Politecnico di Torino

Master’s degree Program in

Digital Skills for Sustainable Societal Transitions

Master's Degree Thesis

Backend Design of a Multi-Scale Energy Data Platform for Territorial Energy Analysis

Supervisor:

Guglielmina Mutani

Candidate:

Keyvan Sharokhi

A.Y. 2025/2026

Contents

Abstract 10

1. Introduction 11

1.1. Context and Motivation 11

1.2. Objectives 12

1.3. Contributions 12

1.4. Scope and Assumptions 14

1.5. Thesis Organization 14

2. ackground and Related Work 16

2.1. Territorial Energy Planning and Data Challenges 16

Multi-Scale Governance and Spatial Aggregation 16

Temporal Heterogeneity and Analytical Complexity 17

Backend Implications for Analytical Systems 18

2.2. Multidimensional Data Models for Analytical Systems 19

Analytical vs. Transactional Data Modeling 19

The Star Schema as a Reference Model 20

Dimensions as Explicit Analytical Concepts 21

Multidimensional Models in Energy Analytics 21

Addressing the Limitations of Domain-Specific Tables 22

Extensibility and Long-Term Maintainability 22

2.3. Spatial Data in Web-Based Energy Platforms 22

Vector Geometries and Administrative Boundaries 22

GeoJSON as a Web-Oriented Spatial Format 23

Performance Challenges in Spatial Data Delivery 23

Geometry Simplification for Interactive Visualization 24

Linking Spatial and Analytical Data 24

Implications for Backend Design 25

2.4. Backend Architectures for Interactive Dashboards 26
Separation of Concerns Between Backend and Frontend 26
Analytical APIs Versus Data-Centric APIs 28
Balancing Flexibility and Control 28
Implications for Energy Planning Platforms 29
2.5. Scenario-Based Energy Analysis 29
Baseline and Scenario Conceptualization 30
Derived Indicators in Energy Planning 30
Backend Implications for Scenario Computation 32
2.6. Summary 32
3. System Requirements and Architecture Overview 34
3.1. Objectives of the Platform 34
3.2. Functional Requirements 35
Multi-Scale Spatial Querying 35
Multi-Temporal Analysis Support 36
Dynamic Filtering and Parameter Validation 36
Scenario Preview and Persistence 37
3.3. Non-Functional Requirements 37
Performance 38
Scalability 38
Maintainability and Clarity 38
Interoperability 38
3.4. High-Level System Architecture 39
3.5. Data Flow and Request Handling 41
3.6. Design Rationale 41
3.7. Summary 42
4. Database Design and Data Modeling 43

4.1. Motivation for the Database Design 43
4.2. Choice of a Multidimensional Data Model 43
4.3. Fact Table Design 45
4.4. Time Dimension and Temporal Resolution 46
4.5. Territory Dimension and Spatial Hierarchies 47
4.6. Energy Categories and Analytical Domains 49
Analytical Domains 49
Base Groups and Energy Sources 50
Role in Scenario Computation 50
Supporting Derived Indicators 51
Design Considerations and their Trade-offs! 51
4.7. Scenario Dimension 52
4.8. Keys, Constraints, and Data Integrity 52
4.9. Summary 53
5. Backend Implementation and API Design 54
5.1. Role of the Backend in the Platform 54
5.2. Technology Stack and Design Choices 54
5.3. Backend Structure and Modularization 56
5.4. API Design Principles 58
5.5. Request Validation and Parameter Handling 58
5.6. Data Aggregation and Query Execution 59
5.7. Scenario Preview and Persistence Logic 61
5.8. Summary 62
6. API Design and Endpoints 63
6.1. Design Principles of the API 63
6.2. Map Endpoints 64
Purpose and Scope of Map Endpoints 64

Separation Between Analytical Values and Geometries 65
Territorial Aggregation Strategy 66
GeoJSON Delivery for Web Mapping 67
Validation and Constraints 67
Design Implications 68
6.3. Chart Endpoints 68
Purpose and Analytical Scope 68
Time-Series Versus Aggregated Views 69
Territorial Context in Chart Queries 69
Output Structure and Ordering Guarantees 70
Interaction with Scenarios 70
Design Implications 70
6.4. Scenario Endpoints 71
Purpose of Scenario Endpoints 71
Baseline and Scenario Context 71
Scenario Listing and Metadata 71
Scenario Preview Endpoints 72
Scenario Persistence Endpoints 73
Territorial Scope of Scenarios 73
Output Structure and Indicator Integration 73
Design Rationale 74
6.5. Parameter Metadata Strategy 74
Motivation for Explicit Parameter Metadata 74
Types of Parameters Described by Metadata 74
Structure and Exposure of Parameter Metadata 75
Validation and Constraint Enforcement 76
Benefits for Maintainability and Extensibility 77

Relationship with Scenario and Indicator Computation 77
6.6. API Endpoints Summary 77
Endpoint Categories and Responsibilities 77
Consistency Across Endpoints 78
Error Handling and Analytical Integrity 79
Role of the API Layer in the Overall Architecture 79
7. Scenario Computation and Indicators 80
7.1. Scenario-Based Reasoning in Territorial Energy Planning 80
Scenarios as Computational Layers 80
Analytical Scope of Scenario Computation 81
Role of Indicators in Scenario Evaluation 81
7.2. Baseline Data and User-Defined Scenarios 81
Baseline Data as Reference State 81
User-Defined Scenarios as Transformations 82
Analytical Implications of the Separation 83
Traceability and Validation 83
Position Within the Scenario Pipeline 83
7.3. Scenario Preview vs. Scenario Persistence 84
Scenario Preview: Exploratory Analysis 84
Scenario Persistence: Formalized Scenarios 85
Analytical Integrity and System Design Implications 85
Position in the Scenario Computation Pipeline 85
7.4. Indicator Definitions and Computation Logic 86
General Computation Principles 86
Core Indicators 86
Ratio-Based Indicators 87
7.5. Scenario Computation Pipeline 88

Pipeline Overview 88
Baseline Data Retrieval 89
Scenario Parameter Application 89
Value Aggregation 89
Indicator Computation 90
Result Packaging and Delivery 90
Why This Pipeline Matters 92
8. Spatial Data Administration 93
Even though spatial data are shown on maps, they are not handled as a separate visualization layer. Instead, spatial processing is built into the backend so that geographic features match the analytical results. This setup lets the platform provide accurate maps, process data efficiently, and stay flexible for different analysis and visualization needs 93
8.1. Role of Spatial Data in Territorial Energy Analysis 93
Design Principles for Spatial Data Handling 94
8.2. Geometry Storage and GeoJSON Generation 94
Geometry Storage Strategy 95
GeoJSON as a Delivery Format 95
On-Demand Geometry Generation 95
Payload Control and Efficiency 96
8.3. Geometry Simplification and Performance Optimization 97
Motivation for Geometry Simplification 97
Simplification Strateg 97
Caching and Reuse 98
Trade-offs and Limitations 98
Role in the Overall Architecture 98
9. Discussion, Limitations, and Future Work 100
9.1. Discussion of Design Choices 100
9.2. Limitations of the Current Implementation 100

Spatial Data Handling Constraints 100
Performance and Scalability Considerations 101
Scenario Modeling Scope 101
9.3. Future Work and Extensions 101
Database and Performance Enhancements 101
Advanced Scenario Families 102
Spatial and Analytical Integration 102
Interoperability and Governance 102
9.4. Concluding Remarks 102
10. Conclusions 104
10.1. Summary of Achievements 104
10.2. Relevance for Territorial Energy Planning 105
10.3. Lessons Learned 105
10.4. Final Remarks 105
References 106

Abstract

As more energy-related data becomes available at local and regional levels, there are new opportunities for evidence-based energy planning. To use this data effectively, backend systems must handle different types of information across various locations, time periods, and analysis needs. This thesis describes how the backend for a multi-scale energy information system was designed and built in Italy to support analysis and decision-making for municipalities, provinces, and regions.

At first, the data was stored in several domain-specific tables, each with its own structure for energy consumption, production, and future scenarios. While this worked, it led to repeated data, less flexibility, and made it harder to combine data from different administrative levels or compare scenarios. To solve these problems, the backend was redesigned using a multidimensional database model based on data warehousing principles. A star-schema was used, with a main fact table for energy values in megawatt hours (MWh), and dimension tables for territory, time, energy category, and scenario details (Kimball & Ross, 2013). This structure makes it easier to combine data across locations and time periods, and keeps data sources clear.

The backend uses Python and the Flask framework and sits above the database layer. Its API provides endpoints to get aggregated values for choropleth maps, time-series data for interactive charts, and detailed scenario parameters. Spatial data from the database is sent to the frontend as GeoJSON, which works well with web-based mapping libraries (H. Butler, 2016). The backend lets users filter data by year, month, season, day type, and energy category, making it easier to analyze energy patterns. Users can preview and save scenarios, allowing them to test changes to energy production mixes and see how these changes affect self-consumption, self-sufficiency, and over-production. This new backend is much easier to maintain, scale, and extend than the original version (Lorenzo Bottaccioli, 2025). This setup creates a strong base for adding more datasets, analysis methods, and planning scenarios in the future. It also helps make territorial energy assessments more transparent and data-driven.

1. Introduction

1.1. Context and Motivation

Energy planning increasingly requires an evidence-based and territorial approach. Public administrations, energy agencies, and local stakeholders have to understand patterns of energy consumption and renewable energy generation across administrative levels, including municipalities, provinces, and regions. Energy datasets often differ in format, timing, and meaning. For example, there are differences between consumption and production data, or between past records and future projections (Di Paolo, Di Martino, Di Battista, Carapellucci, & Cipollone, 2023). These differences make it hard to combine data from various sources, which are often collected at different times and organized in different ways. As a result, it becomes difficult to answer key planning questions, like finding areas with the highest energy use, studying how production changes by month or season, identifying the most important renewable sources, or seeing how things might change under different scenarios (European Commission, 2018).

From a software engineering perspective, answering these evaluative questions requires more than high-quality data visualization. Interactive maps and charts rely on a backend capable of storing energy data consistently, aggregating it over multiple spatial hierarchies and temporal resolutions, and exposing reliable application programming interfaces (APIs). Because the same dataset supports multiple frontend components including choropleth maps, time-series charts, and scenario comparisons, the design of both the database and the API has a direct impact on the system's maintainability consistency and correctness (Valdes, 2020).

This thesis focuses on the design of the backend for a multi-scale energy platform developed to support territorial energy analysis in Italy. The platform enables users to explore energy data at the municipality, province, and regional levels while working with various temporal resolutions, including hourly, monthly, and annual data. It also provides a clear distinction between the main energy domains considered in the analysis, namely consumption, production, and future production.

Another important motivation for this work is the growing role of scenario-based analysis in energy planning. Evaluating alternative assumptions—such as increasing renewable energy capacity or modifying the production mix—has become an essential part of the planning process. To support this type of analysis, the backend includes mechanisms for previewing, comparing, and storing scenarios in a way that supports transparency and makes certain that results can be reproduced consistently (Morrison, 2018).

1.2. Objectives

The main objective of this work is to design and implement a backend architecture that enables multi-scale territorial energy analysis by a web application. The specific objectives are as follows:

- **O1 — Data modeling for analytics**
  The application provides a database structure that represents energy measures with **consistent semantics** over spatial scales, temporal resolutions, energy categories, and scenarios, while limiting redundancy and assisting analytical queries.
- **O2 — Multi-scale aggregation**
  Enable aggregation queries that work flawlessly through municipalities, provinces, and regions, as well as across multiple temporal resolutions (hourly, monthly, and annual), with some optional filters such as month and day type.
- **O3 — Spatial data for web-based maps**
  It delivers territorial geometries in **GeoJSON format** suitable for interactive mapping and implements strategies to simplify geometry shapes and reduce payload size, thereby ensuring to guarantee responsive client-side visualization.
  .
- **O4 — API layer for interactive dashboards**
  At this part, it designs and implements a set of **RESTful API endpoints** and supplies aggregated values choropleth-ready, chart-oriented time series, and scenario-related parameters, including explicit validation rules and predictable response structures.
- **O5 — Examining new scenarios and their persistence**
  At last, support both predefined and user-defined scenarios, persistence, enabling scenario preview and storage of computed outputs and indicators, while preserving transparency and reproducibility.

1.3. Contributions

This thesis provides the following contributions to the backend design for territorial energy analysis:

1. **A multidimensional database model for energy analytics**

   The thesis introduces and develops a multifaceted database model centered on a fact table that records energy measurements, supported by dimension tables that detail territory, time, energy categories, and scenarios. This structure enables consistent analysis across

geographic and temporal levels and decreases redundancy relative to domain-specific table structures. (Cavalheiro & Carreira, 2016)

2. **A modular backend architecture based on Flask**

   A backend system is implemented using the Flask framework and structured around a modular, blueprint-based architecture. The backend includes configuration for cross-origin requests and response compression and is designed to operate as an independent analytical service consumed by a separate web frontend. This design approach stresses maintainability, allocation of responsibility, and extensibility. (Crudu, 2025)

3. **A set of RESTful API endpoints for interactive analysis**

   The backend exposes a collection of RESTful endpoints that provide:

   - Aggregated territorial values for choropleth mapping.
   - Time-series values for charts at different resolutions.
   - Access to scenario metadata and parameters.
   - Scenario preview and persistence functionality.

The API's design stresses analytical concepts over detailed database structures, making sure a consistent experience for the frontend.

4. **A scenario computation layer**

   The system includes a computation layer that derives planning-relevant indicators, such as self-consumption and self-sufficiency indices, based on explicit relationships between consumption and production values. Scenario logic is implemented transparently at the backend level, supporting both exploratory previews and persistent scenario storage. (Ardebili, A., M., & A.I.H.A., 2024)

5. **A spatial data administration approach**

   The thesis proposes an approach for handling and delivering territorial geometries in GeoJSON format. It includes geometry simplification techniques that reduce payload size without sacrificing usability in interactive maps. This enables efficient integration of spatial data with analytical results on web-based visualization platforms. (Butler, 2016) (McMaster & Shea, 1992)

1.4. Scope and Assumptions

This thesis focuses on the design and implementation of the backend architecture and data layer of a territorial energy analysis platform. The frontend is treated as a consumer of the exposed APIs and is therefore not the primary subject of the implementation discussion. Frontend-related aspects are considered only insofar as they influence backend requirements and interface design.

The platform targets the Italian territory and operates at three administrative levels: municipalities, provinces, and regions. Energy analysis is managed using values expressed in megawatt-hours (MWh), covering consumption, production, and future production domains. It is assumed that input datasets are available in a structured form suitable for amalgamation into the database, following prior ingestion and data cleaning steps that are outside the scope of this work.

The proposed architecture prioritizes analytical flexibility, clarity, and long-term maintainability over complete performance optimization. Performance factors are addressed in the database schema design, query formulation, aggregation strategies, and spatial data delivery geometry simplification levels. More sophisticated optimization methods, such as materialized views, extensive caching layers, or distributed query execution, are intentionally excluded from the implementation and are discussed as directions for future work.

No assumptions are made regarding real-time data collection or transactional workloads. The platform is primarily designed for exploratory and relative analysis rather than for operational monitoring.

1.5. Thesis Organization

The remainder of this thesis is organized as follows.

- **Chapter 2** – Background and Related Work discusses the conceptual and technical foundations of the work. It reviews key aspects of territorial energy planning, multidimensional data modeling for analytical systems, and backend architecture for interactive, map-based dashboards. Relevant academic and technical literature is discussed in this chapter.
- **Chapter 3** – Requirements and System Overview presents the platform's functional and non-functional requirements and provides a high-level overview of the adopted system architecture, describing the interactions among the database layer, the Flask-based backend, and the web frontend.

- **Chapter 4, Database Design and Data Modeling** introduces the evolution from domain-specific tables to a multidimensional star schema. The backend presents the design of fact and dimension tables, data sources, keys, constraints, and indexing strategies. The chapter also covers the main ideas behind aggregation and how indicators are calculated.
- **Chapter 5, Backend Implementation and API Design,** looks at how the backend application is organized. It covers the structure of the Flask project, the modularization approach, tools for accessing the database, how requests are validated, how errors are handled, and the general request and response process.
- **Chapter 6, API Design and Endpoints,** describes the REST API built for the platform. It introduces the main endpoints used for maps, charts, and scenario analysis. The chapter also explains how parameters and metadata are handled and ends with a summary of the available API endpoints.
- **Chapter 7, Scenario Computation and Indicators,** focuses on how the platform handles scenario analysis. It explains how baseline data and scenarios are managed, the difference between previewing and saving scenarios, the indicators that have been implemented, and how the scenario computation process works.
- **Chapter 8, Spatial Data Administration,** discusses how spatial information is managed in the platform. It covers how geometry is stored, how GeoJSON files are generated, techniques for simplifying geometry, performance considerations, and the main limitations of the current setup.
- **Chapter 9, Discussion, Limitations, and Future Work,** offers a critical look at the proposed design. It discusses the main limitations of the current implementation and suggests possible directions for future development and improvements.
- **Chapter 10, Conclusions,** sums up the main contributions of the thesis, highlights their importance for territorial energy planning, and ends with some final thoughts.

2. ackground and Related Work

2.1. Territorial Energy Planning and Data Challenges

Territorial energy planning helps public authorities, energy agencies, and local stakeholders analyze energy use, production, and renewable generation over time and across different areas. At the municipal, provincial, and regional levels, planning should be based on evidence, be transparent, and support long-term climate and energy goals. Planners often need to find energy-intensive areas, map where renewable energy is produced, and consider how different policies or infrastructure changes could affect future energy use. But this work often faces challenges because the data is fragmented and inconsistent. Local and regional energy data usually come from utility companies, national statistics, sector studies, and scenario reports. These sources often use different formats, time frames, levels of detail, and definitions. For example, energy use might be reported once a year for each municipality, while renewable energy production could be tracked monthly or even hourly at the provincial or regional level. Past data and future projections are also often stored separately and organized in different ways (Scaramuzzino, Garegnani, & Zambelli, 2019).

These differences make it difficult to compare data between regions, identify trends over time, or accurately assess the impact of policies and infrastructure. Because of this, planners often have to process data by hand, use spreadsheets, or create their own scripts. This approach is harder to repeat and increases the chance of errors.

Multi-Scale Governance and Spatial Aggregation

One main challenge in territorial energy planning is working across different geographic levels at the same time. Municipalities, provinces, and regions each have their own planning roles and authority. Municipalities usually handle local actions and putting plans into practice. Provinces manage infrastructure or policies that cover a wider area, while regions focus on overall strategy and making sure regulations match up (Marks, 1993; ESPON, 2018).

However, energy data are usually collected at the most detailed level possible and then combined for planning. This combining process is often done by hand or with methods that are not well documented. Problems can arise when the rules for combining data are built into software or repeated in different datasets, leading to inconsistencies. For example, municipal data might be added up in different ways depending on the analysis.

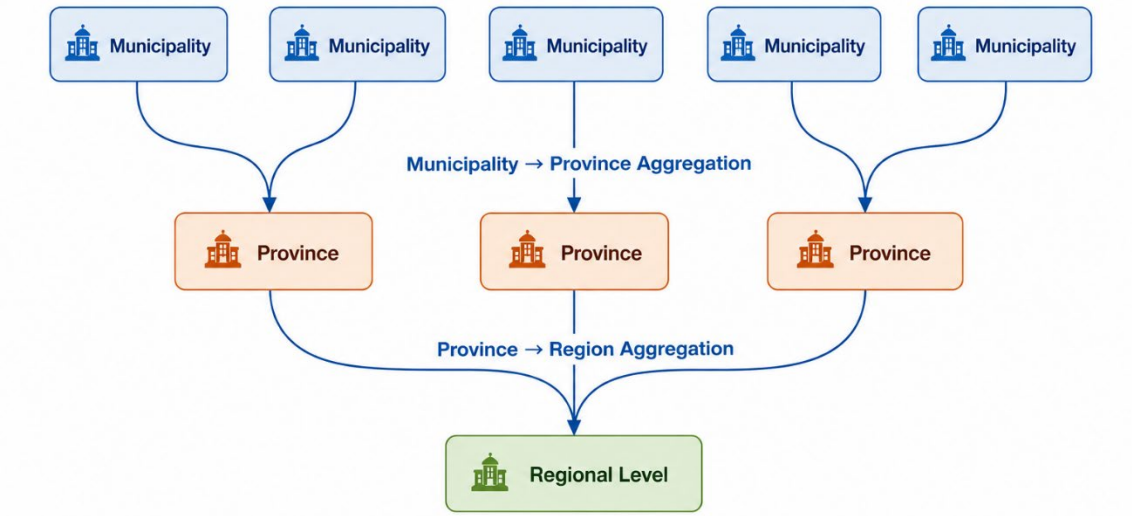

Figure 2—1 Multi-scale governance and aggregation flow illustrating the hierarchical aggregation of municipal data into provincial and regional planning levels.

Temporal Heterogeneity and Analytical Complexity

The variation in temporal resolution introduces additional complexity in analysis. Depending on the specific energy domain and sources, the dataset may be available in different options like hourly, daily, monthly, or yearly intervals. Frequently, analysts can have transitions between these temporal resolutions or combine them with categorical variables, such as seasons and day type (e.g., weekdays and weekends). For example, annual consumption numbers are usually enough for long-term planning, but hourly production data are needed to study how renewable energy changes or to check self-consumption. In city power grid management, operators use hourly electricity use and solar generation data during the summer to keep the grid reliable and reduce peak loads. At the same time, they look at monthly or yearly totals for bigger infrastructure decisions. To make these analyses work well, it is important to store time details as clear fields in the data model, not just in file names or combined columns. For example, adding separate fields like 'date', 'hour', and 'season' in the database makes it easier to group and compare data, so you can analyze it at different time scales (Hyndman & Athanasopoulos, 2021; Kimball & Ross, 2013).

Temporal Resolution	Typical Data Sources	Planning Questions Supported
Hourly	Smart meters, grid operator logs, and weather data	How does demand fluctuate during the day? When is the solar output highest?
Monthly	Utility billing records, regional energy portals	How does seasonal variation affect consumption? What's the monthly yield?
Annual	National statistics, SECAP reports	Are renewable targets being met? What is the total demand by region or sector?

Table 2-1 Examples of temporal resolutions and their planning relevance

Backend Implications for Analytical Systems

Building on the discussion of temporal heterogeneity, these spatial and temporal challenges together highlight the limitations of traditional data storage approaches in territorial energy planning. Treating energy data solely as static datasets for visualization overlooks the analytical effort required to support interactive exploration, comparison, and scenario evaluation (Nielsen & Schiemann, 2018).

In software engineering, territorial energy analysis is mainly an analytical backend challenge rather than a visualization issue. For example, regional grid operators need backend systems that can aggregate and filter data on energy consumption and generation by time, location, and energy source. This helps them make better decisions about load balancing and infrastructure investments. Good systems should allow users to aggregate, filter, and compare data across different dimensions like space, time, energy type, and scenario, while keeping results consistent and traceable.

This shows why it makes sense to use backend architectures built for analytical workloads, though this choice comes with trade-offs. Focusing on multidimensional modeling, clear aggregation rules, and well-structured APIs as main design goals can improve both flexibility and performance. On the other hand, such architectures may require more complex data integration, increased storage demands, and specialized expertise in data warehousing or OLAP systems. Therefore, adopting these analytical backends necessitates carefully weighing the benefits of advanced analysis capabilities against increased system complexity and resource requirements (Kimball & Ross, 2013).

