scraping functions. It performs normalization and validation of inputs to maintain consistency throughout the pipeline.
- `get_urls_by_date_limit`
  This task groups URLs based on the associated dates and identifies which ones require extraction. It applies predefined temporal filters to determine the subset of data that falls within the scraping window.
- `fetch_and_store_reviews`
  The final step performs the actual scraping of review data, retrieving content from the specified URLs. Once extracted, the reviews are processed and stored in either MongoDB or PostgreSQL, depending on the storage schema and indexing strategy defined for downstream analytics and retrieval.

3.2. Data Collection

The data infrastructure operates on a hybrid methodology that combines automated scraping, direct API access, and structured manual ingestion. This multi-source approach ensures reliable, timely, and scalable acquisition of tourism-related, mobility, climate, and environmental datasets. The system's architecture is designed to support scheduled updates, normalization workflows, and metadata traceability, regardless of the collection method used.

3.2.1. Automated Collection via APIFY Platform

A large volume of data is collected through the APIFY platform, which automates browser-based extraction tasks from dynamic websites (**Error! Reference source not found.**). C ustom actors are developed to handle data extraction at varying frequencies depending on the volatility of each source. These tasks are scheduled and monitored to guarantee completeness and to detect failures resulting from layout or structural changes.

APIFY-based scrapers are used to collect comprehensive data from key global platforms in the tourism and accommodation domain, specifically:

- **Tripadvisor**: Extracted data includes business listings for hotels, restaurants, and attractions, user reviews, star ratings, review volume, popularity rankings, business categories, geolocation data, and operational metadata. Additional metadata, such as traveller type and seasonal satisfaction trends, is collected where available.
- **Booking.com**: The system extracts data on hotel and apartment listings including accommodation names, pricing (base rate and dynamic changes), guest ratings, review volume, cancellation policies, amenities, and availability calendars. Room types and capacity data are also collected when accessible.
- **Google (Places and Maps)**: Front-end scraping is used to acquire business metadata from Google's public listings. This includes names, categories (e.g.

hotel, museum, café), average ratings, number of reviews, peak visiting hours, photo coverage, and user-contributed comments.

- **Airbnb**: Extraction includes listing IDs, nightly pricing (standard and seasonal), host profile data, occupancy calendars, listing type (entire home, private room, etc.), user reviews, guest ratings, location metadata, and cancellation policies.

In all cases, the guiding principle is to collect publicly available data presented by each platform, as permitted under usage guidelines and legal frameworks. The focus is on tourism accommodation and lodging facilities, including hotels, B&Bs, vacation rentals, and alternative lodgings. The actors are built to adapt dynamically to content changes and are capable of parsing structured data embedded within JavaScript, APIs, or page metadata (Figure 7).

Collected data from these platforms is processed to align with a unified schema, ensuring comparability across providers. Post-processing includes deduplication, sentiment normalization (for review data), and structured categorization by accommodation type and location.

Status	Actor	Task	Results	Usage	Started	Finished	Duration	Build	Origin
Finished! Total 1842 requests: 1842...	Tripadvisor Scraper maxcopell/tripadvisor Pay per result		459		2025-07-14 15:06:17	2025-07-14 15:11:55	5 m 37 s	0.0.168	API
Finished! Total 719 requests: 719...	Booking Scraper voyag...raper Pay per result		711		2025-07-13 10:51:05	2025-07-13 11:10:44	19 m 39 s	0.0.79	API
Finished! Total 1170 requests: 1170...	Booking Scraper voyag...raper Pay per result		1.136		2025-07-13 10:51:00	2025-07-13 11:24:25	33 m 25 s	0.0.79	API
Finished! Total 1692 requests: 1692...	Booking Scraper voyag...raper Pay per result		1.647		2025-07-13 10:50:54	2025-07-13 11:39:21	48 m 27 s	0.0.79	API
Finished! Total 1568 requests: 1568...	Booking Scraper voyag...raper Pay per result		1.528		2025-07-13 10:46:52	2025-07-13 11:23:15	36 m 23 s	0.0.79	API
Finished! Total 961 requests: 960...	Booking Scraper voyag...raper Pay per result		935		2025-07-10 03:13:27	2025-07-10 03:39:23	25 m 56 s	0.0.79	API
Finished! Total 1455 requests: 1454...	Booking Scraper voyag...raper Pay per result		1.425		2025-07-10 03:13:19	2025-07-10 03:52:14	38 m 54 s	0.0.79	API
Finished! Total 1185 requests: 1184...	Booking Scraper voyag...raper Pay per result		1.157		2025-07-10 03:13:12	2025-07-10 03:46:23	33 m 11 s	0.0.79	API
Finished! Total 678 requests: 676...	Booking Scraper voyag...raper Pay per result		654		2025-07-10 01:33:53	2025-07-10 01:51:31	17 m 37 s	0.0.79	API
Finished! Total 116 requests: 93...	Booking Scraper voyag...raper Pay per result		71		2025-07-10 00:20:52	2025-07-10 00:23:58	3 m 5 s	0.0.79	Web

Figure 6 ApiFy Management Console: List of registered actors

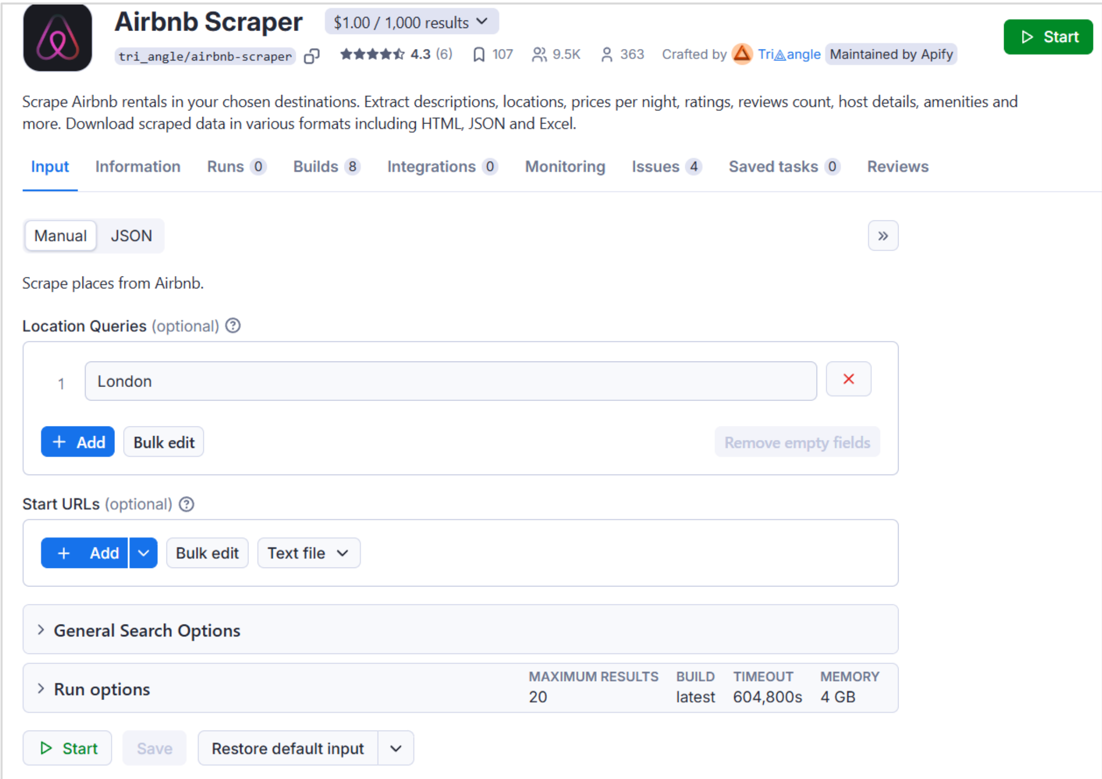

Figure 7 ApiFy Management Console: Example of actor parameterization

3.2.2. API-Based Integration

Where formal APIs are available, the system integrates with external data sources through custom API clients tailored to each provider's API specifications designed to each provider's specifications. These clients manage authentication, enforce rate limits, handle retries and errors gracefully, and normalize response structures for consistent downstream use. Encapsulating this logic within dedicated modules ensures high-quality, machine-readable data ingestion with predictable structure and refresh cycles, while also enhancing maintainability and modularity. Access is authenticated and governed by usage agreements to ensure compliance and long-term sustainability.

The system currently integrates the following APIs:

- Google Directions API: Used to retrieve real-time and historical traffic conditions, travel times, route congestion, and transportation alternatives. This data supports traffic monitoring, access time calculations, and scenario modelling for mobility infrastructure.

- Tourism Related to the Region APIs: Some Regions have their owns APIs that store structured information on tourism, products, attractions, routes or even cultural content. Such information can be accessed by the platform when credentials and permission of access are given.

- Copernicus Climate Data: Access to the Copernicus Climate Data Store enables retrieval of climate indicators such as surface temperature, rainfall patterns, and drought indices. These datasets are used to correlate tourism demand with environmental and seasonal patterns.

- Eurostat Air and Maritime Passenger Data: Eurostat provides semi-automated datasets covering passenger volumes in both air and maritime transport across European countries.
  - The air transport data include monthly passenger counts by airport and transport type, as well as route-level details that capture passenger flows between specific country pairs. These data complement national airport statistics by offering a broader international perspective on air travel patterns.
  - Maritime transport data consist of quarterly statistics on passenger movements through major European ports, excluding cruise passengers, and provide a comprehensive overview of sea passenger traffic within the EU. However, smaller ports may not be covered in these datasets. To assess cruise tourism impacts specifically, data from local port authorities are integrated alongside Eurostat figures to isolate cruise-related passenger volumes.

    These datasets are typically obtained in CSV, Excel or JSON format through the EUROSTAT API and undergo validation and schema mapping before integration, ensuring consistency and usability within the overall data ecosystem.

