HORIZON-CL5-2022-D4-02

EUROPEAN COMMISSION

European Climate, Infrastructure and Environment Executive Agency

Grant agreement no. 101123238

VELIXIA

BUILDINGS AS ACTIVE UTILITY NODES

Smart Grid-Efficient Interactive Buildings

Deliverable D4.7

Integrated EVELIXIA B2G and G2B Services Layer and Orchestration

Project acronym	EVELIXIA
Full title	Smart Grid-Efficient Interactive Buildings
Grant agreement number	101123238
Topic identifier	HORIZON-CL5-2022-D4-02-04
Call	HORIZON-CL5-2022-D4-02
Funding scheme	HORIZON Innovation Actions
Project duration	48 months (1 October 2023 – 30 September 2027)
Coordinator	ETHNIKO KENTRO EREVNAS KAI TECHNOLOGIKIS ANAPTYXIS (CERTH)
Consortium partners	CERTH, RINA-C, CEA, CIRCE, UBE, HAEE, IESRD, UNIGE, SOLVUS, R2M, EI-JKU, FHB, EEE, EG, ÖE, PINK, TUCN, DEER, TN, ENTECH, SDEF, EGC, KB, AF, Sustain, NEOGRID, MPODOSAKEIO, DHCP, HEDNO, BER, MEISA, ITG, NTTDATA, TUAS, NEOY, HES-SO
Website	https://www.evelixia-project.eu/
Cordis	https://cordis.europa.eu/project/id/101123238

VELIXIA
BUILDINGS AS ACTIVE UTILITY NODES

Funded by the European Union

Disclaimer

Copyright Message

ACKNOWLEDGMENT

This project has received funding from the European Union's Horizon Europe Framework Programme for Research and Innovation under grant agreement no 101123238. ***Disclaimer:*** *The European Commission is not responsible for any use made of the information contained herein. The content does not necessarily reflect the opinion of the European Commission.*

VELIXIA
BUILDINGS AS ACTIVE UTILITY NODES

Funded by the European Union

Deliverable D4.7

Integrated EVELIXIA B2G and G2B Services Layer and Orchestration

Deliverable number	D4.7
Deliverable name	Integrated EVELIXIA B2G and G2B Services Layer and Orchestration
Lead beneficiary	UBE
Description	This deliverable is directly linked to the activities foreseen in Task 4.6, consolidating all foreseen technical developments on EVELIXIA's Services Layer Integration and Orchestration. The finalized version of this report will be submitted on M33 with D4.8.
WP	4
Related task(s)	T4.6
Type	Document, Report
Dissemination level	Public
Delivery date	24.04.2025.
Main author	Eleni Kotali (UBE)
Contributors	Stavros Koltsios (CERTH)

Document history

Version	Date	Changes	Author
V1- fisrt draft	28.02.2025		UBE
V1 – Review	01.03.2025		CERTH
V1- consolidated version	11.03.2025		UBE
V2 – review	14.02.2025		SOLVUS
V2-consolidated version	25.03.2025		UBE
Final Version	16.04.2025		UBE
Final deliverable submission	24.04.2025		CERTH

EXECUTIVE SUMMARY

This deliverable presents the design, development, and implementation of the EVELIXIA **Service Layer**, the core software infrastructure enabling the coordination of **Building-to-Grid (B2G)** and **Grid-to-Building (G2B)** services within the EVELIXIA ecosystem. The Service Layer forms the central integration and execution backbone of the platform, interconnecting heterogeneous actors, domain-specific Innovative Solutions (IS), and various data sources to deliver intelligent, data-driven energy services for buildings and grids across multiple pilot sites.

The aim of the Service Layer is to enable automated, scheduled, or on-demand workflows that orchestrate complex energy simulations, forecasting tools, optimization algorithms, and user-facing analytics, all within a microservices-based, modular, and scalable architecture. This infrastructure has been developed with interoperability, replicability, and extensibility in mind, serving both as a technical framework and an execution environment for the full suite of EVELIXIA services.

The implementation follows an event-driven architecture based on Apache Kafka as the messaging backbone, dockerized IS modules as stateless processing units, and a custom-built Orchestration Layer that governs workflow logic and execution. A MongoDB database is used for structured storage of metadata, results, and workflow states, while an API Gateway manages secure communication between external clients and internal services.

This MVP version of the Service Layer integrates and validates a set of representative services, including both B2G and G2B examples, covering forecasting, optimization, investment planning, and control signal generation. These include:

- **S1 – Building Optimization Services**: A daily chain of forecasting, control action generation, and building-level energy recommendations using modules IS1, IS3, IS4, IS5, IS9, and IS10.
- **S2 – Building Investment Planning**: On-demand tools for upgrade recommendations (IS7), KPI analysis (IS6), and scenario evaluation using both static and real-time building data.

- **S3 – Equipment Maintenance**: Monitoring of energy equipment health using IoT data and predictive diagnostics (IS2).
- **S4 – Grid Services**: Including Grid Congestion Management (S4.1), Portfolio Management (S4.2), and Peer-to-Peer Trading (S4.3), all leveraging simulation, optimization, and economic assessment tools (IS15, IS12, IS13).
- **S5 – Grid Investment Planning**: A G2B scenario assessing grid upgrade options and their financial/technical outcomes using IS11 and IS15.
- **S6 – Grid Maintenance**: Predictive scheduling of grid infrastructure maintenance tasks based on simulated grid performance and aging models (IS14).

Each service is defined through a standardized execution workflow, which includes user input ingestion, data retrieval (either from the Middleware or static repositories), triggering of relevant IS tools, publishing of intermediate and final results via Kafka topics, and aggregation for presentation on the Stakeholder Platform.

Notably, this deliverable also introduces:

- A service-oriented topic naming convention to manage multiple pilots and isolate data contextually.
- A structured JSON message format for consistent communication and traceability across all modules.
- A configurable YAML-based orchestration logic, enabling dynamic workflow configuration per service and pilot.
- Scalable architectural patterns that are already being prepared for transition into production deployments.

This deliverable serves as the first milestone toward the realization of the EVELIXIA Service Layer as a modular and interoperable digital backbone for coordinated B2G and G2B service execution. Building upon the results of this MVP implementation, future development efforts (towards M33) should prioritize the seamless integration with the project's Middleware layer and Stakeholder Platform, ensuring a unified data flow and user interaction experience across all services. In parallel, emphasis should be placed on consolidating heterogeneous data inputs, aligning multiple interconnected innovative solutions, and maintaining reliable orchestration within varied operational environments to deliver scalable, field-ready deployments.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION AND OBJECTIVES

1.1 Scope and Objectives

To detransition from isolated Energy Systems and to effectively provide Buildings as Active Utility Nodes (BAUN), EVELIXIA needs to develop sophisticated ways of interconnecting the vast number of components and services across all three Layers (Field, Middleware, Service) of its Platform, all while ensuring interoperability and reducing the computational burden.

D4.7, which is directly related to T4.6 (Integrated B2G and G2B Services Layer), focuses on the technical methodologies and architectural framework employed for the integration and orchestration of the Service Layer and aims to provide the blueprint for the dynamic coordination of its diverse Innovative Solutions (IS).

The core objectives addressed within this deliverable can be summarized below:

- **Integration of Heterogeneous Services:** Establish a standardized, scalable, and resilient integration framework capable of seamlessly connecting diverse analytical and predictive tools, each developed and provided independently by various stakeholders within Docker containers.
- **Dynamic Orchestration of Workflows:** Design and implement a flexible orchestration mechanism that dynamically manages both scheduled (continuous) and user-initiated (on-demand) workflows, effectively coordinating interactions among IS tools to deliver energy services such as forecasts, optimizations, analytics, and real-time decision support.
- **Event-Driven Communication:** Adopt an event-driven architecture using Apache Kafka as the central message broker, enabling loose coupling between services, improving system resilience, and facilitating future expansions and scalability without disruption to existing services.
- **Persistent and Accessible Data Management:** Provide a robust and flexible storage solution, facilitating long-term persistence of analytical outcomes and enabling easy data retrieval to stakeholders through well-defined API endpoints.
- **User-Friendly Service Access:** Offer user and external stakeholder access to the integrated service ecosystem, employing an API Gateway pattern to mediate interactions, manage authentication, authorization, and effectively shield the internal service infrastructure.

To make the document concise and avoid overlaps with other tasks and deliverables, D4.7 explicitly excludes:

- In-depth technical implementation or source code-level detail of individual ISs.
- User-interface designs or front-end application development considerations beyond interactions with the API Gateway.
- Detailed operational or financial evaluations of the energy services themselves, except as directly relevant to integration and orchestration challenges.
- Detailed security aspects and mechanisms, like authentication, privacy and user/identity management.

1.2 Structure of this Deliverable

This deliverable has been structured to comprehensively address the key architectural and implementation aspects related to the integration and orchestration of EVELIXIA's Service Software Layer, starting with the theoretical principles, to the details of the core components that were developed, until the specifics of the first version of the implementation. These are organized in the following sections:

- **Section 1 (Introduction and Objectives)**: Contextual overview, clearly defining the scope, purpose, and main objectives of this document. It also clarifies how this deliverable interacts and aligns with related tasks and previously submitted or upcoming project deliverables.
- **Section 2 (Service Layer Architectural Overview):** High-level view of the chosen integration and orchestration architecture, outlining essential architectural principles and the overall approach.
- **Section 3 (Service Layer Components)**: Descriptions of each component within the architecture. Components such as the API Gateway, Orchestration Layer, Message Brokerage System, Service Layer Database, Security Layer, and specific Innovative Solutions are discussed, including their functions, internal operations, and interdependencies.
- **Section 4 (Architectural Workflow Patterns)**: Details regarding the operational workflows enabled by the architecture, distinguishing clearly between Scheduled Continuous Workflows and On-Demand (User-

Triggered) Workflows. Additionally, this section introduces the key Building-to-Grid (B2G) and Grid-to-Building (G2B) services provided by EVELIXIA, establishing their relevance and roles within the Service Layer.

- **Section 5 (MVP Implementation)**: Addresses the practical deployment of the Minimum Viable Product (MVP). It outlines chosen technology stacks, details on the Docker Compose setup and configuration, detailing the execution on both on-demand and scheduled workflows.
- **Section 6 (Next Steps)**: Future directions to achieve functional integration, scaling and interoperability, preparing for the v2 of this deliverable.
- **Section 7 (Conclusion)**: Summary of the achieved objectives, key architectural benefits, and the overall suitability of the described integration and orchestration approach.

1.3 Interactions with other Tasks and Deliverables

D4.7 sits at the center of the other tasks and deliverables of WP4, providing the orchestration blueprint for the Service Layer. The deliverable is also closely related to the equivalent WP3 deliverable, D3.7 (EVELIXIA platform external communication and common information management), which similarly details the integration components of the second EVELIXIA Layer, the Middleware.
Additionally, D4.7 adheres to the architectural principles detailed in D1.7 (Platform Architecture and integration roadmap). Last but not least, D4.7 will also help guide the holistic integration efforts of the whole EVELIXIA Ecosystem that will be developed in T5.1 and its corresponding deliverable D5.1 (Integrated Holistic EVELIXIA Platform).

2 SERVICE LAYER ARCHITECTURAL OVERVIEW

This chapter focuses on providing the key architectural requirements of the Service Layer to be inline with the original EVELIXIA principles detailed in the DoA and in D1.7 (Platform Architecture and integration roadmap). The role of the Service Layer within EVELIXIA will be detailed first, followed by the design principles we implemented and finally closing with the architectural overview of the Layer and its main components.

2.1 Service Layer Role within the EVELIXIA Architecture

The Service (or Application) Layer is positioned between Middleware Layer and the User Interface (UI) of the EVELIXIA Ecosystem, and its primary role is to enable all the data driven services of the Platform, ensure sophisticated coordination and data exchange between the ISs and deliver analytical results and business value to the end users through EVELIXIA's Stakeholder Platform (Frontend). At a high-level view, the multi-tiered architecture is structured as followed:

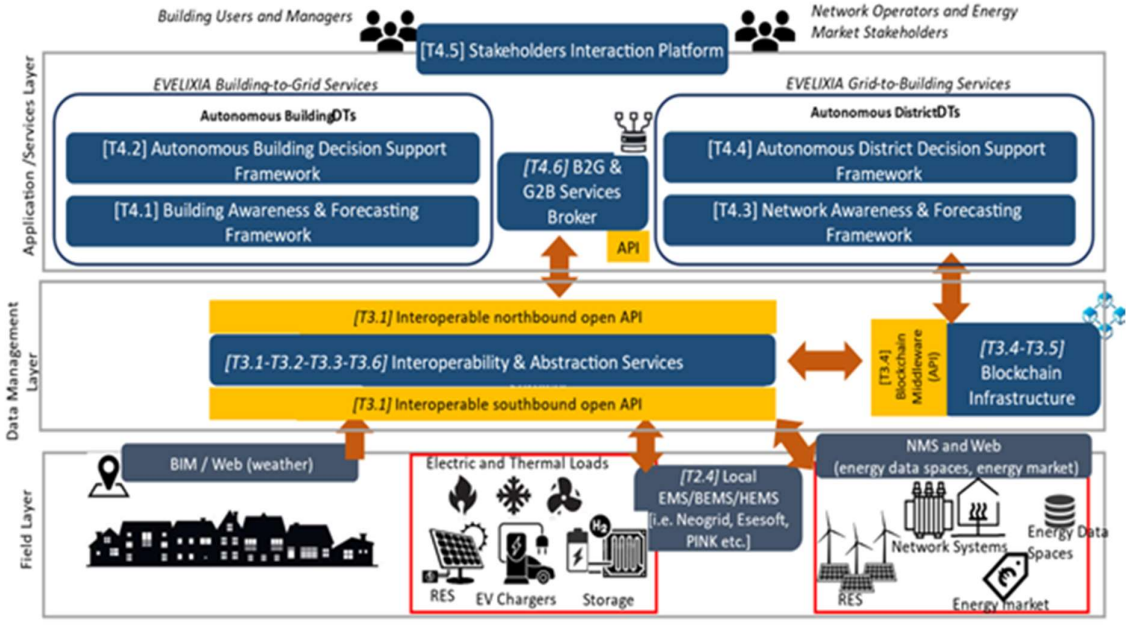


Figure 1 - EVELIXIA High Level Architecture

- **Field Layer**: Contains the equipment as well as the digital assets deployed across the different demonstration sites (referred to as Pilot Sites). This layer

generates raw and contextual data collected through various off-the-shelf smart controllers, IoT sensors, devices, and site-specific IT systems.