2.2. Multidimensional Data Models for Analytical Systems

Multidimensional data modeling is a well-established approach in decision-support and analytical information systems. Transactional database models work best for frequent inserts and updates and for maintaining consistent records. In contrast, multidimensional models are built to handle complex analytical queries that summarize large amounts of data across several dimensions. These models are ideal when users want to look at data from different angles, like time, geography, category, or scenario, instead of just pulling up single records (Kimball & Ross, 2013; Inmon, 2005).

Analytical vs. Transactional Data Modeling

To see why multidimensional modeling matters in territorial energy planning, it helps to compare analytical and transactional database systems. Transactional systems, also known as OLTP systems, focus on keeping data accurate during frequent updates. Their structures are highly organized to avoid repeating data, which can make queries more complicated. This setup works well for day-to-day operations, like billing systems or collecting sensor data.

Analytical systems, by contrast, prioritize, analytical systems, on the other hand, are designed to make complex queries easier and faster. These systems are mostly used for reading data, not writing it, and often involve grouping, summing, or comparing large datasets. For instance, in energy planning, an analyst might compare electricity consumption and renewable energy production across regions and time periods to identify trends or forecast future needs. Implementing such queries in a strictly normalized schema would require complex joins across multiple tables, resulting in slower performance and less transparent query logic. Therefore, analytical schemas deliberately trade strict normalization for simpler, more intuitive query structures that support efficient and comprehensible aggregation.

Aspect	Transactional Systems	Analytical Systems
Write Frequency	High (frequent inserts/updates)	Low (batch updates or periodic loading)
Query Type	Point lookups, inserts, updates	Aggregations, groupings, trend analysis
Normalization Level	High (3NF or higher)	Low (denormalized or star schema)
Typical Use Case	Banking, e-commerce, and inventory management	Dashboards, business intelligence, energy planning

Table 2-2 Comparison between transactional and analytical database models

The Star Schema as a Reference Model

The most common representation of multidimensional data modeling is the **star schema**. A star schema could consist of a central **fact table** that stores quantitative measures, connected to multiple dimension tables that provide descriptive context for those measures. The name derives from the schema's visual structure, in which dimensions radiate outward from the central fact table.

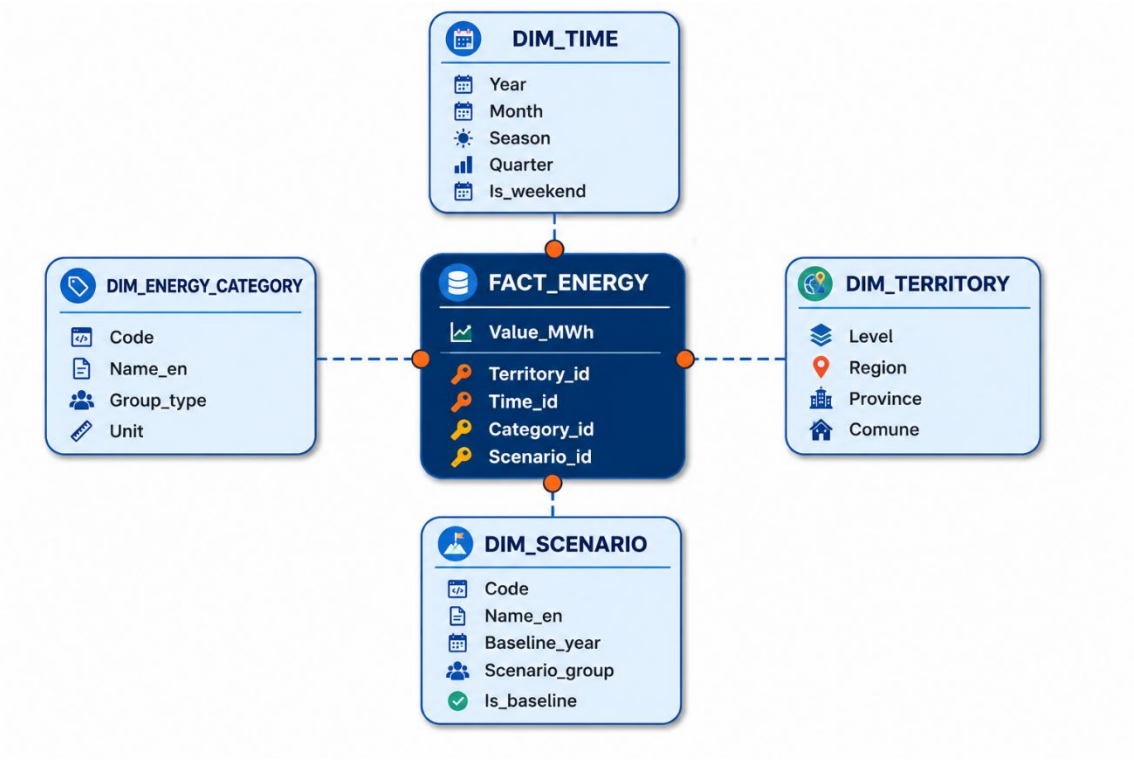

Figure 2—2 Simplified star schema used for multidimensional energy analytics, showing a central fact table connected to temporal, territorial, category, and scenario dimensions.

Dimension tables store descriptive details. For example, a time dimension might include year, month, season, and day-of-week types, while a territory dimension could show how municipalities, provinces, and regions are related.

Separating measures from context lets analytical queries combine and summarize data more flexibly. This also prevents repeating descriptive details in different records.

Dimensions as Explicit Analytical Concepts

One main benefit of multidimensional modeling is that it clearly shows analytical concepts in the database design. In the confrontation of guessing from column names or file layouts, dimensions make it clear how data can be filtered, grouped, and understood.

For example:

- The **time dimension** is the aggregation of valid temporal,
- The **territory dimension** is the spatial hierarchy,
- The **category dimension** shows a semantic difference between energy domains or sources,
- The **scenario dimension** determines whether to use baseline data from hypothetical or user-defined configurations.

By encoding these concepts directly in dimension tables, the system avoids embedding analytical assumptions in application code. This improves transparency and reduces the risk of inconsistent logic across different queries or interfaces. (Kimball & Ross, 2013; Golfarelli & Rizzi, 2009)

Multidimensional Models in Energy Analytics

Several studies show that multidimensional data models work well for energy analytics and planning. Energy systems are naturally complex, with values changing over time, space, energy type, and scenario. Using separate tables for each area often leads to scattered logic and repeated aggregation rules.

In contrast, multidimensional schemas let energy data be stored in a single, unified framework. Spatial and time details are set up once and used again in different analyses. This is especially helpful in territorial planning, where comparing different areas and time periods is common (Keirstead, Jennings, & Sivakumar, 2012; Connolly, Lund, Mathiesen, & Leahy, 2010).

Addressing the Limitations of Domain-Specific Tables

This thesis uses a multidimensional data model to address the problems found in earlier methods that created a separate table for each domain. While having separate tables for consumption, production, and future scenarios may seem straightforward, it actually creates redundant structures and makes analysis more difficult. Each new analysis often requires extra tables or special rules.

A single fact table with domain-aware dimensions offers a clearer solution. Rather than making many tables, the dimension attributes provide the needed meaning. This approach allows you to use the same query patterns across different energy domains and scenarios, reducing code duplication and making the system easier to maintain.

Extensibility and Long-Term Maintainability

Another major benefit of multidimensional modeling is its flexibility. You can add new energy categories, scenario types, or time groupings just by updating the dimension tables, so there is no need to redesign the entire database.

From a software engineering point of view, this is very useful for planning platforms, where needs change and new data is often added. A multidimensional schema gives a stable base that can grow over time.

2.3. Spatial Data in Web-Based Energy Platforms

Spatial information is key in territorial energy analysis because results are usually shown with maps, spatial comparisons, and geographic indicators. For public officials and planners, maps are important tools for spotting patterns, differences, and priorities across areas, not just for showing results. As a result, web-based energy platforms often use maps of administrative boundaries, combined with energy data, to create choropleth maps and other interactive visuals. These tools help users see differences in energy use, production, or scenarios across municipalities, provinces, and regions (Sugumaran & DeGroote, 2011; Malczewski, 1999; Keirstead, Jennings, & Sivakumar, 2012).

Vector Geometries and Administrative Boundaries

In territorial energy projects, spatial data is typically shown with vector geometries. Administrative areas like municipalities, provinces, and regions are modeled as polygons or

multipolygons that match their real boundaries. These shapes form the base of choropleth maps, where numbers are shown with color scales.

On the backend, vector geometries are usually kept in a spatial database or a special data store. This setup lets you search, filter, and connect spatial features with analytical data using shared codes or names (Longley, Goodchild, Maguire, & Rhind, 2015; Obe & Hsu, 2020). Keeping spatial hierarchies consistent is very important. Municipalities need to group correctly into provinces, and provinces into regions. If boundaries or codes are inconsistent, it can lead to wrong results and misleading analysis (Marks, 1993; ESPON, 2018).

GeoJSON as a Web-Oriented Spatial Format

To send spatial data to web frontends, it is often converted to web-friendly formats. GeoJSON is now the main standard for showing vector geometries online. The application works well with almost all mapping tools and is developed in a lightweight, easy-to-read format that integrates well with most web mapping tools.

Its advantages include:

- Supports polygons and multipolygons,
- Compatibility with standard JSON-based APIs,
- direct usability in browser-based visualization frameworks.

For this reason, many web-based energy platforms store spatial data in a backend system and convert it to GeoJSON when a query is made, then send it to the client (Butler, 2016; Haklay, 2008).

Performance Challenges in Spatial Data Delivery

GeoJSON offers many advantages, but it can create performance issues when dealing with complex shapes. For example, administrative boundaries for municipalities often include thousands of points. Sending these detailed shapes for large areas can result in:

- large payload sizes
- increased network transfer times
- slower rendering performance in the browser

These problems are even more pronounced on interactive platforms, where users often change map scales, filters, or scenarios. If not managed well, sending spatial data can slow down the whole system and make it harder to use.

Geometry Simplification for Interactive Visualization

To solve these issues, geometry simplification is usually done before sending spatial data to the frontend. These algorithms reduce the number of points in a shape but keep its overall look and correct connections within set limits.

In web-based energy platforms, the aim of simplification is to make maps look good enough, not to be perfectly precise. For most analysis, simplified boundaries show spatial patterns well, especially when looking at regional or provincial maps (McMaster & Shea, 1992).

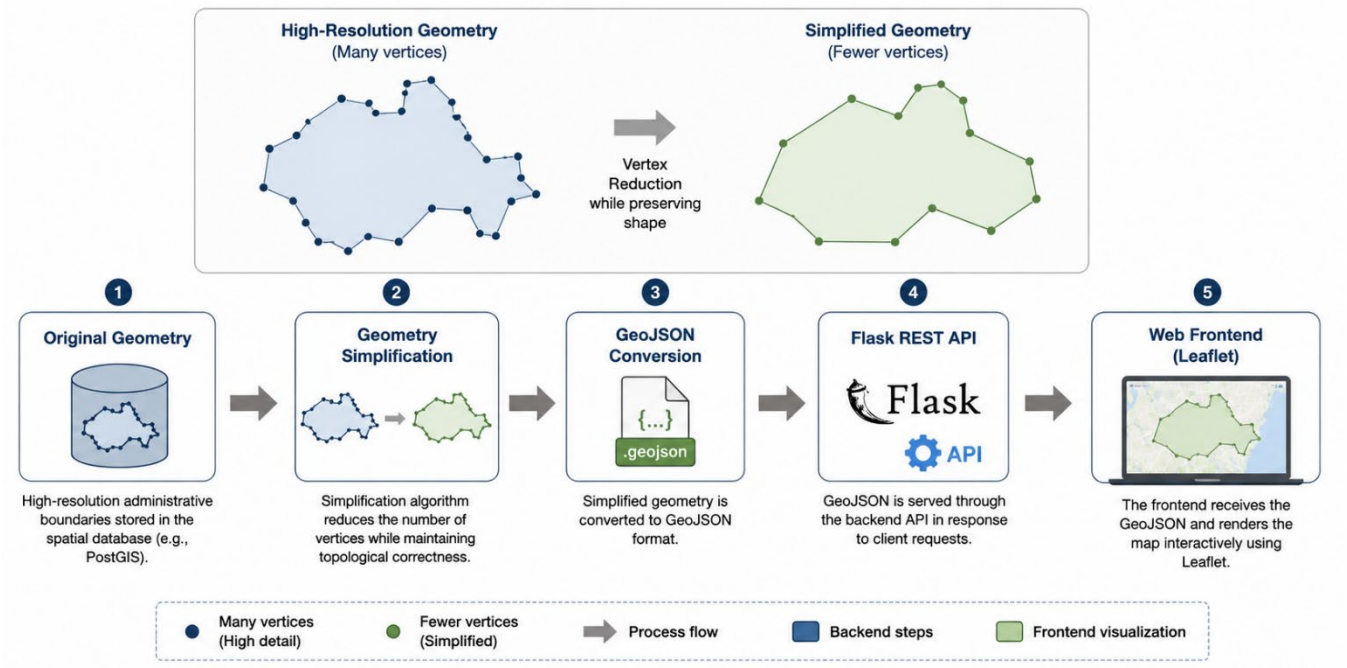

Figure 2—3 ***Geometry simplification and GeoJSON delivery pipeline.***
High-resolution administrative boundaries are simplified, converted to GeoJSON, and delivered through the backend API for interactive web visualization.

Linking Spatial and Analytical Data

Beyond geometric handling, a fundamental challenge in spatial energy platforms is integrating spatial and analytical data. Energy values stored in analytical databases must be reliably linked to their corresponding spatial units.

This linkage typically relies on:

- Standardized administrative codes.
- Consistent naming conventions
- When datasets come from different sources, explicit mapping tables are used.

If spatial and analytical identifiers do not match, joins can fail and visualizations may be missing or wrong. Because of this, integrating spatial data is both a data governance and a technical challenge.

Identifier Type	Source Dataset	Role in Spatial Join
ISTAT code (municipality)	National statistical datasets (ISTAT)	Primary key to link energy data with municipality-level geometries
Province code	Regional planning reports	Used as a grouping field; needs normalization for consistency
Region code	European administrative datasets (NUTS2)	Aggregation level for reporting and map display
Composite key (name + year)	Local scenario files (CSV)	Used for matching custom inputs with standardized territorial units
Geometry ID	Spatial database (PostGIS)	Internal key for retrieving and simplifying GeoJSON representations

Table 2-3 Examples of Spatial Identifiers and Their Role in Data Integration

Implications for Backend Design

The points above directly affect backend architecture. Spatial data should be:

- decoupled from analytical queries whenever possible,
- delivered only when required for visualization,
- simplified appropriately based on the analytical scale.

From an architectural perspective, this reinforces the need to treat spatial data handling as a dedicated concern rather than implicitly embedding it in analytical queries.

"These considerations inform the spatial data handling strategies adopted in this thesis, which are described in detail in Chapter 8."

2.4. Backend Architectures for Interactive Dashboards

Interactive energy dashboards require backend architectures that are different from traditional reporting systems. While static reports run set queries and show results in fixed layouts, interactive dashboards must react in real time to user actions. These actions include filtering, drilling down across different areas and times, and quickly updating maps and charts.

Because of this, the backend needs to handle both data retrieval and real-time analysis of large datasets. Every user action might start a new aggregation, change the grouping, or shift the analytical focus. Backends that are not built for this often have trouble with performance, accuracy, and maintenance (Few, 2006; Thomas & Cook, 2005).

Separation of Concerns Between Backend and Frontend

A common design principle in web-based analytical platforms is to separate data processing from data visualization. In this setup, the backend manages data access, aggregation, validation, and transformation. The frontend is responsible for showing results and handling user interactions.

RESTful APIs are often used to create this separation. By providing analytical results through HTTP endpoints, the backend can support different frontend components like maps, charts, and tables with a single interface. This method also lets the frontend change on its own, as long as the API stays the same (Richardson & Ruby, 2007; Frontend, 2012).

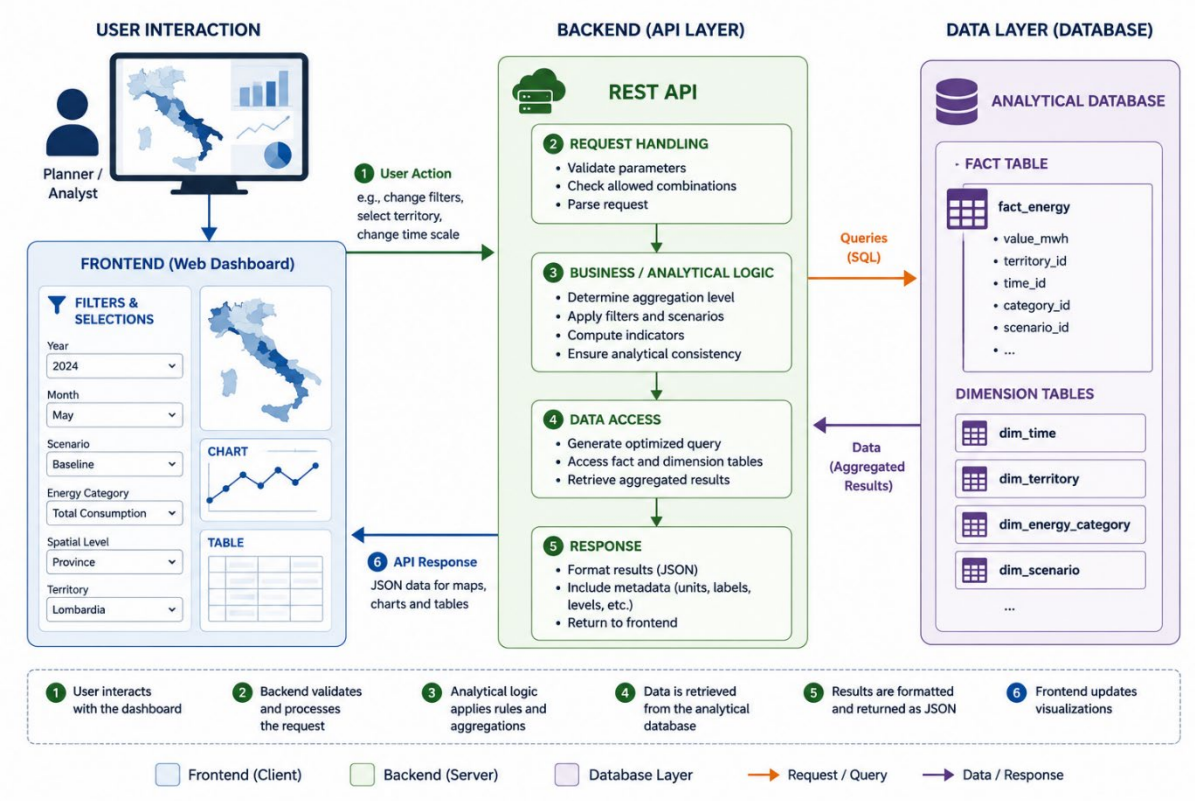

Figure 2—4 High-level interaction between the frontend, backend API, and analytical database in a web-based energy dashboard.

Analytical APIs Versus Data-Centric APIs

Research shows that APIs for analytical dashboards work best when built around analytical concepts, not just database structures. If you expose raw table rows or generic queries, the frontend becomes too dependent on the database design, making the system fragile as data models change.

Instead, analytical APIs usually provide domain-specific resources, such as:

- values aggregated for choropleth mapping,
- time series for chart visualization,
- scenario parameters and derived indicators.

With this design, the backend handles aggregation, validation, and analytical rules. The frontend then receives analytical results that are ready to use, instead of raw data that needs more processing. (Richardson & Ruby, 2007; Amundsen, 2013)

Balancing Flexibility and Control

A key challenge in backend design for interactive dashboards is finding the right balance between flexibility and control. Planners and analysts need to explore data using filters like year, month, energy category, spatial level, and scenario. However, not every combination of these filters makes sense or is valid.

If there are no clear control mechanisms, backend systems might run unclear queries, give misleading results, or use too many resources. For example, asking for hourly data at a regional level could make sense in some cases but not in others.

To address this issue, backend architectures often implement:

- explicit validation of allowed parameter values
- predefined combinations of spatial and temporal resolutions
- explicit constraints on analytical domains

By enforcing these rules at the API level, the backend acts as a guardian of analytical consistency rather than a passive data provider (Thomas & Cook, 2005).

User Action	Backend Task	Potential Risks if Unmanaged
Selecting a year and municipality	Validate parameters, query the fact table, and return values for the map or chart	Inconsistent or missing values if the year or spatial ID is invalid
Switching from monthly to annual	Recompute aggregations using the time dimension hierarchy	Incorrect totals or misleading comparisons if aggregation is not aligned
Filtering by energy domain	Filter based on energy category dimension	Inclusion of unintended data (e.g., mixing future production with observed data)
Previewing a scenario	Apply uplift to selected categories, recompute derived indicators	Overwriting baseline data or incorrect scenario assumptions
Requesting GeoJSON for the map	Retrieve and simplify geometry, attach values for visualization	Slow performance or visual clutter if simplification and caching are not applied
Navigating from region to province	Adjust aggregation level, recompute results using spatial hierarchy	Duplicate values or broken comparisons if spatial joins are ambiguous

Table 2-4 Examples of User Interactions and Backend Responsibilities, tailored to your platform's logic and interaction model

Implications for Energy Planning Platforms

These architectural choices are especially important in territorial energy planning. Energy indicators rely on clear definitions, consistent aggregation, and transparent assumptions. If analytical logic is spread across the frontend or duplicated in different services, results can quickly become inconsistent.

A backend built around analytical APIs helps energy planning platforms to:

- centralize computation logic.
- ensure reproducibility of results.
- support scenario-based reasoning.
- maintain consistency across maps and charts.

"These architectural principles form the basis for the backend design adopted in this thesis, which is described in detail in Chapters 5 and 6."

2.5. Scenario-Based Energy Analysis

Scenario analysis is a key tool in energy planning. It lets public authorities and stakeholders explore how different assumptions, policies, or actions might affect outcomes. Scenarios often

represent policy goals, investments, or technology changes, and are usually compared to a baseline setup.

For planners, scenarios provide a practical way to explore the possible consequences of different assumptions before they are implemented. Questions such as how an increase in photovoltaic capacity would affect local self-consumption or how a greater share of renewable energy might influence the regional energy balance are key to this type of analysis. Dealing with these questions requires more than a simple comparison of energy production and consumption values. Instead, meaningful evaluation depends on a set of indicators that represent the relationship between energy demand and local generation, delivering a clearer picture of system performance under different planning assumptions.

Baseline and Scenario Conceptualization

Analytically, a **baseline** is the reference setup for a territory under current or official conditions (called scenario 0). A **scenario** is a changed setup where one or more factors like energy production, consumption, technology, or policy are adjusted based on planning ideas.

It is important to note that scenarios are not separate datasets. They are changes made to baseline values using set rules. This matters for the backend: instead of copying whole datasets, scenarios can be handled as layers that calculate new values and indicators from existing data (Mutani, Morando, & Zhou, 2024; Connolly, Lund, Mathiesen, & Leahy, 2010).

Derived Indicators in Energy Planning

Evaluating baseline conditions alongside alternative scenarios requires more than comparing energy values directly. In practice, planners rely on a set of derived indicators that reconcisely and meaningfully present the relationship between energy consumption and production. These metrics supply a clearer basis to aid in interpreting system performance and assessing the effects of different planning assumptions. Among the indicators most widely adopted in renewable energy communities and territorial energy planning are self-consumption, self-sufficiency, and surplus (or over-production).