- Environmental data is integrated from various open sources, including public APIs and static datasets. These resources cover topics such as:
  - *Human impact on land use and urban waste generation*
  - *Monitoring of water quality, including harmful algal blooms and bathing water conditions*

API responses are normalized into internal schemas for longitudinal and geospatial analysis. Access logs, versioning, and dataset provenance are stored as metadata for audit purposes.

3.2.3. Manual and Semi-Automated Data Import

Some of the datasets are acquired manually or via structured uploads from institutional partners. In these cases, data is collected through CSV or Excel formatted files, or even via submission forms and portals maintained by third parties. Users upload these file through dedicated web (Figure 8) form and automated pipelines process these files into the central data system after applying validation and formatting logic.

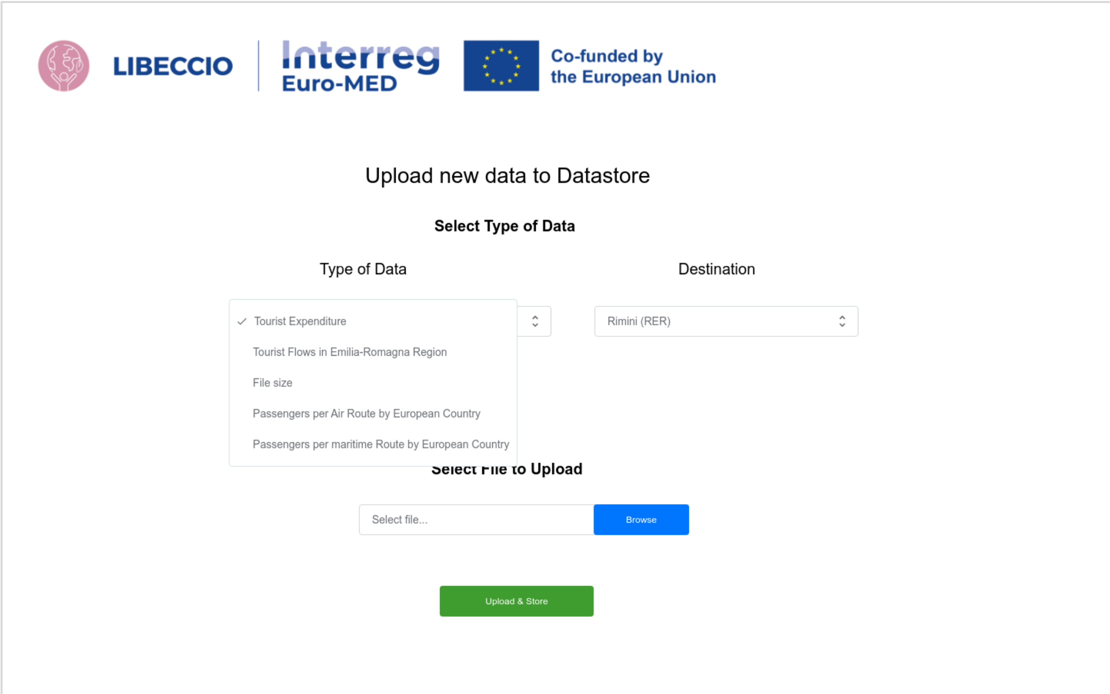

Figure 8 Web-enabled form for uploading data to DMSS

Examples include:

- **SIMC Meteorological Data (Dext3r platform):** This source provides meteorological time series including precipitation, wind, temperature, and relative humidity. These data are crucial for event correlation and seasonal modelling (Accessed from https://simc.arpae.it/dext3r/).
- **ISTAT Statistical Reports**: Datasets include standardized tables for accommodation capacity, tourist flows (arrivals and nights), demographic breakdowns, and national vs international trends. Data is either downloaded or received through curated upload channels.
- **Bank of Italy – Tourism Expenditure Reports**: Periodic reports provide macroeconomic indicators related to tourism spending by category and nationality. These are accessed in report or spreadsheet format and parsed for inclusion in financial behaviour models.

In all such cases, ingestion is supported by schema mapping tools and version control to preserve historical consistency. Data validation scripts are executed prior to merging new records into the core system.

3.2.4. Scheduling and Versioning

Collection tasks are managed through automated schedulers and monitoring agents. **Apache Airflow** is used to orchestrate API-based and APIFY actor tasks, offering fine-grained control over scheduling, dependencies, and retries. APIFY actors are typically run on a daily or weekly basis, while API-based scripts are executed in either batch or streaming mode, depending on the frequency and nature of the data source. Manual and semi-automated sources are aligned with institutional publication cycles, with alerting mechanisms in place to notify the system when new versions or updates become available.

The Figure 9 Airflow Scheduling Interface shows the Airflow scheduling interface, where data pipelines are configured to run at specific intervals (e.g., hourly, weekly). This setup ensures consistent data ingestion and allows for immediate detection and recovery in case of failures.

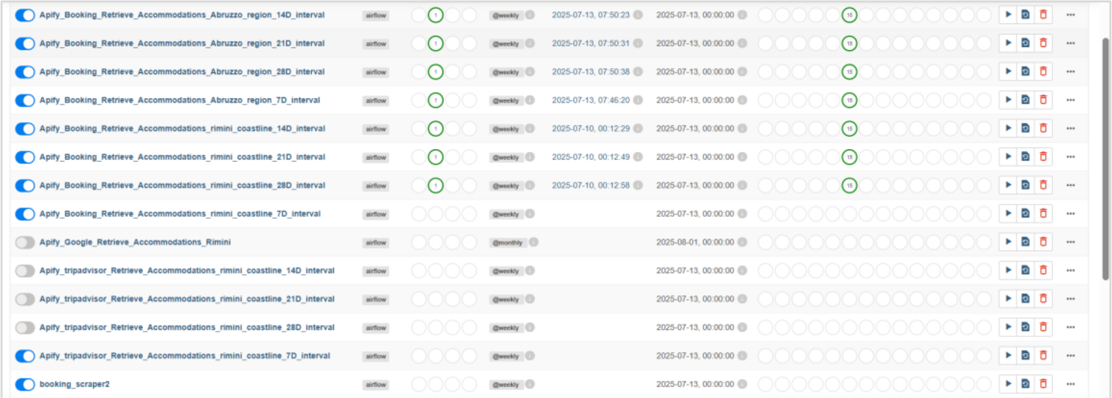

Figure 9 Airflow Scheduling Interface

Each dataset includes metadata fields that track update timestamp, source link, ingestion method, and data volume to ensure transparency and traceability over time.

This multi-channel approach to data collection ensures the system maintains broad coverage across tourism, mobility, climate, and environmental data domains. The integration of automated scraping, structured APIs, and controlled manual input allows the infrastructure to operate with resilience, adaptability, and compliance, supporting a wide range of analytical applications and policy-driven insights.

3.3. Data Preprocessing

Effective data preprocessing is fundamental to ensuring the reliability, consistency, and usability of information integrated into the system. Given the heterogeneous nature of the data sources—ranging from real-time APIs and automated scrapers to manually submitted Excel files—the platform implements a dual-track preprocessing strategy. This strategy includes automated pipelines for handling structured, machine-readable inputs, and semi-automated routines for validating and cleaning manually acquired datasets.

3.3.1. Automated Collected Data Preprocessing

Where formal APIs are available, they are integrated into the system to provide **high-quality, machine-readable data** with predictable structure, refresh cycles, and well-defined schemas. These APIs facilitate direct access to authoritative sources, reducing manual intervention and enabling near real-time data ingestion. API access is typically **authenticated**, governed by **rate limits**, and subject to **usage agreements** to ensure data security, sustainability, and regulatory compliance.

The preprocessing of API-based data focuses on:

1. Parsing and transforming JSON, XML, or CSV payloads

2. Normalizing date and geographical fields
3. Mapping source schemas to the internal data model
4. Handling missing, malformed, or redundant entries
5. Monitoring API status and rate-limit thresholds

The system currently integrates several critical APIs:

- **Google Directions API**: Provides real-time and historical traffic data, including travel times, congestion levels, and transportation alternatives. The data is used for mobility modelling, accessibility analysis, and infrastructure planning. Route-specific metadata such as duration, mode, and traffic impact is parsed and stored in time-series form.
- **Regional Tourism APIs**: Some regional administrations maintain dedicated APIs for tourism products, points of interest, and event calendars. Where access credentials are available, these sources are integrated into the platform. These APIs often follow bespoke formats, requiring custom adapters to translate data into the common schema.
- **Copernicus Climate Data Store (CDS)**: Enables access to climate indicators such as temperature anomalies, drought indices, and rainfall patterns. These datasets are used for environmental modelling and correlation with tourism seasonality. The data is typically retrieved in NetCDF, GRIB or CSV format and processed into standardized spatial-temporal models.
- **Eurostat APIs – Air and Maritime Passenger Transport**:
    - *Air transport data*: Monthly statistics on passenger volumes by airport and route (including transport type) are retrieved through the SDMX-based Eurostat API. These are essential for modelling international mobility trends and complement national airport records.
    - *Maritime transport data*: Quarterly figures for sea passenger movement across EU ports are processed similarly. The system isolates cruise tourism impacts by cross-referencing local port authority data, where available, with the general Eurostat datasets.

All datasets obtained via these APIs are subjected to validation routines and schema mapping procedures to ensure full integration with the platform's data warehouse.

In parallel, automated web scraping tasks are managed using Apify actors and orchestrated through Apache Airflow. These workflows rely on modular configuration schemes, defined in JSON format, which specify the data source, extraction logic, destination table, and field mappings. Rather than hardcoding source-specific logic, the platform reads these dynamic schemes to control how scraped data is retrieved, parsed, and transformed. This approach ensures a flexible and scalable integration pipeline for semi-structured or unstructured sources, particularly where formal APIs are unavailable or incomplete.

These schemes enable the dynamic transformation of nested JSON documents into structured SQL tables, offering a flexible and scalable approach to data ingestion. This design is particularly effective for handling complex entities that include multiple sub-records—such as hotels with multiple rooms, amenities, or reviews—by flattening nested structures into relational formats automatically.