- **Middleware Layer**: Serves as the platform's Data Management and Interoperability layer. This Layer is responsible for filtering, aggregating, contextualizing, and securing data received from the Field Layer. It handles external integration with the Field Layer via the Southbound API as well as expose relevant data to the Service Layer through the Northbound API. In addition to data management, this layer also incorporates the platform's blockchain infrastructure.
- **Service Layer**: The focus of this deliverable, the Service Layer acts as the platform's main data processing and analytical layer. It integrates a heterogenous IS toolset, providing tailored made analytics, simulation, and forecasting capabilities, based on the semantically harmonized data provided by the Middleware. The Service Layer enables all of its software solutions to work together coherently through service workflows, where data, whether originating from the Middleware or generated by other ISs, is consumed, processed, and re-published as intermediate or final results.

The most important role of the Service Layer is to coordinate and provide the data-driven Services foreseen by EVELIXIA, which fall primarily into two categories:

- **Building-to-Grid (B2G) Services**
- **Grid-to-Building (G2B) Services**

B2G services aim to enable buildings to actively contribute to grid operations as Active Utility Nodes (BAUNs), rather than merely reacting to grid signals. Buildings and aggregators can then act as flexibility providers, offering services to the grid such as demand shifting, building operation optimization, and participation in market mechanisms.

The Service Layer supports B2G services through the **Building Awareness and Forecasting Toolbox (BAFT)** and the **Autonomous Building Decision Support Toolbox (ABDST)**, developed as part of EVELIXIA's Service Layer in T4.1 and T4.2 respectively. These toolboxes integrate:

- Advanced building-level forecasting, demand planning, and flexibility assessment capabilities.
- Real-time operational control, supporting both on-demand and automated actions by building systems.

- Decision-support mechanisms for investment planning and long-term energy optimization.

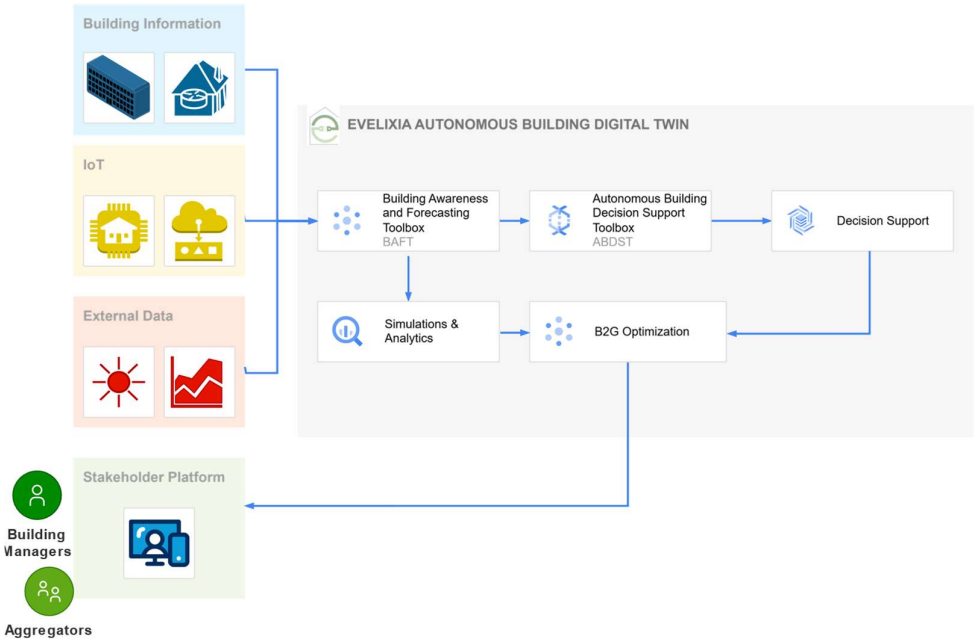


Figure 2 - EVELIXIA Autonomous Building Digital Twin

Furthermore, B2G services within the Service Layer enable:

- Demand flexibility provision to the grid, both implicit (price-based) and explicit (market-based or operator-triggered).
- Distributed optimization of building operations considering grid requirements.
- The aggregation of building contributions to provide coordinated services to Distribution System Operators (DSOs), aggregators, or other grid stakeholders.

In parallel, the Service Layer also supports a variety of Grid-to-Building (G2B) services, where information, forecasts, or control signals are provided from grid-level stakeholders to buildings. These services aim to optimize the operations of multiple buildings while considering network conditions, enhancing both energy efficiency and grid stability.

In the EVELIXIA context, G2B services are primarily supported through the development of the Autonomous District Digital Twin (ADDT), which integrates two key frameworks:

- The **Network Awareness and Forecasting Framework (NAFF)**, which provides simulations, forecasts, and network profiling at the district level, developed at T4.3.
- The **Autonomous District Decision Support Framework (ANDSF)**, which leverages NAFF outputs to generate decision-support services aimed at grid operators, aggregators, and other stakeholders, developed at T4.4.

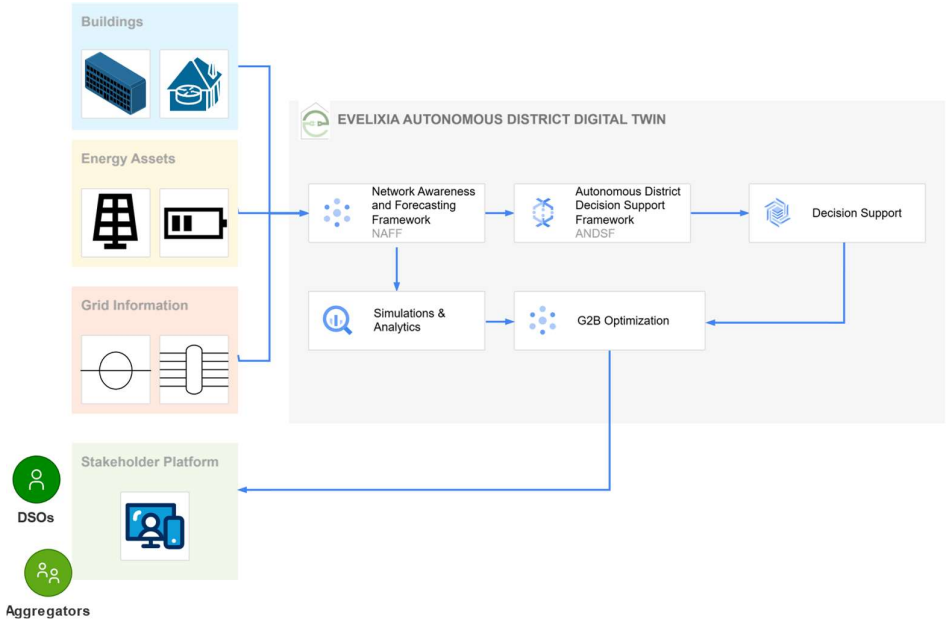


Figure 3 - EVELIXIA Autonomous District Digital Twin Overview

Through these frameworks, the Service Layer enables:

- Simulation-based grid operation planning, supporting services such as Grid Investment Planning, Multi-Vector Network Management, and Aggregated Demand Portfolio Management.
- The provision of dynamic signals, such as flexibility requests, forecasts, or constraints to buildings or aggregators.
- The optimization of grid-aware building operations, enabling buildings to respond optimally to grid needs.

On the other hand, in the context of information flow and for the purposes of the present deliverable, a Service represents more than a single computational output or analytic result. Here, we define it as a coordinated workflow composed of multiple ISs that belong in either BAFT, ABDST, NAFF or ANDSF of the B2G/G2B Layers, that are working in sequence to process data and deliver results to the EVELIXIA's UI, the **Stakeholder Platform**. Each Service:

- Is designed to address a specific business need, such as demand flexibility provision, portfolio optimization, or network investment planning, among others.
- Combines the functionalities of several ISs, each contributing specialized recommendations, analytics, or decisions.
- Operates within the framework of either a B2G or G2B interaction, depending on which types of actors are benefitting or the information flow.

These Services then can be classified in two broad categories, regardless of context, based on their operational patterns:

- **On-Demand Services**: Triggered manually by end-users (e.g., energy managers, aggregators) through the Stakeholder Platform, typically for scenario evaluation, investment planning, or ad-hoc simulations.

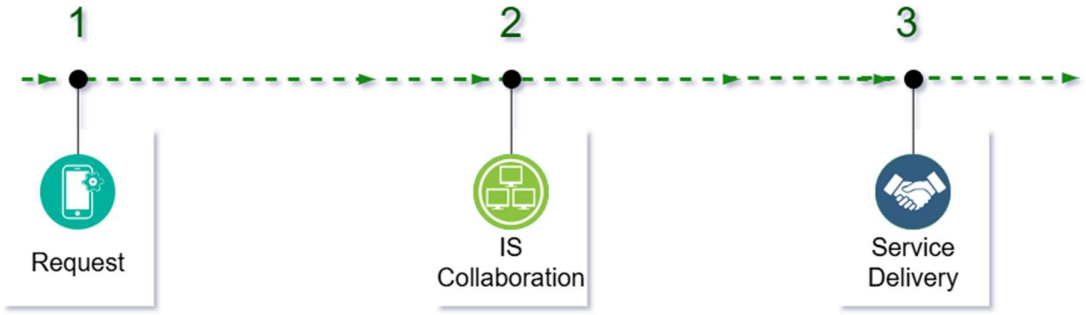


Figure 4 - On-Demand Service Logic

- **Continuous Services**: Operate autonomously, running at pre-defined intervals (e.g., daily) to provide forecasts, recommendations, or operational alerts without user intervention.

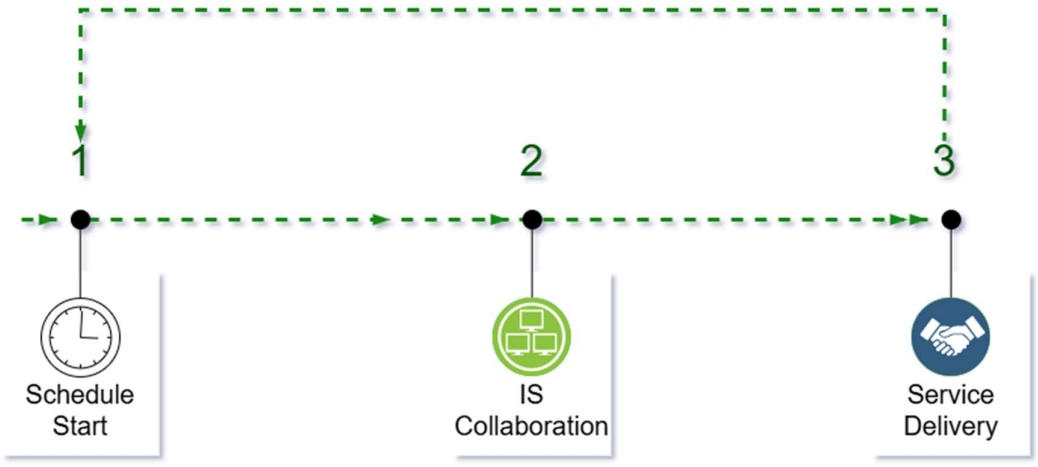


Figure 5 - Continuous Service Logic

The purpose of T4.6 is to enable these Services by:

- Dynamically initializing the required ISs for each requested service.
- Orchestrating the sequence in which the ISs consume, process, and exchange data.
- Delivering aggregated results to end-users via the Stakeholder Platform or to other systems in the EVELIXIA Ecosystem.

In the following sections of this document, we will explain the key design and integration principles we have implemented to ensure coherent, reliable and repeatable Service results.

2.2 Design Principles of the Service Layer

To facilitate the integration of the project's diverse toolset and needs, we have devised specific design principles that were directly shaped by the complexities and challenges posed by EVELIXIA's specificities and objectives.