All these indicators are derived from two fundamental quantities:

- **Energy consumption**

  representing final energy demand within a given territory and time period.

- **Energy production**

representing locally generated energy, typically from renewable sources.

Let the following notation be defined for a given territory *t* and time interval *τ*:

- C (t, τ) = total energy consumption
- P (t, τ) = total local energy production

Generic Energy Balance and Indicator Definitions

The relationship between consumption and production can be expressed through a generic energy balance, from which multiple indicators are derived.

FORMULA 2.1 — Generic relationship between consumption, production, and derived indicators

These expressions capture three complementary aspects of territorial energy performance:

- **Self-consumption**

  The amount of locally produced energy that can be directly consumed within the territory:

$$\boldsymbol{Self - Consumption}(\boldsymbol{t}, \boldsymbol{\tau}) \;=\; \boldsymbol{min}(\boldsymbol{C}(\boldsymbol{t}, \boldsymbol{\tau}), \boldsymbol{P}(\boldsymbol{t}, \boldsymbol{\tau}))$$

- **Self-sufficiency**

  The share of local energy demand that can be covered by local production:

$$\boldsymbol{Self - Sufficiency}(\boldsymbol{t}, \boldsymbol{\tau}) = \frac{\mathbf{min}\big(\boldsymbol{C}(\boldsymbol{t}, \boldsymbol{\tau}), \boldsymbol{P}(\boldsymbol{t}, \boldsymbol{\tau})\big)}{\boldsymbol{C}(\boldsymbol{t}, \boldsymbol{\tau})}$$

- **Surplus production**

  The amount of locally produced energy that exceeds local demand:

$$\boldsymbol{Surplus\ Production}(\boldsymbol{t}, \boldsymbol{\tau}) = \boldsymbol{max}(\mathbf{0}, \boldsymbol{P}(\boldsymbol{t}, \boldsymbol{\tau}) - \boldsymbol{C}(\boldsymbol{t}, \boldsymbol{\tau}))$$

These expressions describe three complementary aspects of territorial energy performance. Self-consumption captures the absolute quantity of energy used locally, self-sufficiency expresses the relative coverage of local demand, and surplus production represents excess energy that must be exported, stored, or curtailed. The use of minimum and maximum operators ensures that all indicators remain physically meaningful and bounded under all production–consumption configurations. (Mutani, Morando, & Zhou, 2024; European Commission, 2018)

Backend Implications for Scenario Computation

For backend design, these indicator definitions are important. First, consumption and production values must be added up in the same way, using the same area and time period. If not, the indicators will not be valid.

Second, since scenarios often change production values but not consumption, the calculation of indicators should be done in a clear, reusable way in the backend, not hidden in the visualization code.

By making these formulas clear in the backend, you improve transparency, reproducibility, and flexibility. It also lets you recalculate indicators on the fly during scenario previews and save them when scenarios are finished.

“The operationalization of these generic indicator definitions within the backend architecture is described in detail in Chapter 7.”

2.6. Summary

This chapter reviews the conceptual and technical background necessary to frame the design choices adopted in this thesis. Starting in the domain of territorial energy planning, it highlighted the analytical challenges of working across multiple spatial scales, heterogeneous data sources, and diverse temporal resolutions. Municipalities, provinces, and regions represent distinct governance levels with different planning responsibilities, yet energy data is rarely collected or structured to directly support coherent multi-scale analysis. As mentioned in Section 2.1, splitting data in this way can result in ad hoc aggregation, make results harder to reproduce, and increase the risk of inconsistency.

To address these issues, Section 2.2 introduced multidimensional data models, which are a solid foundation for analytical systems. By keeping quantitative measures separate from descriptive dimensions, star-schema designs make it easier to aggregate data across space, time, and different analytical areas. This modeling approach is particularly suited to energy planning applications, where the same data must support multiple perspectives, resolutions, and scenario configurations. The discussion in this section provides the theoretical justification for adopting a unified fact table combined with explicit dimension tables in later chapters.

Section 2.3 looked at how spatial data is used in web-based energy platforms. It stressed that spatial shapes are not just for visuals—they are a key part of analysis. Handling boundaries well, using consistent spatial IDs, and simplifying shapes are all important for making interactive maps

accurate and easy to use. These points shape the spatial data strategies discussed in later chapters.

Section 2.4 focused on backend design for interactive dashboards. It showed that the backend should act as an analytical engine, not just a data source. Important ideas include separating concerns, designing APIs around the domain, and clearly checking analytical parameters. These patterns make it possible to use features such as filtering, drill-down, and scenario comparison, while also ensuring the analysis stays reliable.

Section 2.5 introduced scenario-based energy analysis as an important part of decision-support systems. By explaining how consumption, production, and indicators like self-consumption and self-sufficiency are connected, this section provided a strong foundation for evaluating scenarios. Clear definitions of indicators and transparent calculations are important for repeatability and trust, especially when the results are used to guide public decisions.

Overall, this chapter shows why backend architecture needs to combine multidimensional data models, organized spatial data, analytical API design, and clear ways to handle scenarios. These needs set the stage for Chapter 3, which covers the system requirements and main architecture of the proposed platform.

3. System Requirements and Architecture Overview

3.1. Objectives of the Platform

The main goal of this platform is to help with territorial energy analysis using an interactive, data-driven web application. It works as a decision-support tool, allowing public authorities, planners, and analysts to explore energy use, production, and future scenarios throughout various areas and time periods in a clear and consistent way.

Instead of focusing on just one type of analysis, the platform is built to support many different tasks. Users can compare energy performance between municipalities or regions, look at trends over time, and test the effects of different scenarios. This flexibility is important because energy planning needs often change as new policies, data, or priorities come up.

A central objective of the platform is to manage energy-related data across multiple spatial scales—municipality, province, and region—and multiple temporal rA key goal of the platform is to handle energy data at different levels, such as municipality, province, and region, and at different time scales like hourly, monthly, and yearly. The backend lets users group, filter, and compare data across different levels without extra steps or repeating work. This helps solve the challenges of managing data from different areas and time periods, as discussed in Chapter 2. The backend handles tasks like data aggregation, checking query parameters, and calculating indicators. The frontend, on the other hand, focuses on showing results through interactive maps, charts, and controls. Keeping these parts separate makes the system easier to maintain and lets the frontend and backend change independently as long as their connection stays the same. The platform is also designed so results can be reproduced and traced. If you use the same input, data sources, and assumptions, the backend will return the same results. This matters in planning, where results might guide policy or be shared with the public. To help with this, the backend uses clear aggregation rules and states scenario assumptions openly instead of hiding them in the visualization code.

The platform is also designed to be easy to expand. You can add new datasets, energy categories, indicators, or scenario types without needing to make big changes to the backend. This goal has shaped several design choices, such as using a multidimensional data model and domain-focused API endpoints, which are explained in later chapters.

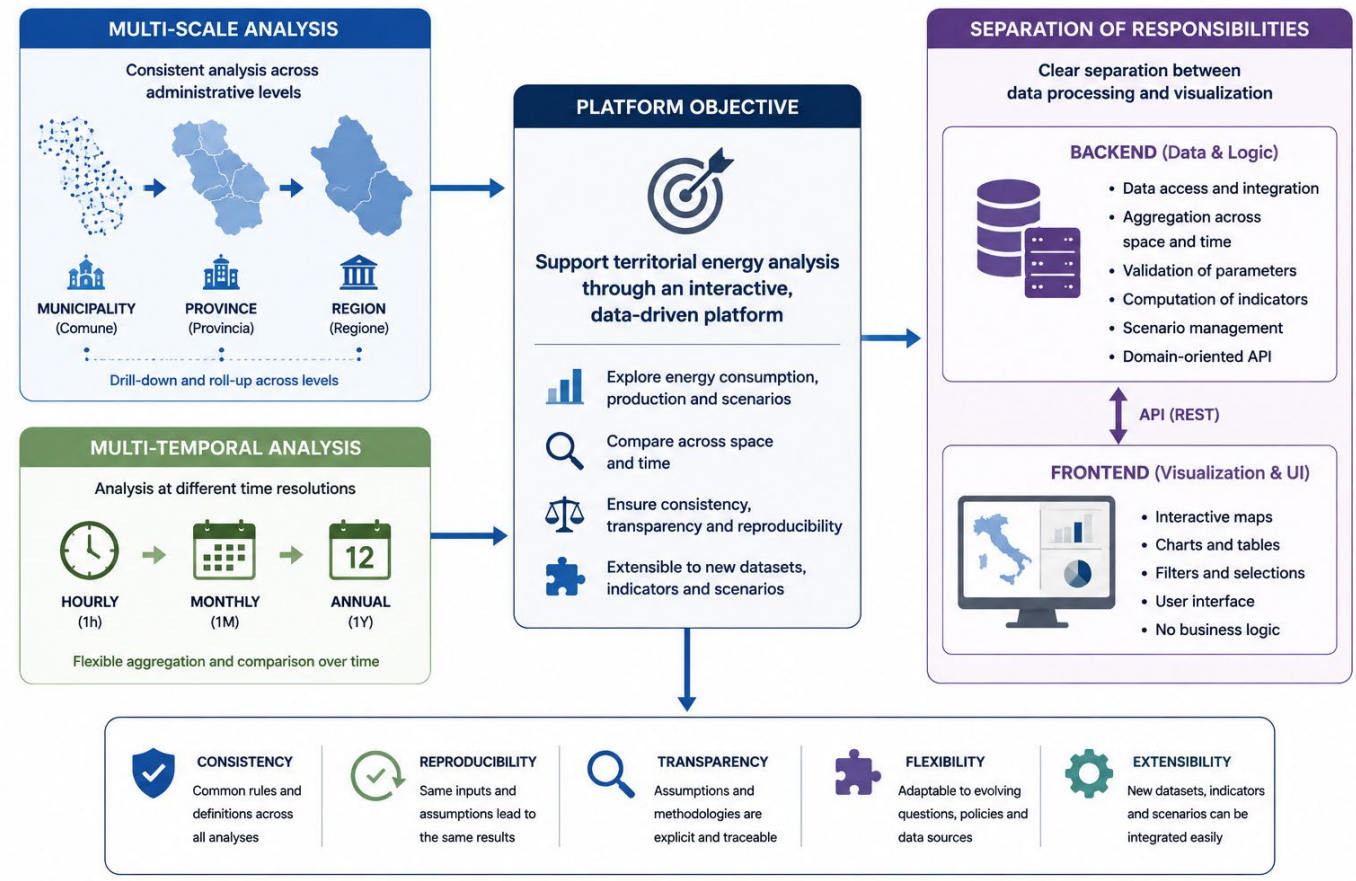

Figure 3—1 Conceptual objectives of the platform, illustrating multi-scale and multi-temporal energy analysis together with the separation of responsibilities between frontend visualization and backend analytical processing.

3.2. Functional Requirements

The functional requirements were defined by looking at the platform's goals and the analytical challenges from Chapter 2. These requirements describe what the backend needs to do to support the frontend and ensure a smooth territorial energy analysis workflow.

These functional requirements are set without tying them to any specific technical solutions. They focus on how the system should behave for users and use cases, not on internal software details. This approach helps keep the requirements stable, even as technology changes

Multi-Scale Spatial Querying

One key requirement is that the backend can query energy data at different territorial levels. It must support analysis at municipal, provincial, and regional levels to match the various governance and planning needs in energy policy.

For each territorial level, the backend must provide:

- Aggregated spatial values suitable for map-based visualization, such as choropleth maps.
- Aggregated time-series data suitable for chart-based visualization

Aggregation rules need to be consistent at all levels. This ensures that results at higher administrative scales accurately reflect the combined data from lower levels.

Multi-Temporal Analysis Support

Besides spatial scale, the backend also needs to support analysis at different time resolutions. These include:

- hourly
- monthly
- seasonal
- annual views

Users should be able to switch between different time resolutions without affecting the analysis logic. The backend handles these changes by creating the right aggregation queries and making sure results can be compared across resolutions.

This requirement tackles the differences in time data found in energy datasets, as discussed in Chapter 2, and supports the need to model time attributes clearly in the system.

Dynamic Filtering and Parameter Validation

Another important requirement is letting users filter analysis results in real time based on their chosen parameters. The backend must support filtering by:

- year
- month
- day type (weekday or weekend)
- energy domain (consumption, production, future production)
- energy category
- territory

However, flexibility needs to be balanced with accuracy. The backend must check incoming parameters and make sure only allowed combinations are used. For example, sometime resolutions may not make sense for certain spatial levels or energy domains.

By checking parameters in the backend, the system avoids unclear or invalid queries and keeps analysis results clear and consistent.

Scenario Preview and Persistence

Managing scenarios is another key requirement for the platform. The backend must let users:

- preview hypothetical scenarios by modifying production assumptions!
- compute updated indicators dynamically based on these modifications.
- observe immediate changes in analytical outputs!

In addition to previewing scenarBesides letting users preview scenarios, the backend must also save them. Scenarios should be stored in the database with enough details so users can find, compare, and reuse them in future analyses.nuity in planning workflows and supports comparative analysis across multiple scenarios over time.

Requirement ID	Requirement Category	Description	Related Chapters
FR-1	Spatial querying	Support aggregation and visualization at the municipality, province, and region levels	Ch. 4, Ch. 6
FR-2	Temporal Analysis	Support hourly, monthly, seasonal, and annual resolutions	Ch. 4
FR-3	Dynamic filtering	Enable filtering by time, domain, and category with validation	Ch. 5, Ch. 6
FR-4	Scenario preview	Allow dynamic modification of production assumptions	Ch. 7
FR-5	Scenario persistence	Store and retrieve scenarios for later comparison	Ch. 7

Table 3-1Summary of Functional Requirements for the Backend System

3.3. Non-Functional Requirements

Along with functional features, the backend design also followed non-functional requirements. These focus on how the system should work, not just what it does, and are key to making sure the platform stays usable, reliable, and sustainable.

Performance

Performance was a primary concern, aPerformance was a top priority because interactive energy dashboards need to respond quickly to stay user-friendly and support exploration. Users expect maps and charts to update right away when they change filters or settings. This need shaped both the database design and backend query strategy. Common aggregation patterns and analysis resolutions were planned ahead of time to make queries simpler and faster.l-time performance, the system was designed to provide consistent and predictable response times under typical analytical workloads.

Scalability

Scalability was seen as essential so the platform can grow beyond its original purpose. While it currently uses Italian territorial data, the design allows for more datasets, longer time periods, and new energy domains without needing to change the database structure.

This need led to choosing a multidimensional data model instead of domain-specific or fixed tables. By separating spatial, temporal, and category data, the system can handle more content and complexity while keeping its structure stable.

Maintainability and Clarity

Maintainability and clear code were also important, since energy datasets and scenario analysis can be complex. The backend logic had to be easy to understand, modular, and simple to extend for future updates and maintenance.

To meet this need, the backend was organized with clear separation of tasks. Different modules handle database access, filtering and aggregation, validation, and API endpoints. This setup keeps components independent, so parts of the system can be changed or updated without affecting everything else.

Interoperability

Finally, interoperability with web-based mapping and visualization tools was a key nonFinally, working well with web-based mapping and visualization tools was a key non-functional requirement. The backend must communicate smoothly with frontend parts and external libraries used for spatial visualization.ted formats, including JSON for analytical data and GeoJSON for spatial geometry. This choice enables seamless integration with web mapping libraries and ensures that the backend remains frontend agnostic, allowing visualization technologies to evolve independently.

3.4. High-Level System Architecture

The backend uses a layered architecture, with each layer handling its own tasks. This design makes the system clear, modular, and easy to update, while meeting the analytical needs described earlier.

At the base, a PostgreSQL database stores both energy data and spatial information. This layer only handles keeping data safe and accurate. Analytical data is set up for aggregation by space and time, while spatial data shows administrative boundaries for maps.

On top of the database, backend utility modules handle connecting to the database, running queries, and transforming data. These modules manage low-level data access and give a consistent interface to the application layer. Keeping database communications in this layer helps separate data storage from application logic.

The application layer is a web service that offers several RESTful API endpoints. Each endpoint handles a specific analytical task, like getting map values, creating time-series data for charts, or managing scenarios. All user actions from the frontend go through these endpoints, and the frontend never accesses the database directly.

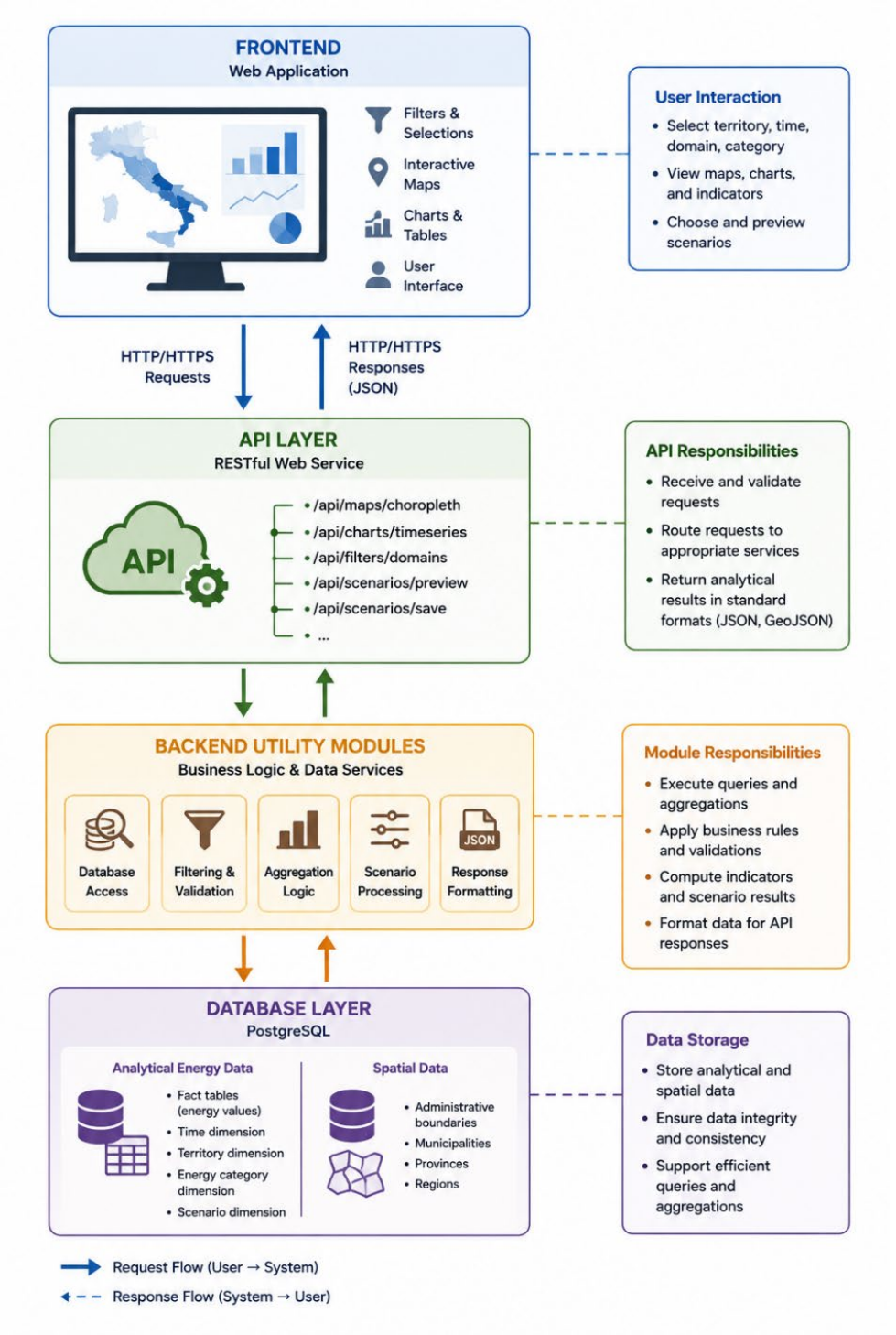

Figure 3—2 This diagram shows the main backend architecture and how data moves between the frontend, API layer, and other components.

This layered architecture enables the backend to evolve independently of the frontend, as long as API contracts remain stable. It also facilitates testing and debugging by allowing individual components to be analyzed in isolation. Furthermore, the clear separation between layers makes

it easier to identify performance bottlenecks and to introduce optimizations without affecting the overall system structure.

3.5. Data Flow and Request Handling

This section explains how data moves through the system when users interact with the platform. It aims to clarify the steps from frontend actions to backend processing and data retrieval, without focusing on technical implementation. For example, when a user selects a territory, changes time filters, or picks an energy domain, the frontend sends a request to the right API endpoint on the backend. Each request includes parameters that reflect what the user wants to analyze, such as location, time range, and filters.

When the backend receives a request, it first checks that all required parameters are included and that their values are within allowed ranges. This step helps prevent unclear or invalid queries and keeps results reliable. If any input is invalid, the backend sends back clear error messages so the frontend can inform the user. After validation, the backend turns the user's request into a database query. This query joins the main data with related information from other tables, and handles aggregation and filtering directly in the database to keep things efficient and reduce extra data transfer.

The backend then formats the query results so the frontend can use them easily. For non-spatial data, it sends responses in JSON format, which lets charts and summaries be shown right away. For spatial data, it retrieves the shapes of the selected territories for map displays. To make web maps load faster, these shapes might be simplified before sending, which keeps the main structure but reduces file size and improves performance without changing the analysis.

This organized request and response process keeps frontend and backend roles separate, and allows for flexible, efficient, and reliable analysis across different locations and time periods.

3.6. Design Rationale

The design choices in this chapter were shaped by technical needs and the specific goals of territorial energy planning. The main aim was to build a backend that supports flexible, multi-level analysis and stays reliable and easy to update. Using a multidimensional data model helps combine and compare energy data across space, time, energy types, and scenarios. Traditional table designs often hide aggregation rules, making flexible analysis harder. A dimensional model, on the other hand, makes these rules clear and reusable in different situations.

In the same way, choosing to present analytical concepts directly through API endpoints, instead of showing raw database structures, was meant to match how users and frontend developers think about the system. By organizing endpoints around tasks like map views, time-series analysis, and scenario checks, the backend keeps system layers separate and makes future changes easier.

The layered architecture and clear data flow were chosen to balance flexibility with control. Backend validation keeps analysis consistent, and modular design makes it easier to add new indicators, datasets, or spatial levels in the future. Overall, this setup is a practical balance between powerful analysis and system simplicity, designed for territorial energy analysis.

3.7. Summary

This chapter explained the goals, requirements, and main structure of the backend system built for this thesis. It started by describing what the platform aims to achieve and what it needs to support interactive, multi-level energy analysis. The chapter then introduced a layered system, showing how data storage, backend logic, and frontend interaction are kept separate. It also covered how data moves through the system when users interact with it, and how validation, aggregation, and structured API responses help keep the analysis accurate and easy to use.

All these parts together form the main ideas and structure of the platform. They set the stage for the next chapter, which will go into detail about the database design and the choices made during implementation. The next chapter will focus on how the multidimensional database schema supports consistent aggregation and analysis across different locations, times, and scenarios.

4. Database Design and Data Modeling

4.1. Motivation for the Database Design

Designing the database was a key part of building the platform because it affects how well the system can handle flexible, consistent, and scalable energy analysis. At first glance, storing energy data might seem simple—just save numbers in tables by territory and year, and pull them out when needed. But after defining the platform's analysis goals, it was clear that this basic approach would not be enough.