3.3.2. Manual and Semi-Automated Data Preprocessing

Depending on the datasets in hand, different types of preprocessing are necessary. As mentioned before, institutional partners offer data in various formats via CSV or Excel files and through structured, supported submission forms. These datasets are often obtained through periodic downloads, direct uploads, or third-party portals, and they vary in terms of structure, completeness, and formatting consistency.

The preprocessing pipeline for such data includes both **manual checks** and **semi-automated transformations** to ensure integration with the central data system. Manual preprocessing typically involves:

1. Verifying file consistency and completeness
2. Identifying and correcting structural discrepancies (e.g., misaligned columns, merged cells, invalid headers)
3. Normalizing date and location formats
4. Flagging missing or anomalous entries for review

Once initial manual checks are complete, semi-automated scripts perform tasks such as:

1. Parsing multi-sheet Excel workbooks or delimited text files
2. Standardizing naming conventions (e.g., municipality codes, indicator labels)
3. Validating against reference taxonomies (e.g., administrative regions, tourism categories)
4. Reformatting temporal data into unified time-series schemas

Overall, this hybrid approach balances flexibility and control, ensuring that data from diverse institutional sources is validated, cleaned, and formatted before being merged with the larger automated data infrastructure.

3.3.3. Data Processing

Following preprocessing, the system proceeds with data processing operations that transform validated inputs into enriched, analysis-ready formats. This stage is essential for integrating data from heterogeneous sources into a coherent and structured model that supports both analytical tasks and visual representation.

In cases where datasets include spatial attributes, geospatial processing routines are applied to associate records with predefined regions or municipalities using spatial joins and area matching. This is particularly relevant for datasets such as accommodation locations and traffic routes. Geocoding is performed when necessary to convert address-based inputs into standardized coordinates, enabling spatial indexing and map-based analytics.

The processed data is then indexed and structured according to the system's relational schema, making it available for querying, reporting, and real-time visualization through platforms such as Grafana. At the visualization stage, descriptive statistics are generated to assist users in interpreting patterns and identifying outliers.

Natural Language Processing (NLP) tasks, including language detection and sentiment analysis, are an integral part of the system's data enrichment pipeline. These tasks are exclusively orchestrated through Apache Airflow, which schedules and manages the execution of model-based processing workflows. The primary application of these NLP tasks is the analysis of user-generated reviews, typically collected through web scraping tools or third-party APIs. Language detection ensures that content is accurately categorized and processed according to its linguistic context, supporting multilingual data integration. Sentiment analysis is performed using pre-trained machine learning models to evaluate the emotional tone of the reviews. These outputs are stored as additional metadata fields in the database, enabling filtered queries, aggregated sentiment scoring, and correlation with quantitative indicators.

In addition to NLP, Apache Airflow is used to orchestrate other data analysis tasks, including the execution of machine learning models. Analytical workflows are defined as DAGs, allowing data processing and model inference steps to be integrated into a unified pipeline.

3.4. Data Storage

The system's data storage architecture is built to accommodate diverse data sources and meet the scalability demands of an analytics-driven platform. To achieve this, a hybrid model is implemented, leveraging the flexibility and redundancy of a NoSQL database (MongoDB) alongside the structure, referential integrity, and optimized querying of a relational database (PostgreSQL). MongoDB serves as a temporary repository for raw and semi-structured data, allowing for asynchronous ingestion and resilient buffering. After validation and transformation, the data is transferred to PostgreSQL for durable storage, structured queries, and integration with analytics and visualization tools. Both databases are indexed to support fast, high-volume data retrieval. The architecture is further reinforced by routine backups and built-in redundancy to protect data integrity, ensure availability, and support continuous operation.

3.4.1. Data Redundancy – MongoDB

This section describes the architecture and technologies employed for data storage within the system. The data layer is designed to accommodate diverse data types, ensure scalability, and support both high-throughput operations and long-term persistence. To achieve these goals, a hybrid storage approach is implemented, combining a NoSQL database (MongoDB) for flexible, redundant, and rapid access to latent information, alongside a relational database (PostgreSQL) optimized for structured querying, reporting, and visualization.

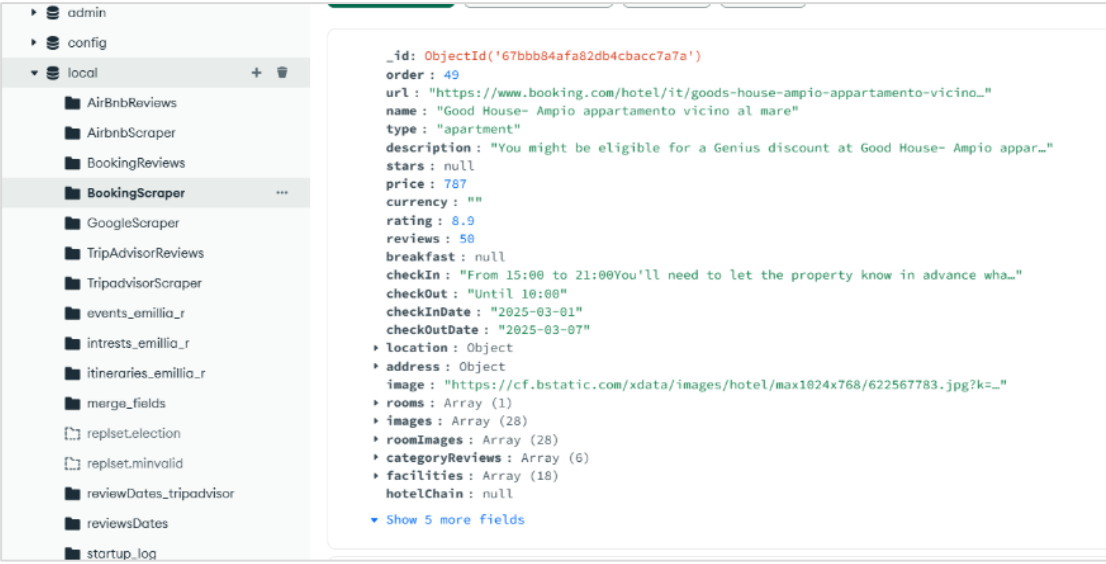

Figure 10 Mongo DB example record

MongoDB serves as a transitional data store within the architecture, primarily supporting early-stage data ingestion and processing. Its document-oriented model and schema flexibility make it particularly well-suited for managing heterogeneous, semi-structured, and evolving data formats. This includes raw API responses, semi-automatically collected data, and unprocessed JSON files generated by Apify actors, which capture detailed reviews and accommodation information.

The key collections maintained in MongoDB encompass:

- **Events from open APIs**: capturing dynamic and real-time event data streams,
- **Data from semi-automatic sources**: aggregating partially processed or manually curated inputs,
- **Raw Apify actor data**: storing untransformed JSON payloads related to reviews and accommodations.

Additionally, MongoDB is utilized to store various index structures that facilitate efficient data retrieval. These indexes support both API-driven queries and SQL-based lookups, bridging the NoSQL and relational layers.

The core rationale for utilizing MongoDB at this stage lies in its ability to provide fast, resilient storage for incoming data prior to transformation or validation. This design enables asynchronous ingestion and preprocessing workflows, effectively decoupling real-time data collection from downstream operations and mitigating data loss risks during transient failures.

Following validation, cleaning, and structuring, data is migrated into the PostgreSQL database for long-term persistence, indexing, and analytical processing. This layered architecture leverages MongoDB's flexibility and redundancy alongside PostgreSQL's relational power and query optimization, ensuring a robust and performant data ecosystem.

3.4.2. Data Organization and Schema Design

This section describes how data is organized internally following ingestion, focusing on structural conventions, levels of granularity, and normalization practices adopted across the system. After collection, all raw data—whether retrieved via APIs or uploaded manually—is first stored in MongoDB in its original semi-structured format. This transitional stage allows for asynchronous ingestion workflows and ensures that no data is lost due to transient errors or delays in downstream processing.

Once data is validated and transformed, it is migrated to the PostgreSQL database, where it is stored in accordance with a normalized relational schema. This structured representation ensures consistency, reduces redundancy, and supports efficient querying, analysis, and reporting across multiple domains, including accommodations, mobility, cultural events, and traffic.

The data model supports both temporal and spatial granularity. Temporal resolution varies by data source and use case: traffic and environmental datasets may be updated hourly, while tourism statistics are typically aggregated at quarterly intervals. Spatial referencing is achieved through a hierarchical area model, allowing each record to be consistently associated with a geographic location—ranging from national regions down to municipalities—via the `Area` entity. This spatial structure supports nested relationships and provides a unified geolocation reference across all functional modules.

To enforce semantic consistency and facilitate analysis, the schema employs fully normalized lookup tables, such as those used for event themes, accommodation amenities, or administrative classifications. Referential integrity is maintained through foreign keys, bridge tables, and recursive relationships, particularly in the modelling of area hierarchies and many-to-many associations.

The volume and frequency of data vary by domain. High-frequency sources, such as traffic data or weather feeds, generate large volumes of timestamped records, while other domains, such as accommodations and events, consist of less volatile but semantically rich datasets. All records include metadata indicating the source, ingestion time, validation status, and—where applicable—versioning information. This metadata ensures traceability and supports incremental updates, auditability, and quality control.

3.4.3. Entity-Relations Diagram

The relational database schema is designed to support an integrated system that models accommodation services, mobility infrastructure, event organization, and transportation statistics. The schema follows a normalized structure with clearly defined entities and relationships, and its design is visually represented in following figures, where the tables are separated in groups according to the scope of the stored data.