As the DoA details, the EVELIXIA ecosystem is expected to serve multiple Pilot Sites and integrate a wide range of diverse ISs, each having different computational needs and developed with different technology frameworks. Additionally, as all distributed systems, EVELIXIA has to be prepared for many of its components operating in different hosting environments and premises across multiple countries.

Building on the main guidelines detailed in D1.7, and simultaneously addressing the above challenges, the following fundamental design decisions for the Service Layer were taken.

2.2.1 Modularity and Containerized Deployment

The Service Layer involves 19 different ISs, each developed by different organizations, using different technology stacks and principles. Apart from the ISs, a number of backend/integration components must be developed to support the smooth operation of the whole layer. All these components must:

- Harmoniously interact as steps within complex workflows to produce the end-results of the Energy Services detailed in D1.7.
- Operate across different Pilot Sites using different data
- Be adaptable to the evolving requirements of the project

To address this challenge, each component within the Service Layer, whether it's an IS or a software solution responsible for orchestrating IS interactions, handling user requests, or delivering outputs, has been designed to operate as an independent and reusable module. This means that different parts of the system can evolve separately without causing disruptions and the IS developers can work independently.

This modularity also ensures that the platform can adapt to each Pilot's specific configuration, as different regions might deploy slightly different services depending on their needs.

2.2.2 Orchestration-Centric Coordination

As mentioned before, EVELIXIA wants to support a vast number of Services, utilizing data from dissimilar sources, all while having different activation timeframes and for different time windows throughout the day. These Services involve complex computational workflows utilizing multiple ISs and components, that need to exchange data either periodically in an automated manner (Continuous Services) or when the end-user requires on-demand action (On-Demand Services). These requirements pose the need for introducing a dedicated orchestration mechanism for the Service Layer, instead of relying on direct connections between each IS.

This Orchestration Layer within the EVELIXIA Service Layer should manage the sequence of IS interactions within each Service to produce their end-results by:

- Automatically initializing each workflow by connecting the ISs when the end-users subscribe to a Service through the Stakeholder Platform.
- Managing the data flows between the different involved ISs.
- Ensuring the correct sequencing and execution of both automated and user-triggered workflows and processes.

2.2.3 Hybrid Communication Model

Since the ISs will be deployed across distributed and heterogenous IT environments, with varying levels of direct access to EVELIXIA's core Service Layer components, there is a need to adopt a hybrid communication model to support the different types of data exchanges. This challenge is caused because native event-driven communication schemes that were originally envisioned in the DoA

cannot be technically supported by all ISs or not permitted due to network restraints (e.g, firewall policies, isolated networks etc).

To mitigate this challenge, the Service Layer will integrate two complementary communication mechanisms to ensure this hybrid model of data exchange:

- An event-driven Message Brokerage System that will allow, wherever possible, for the ISs to communicate natively and asynchronously via standard publish/subscribe mechanisms.
- A RESTful API Gateway that will act as a bridge for ISs and components of the EVELIXIA Ecosystem (e.g the Middleware Northbound API) that cannot directly communicate with the Brokerage System, thus enabling them to participate in the workflows through standardized API Requests.

2.2.4 Integrating User Interactions with Background Processes

Apart from the ISs and external components that cannot directly consume or publish to the Brokerage System, the API Gateway addresses another requirement of the EVELIXIA Platform, the user-initiated actions.

These actions, while external, require immediate feedback from the Service Layer and can be considered as synchronous processes. In order to be able to support both the asychronous and synchronous processes, the Service Layer was designed to support all external interactions (e.g the user actions from the Stakeholder Platform or the Middleware) through the API Gateway. This provides:

- A unified entry point.
- A seamless bridge between synchronous and asynchronous workflows.
- Simplified external integration.

2.2.5 Security-by-Design Approach

Although not directly tackled by T4.6, as a general design principle, the Service Layer should take into consideration the need for strict control over which end-users or actors can access what data and in which context, ensuring isolation between different Pilots and Services.

The vertical security principles that EVELIXIA, and consequently, the Service Layer should follow, are detailed in D1.7, D3.5 and will be expanded in D5.1, which is due on M24 of the project, but as a general approach, the system should enforce:

- Authentication via a dedicated security mechanism (e.g., Keycloak and Blockchain).
- Access control at the API Gateway level.
- Preparedness for future requirements such as:
    - End-to-end encryption.
    - Role-based permissions.
    - Per-Pilot isolation, should localized deployments become necessary.

2.2.6 Flexible and Scalable Architecture

To accomodate the future architectural changes that will be detailed in M33 and the updated version of D1.7 and the rest of the updated deliverables of the WP4, the Service Layer needs to be designed to be future-proof, meaning it should be able to:

- Scale to support additional Services or the updated functionalities of the ISs, by utilizing common data schemas like JSON
- Operate both in centralized and per-Pilot configurations
- Allow Pilot-specific variations in the Services, without compromising the overall Service Layer

To achieve this flexible and scalable architecture for the Service Layer, all above design principles must be utilized. Their summary can be found in Table 1 :

Table 1 - Service Layer Architectural Design Principles

Challenge	Design Decision	Purpose
Integration of Diverse ISs and Components	Modular and Independent Components	Enable independent development, deployment, and evolution of ISs and integration components.
Coordination of Complex Multi-IS Workflows	Orchestration-Centric Approach	Manage sequencing, deployment, and execution of workflows both automatically and on-demand.
Distributed and Heterogeneous Deployment Environments	Hybrid Communication Model (Message Broker + API Gateway)	Support both asynchronous (native broker) and synchronous (API) interactions, overcoming network limitations.
Support for User-Triggered Processes	Unified Access through API Gateway	Provide a single entry point for synchronous user interactions and external system requests.
Security Across Services and Pilots	Security-by-Design	Enforce access control, authentication, and isolation following platform-wide security principles.
Future Scalability and Pilot-Specific Customization	Flexible and Scalable Architecture	Allow centralized or Pilot-specific deployments, scalable service addition, and Pilot-specific service variation.

2.3 Service Layer Architecture

The high-level architecture can be visualized as a set of interconnected modules: an entry-point for external interactions, orchestration and processing logic, communication infrastructure, data storage, and the plugged-in Innovative Solutions (specialized tools). Utilizing the basic Design Principles detailed in the previous section, the following high-level architecture has been designed for the Service Layer:

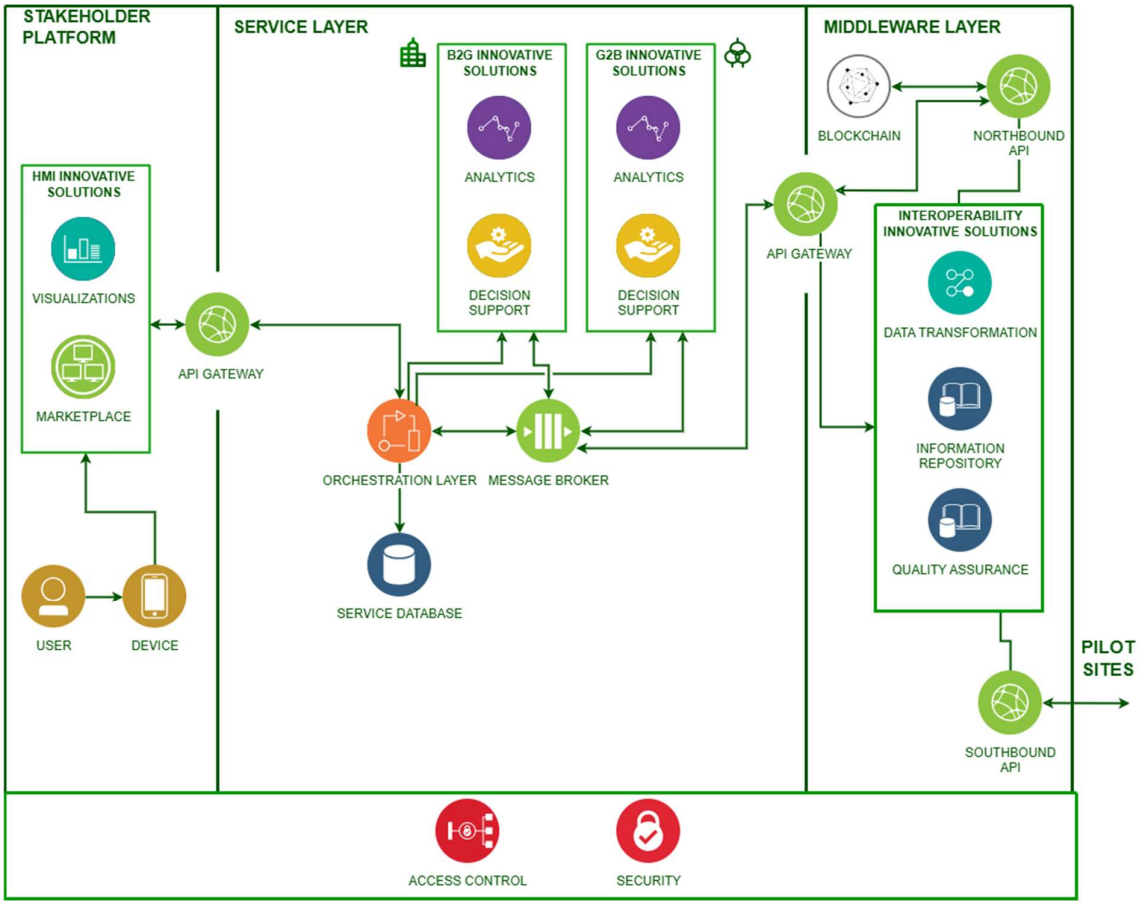


Figure 6 - EVELIXIA Service Layer Architecture and Interactions with Frontend Layer (Stakeholder Platform) and Middleware

The core components of the Service Layer are the following:

- **API Gateway**: Acts as the unified access point for all external interactions (e.g., from the Stakeholder Platform and Middleware), and bridges tools or actors that cannot directly communicate with the internal messaging system.

- **Orchestration Layer**: Responsible for managing workflows, configuring communication links between tools, and monitoring container health.
- **Message Brokerage System**: Facilitates asynchronous, event-driven communication between IS tools. It ensures loosely-coupled data exchange and supports both continuous and on-demand workflows.
- **Service Layer Database**: Stores historical results from the execution of services, making outputs easily accessible to both the Frontend and other IS tools.
- **ISs**: The core analytical, forecasting, optimization, and simulation modules, developed independently and integrated into services via well-defined workflows. For the purpose of this deliverable, we consider the ISs to be containerized "black boxes", where no source code or logic is explored.

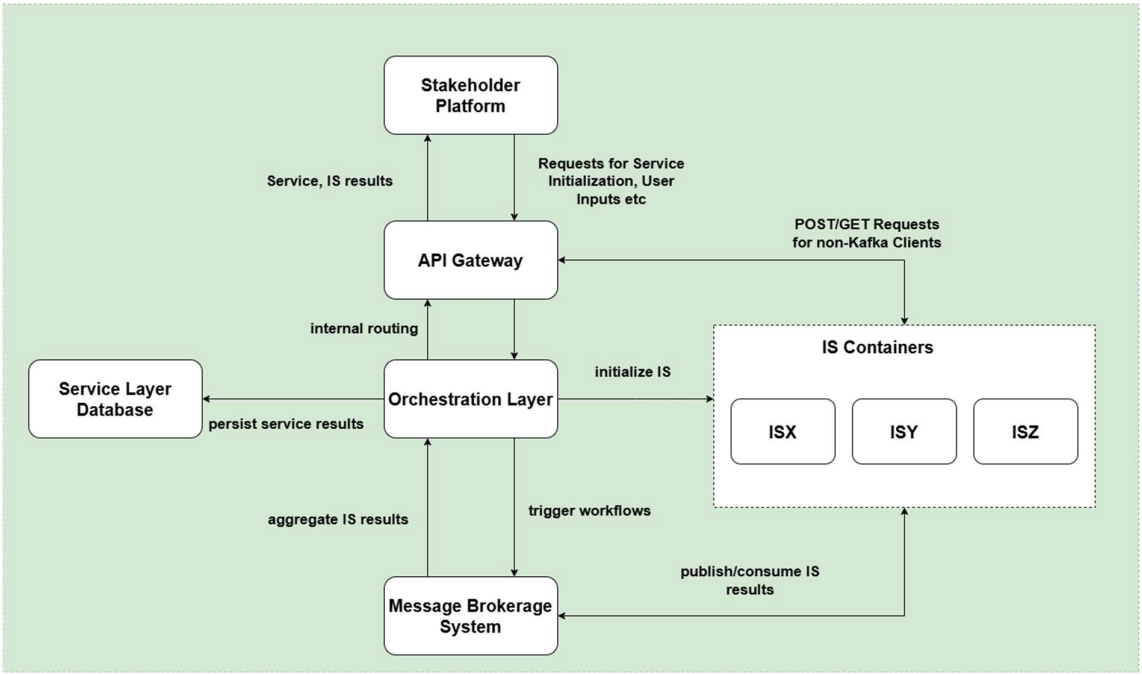


Figure 7 - Service Layer Detail

In the following chapters we will provide detailed breakdowns of each of the components of the Service Layer and highlight their role in facilitating the services.

3 SERVICE LAYER COMPONENTS

The Service Layer consists of several core components, each responsible for specific functionalities. Together, these components implement the principles and patterns described in the previous chapter. This section provides a functional overview of each component, explaining its role and how it interacts with others.