The platform is expected to support analysis across multipleThe platform needs to support analysis at different territorial levels, like municipality, province, and region, as well as various time frames such as hourly, monthly, seasonal, and yearly. It also covers several areas, including consumption, production, future production, and scenario results. Each of these adds complexity to how data is grouped, filtered, and compared. A basic relational database might work for simple tasks, but it would soon become hard to manage with these requirements.se separate tables for each analytical need? This approach was initially considered, as it is common in small-scale or domain-specific applications. For example, one table could store consumption data, another could store production data, and a third could store future scenarios. However, such a design leads to several structural problems. Aggregation logic must be duplicated across tables, spatial and temporal hierarchies are reimplemented multiple times, and the same indicator may be computed differently depending on the context. Over time, this resulted in inconsistent output and increased maintenance effort.

Making changes to the system also gets harder over time. Adding a new energy type, scenario, or time frame would mean changing several tables and queries. This close link between data structure and analysis rules goes against the platform’s goal of being easy to extend in the future.

Because of these challenges, the database design focuses on flexibility, consistent analysis, and long-term maintenance instead of just making things simple at first. Instead of building analysis rules into the tables themselves, the database should offer a stable base for creating analytical views in a controlled and reusable way.

4.2. Choice of a Multidimensional Data Model

To address the challenges identified in the previous section, the database was designed following a multidimensional data modeling approach. This modeling paradigm is widely adopted in analytical and decision-support systems, where data exploration, aggregation, and comparison

are primary requirements. In a multidimensional model, numerical measurements are stored in a central fact table, while contextual information is represented through a set of dimension tables. Instead of spreading energy values across multiple domain-specific tables, a single fact table stores energy-related measurements, and each record references dimensions such as territory, time, energy category, and scenario through foreign keys.

This separation between measures and context is fundamental. By explicitly encoding spatial hierarchies, temporal attributes, and analytical classifications in dimension tables, aggregation logic can be expressed once and applied consistently across all analyses. For example, aggregating municipal values at the provincial or regional level involves traversing predefined hierarchies rather than writing custom queries for each dataset. From an analytical perspective, this structure mirrors how users interact with the platform. When a user selects a year, a territorial level, and an energy category, they are effectively navigating a multidimensional data space. The database design reflects this mental model, simplifying both backend query formulation and API design. Analytical operations such as filtering, grouping, and comparison become natural operations over dimensions rather than ad hoc query logic.

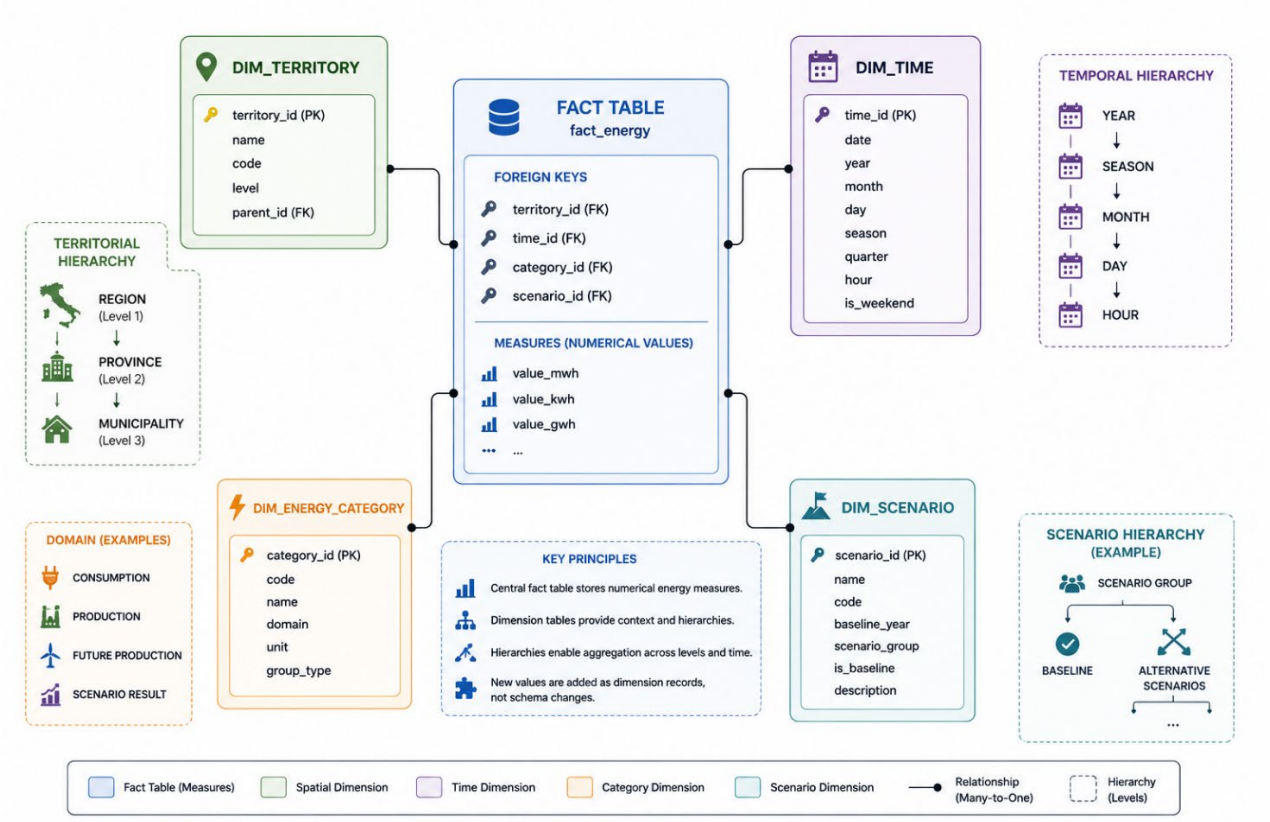

Figure 4—1 Conceptual multidimensional schema used for energy analytics, showing a central fact table linked to temporal, territorial, category, and scenario dimensions, together with their analytical hierarchies.

The multidimensional approach makes analysis clearer and also makes it easier to expand the system. You can add new energy categories, scenarios, or time periods just by adding new dimension records, so there is no need to change the entire database structure. This helps keep the database stable as your analytical needs grow.

A multidimensional data model provides a solid and flexible foundation for analyzing energy data at different levels. This method fits both the theory behind the analysis and the practical needs of the system.

4.3. Fact Table Design

The main part of the multidimensional schema is the fact table. It stores energy measurements in megawatt-hours (MWh). Each row in the fact table shows an energy value linked to a specific location, time, energy category, and scenario.

The fact table is designed to be generic, so it does not use column names to show analytical meaning. It does not separate energy consumption, production, or future projections in its structure. IInstead, these differences are shown by linking to the right dimension tables, especially the energy category and scenario tables.

This design raises a key question: does this level of abstraction make queries harder? Some queries might need more joins to dimension tables, but overall, the system is easier to understand and maintain. You can use the same query structure in different situations, which helps avoid repeating code and reduces mistakes when calculating the same indicators in various cases.

The fact table brings together all analytical dimensions. It stores only numbers and foreign keys, not descriptive or hierarchical details. This clear separation keeps the rules for combining data in one place and makes them consistent for all types of analysis.

Along with the main measurement, the fact table also includes extra details such as time resolution and data source. These fields help the system distinguish between raw and summarized data, and between data from different sources. This is important for supporting different levels of analysis and for keeping track of where the data comes from.

Column Name	Description	Type
id	Primary key (unique identifier for each record)	Integer (PK)
time_id	Foreign key to the time dimension	Integer (FK)
territory_id	Foreign key to the territory dimension	Integer (FK)
category_id	Foreign key to the energy category dimension	Integer (FK)
scenario_id	Foreign key to the scenario dimension	Integer (FK)
value_mwh	Energy value in megawatt-hours	Numeric
time_resolution	Resolution of the data (e.g., hourly, monthly, annual)	Text
data_source	Identifier of the data source or aggregation pipeline	Text

Table 4-1 Structure of the fact_energy Table

4.4. Time Dimension and Temporal Resolution

Time is one of the hardest parts of energy analysis. Data is often collected every hour, but planning usually needs monthly, seasonal, or yearly summaries. Handling these different needs in a consistent way is a major design challenge. To solve this, a separate time dimension was added. Instead of putting timestamps directly in the fact table, each record points to a time ID that includes details like year, month, hour, season, and whether it is a weekday or weekend, depending on the data's resolution.

This design lets you group data in different ways without changing the query logic each time. For example, the same fact table can be used for both hourly and yearly summaries by grouping by different time attributes. This makes time-based grouping a clear and controlled process, not something done in a one-off way in queries.

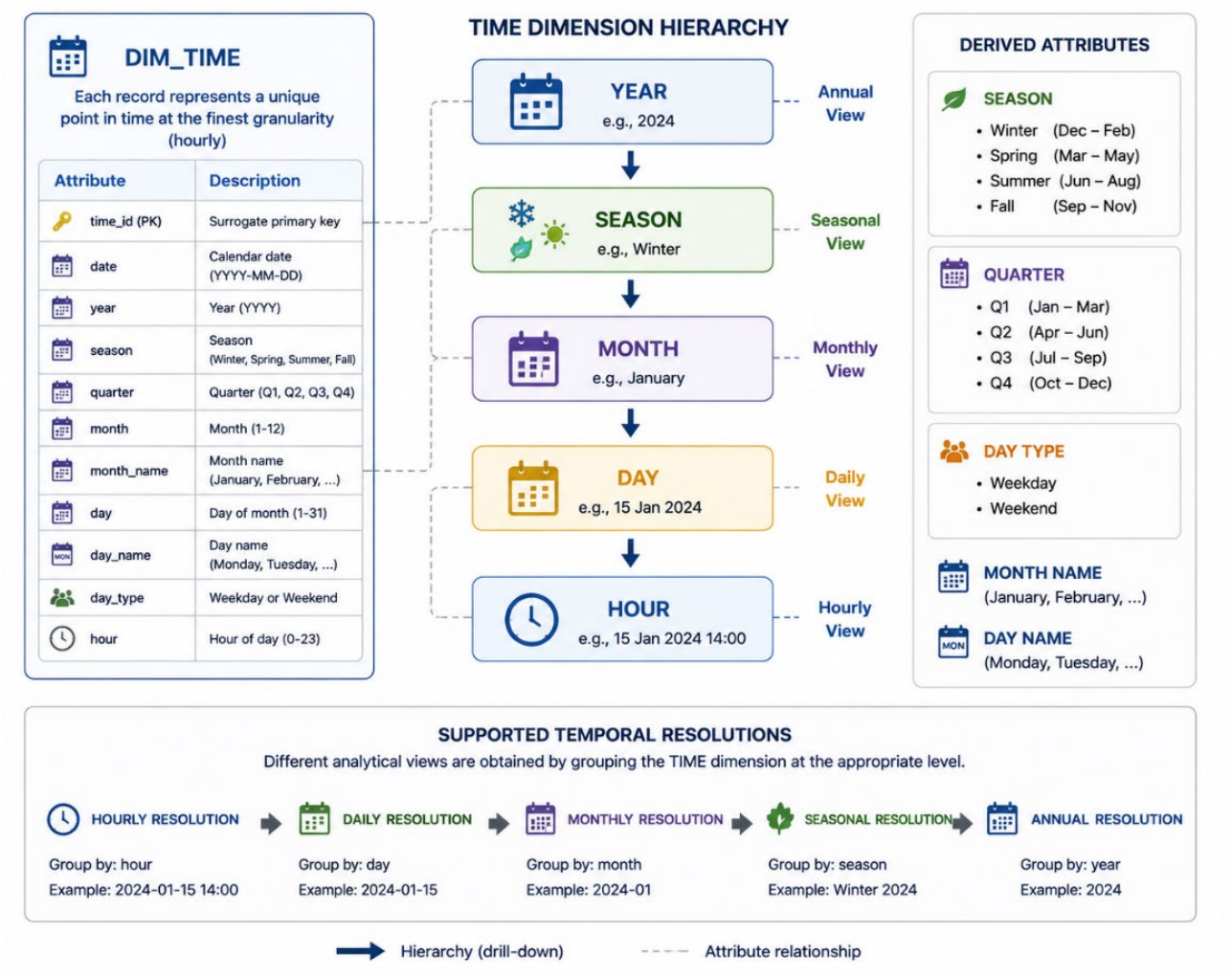

Figure 4—2 Time dimension hierarchy showing how hourly, daily, monthly, seasonal, and annual analyses are supported through a common temporal model.

Another benefit is that concepts like seasons or weekday/weekend are defined once in the time dimension and used everywhere in the system. This prevents small mistakes that could happen if this logic was repeated in different queries or parts of the application.

By moving time-related details into their own dimension, the database stays consistent, reusable, and accurate for all time-based analyses on the platform.

4.5. Territory Dimension and Spatial Hierarchies

Spatial analysis is key in territorial energy planning. The database needs to compare not just municipalities, but also provinces and regions. These areas are connected in a clear administrative hierarchy, which affects planning and decisions.

To show this structure clearly, a single territory dimension is used. Each record stands for a spatial unit and includes IDs for municipality, province, and region, plus standard administrative codes. The level (municipality, province, or region) is stored as an attribute, not guessed from the table or split into separate tables.

This leads to a key question: why not use separate tables for municipalities, provinces, and regions? While this is common, it makes analysis harder because queries have to be repeated for each table or use complex logic, which adds to maintenance work. By putting all territorial units into one dimension, the system can use the same rules for grouping and filtering at any spatial level. Queries stay consistent whether you look at municipal, provincial, or regional data and this reduces repeated work, makes maintenance easier, and keeps rules the same for all levels. Spatial shapes are handled separately from analytical details and are connected to the territory dimension using consistent IDs. This keeps analytical queries fast and simple, and spatial data is only loaded when needed for maps. This way, the system supports both data analysis and mapping without slowing down non-spatial queries.

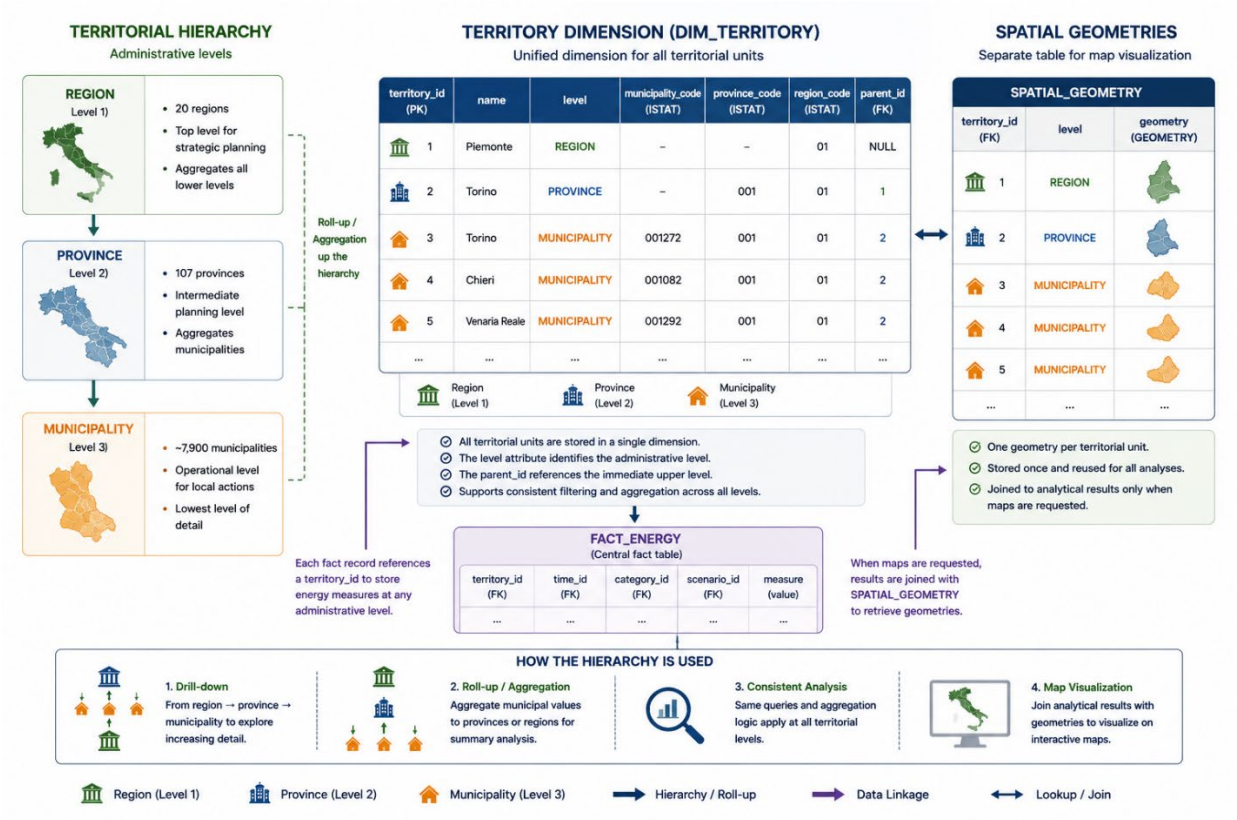

Figure 4—3 Unified territory dimension supporting multi-scale spatial analysis through hierarchical administrative relationships and linked geographic representations.

4.6. Energy Categories and Analytical Domains

A key challenge in database design is how to classify and interpret energy values. While all values are in megawatt-hours and linked to a place and time, treating them as the same can cause confusion. What does an energy value mean? Is it consumption, production, or a future scenario? The numbers may look the same, but their meaning is very different.

To solve this, the platform uses a separate energy category dimension that clearly defines analytical domains. Each energy value in the fact table is linked to an energy category that explains how to interpret it.

Analytical Domains

Right now, energy categories are grouped into a few main domains, such as:

- Consumption represents final energy demand within a territory.
- Production represents actual renewable energy generation.
- Future production represents projected or scenario-based energy outputs.

This difference is important because it changes how values are used in queries and calculations. For example, consumption values are never increased in scenarios, but production values can be changed based on user choices.

By storing these differences directly in the database, you avoid putting domain logic in the application code. This means the meaning of each value is kept in the data model, not spread out in different parts of the backend.

Category Code	Category Name	Analytical Domain	Base Group	Description
C001	Electricity Demand	Consumption	—	Final electricity consumption from all sectors
P001	Solar PV Production	Production	Solar	Electricity produced by photovoltaic systems
P002	Wind Energy	Production	Wind	Electricity generated by onshore wind turbines
FP001	Projected Solar PV	Future Production	Solar	Projected PV output under scenario assumptions
FP002	Projected Wind Output	Future Production	Wind	Scenario-based wind production estimation

Table 4-2 Example Energy Categories and Associated Analytical Domains

Base Groups and Energy Sources

Besides analytical domains, energy categories also show the type of energy source using a base group. In this project, the base groups include renewable sources such as solar, wind, hydroelectric, and biomass.

Why add this extra layer? Many analyses need to group or filter values by source. For example, a user might want to see only solar energy or apply a scenario change just to wind and solar. By making the base group an attribute of the energy category, the backend can handle these cases easily. Queries can filter by base group without knowing the exact category codes, making the system more flexible and easier to maintain.

Role in Scenario Computation

This design choice is especially important for scenario analysis. In the platform, scenarios work by changing production values but keeping consumption the same. The analytical domain linked to each energy category makes sure this rule is followed.

For example, when a user applies a 10% uplift to solar and wind production, the backend identifies all categories that:

belong to the production domain and match the selected base groups!

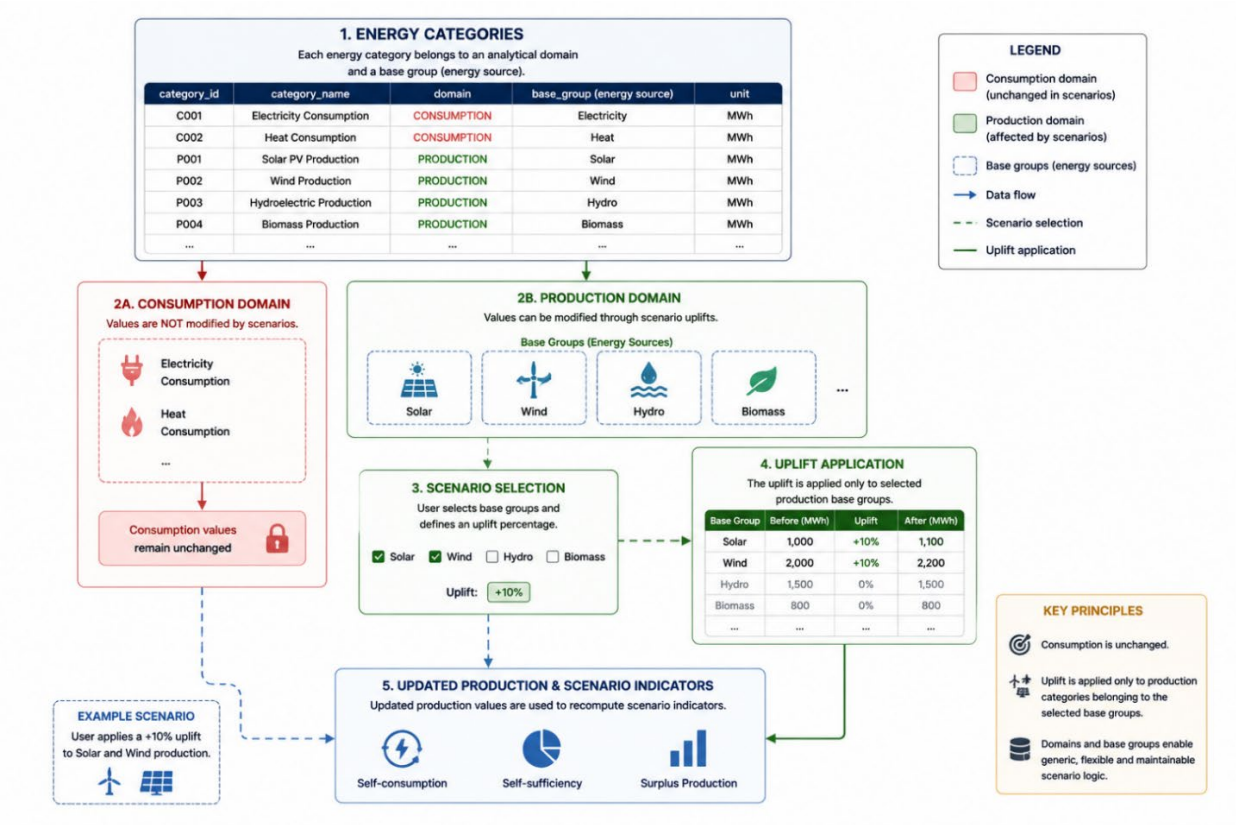

Figure 4—4 ***Example of 10% uplift to solar and wind production,***
Use of energy domains and base groups in scenario computation, illustrating how production categories are selected and modified through scenario uplifts while consumption values remain unchanged.

Supporting Derived Indicators

Energy categories are also key to calculating indicators such as self-consumption, self-sufficiency, and overproduction. These are not stored directly in the fact table but are worked out by combining values from different domains. For example, self-consumption is found by comparing consumption and production values at the same place and time. Since each value's domain is clearly defined, these calculations can be performed consistently and reliably in the backend. Without this setup, these indicators would depend on hidden rules, such as column names or query context, which are hard to verify and maintain as the system evolves.

Design Considerations and their Trade-offs!