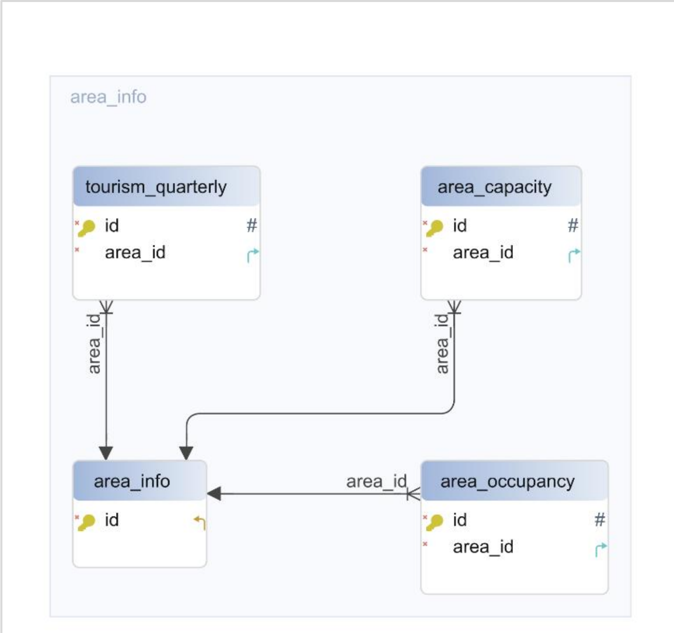

Figure 11 Relational DB Schema: Destination related data

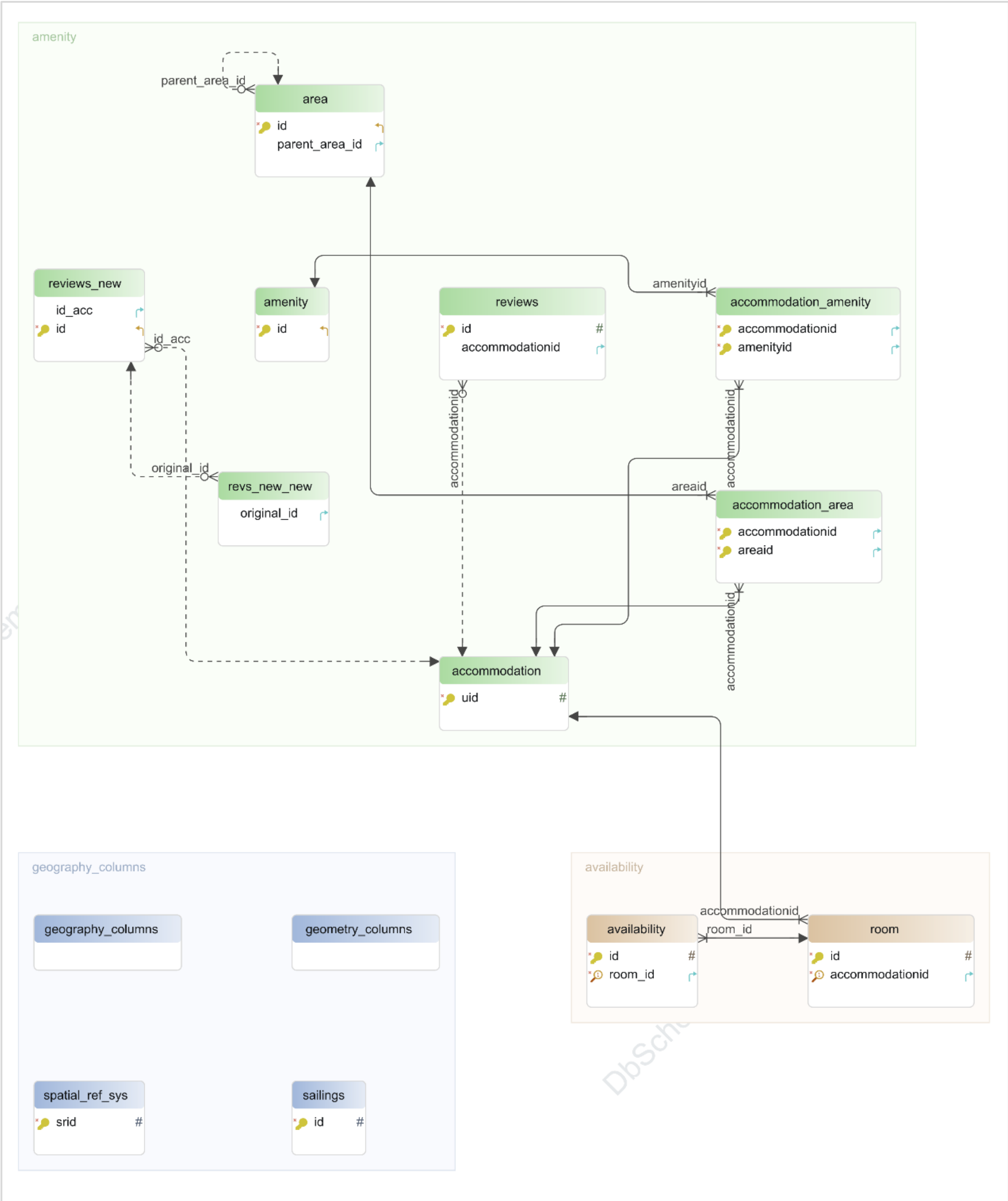

Figure 12 Relational DB Schema: Accommodations related data

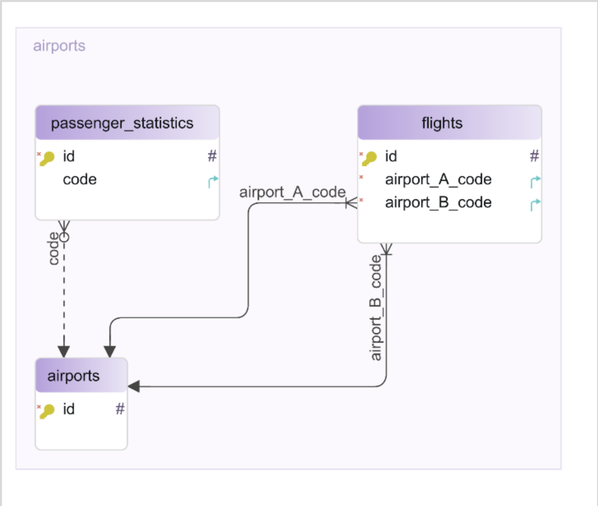

Figure 13 Relational DB Schema: Airports related data

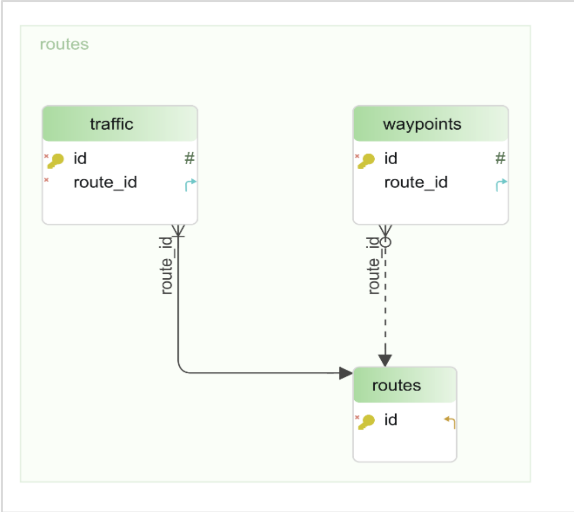

Figure 14 Relational DB Schema: Traffic related data

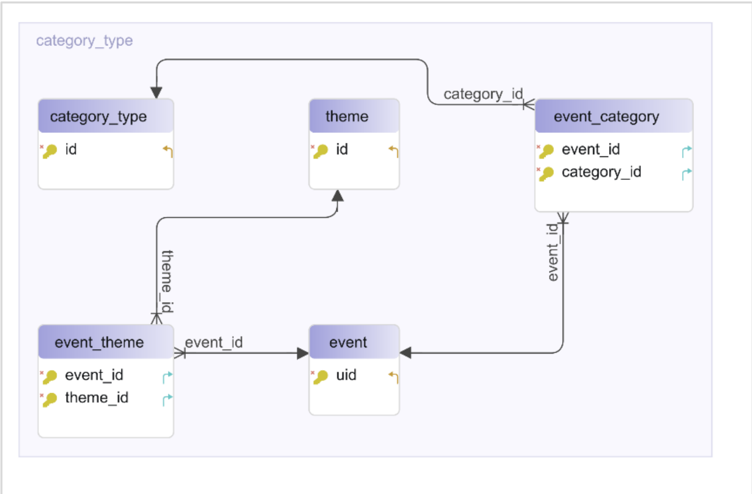

Figure 15 Relational DB Schema: Information categorization related data

The accommodation domain is centred on the Accommodation entity, which models facilities such as hotels, guesthouses, or rental units. Each accommodation is associated with multiple Room entries, and each room is further linked to one or more Availability records, which capture price and date-specific occupancy information. The schema also models the relationship between accommodations and offered amenities through a many-to-many association with the Amenity entity. Feedback mechanisms are supported through Ratings and Reviews, allowing multiple users to evaluate and comment on each accommodation unit.

Spatial referencing is achieved through the Area entity, which serves as the anchor for all geolocation-based associations. The model supports nested geographical hierarchies using a recursive contains relationship, enabling each area to be linked to one or more sub-areas. Furthermore, the Area entity serves as a geographic anchor for multiple other entities in the schema, including accommodations, routes, sailings, and airports. This located_in relationship ensures consistent spatial organization across functional domains.

The mobility layer includes three transportation modalities: air, sea, and road. Airports are modelled through the Airports entity, which is connected to Flights and Airport_Passenger_Statistics through one-to-many relationships. Flights represent the operational aspect of air travel, while passenger statistics provide aggregated data for analytical use. Sea transport is represented in the schema through the Sailings entity, which, although not explicitly expanded in the diagram, is conceptually parallel to the flight model and associated with specific geographic areas. Road transportation is modeled using the Routes entity, which is linked to sequential Waypoints and corresponding Traffic records. These relationships allow for the representation of routes as both spatial sequences and traffic-aware paths.