3.1 Message Brokerage System

The core component of the EVELIXIA Service Layer architecture, and the foremost goal of T4.6, is its event-driven communication model, which enables decoupled, scalable, and asynchronous data exchange between the different ISs. To implement this, the platform employs a Message Brokerage System based on Apache Kafka, an industry-standard distributed event streaming platform. This system serves as the communication message bus of the Service Layer, facilitating the orchestration of complex services, integration of heterogeneous ISs and delivery of data and commands across the ecosystem.

At the core of the Message Brokerage System is the publish/subscribe paradigm. In this model, components (referred to as producers) publish structured messages to Kafka topics, which act as named communication channels. Other components (referred to as consumers) subscribe to these topics to receive and process the messages asynchronously. This design eliminates direct dependencies between producers and consumers, allowing services to operate independently, scale flexibly, and evolve over time without strict integration rules.

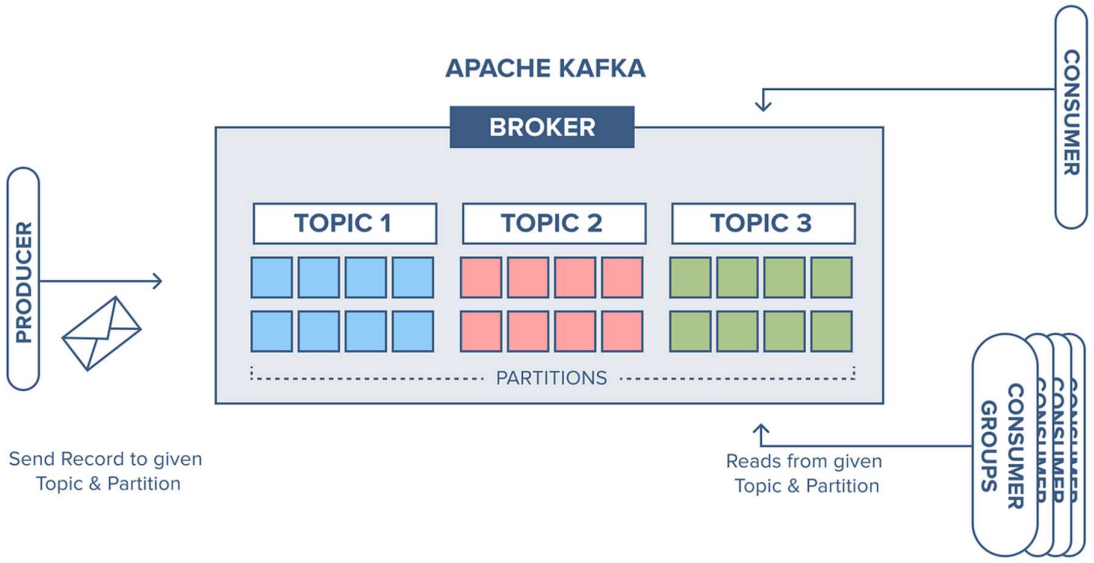


Figure 8 - Kafka Broker Basic Architecture

This asynchronous approach is particularly well-suited to the requirements of the EVELIXIA platform, where ISs are developed by different partners, hosted across distributed environments, and may need to operate independently based on different triggers or schedules. Kafka ensures that all events, whether they originate from user-triggered actions (on-demand services), time-based schedules (continuous workflows), or field data provided via the Middleware Layer and Northbound API, are available when needed by the different ISs and Service Layer Components.

The Message Brokerage System also supports buffering, load balancing, and fault tolerance through features such as message retention and consumer groups. These features make sure that if a service is temporarily unavailable or slow to respond, messages are not lost and can be processed once the service resumes. Furthermore, the architecture is inherently scalable, supporting the future evolution of the platform to handle higher data volumes, more services, and more complex workflows across multiple pilot sites.

In the following subsections, the structure, policies, and role of Kafka within the Service Layer will be described in more detail, along with its integration with other components such as the Orchestration Layer, API Gateway, and the Service Database.

3.1.1 Topic Naming Convention

In EVELIXIA's distributed and modular architecture, communication between components is handled through an event-driven message broker system. One of the key design challenges was determining how to structure Kafka topics to support clean, reusable data flows between the ISs, especially considering that the same IS output may be consumed by multiple different services.

To address this, the naming convention of Kafka topics in the Service Layer follows a pilot-scoped, service-agnostic structure: **{pilot}.{IS}.output**

This naming strategy ensures that:

- Each IS produces results to only one topic per pilot, regardless of how many services use that data.
- Data ownership and origin are clearly expressed (per IS, per pilot), without unnecessarily duplicating outputs per service.
- Other ISs and workflows can reuse existing outputs without changing the IS publishing behavior.
- Service awareness is encapsulated in message metadata, not in topic naming, allowing more flexible and maintainable workflows.

Services that require specific outputs from one or more IS tools subscribe to the relevant topics and apply logic to filter or route messages based on metadata such as *workflow_id, service_id*, or *timestamp*. This decouples producers from consumers and allows multiple services to make use of shared components without needing multiple duplicate topics or extra logic.

Since the Kafka Broker will also accomodate the transmission of structured and contextualized Pilot Data coming from the Middleware via the Northbound API to the different ISs of the Service Layer, we defined a dedicated topic naming convention for them to be published. The pattern we follow for this first version is:

{pilot}.middleware.iot

These pilot-specific topics will carry the structured JSON telemetry data from the Northbound API, following WP3's interoperability specifications. Since the integration of the Middleware and the Service Layer will take place in T5.1, we have started with this simplified approach of a single ingestion pilot topic that will be further developed and designed in later stages of the project.

Regarding service-specific user inputs coming from the Stakeholder Platform, we have also implemented another type of topics to facilitate their correct

dissemination to the different ISs. For these types of messages, the naming pattern is defined as: **{pilot}.{service_id}.trigger**

As the user inputs will usually act as a trigger for the on-demand EVELIXIA Services, this design will allow all ISs participating in a specific service to be activated asynchronously, without requiring direct invocation.

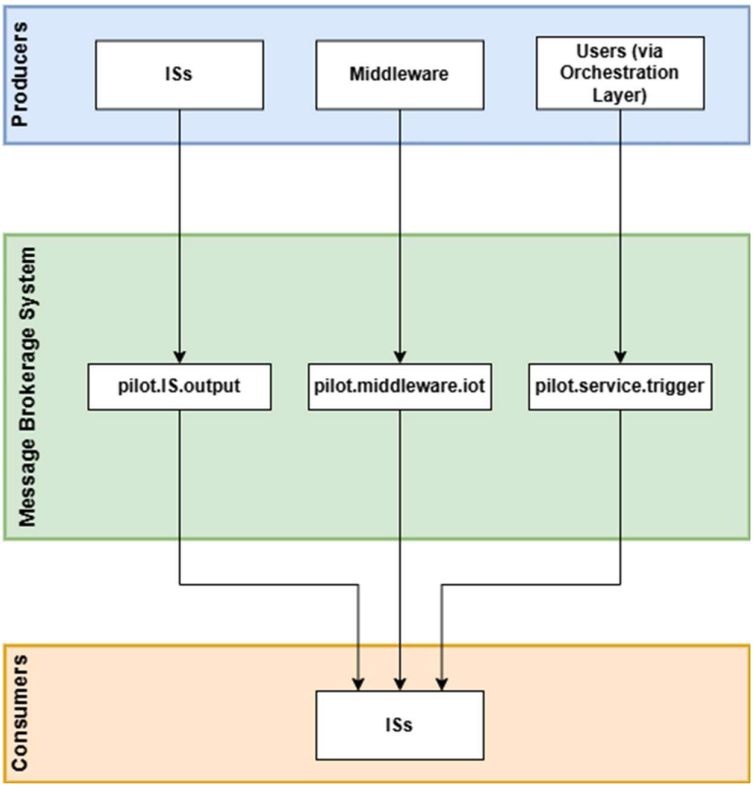


Figure 9 - Broker topics, producers and consumers

3.1.2 Message Format and Metadata

Each Kafka message exchanged within the EVELIXIA Service Layer IS Topics, follows a structured JSON schema to ensure consistency, interoperability, and service-agnostic communication between components. The message format includes both core metadata (for workflow tracing, filtering, and correlation) and dynamic payloads specific to the IS outputs.

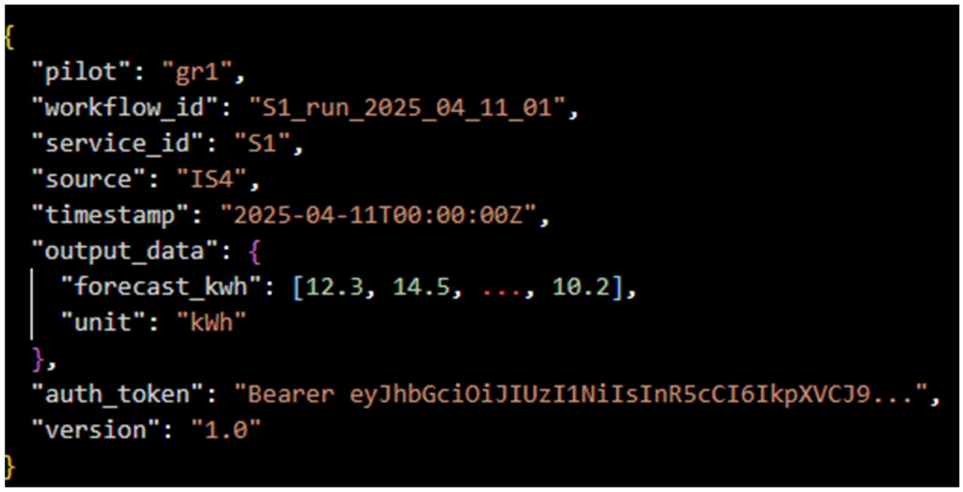


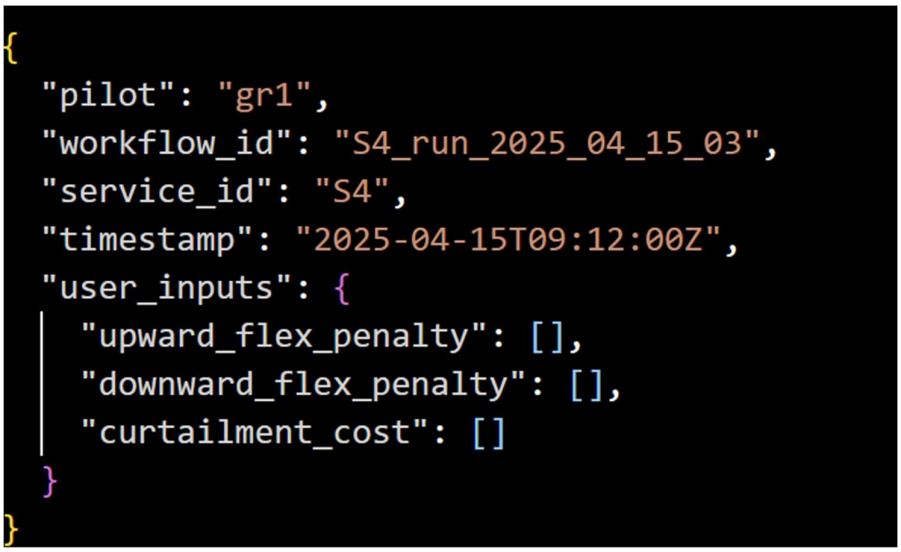
Figure 10 - Example Kafka payload of IS4 participating in service 1

The fields of the payload are described in Table 2:

Table 2 Key fields of a Kafka Payload Message

Field	Description
pilot	Identifier of the pilot site (e.g., gr1)
workflow_id	Unique ID assigned to the service workflow instance
service_id	Service identifier the workflow belongs to (e.g., S1)
source	IS that generated the result (e.g., IS4, Middleware etc)
timestamp	UTC timestamp of result generation
output_data	Dynamic JSON content containing results (e.g., forecasts, KPIs). Flexible depending on the outputs of each IS
unit	Optional unit metadata (e.g., kWh, °C)
auth_token	Token verifying the origin of the message (used for security/auditing). Auth mechanism will be implemented fully in T5.1
version	Schema version for compatibility and evolution

Regarding the Service Trigger topics, although the schema design varies, we keep the same design principles to ensure interoperability while accommodate a wide range of different user inputs.



```
{
  "pilot": "gr1",
  "workflow_id": "S4_run_2025_04_15_03",
  "service_id": "S4",
  "timestamp": "2025-04-15T09:12:00Z",
  "user_inputs": {
    "upward_flex_penalty": [],
    "downward_flex_penalty": [],
    "curtailment_cost": []
  }
}
```


Figure 11 - Example payload for a Service Trigger Topic

The fields of the payload are described in the Table 3:

Table 3 - Key fields of a Service Trigger Payload Message

Field	Description
pilot	Identifier of the pilot site (e.g., gr1)
workflow_id	Unique ID assigned to the service workflow instance
service_id	Service identifier the workflow belongs to (e.g., S1)
timestamp	UTC timestamp of result generation
User_input	Object containing the parameter values provided by the end-user or frontend (e.g., penalty costs, target date). Flexible structure depending on the specific service.