This modeling approach introduces an additional level of abstraction, which may appear complex at first. However, the benefit is that this modeling approach adds another layer of abstraction, which might seem complex at first. But it clearly separates what a value is from how it is used. This makes the system easy to extend: you can add new energy sources, domains, or concepts by adding new category records, without changing the whole schema. In practice, this

was crucial for aligning the database with the platform's goals, especially for scenario analysis and for comparing data across different scales to explore the implications of alternative assumptions and policy choices. In the database design, scenarios are modeled explicitly through a dedicated scenario dimension. Each scenario record represents a distinct analytical context, ranging from the baseline configuration to user-defined hypothetical scenarios. Rather than duplicating entire datasets, scenario identifiers are associated with fact table records, enabling multiple scenarios to coexist within the same schema.

This setup makes sure scenario results can be tracked and compared. Baseline and scenario values can be queried, grouped, and compared using the same logic, with only the scenario reference changing. This makes scenario analysis a main feature, not something added on later.

4.7. Scenario Dimension

Scenarios are a core concept in territorial energy planning, allowing stakeholders to explore the implications of alternative assumptions and policy choices. In the database design, scenarios are modeled explicitly through a dedicated scenario dimension. Each scenario record represents a distinct analytical context, ranging from the baseline configuration to user-defined hypothetical scenarios. Rather than duplicating entire datasets, scenario identifiers are associated with fact table records, enabling multiple scenarios to coexist within the same schema.

This approach ensures that scenario results remain traceable and comparable. Baseline and scenario values can be queried, aggregated, and contrasted using the same analytical logic, differing only in their scenario reference. As a result, scenario analysis becomes a first-class analytical feature rather than an external or ad hoc process.

4.8. Keys, Constraints, and Data Integrity

Maintaining data integrity is essential in analytical systems, particularly when results are used to support planning and decision-making. The database schema, therefore, enforces integrity through a combination of primary keys, foreign keys, and constraints.

Each dimension table defines a unique primary key that is referenced by the fact table. These foreign key relationships ensure that every analytical value is associated with valid contextual information for time, territory, energy category, and scenario.

Additional constraints are used to prevent inconsistent or duplicate records, particularly when storing aggregated values at different temporal resolutions. Together, these mechanisms ensure that analytical queries operate on a coherent and internally consistent dataset.

4.9. Summary

This chapter describes the database design underlying the territorial energy analyThis chapter explains the database design behind the territorial energy analysis platform. Based on the needs of multi-scale energy planning, a multidimensional data model was chosen to allow flexible grouping, clear meaning, and easy future expansion.nd territory dimensions, structured energy categories, and scenario modeling, were informed by both technical considerations and the need for transparent and reproducible analysis. These choices provide a robust foundation for the backend services described in the following chapter.

The next chapter will show how the backend uses this database design to run efficient, reliable, and user-friendly analytical queries.

5. Backend Implementation and API Design

5.1. Role of the Backend in the Platform

Once we defined the database schema, we moved on to designing a backend that could share data in a secure and flexible way. The backend does more than just link the database to the frontend; it also controls how users access and combine analytical concepts. We had to decide how much logic should be handled by the backend and how much by the frontend. Because of using logic too much on the client side maybe there are some issues with duplicated rules, inconsistent results, and less control over query accuracy. By making the backend responsible for data aggregation, validation, and scenario computation we tried to prevent this problem. Then, the frontend part only talks to the backend through specific API endpoints and it never accesses the database directly or handles raw data. The approach that we used keeps results consistent, reproducible, and aligned with the data model.

5.2. Technology Stack and Design Choices

The backend uses the Flask framework in Python. Flask was chosen because it is lightweight and flexible, making it a good fit for building modular APIs without strict architectural limits.

Why not use a larger framework? For this project, clarity and control were more important than advanced features. Flask keeps backend logic clear, which makes it easier to understand how queries and data flow work. This is especially important in an academic setting.

Database access is handled with direct SQL queries through a special utility layer. This method was chosen instead of relying too much on a full object-relational mapping (ORM) system. While ORMs can make development easier, they can also hide how queries work and make it harder to optimize complex analytical queries.

Writing SQL queries directly gives the backend full control over joins, aggregations, and filters. This was especially helpful when handling multidimensional data and large aggregation queries.

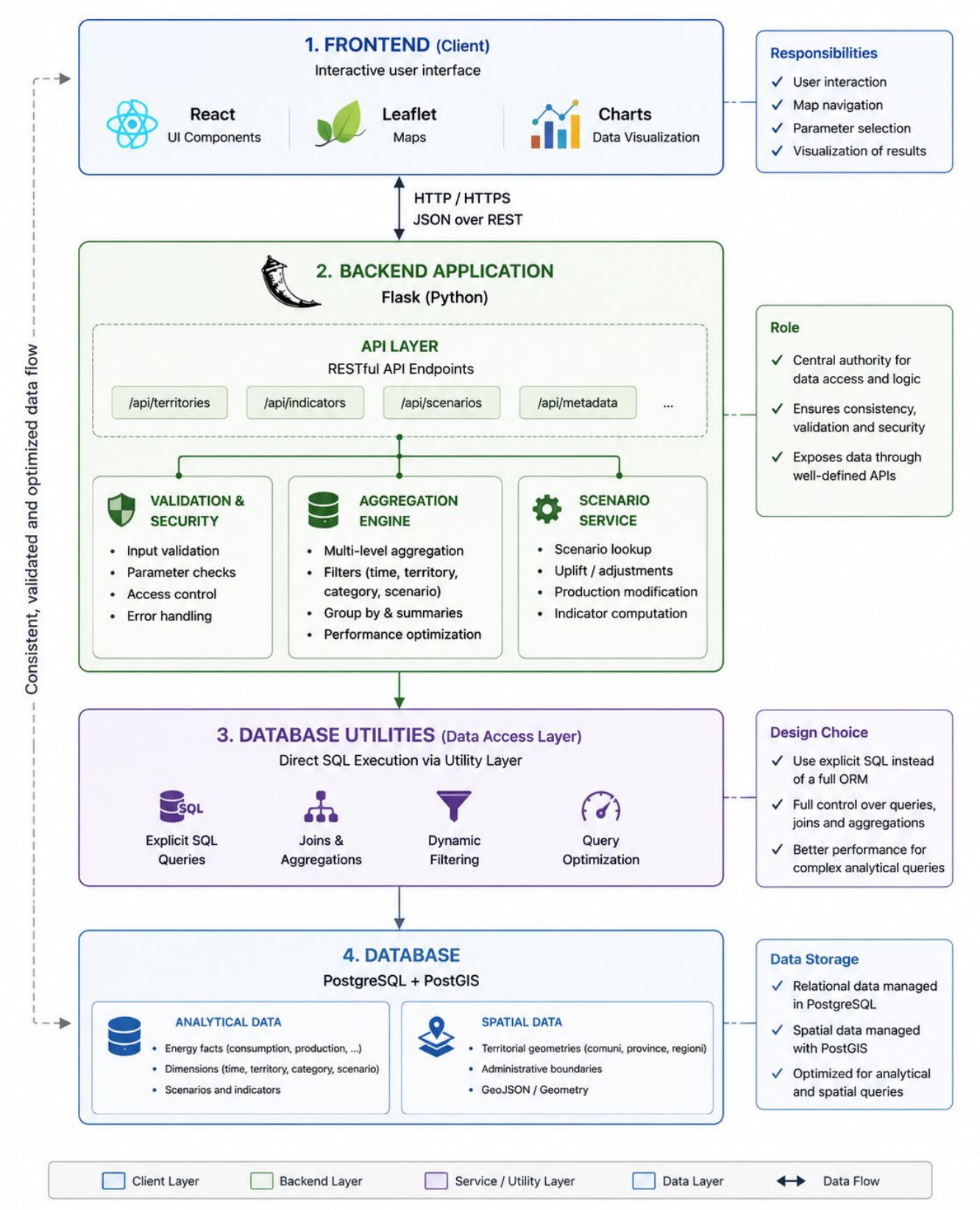

Figure 5—1 Backend technology stack and main components, illustrating the interaction between the frontend, Flask application, analytical services, database utilities, and PostgreSQL database.

5.3. Backend Structure and Modularization

As the backend grew, it soon became obvious that a flat, monolithic setup would not scale, even for a thesis project. Putting database queries, validation, and response formatting directly in API route handlers might seem easy at first, but this method quickly becomes fragile as the number of endpoints grows.

This raised a practical question: how can backend complexity be managed without making the system too complicated? In this project, we find the answer that we can to modularize the backend by function. Rather than sorting files only by technical layers, modules were grouped by their roles in the platform's analytical workflow.

At a high level, the backend is split into three main groups of modules:

- API modules, responsible for handling HTTP requests and responses.
- Utility and service modules, responsible for database access, aggregation logic, and data transformation.
- Configuration modules, responsible for environment variables, database connections, and shared settings.

This separation was not just for style. It came from real development needs. For example, aggregation logic for multi-scale energy data is used by several endpoints. Choropleth maps, time series, and scenario previews all use similar grouping rules. By putting this logic in utility modules, code duplication is avoided and inconsistencies are less likely.

Another advantage of this structure is that it corresponds to the platform's main ideas. API modules are grouped by analytical concepts like energy values, territorial selection, and scenarios, instead of by database tables. This way, reading the backend code shows how the system is meant to be used, not just how it works.

The modular setup we used also helps us make debugging and development easier. For example, imagine that unexpected results showed up in the frontend, it was usually possible to narrow down the problem to a specific layer, such as parameter validation, query construction, or aggregation, without having to search through large, tangled route handlers.

Some might wonder if this level of structure is too much for an academic prototype. In reality, it had the opposite effect. By keeping modules small and focused, the backend stayed readable and

flexible. New features, like extra filters or scenario logic, could be added without having to reorganize existing code.

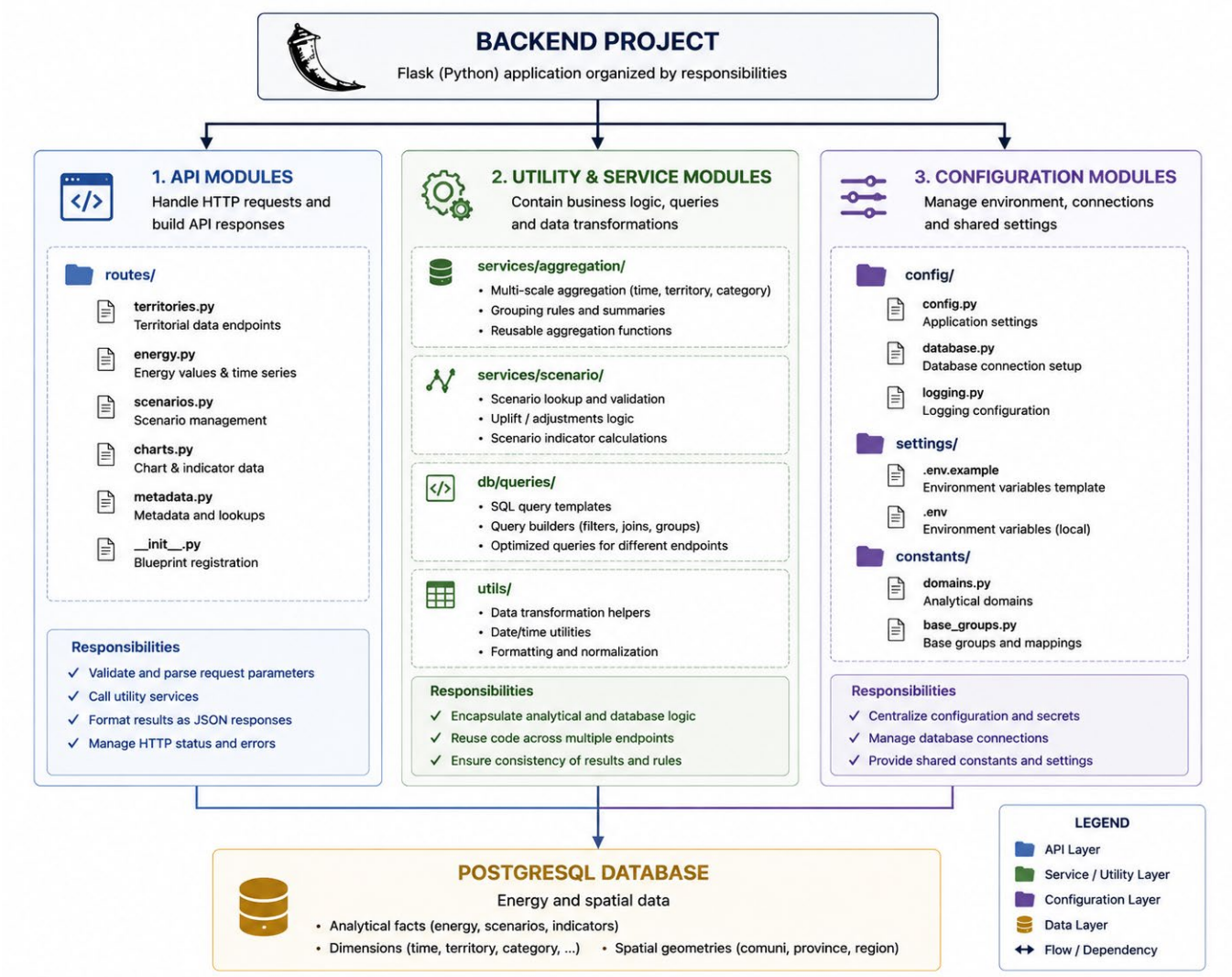

Figure 5—2 Backend module organization showing the separation of responsibilities between API endpoints, utility services, and configuration components.

Overall, backend modularization played a key role in maintaining the system's maintainability while supporting increasingly complex analytical requirements. It provided a balance between simplicity and structure, which is particularly important in projects that evolve incrementally, as was the case for this thesis.

5.4. API Design Principles

The API design follows a resource-oriented approach. Endpoints are not generic database query interfaces; instead, they expose analytical concepts directly. These examples, rather than allowing arbitrary SQL-like filters, the backend provides endpoints such as:

- retrieval of aggregated values for choropleth maps
- Extraction of time series for charts
- preview and persistence of scenario results

Now we should ask, Why is this important? And the answer is, it limited the data access, and by limiting the query shapes, the backend prevents invalid parameter sets and enforces analytical consistency. This is particularly relevant when dealing with multi-scale data, where not all combinations of level, resolution, and domain are meaningful.

Endpoint Path	Purpose Description	Related Component
/map/territories	Retrieve spatial boundaries and metadata for selected territories	Spatial Visualization
/map/values	Return aggregated values for choropleth map rendering	Spatial Analysis
/charts/values	Provide single aggregated values for numeric indicators	Chart Components
/charts/series	Return time-series data for line and bar charts	Time-Series Analysis
/scenarios/preview	Compute temporary scenario modifications for live exploration	Scenario Engine
/scenarios/save	Persist finalized scenarios with metadata	Scenario Management
/parameters/meta	Provide allowed values and validation rules for filters	Parameter Logic

Table 5-1Overview of Primary API Endpoints and Their Purposes

5.5. Request Validation and Parameter Handling

The backend starts by validating every API request and checks that all required parameters are included, makes sure values are allowed, and quickly rejects any incompatible combinations.

This validation step has two main goals. It protects the database from costly or pointless queries and gives the frontend quick, clear feedback so user interfaces can handle invalid input smoothly.

A key design choice was to clearly define allowed values, like spatial levels, time resolutions, and analysis domains, in the backend. This makes assumptions clear and helps prevent silent errors that could cause misleading results.

The backend only turns a request into a database query after it passes validation. Keeping validation and execution separate makes the backend logic easier to understand and maintain.

5.6. Data Aggregation and Query Execution

After a request is validated, the backend builds SQL queries that work directly on the fact table and related dimensions. Aggregation happens in the database using standard SQL, which is well-suited for this. The main idea is to let the database handle as much computation as possible. Rather than pulling raw data and processing it in the application, the backend uses SQL grouping and filtering to get the final results.

This method reduces data transfer, boosts performance, and keeps aggregation logic centralized and consistent. It also fits well with the multidimensional data model discussed earlier.

In Figure 5-3 on the next page, I will show how the backend works in seven steps, and the backend query is more understandable.

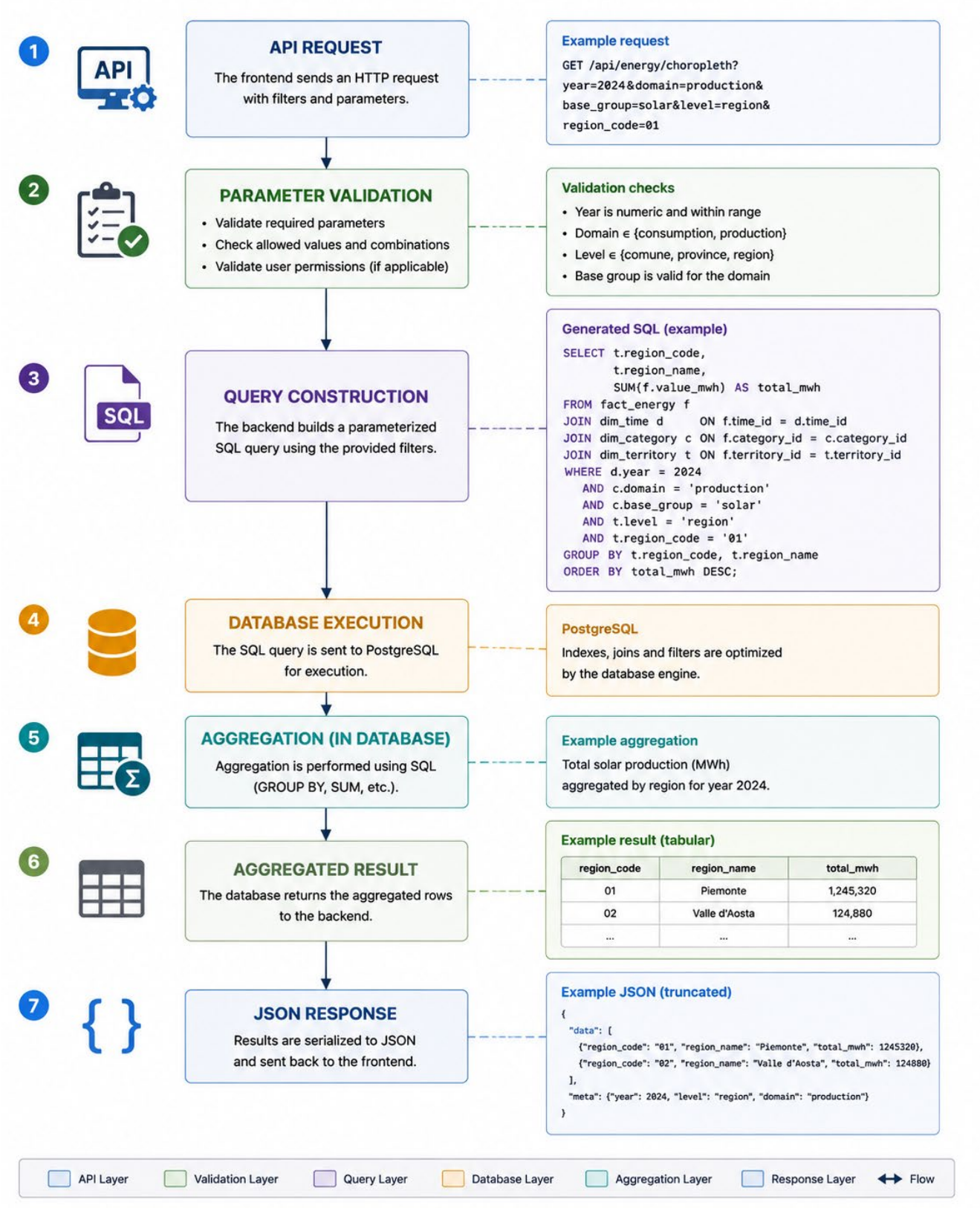

Figure 5—3 Backend query execution workflow illustrating request validation, SQL aggregation, database execution, and response generation.

5.7. Scenario Preview and Persistence Logic

Scenario analysis is more complex than just retrieving data. On this platform, you can preview scenarios in real time before saving them. The backend calculates possible results but does not change the actual data.

To make this possible, the scenario preview uses a read-only process and it gets some baseline values from the fact table, applies the user's assumptions, and sends the results to the frontend without saving any changes.

Results are saved to the database only when the user decides to save a scenario. Then the calculated indicators are entered into a special table linked to the scenario. This keeps exploratory analysis separate from saved data.

This separation is important for data integrity. Temporary tests do not affect stored results, and saved scenarios can always be reproduced and tracked.

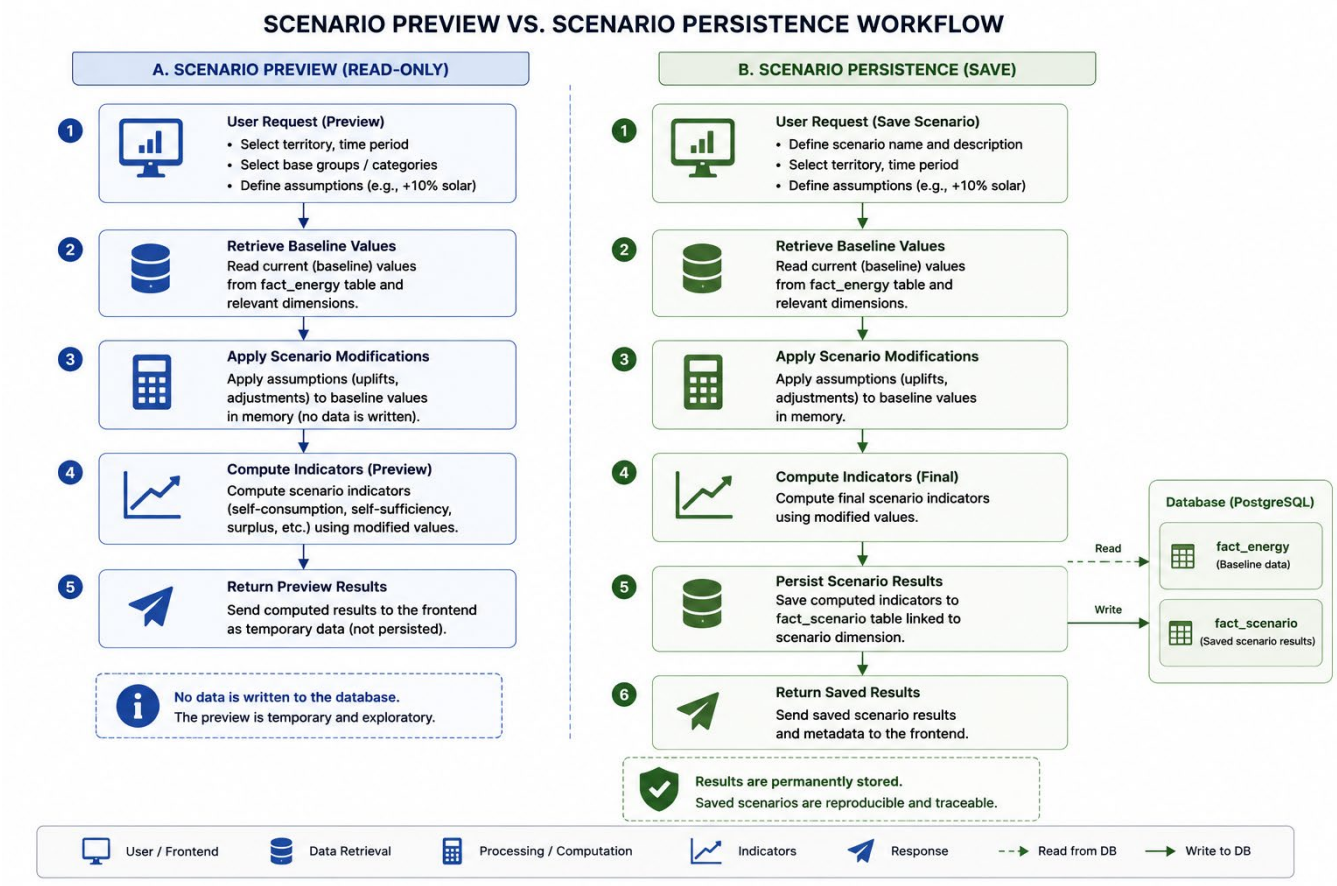

Figure 5—4 Workflow for scenario preview and scenario persistence, illustrating the distinction between temporary read-only computations and permanently stored scenario results.