The schema also includes a cultural and event-related domain. The Event entity models scheduled activities such as festivals, exhibitions, or concerts. Each event is associated with one or more Event_Theme and Event_Category entries. These, in turn, are linked to lookup tables Theme and Category_Type respectively, which provide normalized references for semantic consistency. This structure allows events to be filtered, grouped, or analyzed according to their thematic or administrative classifications.

The relationships in the schema follow well-defined cardinalities. Accommodation is linked to multiple rooms, each of which can have several availability entries. Amenities, reviews, and ratings are also linked to accommodations with appropriate multiplicity. Routes, airports, and events maintain one-to-many relationships with their subordinate or descriptive entities. The Area entity serves as a unifying reference point for spatial organization, allowing recursive nesting and association with all major service and transport structures.

3.5. Data Visualization

Grafana is an open-source analytics and visualization platform widely used for monitoring time-series data from a variety of sources. It provides a powerful, flexible interface for building interactive dashboards, enabling users to explore, query, and visualize metrics, logs, and other data in real time. Grafana supports numerous data backends, including relational databases, time-series engines, and document stores. Through its plugin system, Grafana can also connect to non-native sources such as MongoDB, allowing users to visualize and analyse data stored in NoSQL databases alongside other metrics.

3.5.1. Dashboards

In our deployment, Grafana dashboards are organized in a hierarchical structure, beginning with a central Home Dashboard that serves as the entry point for navigating the monitoring environment. This home dashboard provides a high-level overview of the system, presenting key metrics and general insights through visually distinct panels. These panels are not merely static displays but are interactive and clickable, each acting as a navigation link to more specialized dashboards.

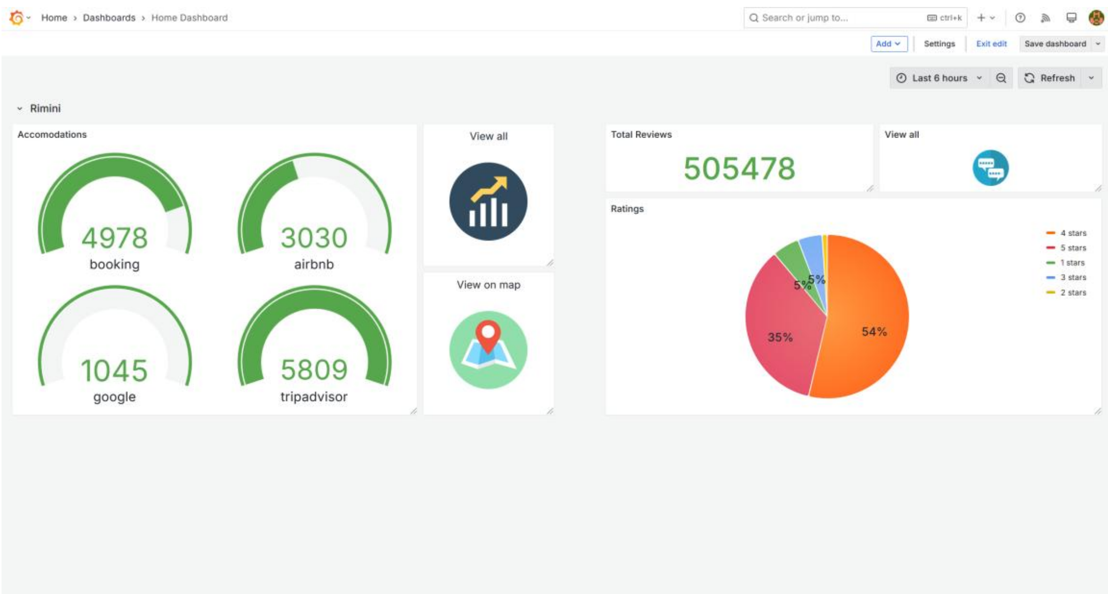

Figure 16 User Interface: Home dashboard

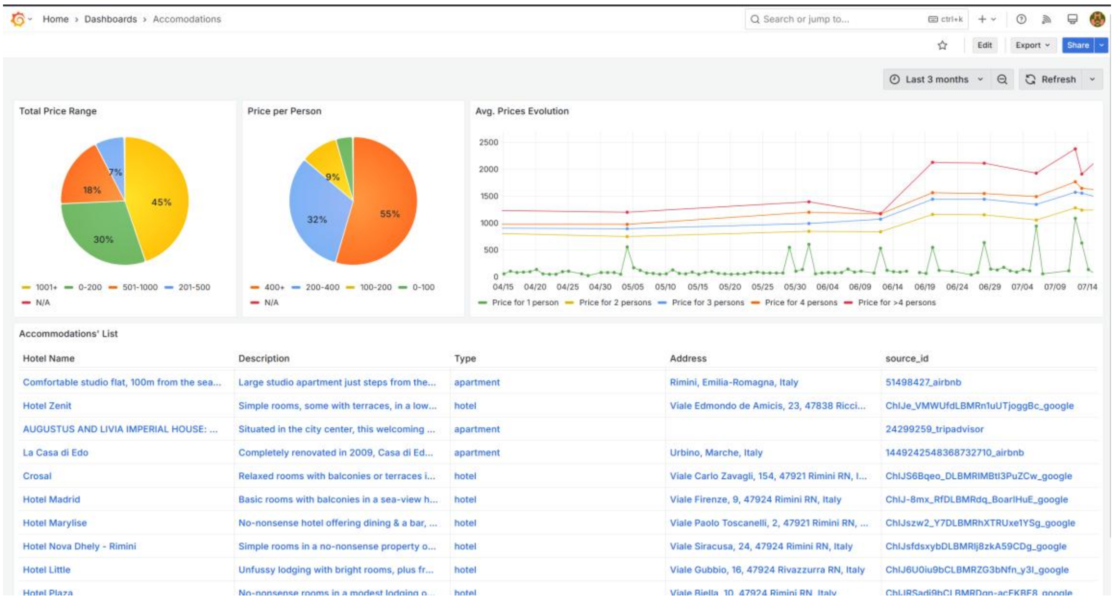

Figure 17 User Interface: Accommodation insights dashboard

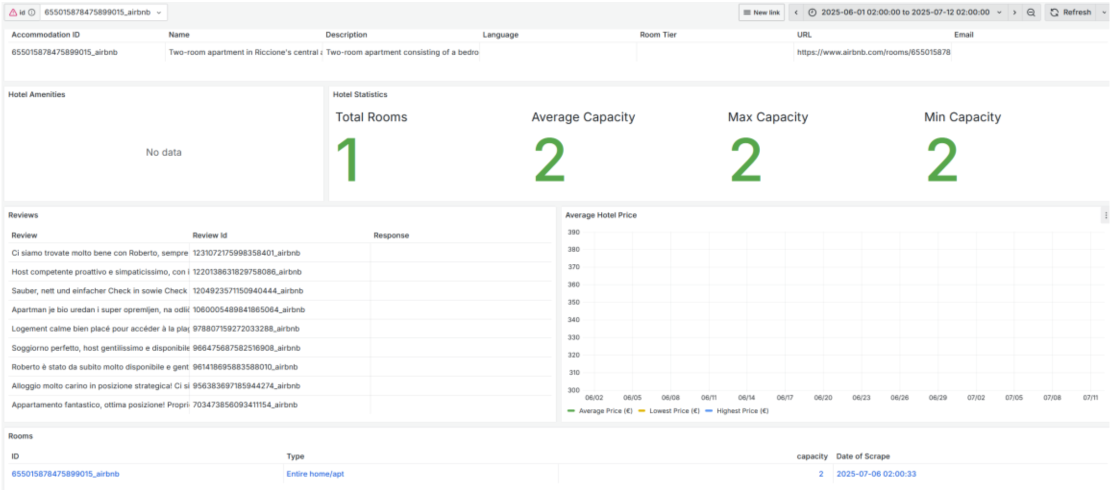

Figure 18 User Interface: Accommodation details dashboard

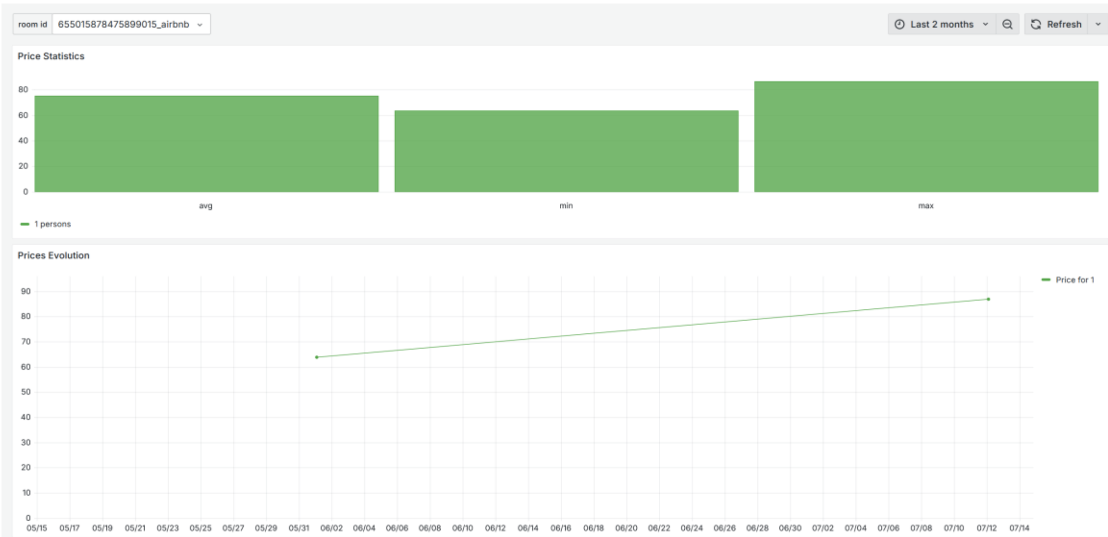

Figure 19 User Interface: Apartment Price Metrics

These linked dashboards delve into specific aspects of the collected data, such as environmental metrics, accommodation statistics, mobility trends, or infrastructure performance. The structure also supports region-specific dashboards, enabling each region to access and view only the data relevant to its own area, offering privacy and variety of presented data amongst them.