3.1.3 Retention, Partitioning and Reliability

To ensure robustness and traceability of the workflows, the Kafka Broker is configured with the core parameters presented in Table 4:

Table 4 Kafka Broker Settings & Policies for low latency and reliability

Setting	Value	Description
Message Retention	24 hours (per topic)	Ensures that results are available for a full workflow cycle. Can be increased in v2.
Max Message Size	~1MB (default)	Sufficient for most payloads (e.g, hourly forecasts, simulations results without visuals).
Topic Partitions	1–3 (per topic, scalable)	MVP starts with 1; partitioning per building/service can be enabled later for scalability.
Consumer Groups	Enabled	Allows multiple ISs or database consumers to read the same topic independently.
Message Durability	Acknowledgment level: acks=all	Ensures messages are fully replicated before acknowledging success.

After the 24-hour window, the retained data inside the various topics are persisted in the Service Layer Database for long-term storing. To avoid duplication, this excludes any data coming from the Middleware and the Northbound API, as those are already saved in the Middleware Database. These parameters were chosen to support both low-latency and reliable workflow execution, all while allowing for future scaling in cases where more demanding services are designed for the second version of the EVELIXIA Platform.

3.2 API Gateway

Another core component of the EVELIXIA Service Layer architecture is the API Gateway, which serves as the unified and secure communication interface between external (to the Service Layer) systems and the internal services of the platform. Its primary role is to enable real-time, request-response interactions for end-users, external services, and tools that are unable (due to technical or security constraints) to interact directly with the Message Brokerage System. The API Gateway complements the platform's asynchronous message-based communications by supporting synchronous workflows, event triggering, data requests, and field-level interactions.

The EVELIXIA platform operates within a heterogeneous and multi-actor environment, where not all tools or users can be tightly coupled to internal event streams. This challenge is particularly relevant in scenarios where:

- End-users (e.g., building managers, aggregators, grid operators) interact with the system via the Stakeholder Platform.
- Certain IS tools are hosted in restricted environments, where direct Kafka connectivity is either unsupported or forbidden by network security policies.
- Synchronous calls are needed to trigger workflows, subscribe to services, or retrieve analysis results in real time.

To accommodate these scenarios, the API Gateway operates as a hybrid integration layer, bridging the asynchronous world of Kafka with REST-based components and users. It allows loosely coupled services to interact without compromising real-time responsiveness, security, or modularity.

At a high level, the API Gateway acts as a unified entry point for all systems that are considered external to the Service Layer and should perform the system-wide functions presented in Table 5:

Table 5 API Gateway Core Functions

Function	Description
Service Orchestration Trigger	Routes validated service subscription or execution requests to the Orchestration Layer to initiate appropriate workflows.
Kafka Proxy Interface	Supports IS tools that do not implement Kafka clients, by acting as a publishing/subscription proxy
Middleware Access Endpoint	Facilitates access to contextual or raw data coming from the Middleware Layer based on pilot and dataset filters.
Security Enforcement	Verifies and enforces authentication/authorization using OIDC-compatible tokens (e.g., Keycloak-issued JWTs), and manages cross-cutting concerns like rate-limiting, CORS, and traffic logging.

It is worth noting that while the Middleware Access Endpoint and the Security Mechanics have been conceptualized, they will be implemented at a later stage for the purposes of T5.1.

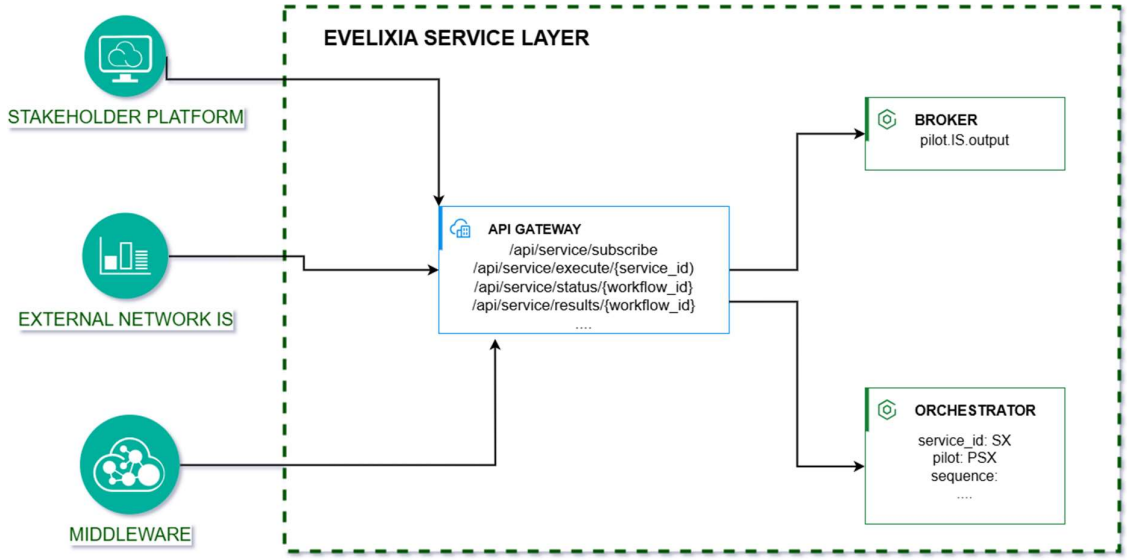


Figure 12 - API Gateway Role

3.2.1 Supported Endpoints

From a communication perspective, the API Gateway complements the event-driven architecture by offering RESTful interfaces that mirror the same core capabilities enabled by Kafka. This means:

- ISs unable to use Kafka natively can use the Gateway's /publish and /consume endpoints to participate in workflows by sending/receiving messages through the Gateway (which acts as a Kafka client on their behalf).
- Service initialization requests (e.g., service subscriptions or workflow executions) are synchronously received and forwarded to the Orchestration Layer, which in turn handles deployment, topic configuration, and message publishing.
- Result queries and monitoring (e.g., polling for results or checking status of workflows) are also handled through REST calls, abstracting the underlying database and orchestration logic.
- Field data requests are routed through the Gateway to the Middleware Northbound API, based on pilot and data type, ensuring standardization and access control across sites.

Table 6 contains an overview of the supported REST API endpoints:

Table 6 Designed Endpoints for the API Gateway

Endpoint	Method	Description
/api/service/subscribe	POST	Triggers the Orchestration Layer to initialize a specific Service.
/api/service/execute/{service_id)	POST	Triggers the execution of an on-demand workflow
/api/service/status/{workflow_id}	GET	Retrieves current status of a workflow (e.g., running, failed, completed).
/api/service/results/{workflow_id}	GET	Retrieves final output of a completed service (fetched from Service Layer DB).
/api/IS/publish/{topic}	POST	Allows ISs that cannot connect to Kafka to push outputs into the broker.
/api/IS/consume/{topic}	GET	Allows Kafka-incompatible tools to fetch messages from broker topics via polling.
/api/data/{pilot}/{dataset}	GET	Provides access to Middleware data for a specific pilot and category.
/api/health	GET	Lightweight service availability check (e.g., for orchestrator status or tool availability).

We have conceptualized the endpoints to be protected by RBAC, by relying on bearer token authentication.

3.3 Orchestration Layer

The Orchestration Layer is a central component of the EVELIXIA Service Layer responsible for executing workflows across the various modules, including IS tools, the Message Brokerage System, and the API Gateway. It acts as the logic and coordination engine of the platform, interpreting user or system-triggered requests, initiating the corresponding processing chains, and managing the sequencing and communication required to produce analytical outputs for each Service.

Unlike the Message Brokerage System (Kafka), which serves as the transport backbone, and the API Gateway, which provides entry and exit points for external communication, the Orchestration Layer is the component that understands what needs to be done in response to incoming requests and how to coordinate the involved services and data flows to make that happen.

Table 7 Core functions of the Orchestration Layer

Function	Description
Workflow Initialization	On user subscription to a Service, the orchestrator triggers the initialization of required IS containers (via Docker Compose for the MVP), and configures Kafka topics and environment variables.
Request Handling	Upon receiving a request (via the API Gateway), the orchestrator determines the correct execution logic (e.g., invoking the right IS tools in order).
Kafka Coordination	Publishes events to relevant Kafka topics to initiate IS actions. Generate trigger events on dedicated trigger Kafka Topics to initialize on-demand workflows. Subscribes to result topics to gather outputs or handle asynchronous responses.
State & Error Management	Tracks workflow execution (status, success/failure), manages retries, and updates result statuses in the Service Layer Database.
Result Aggregation	In workflows involving multiple IS tools, aggregates the final outputs, packages them, and optionally sends them to the database or back to the frontend.
Scheduling & Triggering	For continuous workflows (e.g. daily forecasting), the orchestrator uses internal scheduling logic (via Docker Compose cron-like setups) to launch jobs without frontend input.

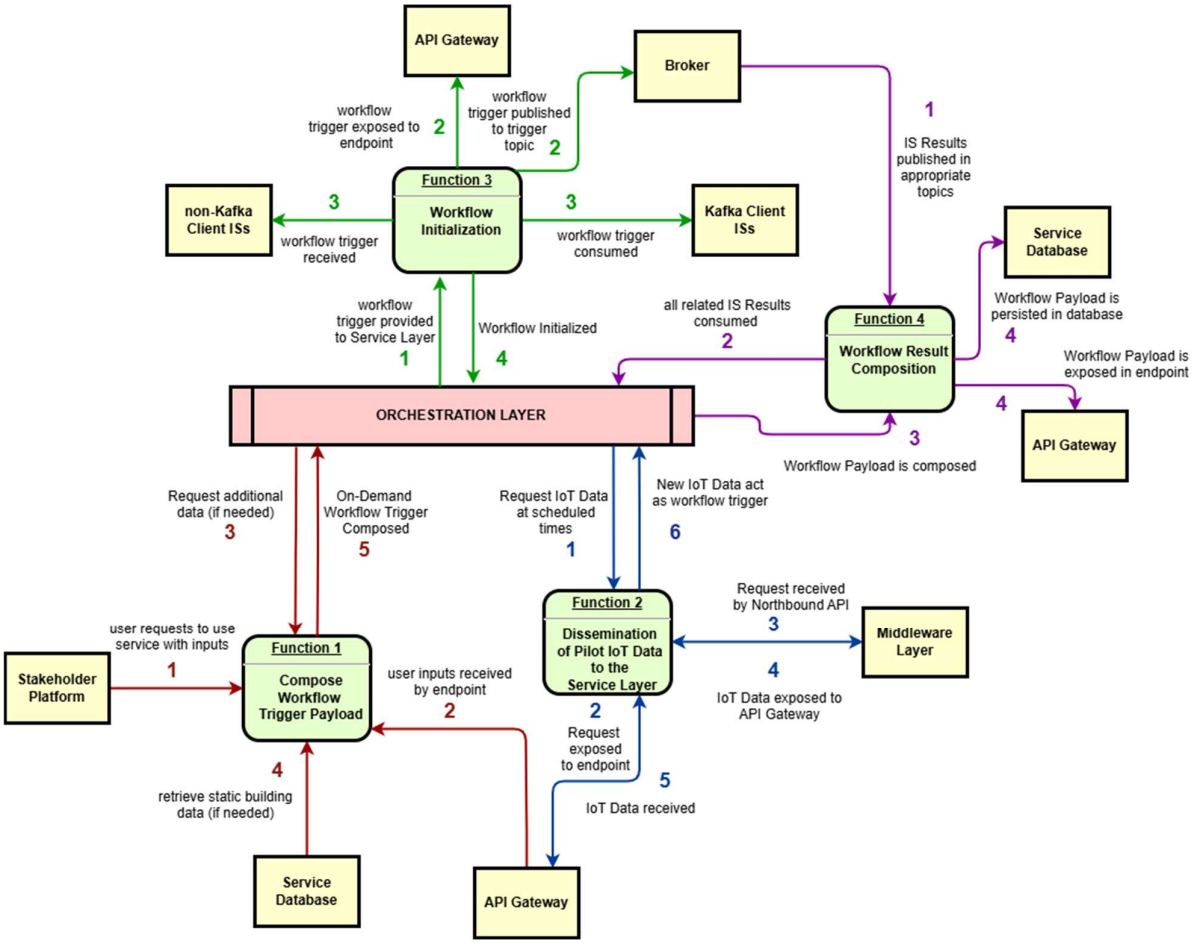


Figure 13 - Orchestration Layer Functions.

Figure 13 illustrates the consolidated functions of the orchestration Layer.

- **Function 1:** Compose Workflow Trigger Payload (Red), Collects necessary user inputs and service configuration to generate a valid workflow trigger message.
- **Function 2:** Dissemination of Pilot IoT Data to the Service Layer (Blue), Handles the reception and publishing of structured pilot data to Kafka topics for IS consumption.
- **Function 3:** Workflow Initialization (Green), Initiates the execution of a defined service by distributing the composed trigger to internal and external components.
- **Function 4:** Workflow Result Composition (Purple), Aggregates IS outputs, formats the final service result, and persists it to the Service Layer database.