5.8. Summary

This chapter describes the implementation of the backend services that expose the database to the frontend application. Design decisions regarding technology selection, modularization, API structure, validation, and aggregation logic were discussed in detail.

By centralizing analytical logic in the backend and exposing well-defined endpoints, the system ensures consistency, reproducibility, and scalability. The backend serves not only as a data provider but also as a guardian of analytical meaning, enforcing the assumptions encoded in the database design.

The next chapter focuses on performance considerations and optimization strategies adopted to support interactive visualization at multiple spatial and temporal scales.

6. API Design and Endpoints

6.1. Design Principles of the API

The API is the main link between the platform's analytical backend and its web-based frontend. Its design is more than just a technical detail; it is a key part of the system's architecture that shapes how analytical features are shared, combined, and reused. For territorial energy analysis, the API needs to turn complex data into clear and useful operations that match the questions planners ask, instead of exposing database details.

The main principle in designing the API is to match it with real analytical tasks. Rather than showing low-level data or generic queries, the endpoints are built around what users actually want to analyze. For example, the API offers endpoints for getting data for choropleth maps or time-series charts, not just for accessing database tables. This way, the API uses terms and structures that make sense to planners and frontend developers, making it easier to use and understand.

Another key principle is keeping clear boundaries between layers and the API acts as a controlled point of contact: the frontend sends structured parameters to show what analysis is needed, and the backend handles validation, aggregation, and interpretation. The frontend never builds SQL logic or decides on aggregation rules, which makes the system easier to maintain and avoids mistakes separating is very useful, and as it could happen if logic were repeated in different places.

Another important design choice is using clear parameters, and also every API endpoint uses well-defined parameters for things like spatial area, time resolution, analytical topic, and optional filters. Instead of allowing free-form inputs, the API only accepts allowed values and combinations. This approach stops invalid or meaningless requests, like asking for time resolutions that do not make sense together. It also makes each endpoint's behavior clear and predictable, which is important for decision-making. The API also follows analytical determinism: with the same parameters, it always returns the same result, regardless of the frontend state or the order of requests. This is crucial for reproducibility and traceability, especially when results are used in planning or compared across scenarios. To support this, the API avoids hidden defaults and makes all important assumptions clear through parameters or metadata.

The API is built to be stateless. Every request includes all the information needed to respond, so there is no need for server-side session state. This makes it easier to scale, debug, and test the system, and it adheres to standard RESTful design principles. Statelessness is especially useful in analytical platforms, where users might view different views simultaneously or repeat queries.

Another important principle is keeping things consistent across endpoints. Parameters for space, time, and categories use the same names and meanings everywhere in the API. For example, the way you specify a territory or time resolution is the same for both map and chart endpoints. This consistency makes frontend development easier and reduces the chance of mistakes.

The API is also built to be easy to extend. New endpoints, analytical areas, or scenario types can be added without breaking existing clients. This is done by keeping endpoints focused, avoiding confusing meanings, and using metadata to define parameters. Because of this, the API can grow and change with the data model and analytical needs, without causing major disruptions.

All these design principles make the API a stable agreement for analytics, not just a simple way to access data. The next sections explain how these ideas are used in map, chart, and scenario endpoints.

6.2. Map Endpoints

Map-based visualization is one of the main ways users interact with the platform. Choropleth maps help users easily compare energy values across different areas and spot patterns that are hard to see in tables or numbers alone. Because of this, the API offers special map endpoints that provide spatially grouped data ready for web mapping tools.

Purpose and Scope of Map Endpoints

The map endpoints are built to answer a clear analytical question:

“What is the value of a given energy measure for each territory at a selected spatial and temporal scope?”

Instead of showing raw geometries or general spatial queries, the API turns this question into a few focused endpoints. These endpoints return values already grouped by the chosen territorial level,

like municipality, province, or region. This way, the frontend can display maps directly without extra processing.

Map endpoints serve two complementary roles:

1. Delivering territorial values required to color map features (e.g., energy consumption per municipality).
2. Delivering spatial geometries that define the shapes of those territories.

These two roles are kept separate to avoid sending extra data and to allow the frontend to work more flexibly.

Separation Between Analytical Values and Geometries

A major design choice is to separate endpoints that give analytical values from those that provide spatial geometries. Analytical endpoints return numbers linked to territory IDs, while geometry endpoints return GeoJSON features that show administrative boundaries.

This separation has several benefits:

- You can request analytical values without also getting geometries, which keeps the data smaller if the client already has the geometries cached.
- Geometries can be used again for different analytical views, such as for various energy domains or years.
- The backend logic stays simpler because spatial and analytical grouping are kept separate.

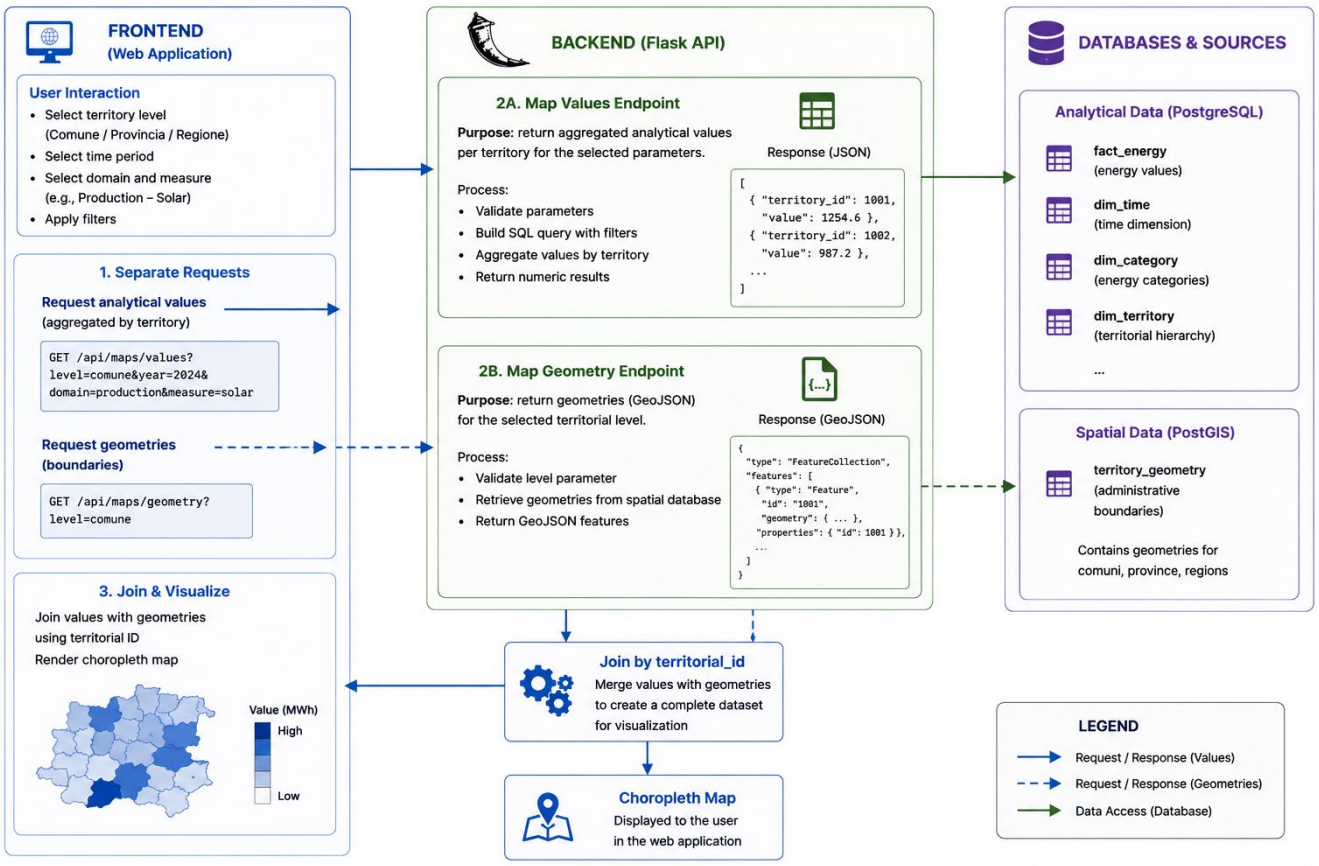

Figure 6—1 Architecture of map-oriented API endpoints illustrating the independent retrieval of analytical values and spatial geometries and their integration in the frontend for map rendering.

Territorial Aggregation Strategy

Map endpoints return data grouped at one clearly defined territorial level. The level you request decides both:

- Which territorial identifiers are used
- How fact-level data is aggregated.

For example, if you request a province-level map, all municipal values are grouped based on set spatial rules before being returned and this grouping happens in the backend and database, so spatial logic stays consistent for everyone.

Importantly, the API doesn't allow automatic aggregation. You must always set the spatial level using parameters. This prevents confusion and keeps each map's meaning clear and repeatable for Web Mapping.

GeoJSON Delivery for Web Mapping

Spatial shapes are sent in GeoJSON format, which most web mapping libraries support. Each feature includes:

- a unique territorial identifier,
- administrative attributes (e.g., name, code, level),
- the polygon or multipolygon geometry.

The backend does not put analytical values directly into the GeoJSON. Instead, shapes and values are connected using shared identifiers. This setup lets the frontend:

dynamically update map coloring without reloading geometries, switch between analytical layers efficiently.

Attribute Name	Description	Used by Frontend
id	Unique identifier for the spatial unit (e.g., municipality code)	Yes
name	Human-readable name of the spatial unit	Yes
level	Administrative level (municipality, province, region)	Yes
geometry	GeoJSON polygon or multipolygon geometry	Yes
centroid	Centroid coordinates for placing labels	Yes
parent_code	Code of the parent territory (for hierarchy reconstruction)	No
source	Origin of the geometry (e.g., ISTAT shapefile)	No
simplified	Boolean flag indicating if geometry has been simplified	No

Table 6-1GeoJSON Attributes Exposed by Map Geometry Endpoints

Validation and Constraints

Map endpoints enforce strict validation rules to ensure analytical correctness. Parameters defining spatial level, energy domain, temporal resolution, and filters are validated before any query is executed. Invalid combinations—such as requesting incompatible time resolutions for a given domain—are rejected with clear error messages.

When validation logic is handled in the map endpoints, the platform makes sure that each choropleth map shown on the frontend matches a clear analytical request.

Design Implications

For designing map endpoints to align with analytical purposes rather than data structures, and the API achieves several goals:

- consistent spatial aggregation across views,
- predictable frontend behavior,
- reduced coupling between visualization logic and database design,
- extensibility to new energy domains or territorial levels.

The characteristics are essential for decision-support systems, where maps are often used as evidence in planning discussions rather than as purely exploratory visualizations.

6.3. Chart Endpoints

While map endpoints support spatial comparison at a given aggregation level, many analytical tasks in territorial energy planning require observing how values evolve. Charts provide a complementary perspective to maps, enabling users to analyze temporal trends, seasonal patterns, and variations across different analytical domains. The platform offers a set of chart-focused API endpoints to support these use cases.

Chart endpoints provide time-indexed data that the frontend can use to create line charts, bar charts, or stacked visualizations. While map endpoints always group values at one spatial level, chart endpoints focus on showing changes over time.

Purpose and Analytical Scope

The primary goal of chart endpoints is to answer questions such as:

- How does energy consumption evolve in a selected territory?
- How does renewable production vary by month or season?
- How do baseline and scenario values compare across time?

To address these questions, chart endpoints return ordered time series where each data point corresponds to a specific temporal unit (e.g., hour, month, year) and a selected analytical context.

Chart endpoints, therefore, focus on:

- temporal aggregation,
- ordering of results,
- consistency of time units across requests.

Time-Series Versus Aggregated Views

Two main types of chart outputs are supported:

1. Continuous time series, where values are returned for each time step at a given resolution (e.g., monthly values across several years).
2. Aggregated temporal views, where values are grouped according to derived temporal attributes such as month, season, or day type.

This distinction is handled entirely by backend logic. The frontend specifies the desired temporal resolution and filters, while the backend ensures that grouping and ordering are applied consistently.

Temporal Resolution	Typical Chart Type	Planning Question Supported
Hourly	Line chart	How does renewable production vary throughout the day?
Daily (average)	Bar chart	What is the average daily consumption by month or season?
Monthly	Line or area chart	What are the monthly trends in energy balance across territories?
Seasonal	Stacked bar chart	How does energy demand shift between summer and winter?
Annual	Column chart	How do yearly totals compare across energy categories or regions?

Table 6-2 Supported Temporal Resolutions and Chart Use Cases

Territorial Context in Chart Queries

Unlike map endpoints, chart endpoints operate on one territorial context per request. A chart always represents values for a single municipality, province, or region and this design choice avoids ambiguity and ensures that time-series data remain interpretable.

Territorial aggregation is applied before temporal grouping. For example, when a province-level chart is requested, all underlying municipal values are aggregated first, and only then organized as a time series. This guarantees that chart values are consistent with map values at the same spatial level.

By enforcing this ordering, the API prevents subtle inconsistencies that could arise if temporal and spatial aggregation were mixed arbitrarily.

Output Structure and Ordering Guarantees

Chart endpoints return structured JSON responses where:

- each data point includes a time reference and a numerical value,
- The results are strictly ordered along the time axis.
- missing time steps are handled explicitly (e.g., zero values or nulls, depending on context).

This explicit ordering is critical for frontend visualization libraries, which rely on consistent time sequences to render charts correctly. The backend, therefore, assumes responsibility for sorting, grouping, and completeness of time-series outputs.

Interaction with Scenarios

Chart endpoints are also used to visualize scenario-based results. When a scenario context is provided, the backend applies scenario-specific transformations before generating the time series. From the frontend's perspective, baseline and scenario charts are retrieved through the same interface, differing only in parameter values.

This design avoids the need for separate "scenario chart" endpoints and ensures that baseline and scenario results remain directly comparable. The implications of this approach are discussed in detail in Chapter 7.

Design Implications

We are separating chart endpoints from map endpoints. The API achieves a clear division of responsibilities:

- map endpoints focus on spatial comparison,
- chart endpoints focus on temporal evolution.

This split makes both backend and frontend work simpler. You can still combine views, like clicking a map area to create a chart. By handling time logic in the backend, all charts use the same rules and time settings, which keeps analysis consistent for planning.

6.4. Scenario Endpoints

Scenario endpoints enable the platform to support *what-if* analysis, which is a core requirement in territorial energy planning. While map and chart endpoints expose baseline analytical values, scenario endpoints allow users to explore alternative configurations, evaluate their impacts, and optionally persist results for later comparison.

From an API design perspective, scenario endpoints must balance flexibility, traceability, and analytical consistency. They are therefore designed around clearly defined scenario states and lifecycle stages rather than around raw data manipulation.

Purpose of Scenario Endpoints

The main objectives of scenario endpoints are to:

- expose predefined scenarios available in the system,
- allow users to preview hypothetical modifications to energy production,
- compute derived indicators under alternative assumptions,
- persist selected scenarios and their results in a reproducible manner.

These endpoints act as a bridge between user intent (scenario assumptions) and analytical outcomes (modified values and indicators), while ensuring that baseline data remain unchanged.

Baseline and Scenario Context

Scenario endpoints operate relative to a baseline context, which represents the reference state of energy data for a given territory and time period. The baseline is immutable and always available for comparison.

A scenario is a controlled modification of baseline values, typically affecting production-related energy categories. Scenarios do not overwrite baseline data; instead, they generate derived values that coexist alongside baseline results.

This distinction is enforced at the API level by requiring scenario context to be explicitly specified in requests that involve scenario-based computation.

Scenario Listing and Metadata

One group of scenario endpoints is dedicated to scenario discovery. These endpoints allow the frontend to retrieve:

- the list of available predefined scenarios,
- descriptive metadata associated with each scenario,
- identifiers required to reference scenarios in subsequent requests.

This functionality is essential for transparency, as users must understand which assumptions underlie a given scenario before interpreting its results.

Scenario Identifier	Scenario Type	Description	Editable Parameters
baseline_it_2022	Baseline	Official territorial data for Italy in 2022	None
nze_targets_2030	Predefined	National Zero Emissions projection for 2030	None
user_abc_solar10	User-defined	User-created scenario with +10% solar production	Uplift (%), Energy Sources
user_xyz_mixed15	User-defined	Mixed scenario with solar + wind increased by 15%	Uplift (%), Energy Sources
draft_temp	User-defined (draft)	Temporary preview for scenario editor	Uplift (%), Base Groups

Table 6-3 Scenario Metadata Exposed by the API

Scenario Preview Endpoints

Scenario preview endpoints allow users to experiment with alternative assumptions without committing them to persistent storage. These endpoints accept scenario parameters—such as uplift percentages or selected energy sources—and return modified analytical results.

Key characteristics of preview endpoints include:

- results are computed on-the-fly,
- No data is written to the database,
- Responses are clearly marked as temporary.

Preview endpoints are critical during exploration analysis, where users may want to test multiple configurations rapidly before selecting a final scenario.

From an architectural standpoint, preview endpoints reuse the same aggregation and indicator logic as persistent scenarios, ensuring that previewed results are analytically equivalent to stored ones.

Scenario Persistence Endpoints

Once a user decides to retain a scenario, persistence endpoints are used to store:

- scenario parameters,
- derived energy values,
- computed indicators.

Persisted scenarios can later be retrieved, visualized, and compared against other scenarios or the baseline. This capability supports longitudinal analysis and collaborative planning workflows.

Importantly, persistence endpoints do not store raw user inputs alone; they store interpretable results linked to explicit assumptions, enabling reproducibility and auditability.

Territorial Scope of Scenarios

Scenario endpoints always operate within a clearly defined territorial scope. A scenario is associated with a specific municipality, province, or region, and its effects are computed accordingly.

This design prevents ambiguous interpretations, such as applying the same scenario simultaneously at multiple spatial levels. It also ensures that scenario results are consistent with baseline aggregation rules used by map and chart endpoints.

Output Structure and Indicator Integration

Responses from scenario endpoints may include:

- modified energy values,
- derived indicators (e.g., self-consumption, self-sufficiency),
- metadata describing applied assumptions.

Rather than exposing raw intermediate calculations, scenario endpoints return results that are directly usable by the frontend for visualization and comparison.

The computation of indicators themselves is not detailed at this stage; it is formally defined in Chapter 7. At the API level, scenario endpoints treat indicators as first-class analytical outputs.

Design Rationale

Separating scenario endpoints from map and chart endpoints provides several benefits:

- Baseline endpoints remain predictable and straightforward,
- Scenario logic is centralized and explicitly invoked,
- Frontend components can manage scenario workflows without embedding analytical rules.

This design also ensures that new scenario types can be introduced without modifying existing baseline endpoints, preserving backward compatibility.

6.5. Parameter Metadata Strategy

Interactive analytical platforms require more than a collection of API endpoints. They also need a shared understanding between frontend and backend about which parameters are allowed, how they can be combined, and what their analytical meaning is. Without this shared understanding, user-driven exploration can easily result in invalid queries, misleading results, or inconsistent interpretations.

To address this challenge, the platform adopts an explicit parameter metadata strategy, in which the backend exposes structured information describing available parameters, valid values, and applicable constraints. Rather than treating parameters as free-form inputs, they are modeled as part of the analytical contract between frontend and backend.

Motivation for Explicit Parameter Metadata

In a multi-scale energy analysis context, parameters are not independent. For example:

- Not all temporal resolutions are meaningful at all spatial levels,
- Some energy categories are valid only for specific analytical domains,
- Scenario-related parameters apply only to production values, not consumption.

Embedding these rules implicitly in frontend logic or backend validation code would make the system difficult to maintain and error-prone. Instead, parameter metadata provides a declarative way to describe analytical constraints once and reuse them consistently.

This strategy improves transparency, reduces duplication of logic, and ensures that frontend interfaces remain aligned with backend capabilities as the system evolves.

Types of Parameters Described by Metadata

The parameter metadata strategy covers several categories of analytical parameters, including:

- Spatial parameters, such as territory level and territorial identifiers.
- Temporal parameters, including year, month, hour, season, and day type.
- Analytical domain parameters, such as consumption, production, and future production.
- Energy category parameters, including base groups and specific energy sources.
- Scenario-related parameters, such as uplift factors or scenario identifiers.

Each parameter is described not only by its name and type, but also by its allowed values, default behavior, and applicability in context.

Structure and Exposure of Parameter Metadata

Parameter metadata is exposed via dedicated API resources that the frontend can query during initialization or during interactions. These resources return structured descriptions of parameters in a machine-readable format, enabling frontend components to:

- populate dropdown menus and selectors,
- turn filters on or off dynamically,
- prevent invalid parameter combinations before requests are sent.

Rather than hard-coding parameter lists in the frontend, this approach allows user interfaces to adapt automatically when new analytical options are introduced at the backend level.

Parameter Name	Description	Allowed Values Example	Applicable Contexts	Dependency Constraints
year	Calendar year of data	2020, 2021, 2022, 2030	Map, Chart, Scenario	Must be available for the selected domain
month	Month of year (1–12)	1–12	Chart (monthly), Scenario	Valid only if resolution = monthly
day_type	Classification as weekday or weekend	weekday, weekend	Chart (daily), Scenario	Depends on time resolution
category	Energy category (e.g., solar, wind)	solar, wind, biomass, total	All	Must belong to the selected domain or base group
scenario_id	Selected scenario context	baseline, user_xyz, nze_2030	Map, Chart, Scenario	Affects filtering and indicator computation
resolution	Temporal resolution for query	hourly, monthly, annual	Chart	Affects allowed filters (e.g., month)
spatial_level	Administrative level	municipality, province, region	Map, Chart, Scenario	Controls aggregation level and geometry type

Table 6-4 Parameter Metadata Attributes and Purpose

Validation and Constraint Enforcement

Parameter metadata plays a key role in validation. Before executing an analytical query, the backend verifies that the provided parameters:

- exist in the metadata definition,
- fall within allowed value ranges,
- respect dependency constraints with other parameters.

For example, requesting hourly data at a regional level may be disallowed or automatically downgraded to a coarser resolution, depending on metadata rules. By centralizing these constraints in metadata, the backend avoids scattering validation logic across individual endpoints.

This approach also simplifies error handling, as invalid requests can be rejected with meaningful, standardized error messages.

Benefits for Maintainability and Extensibility

One of the main advantages of a metadata-driven parameter strategy is extensibility. When new energy categories, temporal classifications, or scenario options are added, the system can expose them by updating metadata definitions rather than modifying endpoint logic or frontend code.