3.5.2. User Management

Grafana supports a centralized user management model scoped within a single organization. An organization in Grafana serves as a logical boundary for dashboards, folders, data sources, and user accounts. Each user account is linked to a specific organization and can be assigned one of three predefined roles:

- **Viewer**: Can view dashboards and panels but cannot edit or create new content.

- **Editor**: Can create and modify dashboards, panels, and alerts, but cannot modify data sources or user accounts.
- **Admin**: Has full permissions within the organization, including user management, data source configuration, and dashboard creation.

User accounts can be created either through the Grafana UI by an administrator or programmatically via the HTTP API. In the default setup, Grafana uses its built-in user authentication system, which stores user credentials in a local SQLite database.

Upon installation, Grafana initializes with a default administrator account (admin). Additional users can be invited or added manually through the UI. Each user must log in with a username and password, and upon successful authentication, is granted access according to the role assigned to them.

Roles are organization-specific, which means the same user could potentially have different roles across multiple organizations — although the Free Edition supports only one organization by default. Role changes can be performed by users with the Admin role.

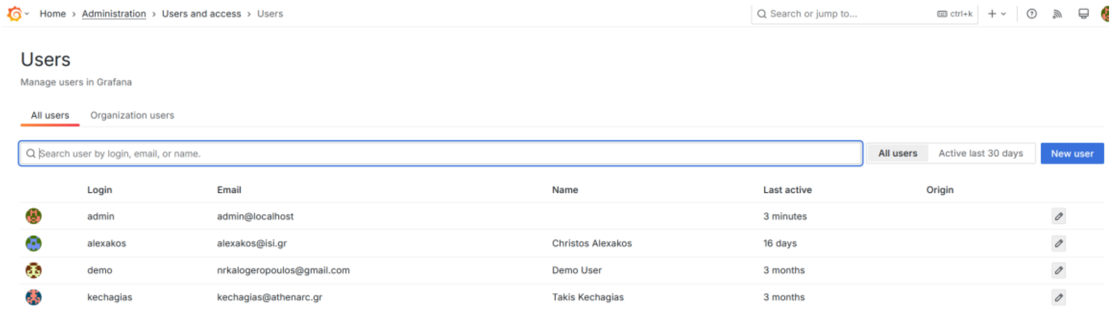

Figure 20 Users Management

In Grafana, dashboard access control is role-based and not granular per-user. Dashboards are grouped into folders, and access to a folder is governed by the role of the user within the organization. For instance, a Viewer cannot modify any dashboard, while an Editor can modify any dashboard in any folder.

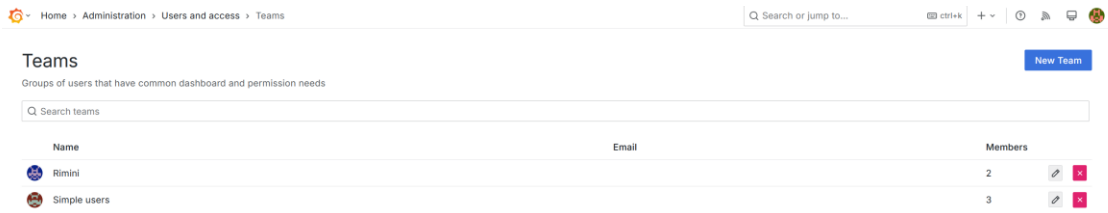

Figure 21 Groups Management

Unlike Grafana Enterprise, which supports fine-grained, per-folder and per-dashboard permissions, the Free Edition lacks support for:

- Custom roles
- Team-based permissions
- Per-dashboard access rules

As a result, access control is coarse-grained, relying entirely on the role hierarchy.

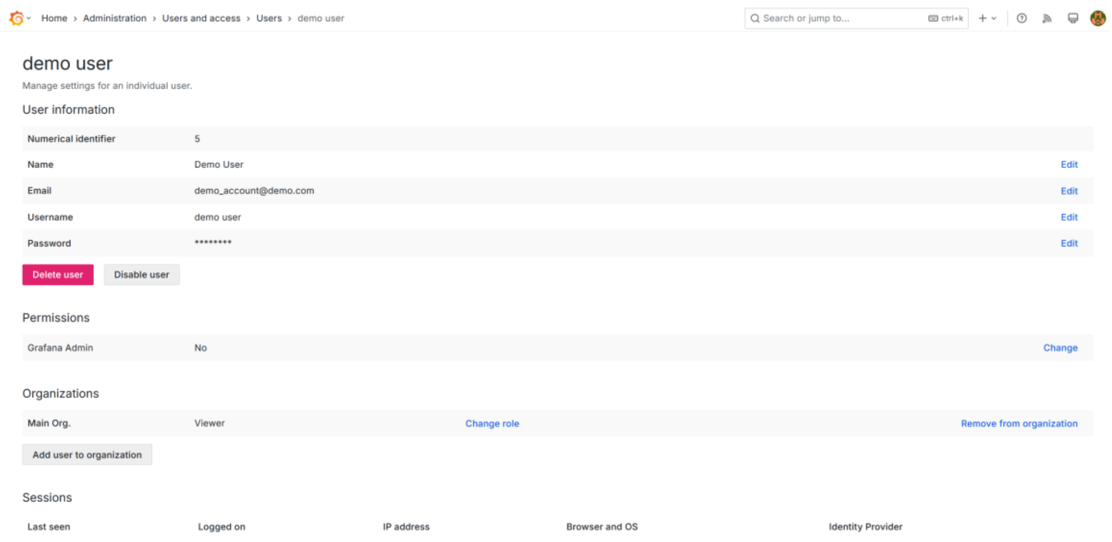

Figure 22 Account Management

3.5.3. User Security

Apache Airflow incorporates multiple layers of security to ensure the safe execution of workflows, protect sensitive configuration data, and control access to its web interface and underlying services. User authentication in Airflow can be configured through various backends, including LDAP, OAuth, Google authentication, and basic authentication. These methods allow integration with enterprise identity providers, ensuring that only authorized personnel can access the Airflow UI and manage workflows.

Role-Based Access Control (RBAC) is a core feature of Airflow's security model. Through RBAC, users are assigned roles such as Admin, User, Viewer, or Op, each with defined permissions that govern what actions can be performed within the system. This enables fine-grained control over tasks like viewing DAGs, triggering executions, editing pipelines, or managing system configurations.

Sensitive information such as credentials, API keys, or database connection strings are managed through Airflow's connection system and can be further protected using environment variables, secret backends, or external secret managers like HashiCorp Vault, AWS Secrets Manager, or Google Secret Manager. These integrations prevent secrets from being hardcoded or stored in plaintext, enhancing the overall security posture of the system.

Airflow also supports encrypted connections to its metadata database and external services. TLS can be enabled for secure communication with the webserver, scheduler, and executor components. Additionally, Airflow logs user activity and task execution status, which can be used for audit trails and monitoring. These logs can be centralized using external logging systems, such as Elasticsearch, Fluentd, or Splunk, to support compliance and forensic analysis.

Access to the Airflow REST API is also secured using authentication tokens and can be further restricted through network-level controls such as firewalls, IP whitelisting, or VPN-based access. Furthermore, when deployed in distributed environments such as Kubernetes or Docker, best practices such as role isolation, container hardening, and network segmentation should be applied to minimize the attack surface.

Moreover, Grafana implements a comprehensive security model to ensure controlled access, data protection, and system integrity within analytics and monitoring environments. User authentication and authorization are managed through various supported mechanisms, including Basic Authentication, LDAP, OAuth2, SAML, and Single Sign-On (SSO) protocols. This allows seamless integration with institutional identity providers and guarantees that access is restricted to verified users. Once authenticated, user actions are governed by Role-Based Access Control (RBAC), with roles such as Admin, Editor, and Viewer determining the scope of each user's permissions.

Active user sessions are handled securely through session tokens or cookies, with configurable timeout policies to limit exposure. Idle sessions can be terminated automatically, and administrators retain the ability to revoke sessions or enforce re-authentication when necessary. To safeguard data during transmission, Grafana supports Transport Layer Security (TLS), enabling encrypted communication between the client and server. When deployed behind a reverse proxy, TLS termination can also be configured at the proxy level to maintain end-to-end encryption.

Authentication attempts, including both successful and failed logins, are recorded in Grafana's internal logging system. These logs provide an audit trail that supports compliance, anomaly detection, and administrative oversight. For broader observability, logs can be exported to external monitoring systems such as Loki, Elasticsearch, or other SIEM platforms.

Finally, Grafana's plugin architecture includes security safeguards to ensure safe extensibility. Only signed and verified plugins from the official Grafana marketplace are recommended for production use. To maintain control over the deployment environment, administrators can restrict plugin installation to prevent unapproved extensions.