For this first version of the Service Layer, we have designed the Orchestration Layer through a custom Node.js/Express service running in a Docker Compose environment, complementary with the other Service Layer Components. To perform the core functionalities depicted in the table above it uses:

- REST API routes to receive external triggers (/api/service/execute, /api/service/status)
- Kafka client libraries to produce and consume messages related to each service workflow
- Lightweight job tracking using internal in-memory state and MongoDB as needed for result persistence

This way, the Orchestration service can manage different types of workflow executions while enabling the different ISs to remain stateless and agnostic. Table 8 lists the types of triggers that have been identified so far:

Table 8 Types of Workflow Triggers that the Orchestrator Supports

Trigger Type	Example	Details
User-Initiated	An end-user on the Stakeholder Platform clicks a button that triggers an action, e.g., "Run Optimization"	Orchestrator service validates requests and publishes initial message to Kafka Broker
Time-Based	An initialized continuous Service executes a new workflow e.g., Daily Forecasting starts at 00:00	Internally scheduled via cron-like timer to initiate event to an IS
Data-Driven	Middleware sends new Pilot Data	Orchestrator manages Middleware Topics and triggers workflows when conditions match

A useful clarification regarding the role of the Orchestration Layer is that it does not act as a Middleware inside the Service Layer Architecture, but rather acts as a

decision-making component of the backend that provides and conducts the internal logic of the whole Layer. Currently, we created a rule-based logic to accommodate existing services through configuration and environment files, but in the next version the aim is to explore more robust orchestration options like Kubernetes or Apache Airflow.

Additionally, for user-initiated (on-demand) workflows, the orchestrator initiates execution by publishing a structured message to a Kafka trigger topic specific to the service and pilot (e.g., gr1.S4.trigger). These trigger topics act as coordination entry points where ISs involved in each service will listen to them to begin processing. This design ensures that no direct service-to-service invocation is required, while allowing the orchestrator to centrally manage the execution logic. The orchestration logic determines the naming and routing of these trigger messages dynamically based on each service's YAML configuration file.

3.3.1 Dynamic Configuration of Workflows

The Orchestration Layer manages the logic of the Service Layer through dynamic configuration. This means that the rules are not hard-coded in each involved IS's source code, but rather some external descriptors are utilized to understand:

- Which ISs are involved in a service
- What is the order they should be invoked
- What Kafka topics should be produced to or consumed from
- What parameters or transformation logic needs to be applied
- Where and how to store final results

This logic is received by each IS container through a set of environment variables during the deployment phase. Those include:

Table 9 Environment Variables to inject in each IS Container

Variable	Description
KAFKA_BROKER_URL	Kafka broker instance for the pilot (currently we have only 1 Broker)
INPUT_TOPICS	Kafka topic(s) to consume from
OUTPUT_TOPIC	Topic to which results should be published
SERVICE_IDS	Which services this tool participates in

WORKFLOW_TIMEOUT	Max time to wait before failing a workflow

Additionally, for each Service, the Orchestration Layer references a specific YAML file that defines the logic behind each workflow.

```yaml
service_id: S1_1
pilot: gr1
sequence:
  - tool: IS5
    input_topic: gr1.middleware.iot
    output_topic: gr1.IS5.output
  - tool: IS1
    input_topic: gr1.middleware.iot
    input_topic: gr1.IS5.output
    output_topic: gr1.IS1.output
  - tool: IS3
    input_topic: gr1.middleware.iot
    input_topic: gr1.IS5.output
    output_topic: gr1.IS3.output
  - tool: IS4
    input_topic: gr1.middleware.iot
    input_topic: gr1.IS5.output
    output_topic: gr1.IS4.output
store_to_db: true
```

Figure 14 - Example YAML file depicting the involved ISs of S1.1, along with their appropriate input/output topics

For each Service and their YAML files the orchestrator:

- Determines the required Kafka topics to configure and monitor
- Establishes the execution chain for the IS tools
- Monitors expected outputs to determine when the workflow is complete
- Logs metadata (timestamps, tool outputs, errors) and updates workflow status in the database.

3.4 Service Layer Database

The Service Layer Database is a core backend component responsible for persistently storing the results of service workflows, workflow statuses, and service metadata. Unlike raw data ingested from the Field Layer via the Middleware (which remains stored within the Middleware infrastructure) the Service Layer Database only stores outputs produced from the execution of workflows across one or more IS. These include aggregated service results, IS-specific outputs, workflow metadata, and execution status.

In the first version, we use MongoDB (a document-based NoSQL database) for its flexibility, schema-less nature, and ability to scale with heterogeneous and dynamic data structures. This structure fits perfectly with the requirements of a platform where multiple ISs may contribute various outputs (with different formats and metadata) to a single workflow result.

- The Service Layer Database was designed to accommodate specific goals:
- Store the final outputs of each workflow in a structured and queryable format
- Retain results per pilot and per service while accommodating multiple contributing ISs
- Allow IS-specific result queries in addition to full workflow queries
- Support easy integration with Orchestration Layer logic and frontend queries

3.4.1 Result Storage Model

Each service execution creates a single document under the Results collection. The document contains metadata about the workflow and stores a list of individual results submitted by different ISs. This allows for querying both full workflow outputs and individual IS contributions.

Each Result Document that is created for each workflow_id contains the necessary workflow-level metadata (e.g., pilot, timestamp, service_id) as well as the results array that contains the results of each participating IS. Each result item contains:

- The originating source (IS ID or other source).
- results with IS-specific fields and format.

- Optional fields such as unit, status, and error_msg (if applicable).

This design ensures that all outputs of the service are stored together while simultaneously keeps IS results separately accessible within the array to support selective access (if needed).


```
{
  "workflow_id": "S4.1_run_2025_04_15_01",
  "pilot": "gr1",
  "service": "S4.1",
  "trigger_time": "2025-04-15T08:00:00Z",
  "user_id": "evelixia_user_1",
  "inputs": {
    "penalty_up": [],
    "penalty_down": [],
    "curtailment_cost": []
  },
  "results": {
    "IS15_baseline": { /* baseline results from IS15 */ },
    "IS12_optimization": { /* results from IS12 */ },
    "IS15_optimized": { /* post-optimization results */ }
  },
  "status": "complete"
}
```


Figure 15 - Example workflow document saved in the Service Database

3.4.2 Database Querying

The platform must support a variety of queries originating from the frontend, the orchestration layer, and possibly even other ISs. The design of the storage model supports the following key scenarios:

1 Retrieve Full Workflow Results (per workflow_id)

This use case supports end-users on the Stakeholder Platform or visualization dashboards who want to review the outcome of a specific service execution. The platform will retrieve the full document based on the workflow_id, including all IS contributions.

2 Retrieve IS-Specific Output (inside a workflow)

When a downstream IS tool or orchestration step only needs the result of one particular IS (e.g., only IS5's dispatch plan), the system can isolate and extract the relevant result item inside the array using internal filtering.

3 List Workflow Executions (per service and pilot)

For navigation or analytics, the frontend may display a history of service executions for a given pilot and service. This requires filtering documents by pilot and service, sorted by timestamp.

4 Analytical/Validation Queries (by IS output type or key)

Some advanced tools or reports may request "all forecasts by IS4 in the past week" or "every time IS10 issued a certain control signal." These queries leverage field-level filtering inside the results.output_data.

5 Support Historical Results Fetching

Since Kafka message retention is temporary (e.g., 24h), any re-evaluation or re-visualization must query the database. The orchestration layer will provide an abstraction layer to serve these through specific request types (e.g., via /service/status).

It is worth noting that direct access to the Service Layer Database is restricted to the Service Layer Components. For this reason the API Gateway routes external calls to the Orchestrator Service which in turn queries the database using workflow_id or other parameters according to the use cases. When the vertical EVELIXIA Security layer is implemented in T5.1, this logic may need to be adjusted to accommodate the new authentication requirements.

3.5 Additional Service Layer Components

While the focus of D4.7 is the design and integration of the Service Layer through the core technical components detailed in the previous sections, there are also two additional types of entities that play an important role within EVELIXIA; the Innovative Solutions (IS) and the Security Components.

3.5.1 Innovative Solutions

The 19 ISs of the Service Layer are developed independently utilizing efforts from the other tasks of WP4 (T4.1-T4.5), providing the core business value of EVELIXIA. Without including technical details, for the purpose of this deliverable we consider the IS to be:

- Independently developed and hosted, most often on partner premises.
- Integrated into workflows via Kafka topics or API calls orchestrated by the backend.

- Containerized and deployed dynamically by the orchestrator when a service is activated. In the context of this deliverable, each IS is treated as a modular processing node that consumes or produces data, without delving into its internal algorithmic details. Integration contracts (schemas, topics, and configuration formats) are agreed during the onboarding of each IS and serve as the primary interface for the orchestration logic.
- The Human-to-Building Interfaces (IS16-IS19) are considered part of the frontend of the EVELIXIA Platform and together are being treated as the Stakeholder Interaction Platform. The key difference between these ISs and the others is that they do not directly take part in workflows or services, but instead 1) provide direct user input to the rest of the Service Layer Components (e.g., triggering workflows, providing user inputs) and 2) utilize service results to present them to the user in a visual manner.

Table 10 lists the ISs involved in the different EVELIXIA Service Layers.

3.5.2 Security Layer

Security in the EVELIXIA platform is handled through a vertical access management system, utilizing Keycloak and Blockchain. These components are developed in T5.1 and T3.5 and provide identity and access management services to all platform components, as well as, handle privacy and trust throughout the Stakeholder Platform. The Security Layer:

- Authenticates users and services via OAuth2 and OpenID Connect tokens.
- Enforces role-based access control policies at the API Gateway level.
- Ensures only authorized requests reach the orchestration and IS logic. Though not developed within this task, the Security Layer is critical for supporting multi-actor usage, secure Pilot operations, and future compliance with cybersecurity regulations in energy systems.

Table 10 Service Layer Innovative Solutions

No.	Name	Type
IS01	Indoor Air Quality measurement	**B2G** - Building Awareness and Forecasting Framework
IS02	Energy Assets Maintenance	**B2G** - Building Awareness and Forecasting Framework
IS03	Demand Forecasting	**B2G** - Building Awareness and Forecasting Framework
IS04	Flex Forecasting	**B2G** - Building Awareness and Forecasting Framework
IS05	Building Energy Modelling and Simulation	**B2G** - Building Awareness and Forecasting Framework
IS06	Building Investment Planning Assistant	**B2G** - Autonomous Building Decision Support
IS07	SRI Advisor	**B2G** - Autonomous Building Decision Support
IS08	DSM Services Advisor	**B2G** - Autonomous Building Decision Support
IS09	Proactive Demand Planning (PDP)	**B2G** - Autonomous Building Decision Support
IS10	Continuous Energy Performance Management (CEPM)	**B2G** - Autonomous Building Decision Support
IS11	Grid Investment Planning Assistant (GIPA)	**G2B** - Autonomous Network Decision Support Framework
IS12	Multi-vector Network Manager	**G2B** - Autonomous Network Decision Support Framework
IS13	Aggregated Demand Portfolio Manager	**G2B** - Autonomous Network Decision Support Framework
IS14	Smart Grid Maintenance	**G2B** - Network Awareness and Forecasting Framework
IS15	Multi-Vector Grids Energy Modelling & Simulation	**G2B** - Network Awareness and Forecasting Framework
IS16	Digital Building Logbook (DBL)	Human-to-Building Interfaces
IS17	Visual Analytics Engine (VAE)	Human-to-Building Interfaces
IS18	Energy Services Marketplace	Human-to-Building Interfaces
IS19	Building Virtual Model	Human-to-Building Interfaces

4 ARCHITECTURAL WORKFLOW PATTERNS

Having defined the technical architecture of the Service Layer and its core components, this chapter focuses on how these components work together to deliver real value through orchestrated service workflows. These workflows follow two main patterns, 1) scheduled and 2) on-demand, each tailored to support different categories of services (e.g., optimization, planning, analytics) throughout the EVELIXIA ecosystem.

This chapter analyses the specifics of the two patterns as well as provides an overview of the revised EVELIXIA Services that will be supported.

4.1 Scheduled Continuous Workflows

In the EVELIXIA Service Layer, Scheduled Continuous Workflows refer to processes that execute automatically and repeatedly at predefined intervals (either time-triggered or event-based) without the need for user interaction.

Scheduled workflows are primarily used in scenarios where services require recurring execution, such as daily building optimization, continuous monitoring of asset health, or forecasting energy demand and flexibility. They ensure the platform remains responsive and autonomous, executing key services on a routine basis even when no explicit user input is given. These workflows share several common traits:

- Automated & Periodic: They are initiated either on a schedule (e.g., daily at midnight) or upon data availability (e.g., arrival of new data). The scheduling can be done through internal orchestration logic or external scheduling tools.
- Defined Processing Logic: The Orchestration Layer manages the full lifecycle of each scheduled job, executing a pre-defined sequence such as: *"Fetch input data → Trigger ISs → Gather results → Persist or dispatch results."*
- Data Aggregation & Input Readiness: Since these workflows are planned, required data (e.g., weather forecasts, grid price signals, building sensor values) can be fetched just-in-time or scheduled to be available in advance via the Middleware integration.
- Asynchronous & Scalable Execution: Heavy computations or long-running simulations are offloaded to Kafka-connected IS tools. The system can run multiple workflows in parallel without blocking operations.