This design significantly reduces the risk of inconsistencies and makes the platform resilient to future changes in analytical scope. It also supports experimentation, as new analytical dimensions can be added incrementally without disrupting existing workflows.

Relationship with Scenario and Indicator Computation

Parameter metadata is also essential for scenario-based analysis. Scenario endpoints rely on metadata to determine:

- Which parameters can users modify?
- Which analytical domains are affected by scenarios?
- Which indicators need to be recalculated?

By encoding these rules declaratively, the platform ensures that scenario computation remains transparent and reproducible.

6.6. API Endpoints Summary

This section provides a consolidated overview of the API endpoints introduced in Chapter 6. While previous sections described endpoints by functional category—maps, charts, scenarios, and metadata—the purpose of this section is to provide a unified reference summarizing each endpoint's role, analytical purpose, and the type of data it returns. Rather than serving as implementation documentation, this summary clarifies how the API surface reflects the platform's analytical structure and supports consistent interaction patterns across frontend components.

Endpoint Categories and Responsibilities

The API endpoints are grouped into four main categories, each corresponding to a core analytical function of the platform:

- Map endpoints are responsible for delivering spatially aggregated values and geometries for territorial visualization.
- Chart endpoints are designed to return time-series or aggregated numerical values suitable for graphical representation.

- Scenario endpoints, enabling preview, persistence, and retrieval of user-defined scenarios.
- Parameter metadata endpoints, which expose valid parameters, constraints, and analytical contexts to the frontend.
- Each category follows a consistent design philosophy: endpoints expose analytical concepts, not raw database structures, and return data in formats directly usable by frontend components.

Overview of Exposed Endpoints

Endpoint Group	Endpoint Path	Purpose	Input Parameters (High-Level)	Output Type	Typical Frontend Use
Map	/map/territories	Retrieve the list of available territories	Spatial level (municipality, province, etc.)	JSON	Populate map filters or dropdowns
Map	/map/geojson	Deliver geometry for selected areas	Spatial level, territory codes	GeoJSON	Render a choropleth map
Chart	/charts/values	Return aggregated values for charting	Year, category, spatial level, scenario ID	JSON	Bar or pie charts
Chart	/charts/series	Generate time-series data	Category, time resolution, territory, scenario	JSON	Line charts, trend visualizations
Scenario	/scenarios	List available scenarios	—	JSON	Scenario selector
Scenario	/scenarios/territory/values	Retrieve computed values for a scenario	Scenario ID, spatial level, category	JSON	Display indicator comparisons
Scenario	/scenarios/territory	Load scenario values for a specific territory	Scenario ID, territory code	JSON	Edit scenario assumptions
Scenario	/scenarios/territory/preview	Preview uplifted production values	Uplift parameters (e.g., +10% solar)	JSON	Temporary chart/map refresh
Scenario	/scenarios/territory/save	Persist user-defined scenario	Scenario metadata, uplift parameters	JSON	Store custom scenario
Metadata	/meta/parameters	Provide allowed values for filters	—	JSON	Populate UI filters dynamically
Metadata	/meta/categories	List all energy categories and groups	—	JSON	Drive category selection logic

Table 6-5 Overview of the API endpoints exposed by the backend, organized by functional category and describing their analytical role, required parameters, response types, and frontend applications.

Consistency Across Endpoints

Despite serving different analytical purposes, all endpoints share common characteristics:

- parameters are validated using the metadata strategy described in Section 6.5,
- spatial and temporal aggregation is handled exclusively at the backend,
- Responses are structured to minimize frontend-side computation.

This consistency simplifies frontend development and ensures that different analytical views—maps, charts, and scenario previews—remain coherent and comparable.

Error Handling and Analytical Integrity

All the enforcement validation rules from the parameter metadata derived by the endpoint. Invalid or ambiguous requests are rejected before query execution, making sure that analytical results are not silently distorted.

Where incorrect aggregation or misinterpretation of indicators can lead to flawed conclusions, design choices are important in planning contexts. By forcing analytical integrity at the API level, the system acts as a safeguard rather than a passive data provider.

Role of the API Layer in the Overall Architecture

The API layer serves as the platform's analytical interface. It encapsulates database complexity, enforces consistency rules, and exposes domain-oriented resources aligned with planning tasks.

As a result, the frontend remains lightweight and flexible, while the backend retains complete control over how energy data is aggregated, combined, and interpreted.

7. Scenario Computation and Indicators

7.1. Scenario-Based Reasoning in Territorial Energy Planning

Scenario-based analysis is essential in territorial energy planning. It helps planners and decision-makers look at the possible effects of different assumptions before taking action. Instead of only describing the current energy system, scenarios allow for what-if reasoning. They make it possible to evaluate policy goals, infrastructure investments, and technology changes in a clear and controlled way.

On this platform, scenarios are not separate datasets that are disconnected from real data. Instead, they are changes made to a baseline setup, where certain parameters, mainly about energy production, are adjusted based on clear user assumptions. This method follows common energy planning practices, where future analysis is based on a clear reference point.

A clear distinction is therefore established between two conceptual elements:

- the baseline, representing observed or officially reported energy data for a given territory and period;
- The scenario represents a modified version of the baseline, in which specific energy components are adjusted.

This distinction is important for clear analysis. Without a baseline, it would be hard to interpret, compare, or check scenario results. Using the same baseline for all scenarios means any differences come from the assumptions used, not from inconsistent data handling.

Scenarios as Computational Layers

In the system, scenarios are implemented as computational layers rather than as separate datasets. The baseline data in the database stays the same. Scenario values are created on the fly by applying transformation rules to certain baseline values, usually those related to production.

This design choice has several advantages:

- it avoids duplication of large volumes of data,
- it ensures traceability between baseline and scenario values,
- It allows rapid iteration during scenario exploration.

By handling scenarios as computations rather than fixed records, the platform lets users preview changes interactively while keeping the original data safe.

Analytical Scope of Scenario Computation

Scenarios on the platform operate across the same spatial and temporal dimensions as the baseline data. A scenario may apply to:

- a single municipality, a province, or an entire region,
- a specific year or a broader temporal aggregation,
- selected energy sources or production technologies.

Right now, scenarios mainly change energy production values, while consumption is assumed to stay the same. This is a common approach in planning, where demand is either handled separately or kept constant when looking at supply changes. The effects of this choice will be discussed later in the chapter.

Role of Indicators in Scenario Evaluation

Raw energy values by themselves are not enough to compare scenarios in a useful way. That's why scenario analysis uses indicators that show the relationship between consumption and production in an easy-to-understand way.

Indicators such as self-consumption, self-sufficiency, and over-production provide planners with concise measures of system performance under different assumptions. On the platform, these indicators are not stored as independent data points but are computed during the scenario evaluation process.

The precise definitions and computation logic of these indicators are introduced in the following sections.

7.2. Baseline Data and User-Defined Scenarios

A clear separation between baseline data and user-defined scenarios is essential to ensure analytical correctness, transparency, and reproducibility. On the platform, this separation is enforced conceptually and computationally rather than through duplicating datasets.

Baseline Data as Reference State

Baseline data represents the reference state of the energy system for a given territory and time period. It consists of observed, reported, or officially validated values stored in the database, including energy consumption and production aggregated at different spatial and temporal resolutions.

The baseline fulfills three key roles:

1. **Reference for comparison**
   All scenario results are interpreted relative to the baseline. Changes in indicators or energy balances are meaningful only when compared against a common reference state.
2. **Source of analytical consistency**
   By maintaining a single authoritative baseline, the platform avoids inconsistencies that could arise from recalculating or duplicating reference values across different analytical contexts.
3. **Anchor for reproducibility**
   Baseline data remains immutable. This guarantees that scenario results can always be recomputed and verified, even after multiple exploratory analyses.

Importantly, baseline data is agnostic to scenarios. It does not encode assumptions, uplifts, or future projections beyond what is explicitly stored as baseline measurements.

User-Defined Scenarios as Transformations

User-defined scenarios are made by changing the baseline data. Rather than creating new records for every scenario, the platform works out scenario values by adjusting specific baseline values according to set rules.

At the moment, scenarios usually change only the production-related energy categories, while the consumption values remain unchanged. This is a standard method in energy planning, where demand changes are either managed gradually or considered by focusing on supply-side changes.

A scenario is defined by the following:

- a scope, specifying the affected territory and temporal extent;
- a selection of energy categories or base groups to which modifications apply;
- a set of transformation parameters, such as percentage uplifts or scaling factors.

Describing scenarios this way keeps the logic clear and easy to understand. Raw energy values by themselves are not enough for useful scenario comparisons. That’s why scenario analysis uses calculated indicators that clearly show the relationship between consumption and production.

Indicators such as self-consumption, self-sufficiency, and overproduction provide planners with simple ways to measure system performance under different assumptions. On the platform, these indicators are not saved as separate data points. Instead, they are calculated during scenario evaluation.

The next sections explain exactly how these indicators are defined and calculated.

Analytical Implications of the Separation

The difference between baseline and scenario values matters for analysis. Because scenario values are calculated on the fly instead of being stored, the platform can:

- compute scenario outcomes dynamically for interactive exploration;
- ensure that multiple scenarios can be compared consistently against the same baseline;
- prevent accidental mixing of baseline and scenario data in analytical results.

Keeping baseline and scenario values separate also helps with step-by-step analysis. For example, a user can see how increasing production affects self-sufficiency without changing the original consumption baseline. This makes it easy to spot the impact of any change right away.

Traceability and Validation

Treating scenarios as changes to the baseline also makes it easier to track where results come from. Each scenario result can be traced back to:

- the exact baseline values used,
- The transformation rules applied to
- the spatial and temporal scope of the computation.

Being able to trace results is especially important when analysis is used to support decisions, such as in policy or planning. By keeping baseline and scenario data separate, the platform helps prevent confusion and makes it easier to check results.

Position Within the Scenario Pipeline

At this stage, the chapter has established what baseline data and user-defined scenarios represent conceptually. The following sections focus on how these concepts are operationalized:

- Section 7.3 explains the distinction between scenario preview and scenario persistence.
- Section 7.4 formalizes the computation of derived indicators based on baseline and scenario values.

Together, these sections complete the conceptual foundation for scenario computation before introducing formulas and pipelines.

7.3. Scenario Preview vs. Scenario Persistence

When using scenario-based analysis for planning, there are usually two main ways to interact: exploratory experimentation and formal evaluation. To support both, the platform clearly separates scenario preview from scenario persistence, so the analysis stays reliable.

This separation matches different user goals and analysis needs, and it is built into the system's design.

Scenario Preview: Exploratory Analysis

Scenario preview lets users quickly and interactively test different assumptions. For example, they can adjust settings like production levels or energy sources and immediately see how these changes affect energy balances and other results.

The defining characteristics of scenario preview are:

- **Ephemeral nature**
  Previewed scenarios are not stored in the database. Results exist only for the duration of the interaction and are recalculated on demand.
- **Rapid feedback**
  Preview calculations are made for quick, repeated exploration. Users can try out several assumptions one after another without having to save any scenario.
- **Baseline dependency**
  All preview results are calculated directly from the baseline data using the transformation rules set by the user. No intermediate values are saved.

This mode helps users make sense of data and test ideas. They can explore what-if questions freely, without worrying about changing the system's main analysis.

Scenario Persistence: Formalized Scenarios

Scenario persistence is used when a user wants to save a specific configuration as an important analytical case that can be stored, retrieved, and compared later.

Persisted scenarios are different from previews in several key ways:

- **Explicit identification**
  A persisted scenario is assigned an identifier and descriptive metadata, allowing it to be referenced consistently in future analyses.
- **Reproducibility**
  The parameters defining the scenario are explicitly stored, ensuring that results can be recomputed exactly as originally described.
- **Comparability**
  Persisted scenarios can be compared against the baseline and against other scenarios using a consistent analytical framework.

Instead of saving all the results, the system only saves the scenario's definition. This avoids extra data and keeps everything traceable.

Analytical Integrity and System Design Implications

Keeping preview and persistence separate helps meet several analytical and technical goals.

First, it stops too much data from being created too soon. Without this separation, every test could create stored data, making things more complicated and using more storage.

Second, it makes the analysis clearer. Preview results are temporary, while persisted scenarios reflect deliberate analytical decisions.

Finally, this design matches how planning workflows operate. Exploratory analysis supports discussion and idea testing, while persistent scenarios are useful for documentation, reporting, and policy review

.

Position in the Scenario Computation Pipeline

In the scenario computation process:

- Preview operates as a transient computation layer applied to baseline data.
- Persistence formalizes selected transformations into reusable analytical objects.

This distinction leads to the formal definition of indicators in the next section. The calculations for indicators are the same in both preview and persisted scenarios; only the scenario's lifecycle is different.

7.4. Indicator Definitions and Computation Logic

Indicators are essential for turning raw energy data into useful information for decision-making. Absolute values show how much energy is used or produced, but indicators reveal relationships that planners and stakeholders can understand more easily. In scenario-based energy analysis, indicators help assess how different assumptions impact the system compared to the baseline.

This section defines the indicators used in the platform and explains the general computation logic that applies to baseline data, scenario previews, and persisted scenarios.

General Computation Principles

All indicators are derived from aggregated energy values computed at a consistent spatial and temporal resolution. For a given territory t and time interval τ, the following base quantities are defined:

- $C_{t,\tau}$: total final energy consumption
- $P_{t,\tau}$: total local energy production

Core Indicators

The platform implements a set of indicators commonly used in territorial energy planning. These indicators quantify energy balance, autonomy, and surplus conditions consistently and transparently.

Self-Consumption (SC)

Self-consumption measures the amount of locally produced energy used within the territory.

$$SC_{(\mathrm{t},\tau)} = min(Ct, \tau, Pt, \tau)$$

This indicator captures the effective local use of renewable production and is bounded by both demand and production capacity.

Over-Production (OP)

Over-production represents the portion of local production that exceeds local consumption.

$$OP_{(t,\tau)} = max\left(0,\ P_{t,\tau} - C_{t,\tau}\right)$$

This indicator highlights situations where energy must be exported, stored, or curtailed.

Uncovered Demand (UD)

Uncovered demand quantifies the portion of consumption that cannot be met by local production.

$$UD_{(t,\tau)} = \max\left(0,\ C_{t,\tau} - P_{t,\tau}\right)$$

This indicator is particularly relevant for assessing dependency on external energy sources.

Ratio-Based Indicators

While absolute indicators express quantities in energy units, ratio-based indicators provide normalized measures that facilitate comparison across territories and scenarios.

Self-Sufficiency Index (SSI)

The self-sufficiency index expresses the fraction of local demand covered by local production.

$$SSI_{t,\tau} = \frac{SC_{t,\tau}}{C_{t,\tau}}$$

This indicator ranges from 0 to 1 and is undefined when consumption is 0; such cases are handled explicitly by the backend logic.

Self-Consumption Index (SCI)

The self-consumption index is the fraction of local production consumed locally.

$$SCI_{t,\tau} = \frac{SC_{t,\tau}}{P_{t,\tau}}$$

This indicator provides insight into how effectively renewable production is absorbed within the territory.

Over-Production Index (OPI)

The over-production index expresses the relative magnitude of surplus production.

$$OPI_{t,\tau} = \frac{OP_{t,\tau}}{P_{t,\tau}}$$

This indicator is handy when comparing scenarios with high renewable penetration.

Consistency Across Baseline and Scenarios

A key design principle of the platform is that indicator definitions do not change between baseline data, scenario previews, and persisted scenarios. Only the underlying production values may differ.

This ensures that:

- Scenario impacts are directly comparable to baseline conditions,
- Preview results are analytically equivalent to persisted scenario results,
- indicators remain reproducible and transparent.

All indicator computations are centralized in the backend logic to prevent discrepancies between visualizations and analytical results.

7.5. Scenario Computation Pipeline

The platform uses a structured computation pipeline for scenario evaluation to keep the process transparent, reproducible, and consistent. This pipeline sets the order for handling baseline data, scenario parameters, and aggregation logic before calculating and sending indicators to the frontend. By following this process, the system avoids unplanned changes and makes sure scenario results can be compared across different areas and time periods.

Pipeline Overview

The scenario computation pipeline is made up of several clear stages, each responsible for a specific analytical task. These stages work the same way for both scenario previews and saved scenarios, but only saved scenarios have their results stored.

Here is a summary of the pipeline steps:

1. Baseline Data Retrieval
2. Scenario Parameter Application
3. Value Aggregation
4. Indicator Computation
5. Result Packaging and Delivery

The following sections explain each stage in more detail.

Baseline Data Retrieval

The pipeline begins by retrieving baseline energy valThe pipeline starts by getting baseline energy values from the database. These values come from observed or officially reported energy data and are grouped based on the area and time period the user requests.re retrieved separately;

- All values are associated with explicit territory and time dimensions.
- No scenario changes have been made yet.

This step keeps the baseline data unchanged so it can be used again for different scenario evaluations.

Scenario Parameter Application

After getting the baseline values, scenario parameters are applied only to production values. These parameters are usually percentage increases or multipliers linked to certain energy sources or analysis areas.

During this stage:

- Consumption values are preserved unchanged,
- production values matching the selected energy categories are adjusted,
- Scenario logic is applied uniformly across all affected records.

Applying scenario parameters before aggregation makes sure that changes are handled consistently across different areas and time periods.

Value Aggregation

After making scenario adjustments, the values are grouped to match the requested level of detail. This includes combining data by area, like from municipality to province, and by time, like from hourly to yearly.

Aggregation uses:

- explicit dimension hierarchies,
- consistent grouping rules,
- predefined aggregation functions.

This setup makes sure that aggregated scenario values can be directly compared to baseline values created with the same method.

Indicator Computation

Once aggregated consumption and productiAfter the consumption and production values are aggregated, indicators are calculated using the definitions from Section 7.4. All indicators come from these same aggregated values and are worked out in one central step.s are mathematically consistent,

- no indicator relies on partial or intermediate results,
- edge cases (e.g., zero consumption or production) are handled explicitly.

Indicator computation represents the final analytical transformation in the pipeline.

Result Packaging and Delivery

In the last stage, the computed values and indicators are put into a structured response and sent to the frontend. For scenario previews, results are sent directly and not saved. For saved scenarios, results can also be stored for future use and comparison.

At this stage:

- Numerical results are formatted in standard ways,
- spatial details are included only when needed,
- and metadata about the scenario setup is added to the response.

Keeping computation and delivery separate allows different frontend components to use the results in flexible ways.

Figure 7—1 Scenario computation pipeline illustrating the transformation of baseline energy data into aggregated indicators through scenario parameter application, aggregation, and result delivery.

Why This Pipeline Matters

Making the scenario computation pipeline more structured has several benefits:

- It helps decision-makers see exactly how the analysis works,
- It avoids differences between baseline and scenario results,
- It makes it easier to debug and add new features later,
- It also makes it possible to repeat scenario evaluations reliably.

When the computation logic is clear, the platform becomes a more trustworthy decision-support tool, not simply a visual interface.

8. Spatial Data Administration

Spatial information is a fundamental component of territorial energy analysis because most planning decisions have a geographic dimension. Understanding how energy is produced and consumed requires more than numerical values alone; those values must be interpreted within the spatial context of municipalities, provinces, and regions. Interactive maps provide an effective way to explore these spatial patterns and support comparisons across different territorial levels.

This chapter explains how the platform manages spatial data, starting with storing administrative boundaries in the database and ending with delivering them as GeoJSON for web visualization. It also covers how geometries are generated, simplified, and optimized before being sent to the frontend.

Even though spatial data are shown on maps, they are not handled as a separate visualization layer. Instead, spatial processing is built into the backend so that geographic features match the analytical results. This setup lets the platform provide accurate maps, process data efficiently, and stay flexible for different analysis and visualization needs.

8.1. Role of Spatial Data in Territorial Energy Analysis

Spatial data play a fundamental role in territorial energy planning by providing the geographic context required to interpret analytical results. Indicators such as energy consumption, renewable production, or self-sufficiency acquire practical meaning only when they are associated with specific territories and visualized across space.

From a user perspective, spatial interaction enables:

- comparison between neighboring territories,
- identification of regional patterns and disparities,
- exploration of results across multiple administrative scales.

From a system perspective, spatial data introduces additional requirements beyond those of purely numerical datasets. Geometries are larger and more complex, and are subject to performance constraints when transmitted to web clients. As a result, spatial data handling must be carefully designed to balance analytical accuracy, rendering performance, and system scalability.

In the platform, spatial data are used exclusively to support visualization and spatial context. Analytical computations—such as aggregation, filtering, and indicator derivation—are performed independently of geometry processing. This separation ensures that spatial complexity does not interfere with numerical correctness or computational efficiency.

At the same time, spatial data must remain tightly linked to analytical entities. Each geometry corresponds to a specific territorial unit and must be consistently aligned with the territory dimension described in Chapter 4. This alignment allows numerical results returned by the backend to be visualized accurately on maps without additional transformation logic in the frontend.

Design Principles for Spatial Data Handling

Three main principles guide the handling of spatial data in the platform:

1. **Decoupling analysis from visualization**
   Spatial geometries are not embedded in analytical tables or computation pipelines. They are retrieved only when required for map rendering.
2. **Consistency across spatial scales**
   Municipal, provincial, and regional geometries are managed using a unified approach, ensuring that map-based representations remain coherent across scales.
3. **Performance-aware delivery**
   Because web-based maps are sensitive to payload size and rendering cost, spatial data must be optimized before being sent to the client.

These principles shape the storage model, processing steps, and delivery mechanisms described in the following sections.

8.2. Geometry Storage and GeoJSON Generation

To support map-based visualization, the platform relies on explicit storage and controlled delivery of spatial geometries representing administrative boundaries. These geometries correspond to the territorial units used throughout the analytical system—municipalities, provinces, and regions—and must remain consistent with the territory dimension described in Chapter 4.

Geometry Storage Strategy

Spatial geometries are stored in the database as polygon or multipolygon objects, each associated with a unique territorial identifier. Each geometry represents the official boundary of a territorial unit and is linked to the analytical data through standardized administrative codes rather than textual names. This choice avoids ambiguity and ensures stable joins between spatial and analytical components, even when naming conventions differ across datasets.

Geometries are stored independently from analytical fact data. This separation is intentional: spatial objects are significantly heavier than numerical values and are not required for most analytical operations. By isolating geometries in dedicated structures, the system minimizes memory usage and avoids unnecessary overhead during non-spatial queries.

This design also allows spatial data to evolve independently. Updated boundary definitions or alternative geometry sources can be introduced without affecting the integrity of the analytical schema or historical energy values.

GeoJSON as a Delivery Format

For communication with the frontend, spatial geometries are delivered in GeoJSON format. Modern web mapping libraries widely support GeoJSON and provides a convenient, standardized representation for polygonal features and their associated properties.

When a spatial request is received, for example, to render a choropleth map, the backend generates a GeoJSON Feature Collection in which:

- each feature represents a territorial unit,
- geometry coordinates describe the boundary shape,
- properties include the territorial identifier required to associate analytical values.

This approach allows the frontend to remain generic and visualization-focused. The frontend does not need to interpret database-specific spatial formats or perform geometry transformations; it simply consumes GeoJSON objects and renders them on the map.

On-Demand Geometry Generation

Rather than precomputing and storing static GeoJSON files, geometries are generated on demand. This strategy provides flexibility in selecting spatial resolution, territorial level, and filtering criteria. For example, a request for municipal-level visualization returns a different geometry set than a request at the regional level.

On-demand generation also ensures that spatial data delivery remains synchronized with analytical queries. When users switch spatial scale or apply filters, the backend can return exactly the geometries required for the current analytical context, without transmitting unnecessary spatial features.