4. Conformance with DMSS Specifications

Table 1 Conformance with DMSS Specifications of D1.3.1

DMSS Specifications (D1.3.1)	Means of conformance
Process Orchestration	
Manage the flow of data between components	Managed through Apache Airflow workflows (section 3.1)
Automate and schedule workflows	Managed through Apache Airflow Scheduling capabilities (section 3.1)
Retry and re-route failed processes	Managed through Apache Airflow workflows, DAGs and scheduling (section 3.1)
Handle increasing data volumes without performance degradation	Handled by Apache Airflow, Storage Redundancy (sections 3.1 and 3.2)
Data collection	
Connect to and extract data from external APIs	Handled by Custom API clients (section 3.3.1)
Convert API responses into system-compatible formats	Handled by Custom API clients (section 3.3.1)
Allow the addition of new APIs with minimal reconfiguration	Implemented using MongoDB, a relational PostgreSQL schema, and dedicated API clients (sections 3.4.1 and 3.3.1)
Track API performance and log errors	Logging is handled by Apache Airflow and the respective API client
Extract data from targeted websites	Data extraction is orchestrated by Apache Airflow and handled mainly by APIFY actors and custom API clients (sections 3.1, 3.2.1 and 3.2.2)
Schedule and automate scraping tasks	The scheduling is a responsibility of Apache Airflow (section 3.2.4)
Remove irrelevant or redundant data from scraped results	Preprocessing to data regardless the source, orchestrated mainly by Apache Airflow and handled by source-driven preprocessing functions (sections 3.3.1 and 3.3.2)

Handle website changes (e.g., DOM structure) gracefully	APIFY actors supports error handling when the target web pages are failed to be parsed, due to changes in their DOM structure. If the changes are permanent, the update of the actors is mandatory (section 3.1)
Data Storage	
Store raw and processed data	Supported by MongoDB collections and PostgreSQL relational schema (sections 3.4.1 and 3.4.3)
Enable quick access and querying of stored data	Indices and views generated for the key elements of each database (section 3.4)
Index data for efficient retrieval	Supported and implemented to both storage options (section 3.4)
Ensure data is backed up and recoverable	Data Redundancy and Regular Backup Policy (sections 3.4 and 3.4.1)
Data Preprocessing	
Perform descriptive and inferential analytics	During visualization, the system delivers analytics tailored to predefined user specifications. (section 3.3.3)
Train and deploy machine learning models	ML tasks can be orchestrated by Airflow using DAGs to define their analysis pipeline (sections 3.1, 3.3).
Support user-defined analytics workflows	Users can define new DAGs in the Airflow defining new data analysis workflows.
Save and deploy models for future use	Any data analysis routine is saved in the DB of Airflow for future use (sections 3.1, 3.3).
Data Visualization	
Convert raw analytics outputs into display-friendly formats	Managed through Grafana using multiple visualization tools, such as dashboards, graphs, and dynamic plots (section 3.5.1)
Ensure smooth integration between backend and frontend	Grafana integrates seamlessly with the PostgreSQL database and, through its plugin system, also supports connections to MongoDB (section 3.5)
Validate data before presentation	The validation is conducted by validating the queries before their execution and data visualization in Grafana(section 3.5)

Log and resolve errors in data preparation	The system and its modules produce error and warning logs. For data preparation, Grafana's logging system is used (section 3.5.3)
Allow users to create and customize dashboards	User access and permissions are administered through Grafana's integrated user management system (section 3.5.2)
Display live updates of data	Real-time data visualization and live updates are managed through Grafana's dynamic dashboard capabilities (section 3.5.1)
Support exporting dashboards and charts	Grafana provides functionality for exporting dashboards and visualizations in various formats for reporting and sharing purposes (section 3.5.1)
Enable users to users to filter and drill down into visualizations, input data, query datasets and view visualizations	Grafana enables users to design, modify, and persist dashboards according to their specific analytical and monitoring requirements (section 3.5.2)
Allow users to create, modify and save dashboards for specific needs	User access and permissions are administered through Grafana's integrated user management system (section 3.5.2)
Users' management	
Authenticate users before granting access	Grafana and Apache Airflow enforces user authentication prior to granting access, ensuring secure and role-based entry to the system (section 3.5.2)
Manage active sessions securely	Grafana and Apache Airflow manages active user sessions securely through a combination of session tokens, configurable timeouts, and role-based access controls. Supports secure transmission using HTTPS, ensuring that session data is encrypted in transit. (section 3.5.3)
Enforce role-based permissions	User roles are managed through Grafana's integrated user and team management system for visualization tasks, while

	Apache Airflow employs a similar role-based access control mechanism to govern workflow creation and execution (see Sections 3.5.2 and 3.5.3)
Record authentication attempts	Grafana records authentication attempts through its built-in logging system, when security and audit logging are enabled (section 3.5.3)

5. Conclusion

The Libeccio DMSS represents a significant advancement in tourism policy development and destination management capabilities, offering a comprehensive solution that addresses the complex data challenges facing modern tourism authorities. Through its integrated architecture combining data collection, processing, analytics, and visualization components, the DMSS transforms fragmented tourism information into a unified intelligence platform that supports evidence-based decision making. The system's ability to seamlessly integrate diverse data sources, from traditional statistical databases to real-time web scraping and API feeds, provides policymakers with unprecedented visibility into tourism dynamics and trends that were previously difficult to capture and analyse.

The modular design and scalable architecture of the DMSS ensure that the system can adapt to varying organizational needs and technical environments while maintaining robust performance and reliability. The combination of Elasticsearch and MongoDB for data storage, coupled with advanced analytics capabilities and Grafana-powered visualization, creates a flexible platform that can evolve with changing requirements and technological advances. This adaptability is particularly crucial in the rapidly evolving tourism sector, where new data sources, analytical techniques, and policy challenges continuously emerge.

Looking forward, the DMSS establishes a foundation for more sophisticated tourism intelligence applications, including predictive modelling for visitor flows, impact assessment tools for policy interventions, and real-time monitoring systems for destination capacity management. The comprehensive data pipeline and analytical infrastructure provided by the DMSS enable tourism authorities to move from reactive policy responses to proactive strategic planning, supporting more sustainable and effective destination management practices.

The successful implementation of the DMSS has the potential to fundamentally transform how tourism policies are developed, implemented, and evaluated, creating a new paradigm for data-driven governance in the tourism sector. By providing stakeholders with timely, accurate, and comprehensive insights, the system empowers more informed decision making that can better balance the competing demands of economic development, visitor experience, community well-being, and environmental sustainability that define successful tourism destinations in the contemporary global marketplace.

6. Appendixes

6.1. Appendix A: Rational Database Schema

Table accommodation

Idx	Column Name	Data Type
*	uid	serial
	name	text
	house_description	text
	language	text
	type	text
	room_tier	text
	email	text
	website	text
	property_url	text
	images	text
	highlights	text[]
	number_of_rooms	integer
	last_updated	timestamp DEFAULT CURRENT_TIMESTAMP
	phone	text
	source	text
	source_id	text
	latitude	double precision
	longitude	double precision
	address_full	text
	address_street	text
	address_region	text
	address_country	text
	address_postalcode	text
Indexes		
	accommodation_pkey	Primary Key ON uid

Idx	Column Name	Data Type
	unique_source_id	Unique Key ON source_id
Referring Foreign Key		
	reviews_accommodationid_fkey	source_id ↙ ❏ reviews(accommodationid)
	accommodation_area_accommodation_source_id_fk	source_id ↙ ❏ accommodation_area(accommodationid)
	accommodationid_fk	source_id ↙ ❏ accommodation_amenity(accommodationid)
	room_accommodationid_fkey	source_id ↙ ❏ room(accommodationid)
	reviews_new_id_acc_fkey	source_id ↙ ❏ reviews_new(id_acc)

Table accommodation_amenity

Idx	Column Name	Data Type
*	accommodationid	text
*	amenityid	text
Indexes		
	accommodation_amenity_pkey	Primary Key ON accommodationid, amenityid
Foreign Key		
	amenityid_fk	amenityid ↗ ❏ amenity(id)
	accommodationid_fk	accommodationid ↗ ❏ accommodation(source_id)

Table accommodation_area

Idx	Column Name	Data Type
*	accommodationid	text
*	areaid	integer

Idx	Column Name	Data Type
Indexes		
	accommodation_area_pkey	Primary Key ON accommodationid, areaid
Foreign Key		
	accommodation_area_areaid_fkey	areaid ↗ ❑ area(id)
	accommodation_area_accommodation_source_id_fk	accommodationid ↗ ❑ accommodation(source_id)

Table airports

General information and mapping for airports

Idx	Column Name	Definition	Description
*	id	integer DEFAULT nextval('"Airports_id_seq"'::regclass)	
*	code	varchar(10)	
	name	text	
	icao_code	varchar(10)	
	country_code	varchar(2)	
Indexes			
	Airports_pkey	Primary Key ON id	
	Airports_code_key	Unique Key ON code	
Referring Foreign Key			
	airport_B_code	code ↙ ❑ flights(airport_B_code)	
	airport_A_code	code ↙ ❑ flights(airport_A_code)	
	passenger_statistics_code_fkey	code ↙ ❑ passenger_statistics	

Table amenity

Idx	Column Name	Data Type
*	id	text DEFAULT nextval('amenity_id_seq'::regclass)
*	amenity	text
Indexes		
	amenity_pkey	Primary Key ON id
Referring Foreign Key		
	amenityid_fk	id ↙ ❑ accommodation_amenity(amenityid)

Table area

Idx	Column Name	Data Type
*	id	serial
*	name	text
	latitude	double precision
	longitude	double precision
	parent_area_id	integer
	coordinates	text
	area_type	text
Indexes		
	area_pkey	Primary Key ON id
Foreign Key		
	fk_parent_area	parent_area_id ↗ ❑ area(id)
Referring Foreign Key		
	accommodation_area_areaid_fkey	id ↙ ❑ accommodation_area(areaid)
	fk_parent_area	id ↙ ❑ area(parent_area_id)

Table area_capacity

Idx	Column Name	Data Type
*	id	serial

Idx	Column Name	Data Type
*	area_id	integer
	type	text
	bathrooms	integer
	bedrooms	integer
	beds	integer
	establishments	integer
	stars	integer
*	year	integer
Indexes		
	area_capacity_pkey	Primary Key ON id
Foreign Key		
	area_capacity_area_id_fkey	area_id ↗ ❑ area_info(id)

Table area_info

Idx	Column Name	Data Type
*	id	serial
	region	text
	province	text
	country	text
	municipality	text
	province_code	integer
	country_code	text
	minicipality_code	integer
	geojson_polygon	geometry
	region_code	integer
Indexes		
	area_info_pkey	Primary Key ON id
	unique_place	Unique Key ON region, province, municipality, country
Referring Foreign Key		