- Always-On Event Streams: Some workflows are continuous listeners to Kafka topics, analyzing data streams in real time and acting upon specific triggers (e.g., detecting equipment failures or grid anomalies).

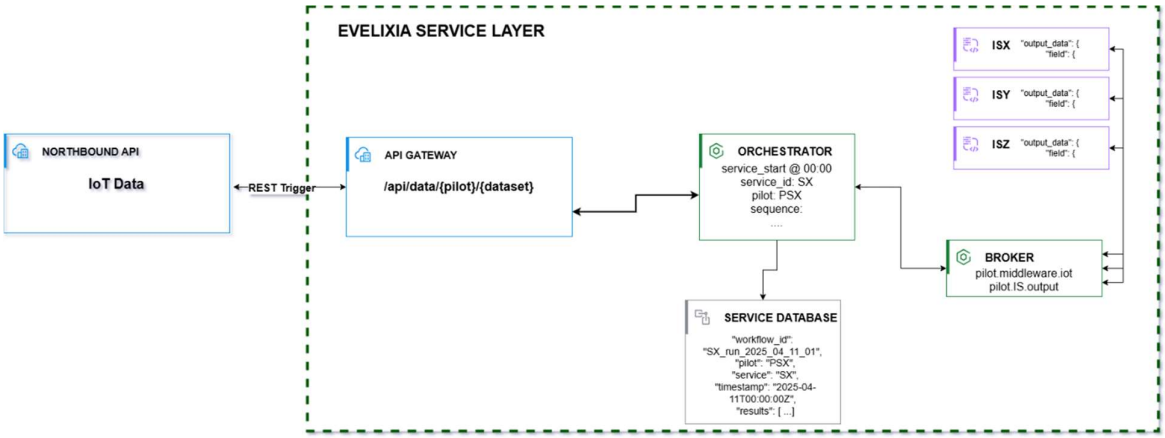


Figure 16 - Components involved in a Scheduled Workflow

For the current version of the Service Layer, the scheduling logic is being implemented inside the Orchestration Layer using lightweight cron-like timers. We define each scheduled service in external configuration files that include:

- Workflow ID
- Execution time
- Participating ISs
- Expected Kafka topics for inputs/outputs

4.2 On-Demand (User-Triggered) Workflows

While scheduled services ensure continuous optimization and monitoring, On-Demand Workflows provide the reactive flexibility needed to support user- or system-initiated analytics, simulations, and operational overrides. These workflows are triggered only when explicitly requested by an external actor, typically via the API Gateway, and are often tied to decision-making processes, analysis needs, or urgent interventions that cannot wait for the next scheduled cycle.

In the context of the EVELIXIA platform, many of the grid-level services (e.g., network planning, P2P trading strategies) and advanced building-side analyses are delivered through on-demand workflows.

The main characteristics of an on-demand service include:

- External Triggers via the API Gateway: When an authenticated external request reaches the API Gateway, an on-demand service begins. The request could either originate for an end-user in the Stakeholder Platform or a Northbound API request (actuation signal).
- Context-sensitive execution: In these types of workflows, there are typically input parameters involved. These input parameters should be verified and included in the workflow results.
- Synchronous/Asynchronous flow: Shorter tasks should be returned on the same request while longer tasks, return a workflow ID and its results are stored in the Service Layer Database.
- Interaction with past results: On-demand workflows often utilize past data that are persisted in the Service Layer Database, such as Digital Twin models or aggregated forecasted loads. The workflow should ensure that these are fetched in execution time to avoid timeouts and fatal errors.

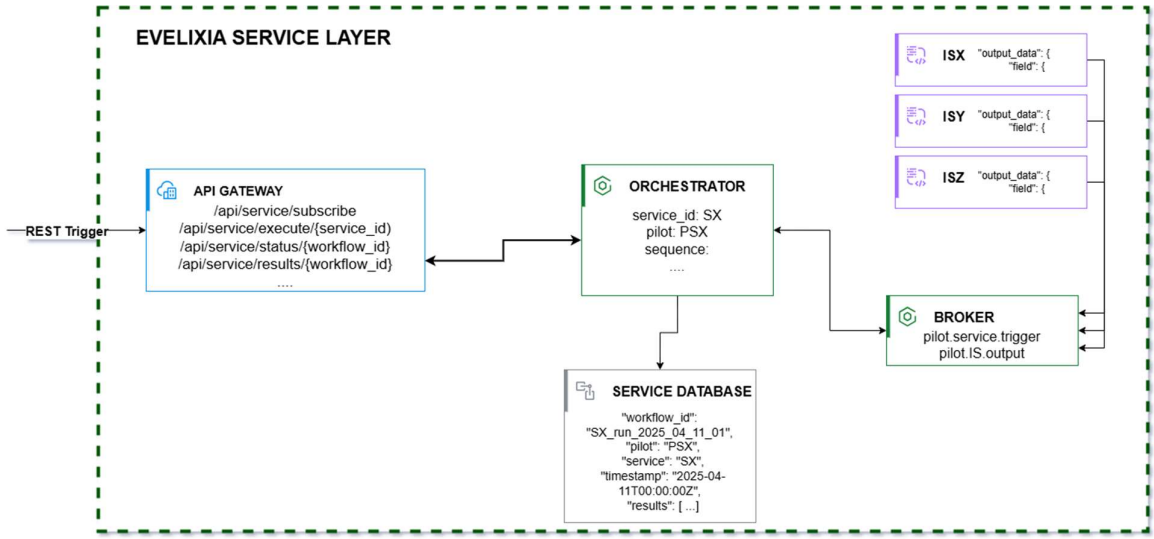


Figure 17 - Components involved in an On-Demand Workflow

For the first version of the Service Layer, we have defined the orchestration logic via:

- REST endpoints exposed by the Orchestration Layer (e.g., /execute, /status).
- In-memory or database-based job tracking.
- Kafka messages used to decouple orchestration from heavy computation.

The configuration files for the on-demand services define the ISs needed, the expected inputs/outputs schema and in later versions, the allowed roles or access level.

4.3 EVELIXIA Services

With the two architectural patterns previously established, this section presents the EVELIXIA Services (which were originally defined in D1.7), as fully implemented and orchestrated workflows within the Service Layer. Each service entry outlines its intent, category (B2G or G2B), type of trigger, involved IS tools, and the expected outputs. Both categories are developed around the triad of Day-Ahead Optimization, Investment Planning, and Maintenance, applied at the Building and District/Grid levels, respectively. Where applicable, services are further decomposed into smaller sub-workflows to reflect their modular structure and highlight specific integration points.

4.3.1 B2G Services

B2G services refer to functionalities that enable the dynamic interaction between the building and the electrical grid, enhancing operational efficiency and flexibility. The EVELIXIA platform engages two main stakeholder groups, building users and building engineers. **Scheduled services** support the building's day-to-day operation, focusing on optimizing energy performance in line with grid requirements and incentive schemes, while ensuring user comfort. This category also includes functions that monitor the condition and performance of building equipment. **On-demand services** are designed to assist users in exploring infrastructure upgrades and investment planning options, considering the added value of grid interaction in a holistic manner. **Table 11** summarizes the types of services offered, along with the corresponding Innovative Solutions (ISs) involved in each case.

Table 11 - B2G Services and sub-services and involved components

Service ID	Service Name	Workflow Type	Involved Components
S1	**Building Operation Daily Optimization**		
S1.0	Day-Ahead Building Forecasting	Scheduled	Middleware → IS5 → IS1, IS3, IS4
S1.1	Day-Ahead Building Optimization - Control Actions	Scheduled	S1.0 → IS10 → Middleware
S1.2	Day-Ahead Building Optimization- Recommendations	Scheduled	S1.0 → S1.1 → IS9
S2	**Building Investment Planning**		
S2.1	SRI Advisor	On-Demand	User Inputs via the Stakeholder Platform, Static Building Data from Database → IS7
S2.2	Building Investment Planning Assistant	On-Demand	Middleware, User Inputs via the Stakeholder Platform, Static Building Data from Database → IS6
S3	**Building Equipment Maintenance**	Scheduled	Middleware → IS2

The execution of all services relies on data retrieved from building IoT systems via the middleware layer, which ensures standardized data access across all included Innovative Solutions. Figure 18 provides a high-level representation of the services workflow within the platform. A detailed, step-by-step execution flow for each service is provided in Section 5. At the current development stage, the focus is on the operational deployment of each service. The initialization phase from the user's perspective will be elaborated in the next version of this delivery, once the services are more mature in terms of required inputs and system integration.

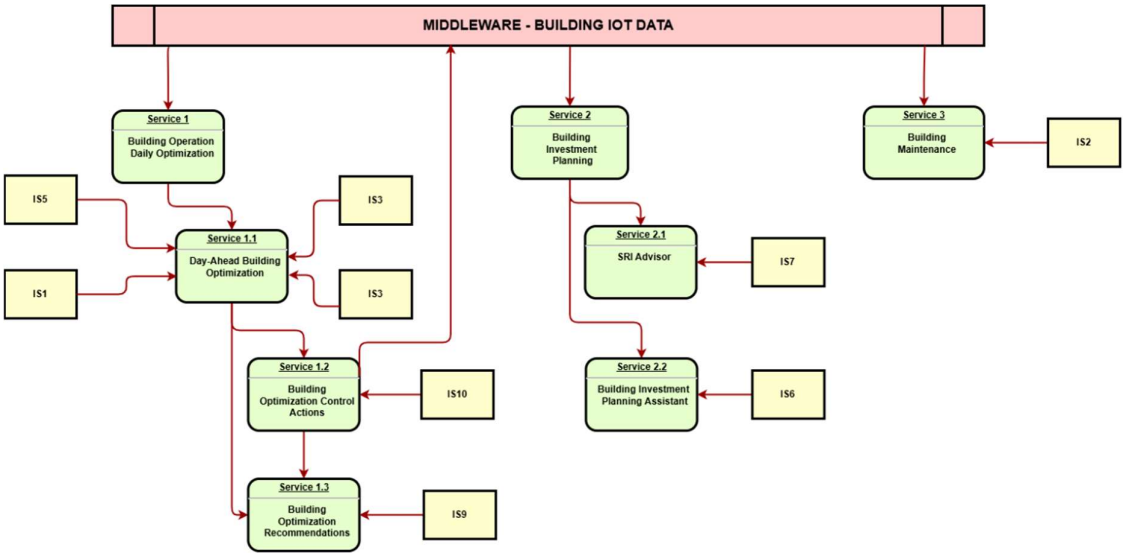


Figure 18 - B2G Services Overview

4.3.2 G2B Services

Following the structure of the previous section, Table 12 analyzes the services provided at the grid level. In this context, the identified stakeholders are the grid operators (for electricity or district heating) and the aggregators of buildings or other grid-scale decentralized energy production and storage assets. Their requirements are addressed by the EVELIXIA platform through a combination of on-demand and scheduled services. Day-ahead operational optimization at the grid level can be deployed either as an on-demand or scheduled service, depending on the extent of engagement from grid-related stakeholders in the decision-making process. Grid Maintenance is performed through continuous grid monitoring and thus is considered scheduled, but an option for the user to perform it upon request is also supported. Grid Investment Planning service is categorized as on-demand and is typically initiated by the local energy network operator.

The execution of these services, as with building-level services, relies heavily on the retrieval of IoT data from the field—in this case, the energy network. In addition to grid-specific data, the Day-Ahead Optimization of energy flows also depends significantly on data generated through the execution of B2G services, as these reflect the anticipated energy demands and operational specifications of the nodes in the energy network. Figure 19 illustrates an workflow overview from the data to the IS level, for the the execution of each G2B service.