Payload Control and Efficiency

Because spatial geometries can be large, especially at acceptable spatial resolution, careful control over payload size is essential. Geometry generation, therefore, focuses on returning only the minimum information required for visualization:

coordinates defining the shape,

identifiers needed to join analytical values.

Additional attributes unrelated to visualization are excluded from spatial responses. This selective delivery reduces bandwidth usage and improves browser rendering performance.

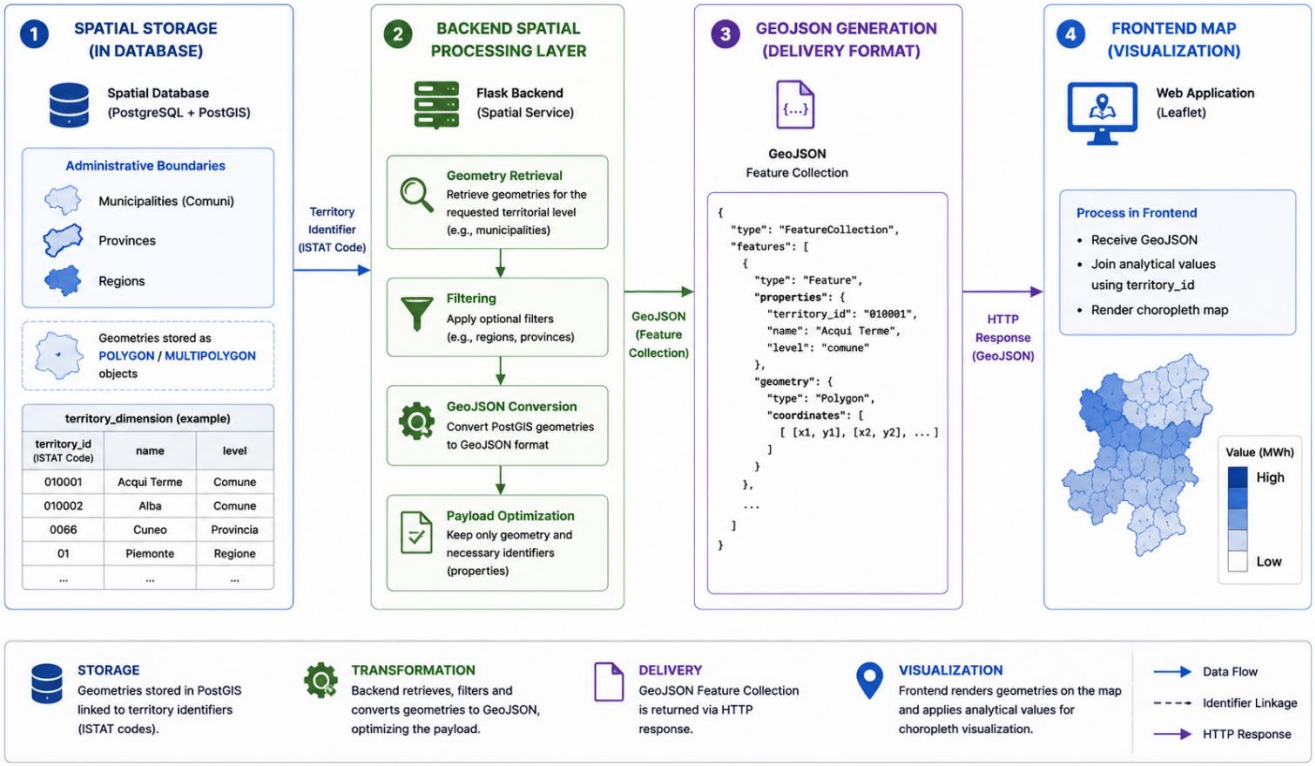

Figure 8—1 Geometry storage and GeoJSON generation workflow, illustrating how spatial geometries linked to territorial identifiers are transformed into GeoJSON and delivered to the frontend mapping component.

8.3. Geometry Simplification and Performance Optimization

Interactive web maps require backend systems to perform well, especially when delivering spatial data at a useful level of detail. Municipal boundaries often have many points, which can make files larger and slow down how quickly maps load in web browsers.

To address these issues, the platform uses a geometry simplification method that keeps maps looking accurate while making them more responsive.

Motivation for Geometry Simplification

Full-resolution geometries are essential for accurate spatial representation and analysis, but they are rarely necessary for interactive visualization. When rendering a national or regional overview, the fine-grained detail of municipal boundaries provides little additional value to users while imposing a substantial computational cost.

If such geometries were transmitted without modification:

- Network latency would increase due to large payload sizes,
- browser rendering would become sluggish,
- User interaction (zooming, panning, filtering) would be negatively affected.

For these reasons, geometry simplification is treated as a presentation-level optimization, rather than a modification of the underlying spatial data.

Simplification Strateg

Geometry simplification is applied during the data delivery phase, after geometries have been selected but before they are serialized as GeoJSON. The simplification process reduces the number of vertices while preserving the overall shape and topological structure of territorial boundaries.

The degree of simplification is chosen according to the spatial scale and analytical context:

- Regional and provincial views apply stronger simplification, as fine boundary details are not visually distinguishable.
- Municipal views use lighter simplification to preserve recognizable shapes while still improving performance.

Bydynamically simplifying, the backend avoids storing multiple geometry variants and maintains a single authoritative geometry source.

Caching and Reuse

To further improve performance, simplified geometries can be cached at the backend level. Because administrative boundaries are static over time, the same simplified geometries are frequently requested across multiple sessions and users.

Caching allows the system to:

- Avoid repeated simplification computations,
- Reduce database access,
- Deliver spatial data with minimal latency.

This approach is efficient for commonly used spatial levels, such as national or regional views, which are repeatedly requested during exploration analysis.

Trade-offs and Limitations

Geometry simplification helps improve performance, but it does cause a small, controlled loss of detail. Some small features, thin boundaries, or irregular shapes might get smoothed out or removed. This trade-off is fine for visualization because all analytical calculations still use the full-resolution data.

Simplified geometries are only used for visualization and **never** for analytical calculations or spatial measurements. This keeps the visual display efficient while making sure the accuracy of the analysis is not affected.

Role in the Overall Architecture

Geometry simplification complements the layered architecture described in Chapter 3 and the GeoJSON delivery strategy introduced in Section 8.2. By isolating simplification logic within the spatial delivery layer, the system preserves a clear separation between:

- analytical data processing,
- spatial data management,
- frontend visualization.

This separation reinforces maintainability and allows future enhancements—such as multi-resolution geometry support or tile-based rendering—to be introduced without redesigning the database schema.

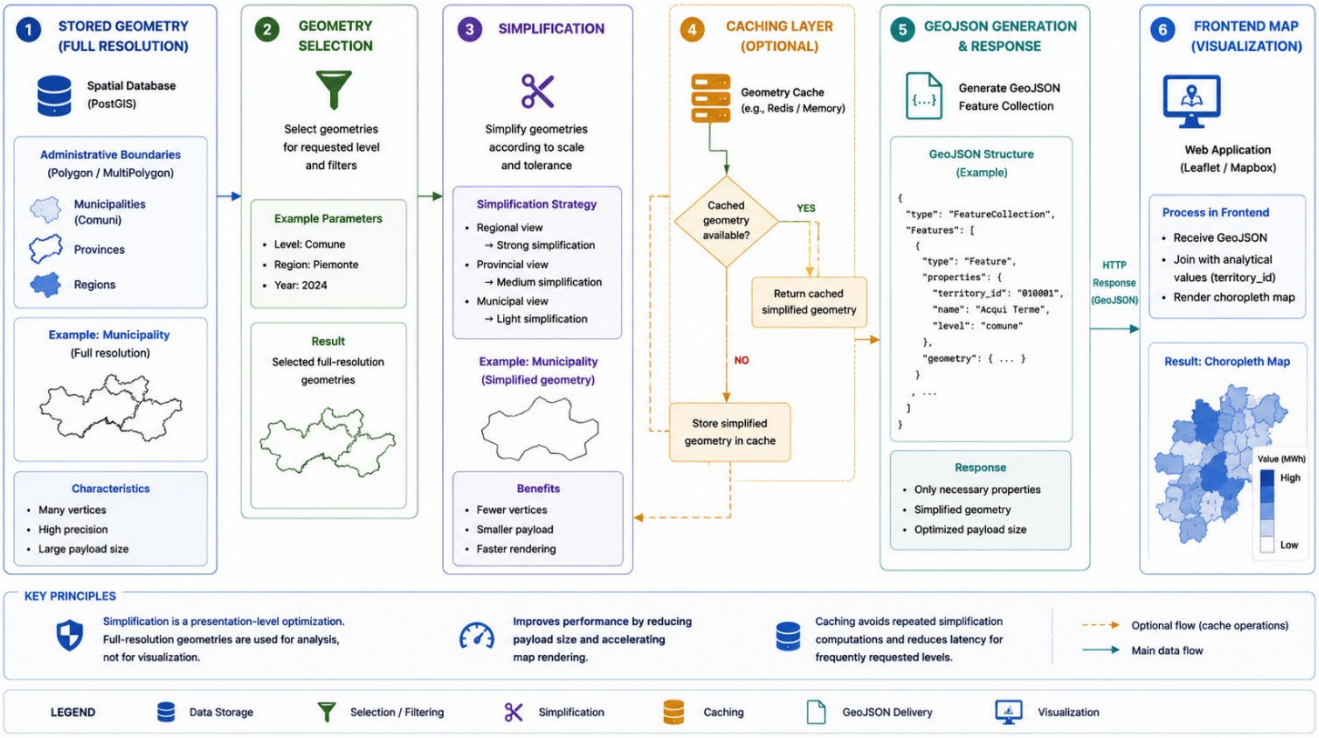

Figure 8—2 Geometry processing and optimization workflow illustrating the separation between full-resolution spatial storage, presentation-oriented simplification, cache reuse, and GeoJSON generation for web-based visualization.

9. Discussion, Limitations, and Future Work

This chapter reflects on the design and implementation choices presented throughout the thesis and critically discusses their strengths and limitations. While the developed platform demonstrates the feasibility and value of a multidimensional, backend-driven approach to territorial energy analysis, it also reveals constraints inherent to the current scope and implementation. The chapter concludes by outlining potential directions for future work that could extend the platform's analytical capabilities, performance, and applicability.

9.1. Discussion of Design Choices

The main design choice in this thesis was to focus on territorial energy analysis as a backend analytical task instead of a visualization issue. This view shaped later decisions, such as using a multidimensional database, analytical API endpoints, and clear models for scenarios and indicators.

Organizing energy data with a central fact table and clear dimensions gives the platform strong analytical flexibility. It allows consistent aggregation across different spaces, times, and energy types without repeating logic or changing the data structure. This method works well for scenario analysis, where both baseline and changed values need to be compared clearly.

In the same way, providing analytical features like choropleth values, time series, and scenario previews through specific API endpoints instead of direct database access separated the frontend from the backend data structures. This made the system easier to maintain and allowed analytical rules to be managed in one place.

Overall, the final architecture strikes a balance between being general enough for different analytical uses and specific enough to meet the needs of territorial energy planning.

9.2. Limitations of the Current Implementation

Despite these strengths, several limitations must be acknowledged.

Spatial Data Handling Constraints

Although spatial geometries are consistently managed and delivered efficiently via GeoJSON, the current implementation relies on simplified polygon geometries for visualization. While this approach is appropriate for interactive dashboards, it imposes limitations on geometric precision

and topological accuracy. Complex spatial operations—such as adjacency analysis or spatial overlays—are outside the scope of the current system.

In addition, the coexistence of analytical data and spatial data introduces ambiguity regarding the appropriate location for certain operations. Some spatial transformations could be performed at the database level, while others are handled during API response preparation. This mixed approach, while pragmatic, could benefit from more apparent separation in future iterations.

Performance and Scalability Considerations

The platform was designed to support exploratory analysis rather than high-frequency transactional workloads. While query performance is acceptable for typical dashboard interactions, the current implementation does not employ advanced optimization techniques such as materialized views, distributed caching, or query parallelization.

As dataset size grows—either through finer temporal resolution, additional years, or expanded geographic coverage—query execution times may increase. Without further optimization, this could limit responsiveness under heavier analytical loads.

Scenario Modeling Scope

Scenario handling in the current system focuses primarily on production-side modifications, with consumption values treated as fixed. This assumption reflects standard planning practices but limits the expressiveness of scenarios involving behavioral change, demand-side management, or policy-driven consumption reductions.

Furthermore, scenarios are modeled as parameterized transformations rather than full-fledged simulation models. While this design supports transparency and reproducibility, it does not capture dynamic feedback effects or interactions between multiple indicators over time.

9.3. Future Work and Extensions

Several directions for future development emerge naturally from these limitations.

Database and Performance Enhancements

At the database level, future work could introduce:

- targeted indexing strategies for frequently queried dimensions,
- materialized views for commonly used aggregations,
- caching layers for repeated analytical requests.

These improvements would enhance performance without altering the conceptual data model.

Advanced Scenario Families

The scenario framework could be extended to support:

- demand-side scenarios affecting consumption values,
- combined scenarios involving both production and consumption changes,
- Scenario families with inheritance and versioning.

Such extensions would allow the platform to support more sophisticated planning exercises while preserving the distinction between baseline data and scenario-derived values.

Spatial and Analytical Integration

Future versions of the platform could explore tighter integration between spatial analysis and energy indicators. This might include:

- spatial clustering of municipalities based on energy profiles,
- proximity-based analysis for infrastructure planning,
- integration of external spatial datasets, such as land-use or environmental constraints.

These features would require more advanced spatial processing capabilities but would significantly expand the platform's analytical scope.

Interoperability and Governance

Finally, the platform could be extended to support multi-user environments, authentication mechanisms, and role-based access control. This would enable its use in institutional contexts where multiple stakeholders interact with shared datasets and scenarios.

9.4. Concluding Remarks

The discussion presented in this chapter highlights that the developed platform should be understood as a foundational analytical system, rather than a finalized decision-support product. Its primary contribution lies in demonstrating how careful backend design—grounded in multidimensional modeling and explicit analytical logic—can support transparent, flexible, and reproducible territorial energy analysis.

The identified limitations do not undermine this contribution; instead, they point to clear, achievable paths for future enhancement. In this sense, the platform provides both a working tool and a structured framework for building more advanced energy planning applications.

The final chapter summarizes the thesis's overall contributions and reflects on their relevance for territorial planners and public administrations.

10. Conclusions

This thesis presented the design and implementation of a backend-oriented platform for territorial energy analysis, with a specific focus on flexibility, transparency, and analytical consistency across spatial and temporal scales. The work was motivated by the growing need for data-driven tools that can support public administrations and planners in understanding energy systems and evaluating alternative development scenarios.

Rather than approaching the problem primarily from a visualization or frontend perspective, the thesis framed territorial energy planning as an analytical challenge on the backend. This perspective guided the adoption of a multidimensional data model, the design of analytical APIs, and the explicit handling of scenarios and derived indicators within the backend architecture.

10.1. Summary of Achievements

The main contribution of this work is the development of a coherent, extensible backend architecture that supports multi-scale energy analysis. Key achievements include:

- The design of a multidimensional database schema that enables consistent aggregation across municipalities, provinces, and regions, as well as across multiple temporal resolutions.
- The implementation of a generic fact-based data model that avoids domain-specific table proliferation while preserving analytical clarity through explicit dimensions.
- The definition of a scenario computation framework that distinguishes baseline data from user-defined scenarios and supports both preview and persistence workflows.
- The formalization and backend-level implementation of energy planning indicators, ensuring transparency and reproducibility of analytical results.
- The exposure of analytical concepts through domain-oriented API endpoints, decoupling frontend visualization from internal data structures and computation logic.
- The integration of spatial data handling strategies that balance performance and usability for interactive map-based visualization.

Together, these elements demonstrate how careful backend design can provide a robust foundation for interactive energy planning platforms.

10.2. Relevance for Territorial Energy Planning

The approach presented in this thesis is particularly relevant for territorial planners and public administrations operating at local and regional levels. Energy planning increasingly requires the ability to dynamically explore data, compare scenarios, and communicate results clearly to diverse stakeholders.

By centralizing analytical logic in the backend and making assumptions explicit through the data model, the platform supports:

- consistent interpretation of energy indicators,
- transparent comparison between baseline and scenario outcomes,
- reproducible analytical workflows.

These characteristics are essential in institutional contexts where analytical results inform policy decisions, planning documents, and public communication.

10.3. Lessons Learned

Several lessons emerged during the platform's development. First, early investment in data modeling pays off significantly as analytical complexity grows. While a simple relational schema may suffice for limited use cases, it quickly becomes a bottleneck when multiple spatial scales, temporal resolutions, and scenarios are introduced.

Second, separating analytical concerns from visualization concerns improves both maintainability and conceptual clarity. Treating the backend as an analytical engine rather than a data delivery service enables more robust validation, logic reuse, and long-term evolution.

Finally, explicit modeling of assumptions, whether related to scenarios, indicators, or aggregation rules, reduces ambiguity and makes the system easier to reason about, extend, and critique.

10.4. Final Remarks

This thesis does not aim to present a finished decision-support product, but rather a structured and extensible foundation for territorial energy analysis. The developed platform demonstrates that a backend-driven, multidimensional approach can effectively support interactive exploration, scenario evaluation, and multi-scale comparison of energy data.

The work contributes both a practical implementation and a methodological reference for designing analytical backends in the context of energy planning. It is hoped that the ideas and structures presented here inform future research and development, supporting more transparent, data-driven, and scalable energy planning practices.

Closing Note

With this chapter, the thesis concludes the presentation of the platform's design, implementation, and implications. The appendices that follow provide supporting technical material, including API summaries, database schema listings, and example queries, to complement the conceptual discussion presented in the main chapters.

References

Amundsen, M. (2013). *RESTful Web APIs*. Sebastopol, CA: O'Reilly Media.

Ardebili, A., A., Z., M., R., & A.I.H.A. (2024, October 01). Digital Twins of smart energy systems: a systematic literature review on enablers, design, management and computational challenges. *Energy Informatics, 7*(1). doi:https://doi.org/10.1186/s42162-024-00385-5

Butler, H. D. (2016, August). *The GeoJSON Format*. Internet Engineering Task Force (IETF). doi:10.17487/RFC7946

Cavalheiro, J., & Carreira, P. (2016). A multidimensional data model design for building energy management. *30*(4), pp. 619-632. Retrieved from https://www.sciencedirect.com/science/article/pii/S1474034616301227

Connolly, D., Lund, H., Mathiesen, B. V., & Leahy, M. (2010). A Review of Computer Tools for Analysing the Integration of Renewable Energy into Various Energy Systems. *Applied Energy, 87*(4), 1059–1082. doi:doi.org/10.1016/j.apenergy.2009.09.026

Crudu, V. (2025, April). Flask and RESTful Design Patterns - A Practical Overview for Developers. Retrieved from https://moldstud.com/articles/p-flask-and-restful-design-patterns-a-practical-overview-for-developers

Di Paolo, L., Di Martino, A., Di Battista, D., Carapellucci, R., & Cipollone, R. (2023). The potential of energy planning at Municipality scale: Sustainable Energy and Climate Action Plans (SECAP) and local Energy Communities to meet the energy demand variability.

ESPON. (2018). *Territorial Governance and Spatial Planning*. ESPON EGTC, Luxembourg.

European Commission, J. R. (2018). *Guidelines for the Development of Sustainable Energy and Climate Action Plans (SECAP)*. Luxembourg: Publications Office of the European Union. Retrieved from https://www.istat.it/en/non-categorizzato/physical-energy-flows-pefa/

Few, S. (2006). *Information Dashboard Design: The Effective Visual Communication of Data*. Sebastopol, CA: O'Reilly Media.

Frontend, S. o. (2012). *Software Architecture in Practice* (3 ed.). Boston, MA: Addison-Wesley.

Golfarelli, M., & Rizzi, S. (2009). *Data Warehouse Design: Modern Principles and Methodologies*. New York: McGraw-Hill.

H. Butler, M. D. (2016, August). *The GeoJSON Format*. IETF. IETF. Retrieved from https://datatracker.ietf.org/doc/html/rfc7946

Haklay, M. (2008). OpenStreetMap: User-Generated Street Maps. *IEEE Pervasive Computing, 7*(4), 12–18. doi:10.1109/MPRV.2008.80

Hyndman, R. J., & Athanasopoulos, G. (2021). *Forecasting: Principles and Practice* (3 ed.). Melbourne: OTexts.

Inmon, W. H. (2005). *Building the Data Warehouse* (4 ed.). Indianapolis, IN: Wiley.

Keirstead, J., Jennings, M., & Sivakumar, A. (2012). A Review of Urban Energy System Models: Approaches, Challenges and Opportunities. *Renewable and Sustainable Energy Reviews, 16*(6), 3847–3866. doi:10.1016/j.rser.2012.02.047

Kimball, R., & Ross, M. (2013). *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. Hoboken, NJ: Wiley.

Kimball, R., & Ross, M. (2013). *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling* (3 ed.). Indianapolis, IN: Wiley.

Longley, P. A., Goodchild, M. F., Maguire, D. J., & Rhind, D. W. (2015). *Geographic Information Science and Systems* (4 ed.). Hoboken, NJ: Wiley.

Lorenzo Bottaccioli, P. R. (2025). Development of a Scalable Modular Backend for an Urban Energy Co-simulation Platform. Turin, Italy. Retrieved from https://webthesis.biblio.polito.it/id/eprint/34441

Malczewski, J. (1999). *GIS and Multicriteria Decision Analysis*. New York: Wiley.

Marks, G. (1993). Structural Policy and Multilevel Governance in the EC. In A. Cafruny, & G. Rosenthal (Eds.), *The State of the European Community: The Maastricht Debates and Beyond*. Boulder, CO: Lynne Rienner Publishers.

McMaster, R. B., & Shea, K. S. (1992). *Generalization in Digital Cartography*. Washington, DC: Association of American Geographers.

Morrison, R. (2018, April). Energy system modeling: Public transparency, scientific reproducibility, and open development. pp. 49-63. Retrieved from https://www.sciencedirect.com/science/article/pii/S2211467X17300949

Mutani, G., Morando, V., & Zhou, X. (2024, December). An Italian Geoportal for Renewable Energy Communities. *JOURNAL OF SUSTAINABILITY FOR ENERGY*(2958-1907). doi:10.56578/jse030404

Nielsen, G. F., & Schiemann, J. (2018). *Spatiotemporal modelling for integrated spatial and energy planning*. Springer.

Obe, R. O., & Hsu, L. S. (2020). *PostGIS in Action* (3 ed.). Shelter Island, NY: Manning Publications.

Richardson, L., & Ruby, S. (2007). *RESTful Web Services*. Sebastopol, CA: O'Reilly Media.

Scaramuzzino, C., Garegnani, G., & Zambelli, P. (2019, November). Integrated approach for the identification of spatial patterns related to renewable energy potential in European territories. doi:https://doi.org/10.1016/j.rser.2018.10.024

Sugumaran, R., & DeGroote, J. (2011). *Spatial Decision Support Systems: Principles and Practices*. Boca Raton, FL: CRC Press.

Thomas, J. J., & Cook, K. A. (2005). *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. Los Alamitos, CA: IEEE Computer Society.

Valdes, J. W. (2020). A framework for regional smart energy planning using volunteered geographic information. Retrieved from https://doi.org/10.5194/adgeo-54-179-2020