Idx	Column Name	Data Type
	tourism_quarterly_area_id_fkey	id ↙ ❑ tourism_quarterly(area_id)
	area_capacity_area_id_fkey	id ↙ ❑ area_capacity(area_id)
	area_occupancy_area_id_fkey	id ↙ ❑ area_occupancy(area_id)

Table area_occupancy

Idx	Column Name	Data Type
*	id	integer DEFAULT nextval('area_occupancy_id_seq'::regclass)
*	area_id	integer
	arrivals_hotels_residents	integer
	arrivals_hotels_foreigners	integer
	arrivals_other_residents	integer
	arrivals_other_foreigners	integer
	nights_spent_hotels_residents	integer
	nights_spent_hotels_foreigners	integer
	nights_spent_other_residents	integer
	nights_spent_other_foreigners	integer
*	year	integer
Indexes		
	area_occupancy_pkey	Primary Key ON id
Foreign Key		
	area_occupancy_area_id_fkey	area_id ↗ ❑ area_info(id)

Table availability

Idx	Column Name	Data Type
*	id	serial
*	room_id	text
	price	numeric(10,2)
	option_cancellationtype	text
	persons	integer
	checkindate	timestamp
	checkoutdate	timestamp
	future_interval_days	integer
Indexes		
	availability_pkey	Primary Key ON id
	unique_room_checkin_checkout_interval	Unique Key ON room_id, persons, checkindate, checkoutdate, future_interval_days
Foreign Key		
	availability_roomid_fkey	room_id ↗ ❑ room

Table category_type

Idx	Column Name	Data Type
*	id	serial
*	name	text
Indexes		
	category_type_pkey	Primary Key ON id
	category_type_name_key	Unique Key ON name
Referring Foreign Key		
	event_category_category_id_fkey	id ↙ ❑ event_category(category_id)

Table event

Idx	Column Name	Data Type
*	uid	integer DEFAULT nextval('event_uid_seq'::regclass)
*	source	text
	times	text
	lang	text
	permalink	text
	title	text
	subtitle	text
	ticketing_website	text
	ticketing_entrance	text
	updated_at	timestamp
	pub_date	timestamp
	description	text
*	source_id	text
	dates_from	date
	dates_to	date
Indexes		
	event_pkey	Primary Key ON uid
Referring Foreign Key		
	event_theme_event_id_fkey	uid ✓ ❑ event_theme(event_id)
	event_category_event_id_fkey	uid ✓ ❑ event_category(event_id)

Table event_category

Idx	Column Name	Data Type
*	event_id	integer
*	category_id	integer
Indexes		
	event_category_pkey	Primary Key ON event_id, category_id
Foreign Key		

Idx	Column Name	Data Type
	event_category_category_id_fkey	category_id ↗ ❑ category_type(id)
	event_category_event_id_fkey	event_id ↗ ❑ event(uid)

Table event_theme

Idx	Column Name	Data Type
*	event_id	integer
*	theme_id	integer
Indexes		
	event_theme_pkey	Primary Key ON event_id, theme_id
Foreign Key		
	event_theme_theme_id_fkey	theme_id ↗ ❑ theme(id)
	event_theme_event_id_fkey	event_id ↗ ❑ event(uid)

Table flights

Idx	Column Name	Data Type
*	id	serial
	last_updated	date
	freq	text
	unit	text
	tra_meas	text
	airp_pr	text
	TIME_PERIOD	date
*	airport_A	text
*	airport_A_code	text
*	airport_B	text
*	airport_B_code	text
	OBS_VALUE	integer
Indexes		

Idx	Column Name	Data Type
	flights_pkey	Primary Key ON id
Foreign Key		
	airport_B_code	airport_B_code ↗ ❑ airports(code)
	airport_A_code	airport_A_code ↗ ❑ airports(code)

View geography_columns

Idx	Column Name	Data Type
	f_table_catalog	name
	f_table_schema	name
	f_table_name	name
	f_geography_column	name
	coord_dimension	integer
	srid	integer
	type	text

View geometry_columns

Idx	Column Name	Data Type
	f_table_catalog	varchar(256)
	f_table_schema	name
	f_table_name	name
	f_geometry_column	name
	coord_dimension	integer
	srid	integer
	type	varchar(30)

Table passenger_statistics

Idx	Column Name	Data Type
*	id	serial

Idx	Column Name	Data Type
	airport_name	text
	domestic_passengers	integer
	domestic_percent	double precision
	international_passengers	integer
	international_percent	double precision
	eu_passengers	integer
	eu_percent	double precision
	general_aviation_passengers	integer
	general_aviation_percent	double precision
	total_passengers	integer
	total_percent	double precision
*	year	integer
*	month	integer
	matched_airport	text
	code	varchar(20)
Indexes		
	passenger_statistics_pkey	Primary Key ON id
Foreign Key		
	passenger_statistics_code_fkey	code ↗ ❏ airports

Table reviews

Idx	Column Name	Data Type
*	id	serial
	accommodationid	text
	rating	integer
	overall_rating	double precision
	review_count	integer
	staff_rating	double precision
	facilities_rating	double precision

Idx	Column Name	Data Type
	cleanliness_rating	double precision
	comfort_rating	double precision
	value_for_money_rating	double precision
	location_rating	double precision
	free_wifi_rating	double precision
Indexes		
	reviews_pkey	Primary Key ON id
Foreign Key		
	reviews_accommodationid_fkey	accommodationid ↗ ❏ accommodation(source_id)

Table reviews_new

Idx	Column Name	Data Type
	id_acc	text
	response	text
	checkoutdate	date
	checkindate	date
	language	text
	title	text
	context	text
	review_date	date
*	id	serial
	rev_id	text
	place_name	text
	place_type	text
	traveler_type	text
	number_nights	integer
	room_info	text
Indexes		
	reviews_new_pkey	Primary Key ON id

Idx	Column Name	Data Type
	reviews_new_rev_id_key	Unique Key ON rev_id
Foreign Key		
	reviews_new_id_acc_fkey	id_acc ↗ ❏ accommodation(source_id)
Referring Foreign Key		
	revs_new_new_original_id_fkey	id ↙ ❏ revs_new_new(original_id)

Table revs_new_new

Idx	Column Name	Data Type
	original_id	integer
	id_acc	varchar(255)
	context	text
	language	varchar(50)
	rev_id	varchar(255)
	translation	text
	sentiment	varchar(50)
Foreign Key		
	revs_new_new_original_id_fkey	original_id ↗ ❏ reviews_new(id)

Table room

Idx	Column Name	Data Type
*	id	serial
*	accommodationid	text
*	room_id	text
	room_type	text
	capacity	integer
	description	text
	images	text[]
	amenities	text

Idx	Column Name	Data Type
	time_scrape	timestamp DEFAULT CURRENT_TIMESTAMP
	checkindate	date
	checkoutdate	date
Indexes		
	room_pkey	Primary Key ON id
	unique_room_id	Unique Key ON accommodationid, room_id
	unique_room_id_text	Unique Key ON room_id
Foreign Key		
	room_accommodationid_fkey	accommodationid ↗ ❑ accommodation(source_id)
Referring Foreign Key		
	availability_roomid_fkey	room_id ↙ ❑ availability

Table routes

Idx	Column Name	Data Type
*	id	serial
	city	text
	street_address	text
	origin_latitude	double precision
	origin_longitude	double precision
	destination_latitude	double precision
	destination_longitude	double precision
Indexes		
	routes_pkey	Primary Key ON id
Referring Foreign Key		
	fk_route	id ↙ ❑ waypoints(route_id)
	fk_route	id ↙ ❑ traffic(route_id)

Table sailings

Idx	Column Name	Data Type
*	id	serial
	structure_name	text
	freq	char(4)
	direct	char(10)
	tra_cov	text
	transport_coverage	text
	unit	text
	unit_of_measure	text
	rep_mar	text
	maritime_entity_reporting	text
	time_period	char(7)
	obs_value	numeric
Indexes		
	sailings_pkey	Primary Key ON id

Table spatial_ref_sys

Idx	Column Name	Data Type
*	srid	integer
	auth_name	varchar(256)
	auth_srid	integer
	srtext	varchar(2048)
	proj4text	varchar(2048)
Indexes		
	spatial_ref_sys_pkey	Primary Key ON srid
Constraints		
	spatial_ref_sys_srid_check	srid > 0) AND (srid <= 998999

Table theme

Idx	Column Name	Data Type
*	id	serial
*	name	text
Indexes		
	theme_pkey	Primary Key ON id
	theme_name_key	Unique Key ON name
Referring Foreign Key		
	event_theme_theme_id_fkey	id ↙ ❑ event_theme(theme_id)

Table tourism_quarterly

Idx	Column Name	Data Type
*	id	serial
*	year	integer
*	quarter	integer
	ft_expenditure	numeric
	ft_number	integer
	ft_number_overnight_stays	integer
*	area_id	integer
Indexes		
	tourism_quarterly_pkey	Primary Key ON id
Foreign Key		
	tourism_quarterly_area_id_fkey	area_id ↗ ❑ area_info(id)

Table traffic

Idx	Column Name	Data Type
*	id	serial
*	route_id	integer
*	distancemeters	integer
*	duration	integer
*	polyline	text

Idx	Column Name	Data Type
*	speedreadingintervals	jsonb
	speedreadingintervals.speed	_string
	speedreadingintervals.endPolylinePointIndex	_number
	speedreadingintervals.startPolylinePointIndex	_number
*	timestamp	timestamp
Indexes		
	traffic_pkey	Primary Key ON id
Foreign Key		
	fk_route	route_id ↗ ❑ routes(id)

Table waypoints

Idx	Column Name	Data Type
*	id	serial
	route_id	integer
	waypoint_index	integer
	latitude	double precision
	longitude	double precision
Indexes		
	waypoints_pkey	Primary Key ON id
Foreign Key		
	fk_route	route_id ↗ ❑ routes(id)