Table 12 – G2B Services and involved components

Service ID	Service Name	Workflow Type	Involved Components
S4	**Grid Optimization**		
S4.1	Grid Congestion Management	On-Demand	User Inputs via the Stakeholder Platform, Middleware, IS4 → IS5 → IS12 → IS15
S4.2	Portfolio Management Service	Scheduled	S4.1, IS3, IS4 → IS13
S4.3	P2P Flexibility Trading	Scheduled	S4.1, IS3, IS4 → IS13
S5	**Grid Investment Planning**	On-Demand	User Inputs via the Stakeholder Platform, Middleware → IS15 → IS11
S6	**Grid Maintenance**	Scheduled/ On-Demand	Middleware → IS15 → IS14

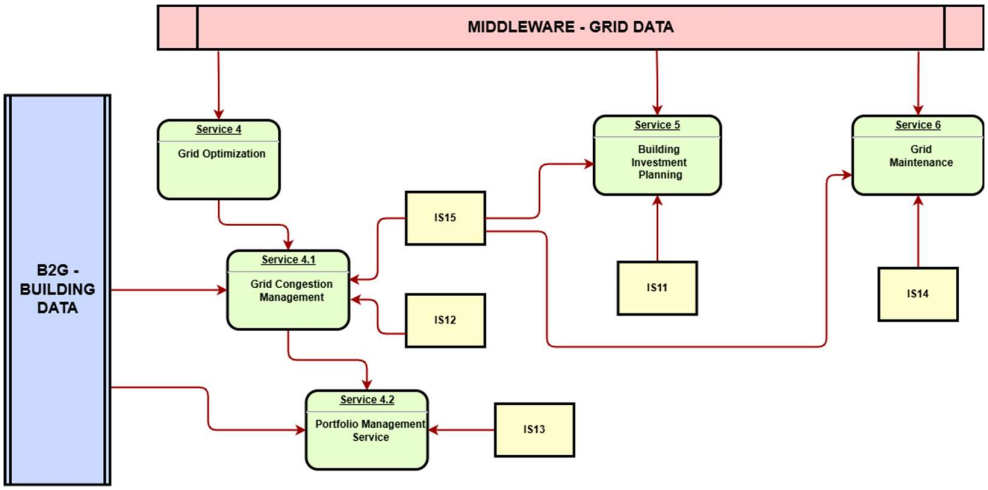


Figure 19 - G2B Services Overview

5 MVP IMPLEMENTATION

This chapter documents the implementation of the Minimum Viable Product (MVP) for the EVELIXIA Service Layer, following the architecture and workflow patterns described in previous sections. The MVP constitutes a working prototype demonstrating key platform capabilities in a centralized, containerized setup using Docker Compose. It validates the design of the orchestration, integration, and communication mechanisms through the envisioned services executions:

1. B2G/ **S1.1 - Day-Ahead Building Forecasting** / Scheduled
2. B2G/ **S1.2 - Day-Ahead Building Optimization - Control Actions** / Scheduled
3. B2G/**S1.3 - Day-Ahead Building Optimization - Recommendations** / Scheduled
4. B2G/ **S2.1 - SRI Advisor** / On-Demand
5. B2G/ **S2.2 – Building Investment Planning Assistant** / On-Demand
6. B2G/ **S3 - Building Maintenance** / Scheduled
7. G2B/ **S4.1 - Grid Congestion Management** / On-Demand
8. G2B/ **S4.2 - Portfolio Management Service** / On-Demand
9. G2B/ **S5 - Grid Investment Planning** / On-Demand
10. G2B/ **S6 - Grid Maintenance** / On-Demand

5.1 Implementation Overview

Each Service Layer component was implemented using open-source tools to ensure interoperability, as described in Chapter 3. Table 13 lists an overview of the implementation.

Table 13 Technology Stack Overview

Component	Technology Stack
Message Brokerage System	Apache Kafka + Zookeeper
API Gateway	Node.js/Express (5) + Kong OSS
Orchestration Layer	Node.js/Express (5) + Docker Compose
Service Layer Database	MongoDB
Innovative Solutions	Docker Containers

A central JSON topic registry has been used by the Orchestration Layer to ensure topic consistency and topics are referenced only through this configuration file. Additionally, all services are defined in a Docker Compose (YAML) file that specify environment variables and other necessary configurations for the internal communication of the Layer. No database ports are directly exposed outside the internal environment and any variables are passed for the Broker to ensure interoperability.

5.2 S1.0 - Day-Ahead Building Optimization - Forecasting

This section describes a complete execution of a B2G service that is scheduled to run daily. The service selected is Day-Ahead Building Optimization - Forecasting, implemented in the first version using IS containers and the full-Service Layer stack. The workflow leverages all core components (API Gateway, Orchestrator, Kafka Broker, and multiple ISs) to coordinate a multi-step optimization and simulation flow, returning results to the user and storing them for future reference.
The high-level Use Case for this service can be found in Table 14:

Table 14 - S1.0 Day-Ahead Building Forecasting

Service Use Case Name	**S1.0 – Day-Ahead Building Optimization - Forecasting**
Type	B2G
Architectural Pattern	Scheduled
Actors Involved	**Main Actor:** Building Engineers **Secondary Actor:** Building Owner, Tenant, User, Facility Manager
Brief Description	1. Every day at 00:00, this service is automatically triggered, by requesting new IoT Building Data from the Middleware. 2. The **Building Energy Modelling and Simulation (IS5)** is triggered by the new IoT data to create the model for the building and the energy simulation**.** 3. The simulation results and the IoT data are fed to the **IAQ (IS1), Demand (IS3)** and **Flex Forecasting (IS4)** to be used for predicting the behaviour of the building for the next 24 hours. 4. The building day-ahead operation profile is generated and the analytic results for the next 24-hours are stored on the EVELIXIA platform for later use.
Alternate Flow/ exceptions	• In the case that there are not available IoT data the training of the forecasting ISs takes place only with the data generated from the **Building Energy Modelling and Simulation (IS5)**
Assumptions and Pre-conditions	• Existence of building documentation • Existence of historic data
Trigger	• Daily Schedule
Goal	• The building model is ready to be used by the rest of the IS in Service 1. • The Forecasting ISs have been trained and are able to generate forecast reports.
Pre-Conditions (UCs)	• None.
Post-Conditions (UCs)	• S1.1 - Day-Ahead Building Optimization - Control Actions • S1.2 - Day-Ahead Building Optimization - Recommendations

For the purposes of this deliverable and to stay within its scope, the following assumptions were made:

- Since the integration between the Service Layer and Middleware will be implemented in T5.1, we have not received any real IoT Pilot data for this workflow. Instead, a test stream has been utilized for this implementation.

- For the same reason, we have also considered that the Building Energy Modelling Digital Twin (IS5) has been initialized previously with the required building BIM files and static metadata from the Pilot Site. The initial Pilot Site for this current execution is the Greek Pilot Site (PS5).
- Similarly, we consider that the training of the Analytical Tools (IS1, IS3 and IS4) has already been performed using historical data during their development phase.
- The user does not need to initiate the analysis through the Stakeholder Platform since this is a scheduled workflow that will begin automatically at 00:00. We have chosen this time in order to produce 24-hour timeseries for the whole day ahead.

The involved components are:

- **Stakeholder Platform** (User Input / UI)
- **API Gateway**
- **Middleware** (test stream)
- **Orchestration Layer**
- **Message Broker**
- **IS5**: Building Energy Modelling and Simulation
- **IS1**: Indoor Air Quality measurement
- **IS3**: Demand Forecasting
- **IS4**: Flex Forecasting
- **Service Layer Database**

Figure 20 illustrates the cross-functional flowchart of the service's workflow:

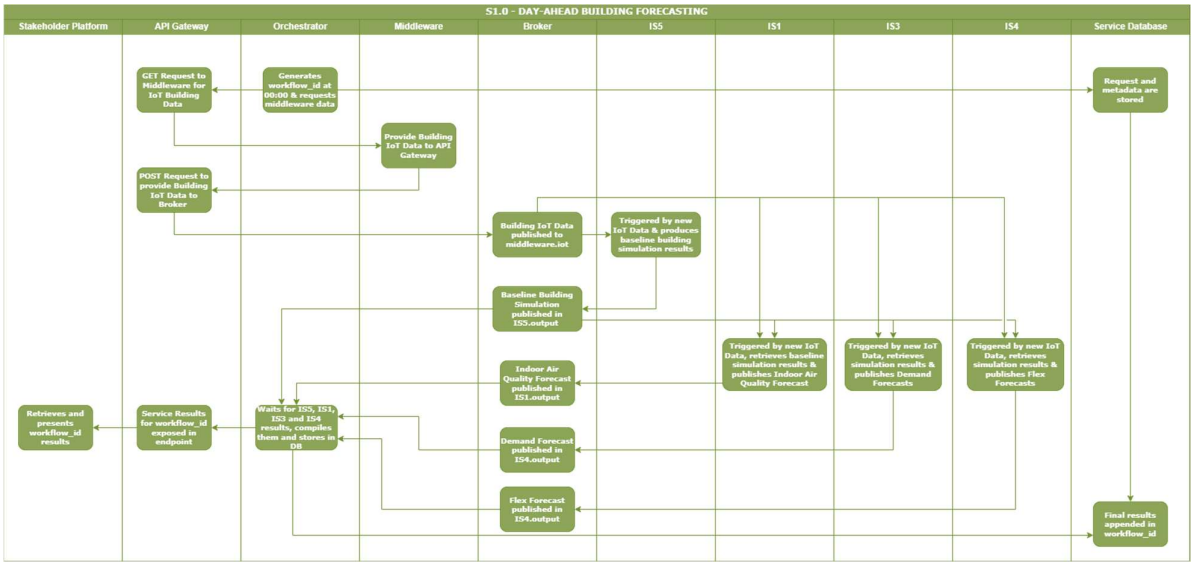


Figure 20: Day-Ahead Building Optimization - Forecasting – Workflow

1. **Orchestrator Triggers Workflow Automatically (00:00)**
   - The Orchestrator generates a new workflow_id
   - Queries the Middleware to retrieve the latest Building IoT Data for the current day
   - Prepares the orchestration context for the scheduled S1.0 service
2. **API Gateway Retrieves Building IoT Data from Middleware**
   - The API Gateway fetches IoT readings from Middleware (simulated test stream in MVP)
   - The data is received and posted into the Service Layer via the topic: gr1.middleware.iot
3. **Broker Publishes IoT Data to Relevant Topic**
   - Kafka publishes the structured JSON data received from Middleware into the topic gr1.middleware.iot
   - This acts as a trigger for all downstream IS tools that subscribe to the topic

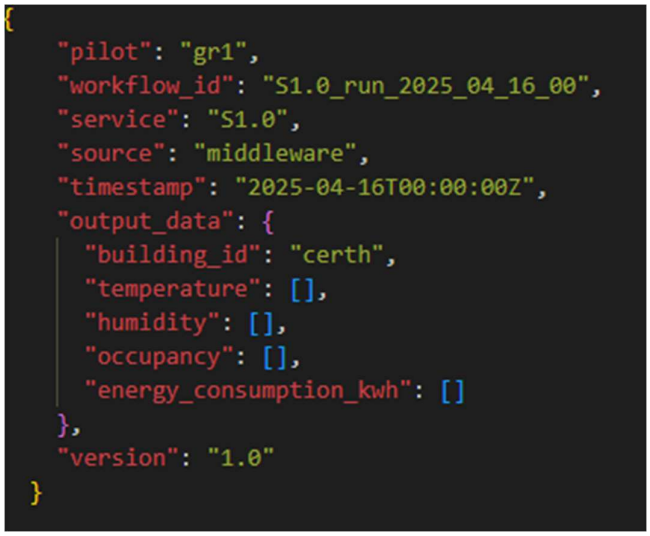


Figure 21 - Sample IoT payload to be used by S1

4. **IS5: Building Simulation Triggered**
   - IS5 listens to gr1.middleware.iot
   - It retrieves the latest building data and runs a baseline building energy simulation
   - The output (hourly building energy metrics for the next 24h) is published to: gr1.IS5.output

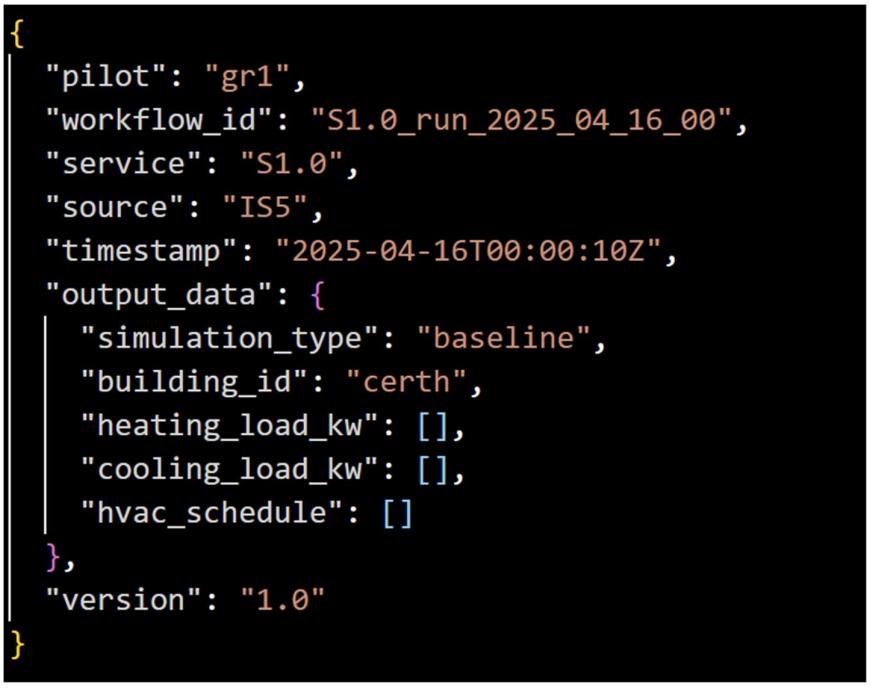


Figure 22 - Sample payload for IS5 for S1.0

5. **IS1, IS3, IS4 Triggered by New IoT + Simulation Data**
    - Each forecasting tool (IS1 for IAQ, IS3 for Demand, IS4 for Flexibility) is triggered by the arrival of data on middleware.iot and IS5.output
    - They compute their respective 24h predictions and publish to:
        - gr1.IS1.output – Indoor Air Quality Forecast
        - gr1.IS3.output – Demand Forecast
        - gr1.IS4.output – Flexibility Forecast


```
{
  "pilot": "gr1",
  "workflow_id": "S1.0_run_2025_04_16_00",
  "service": "S1.0",
  "source": "IS1",
  "timestamp": "2025-04-16T00:00:20Z",
  "output_data": {
    "building_id": "certh",
    "iaq_forecast": {
      "co2_ppm": [],
      "ventilation_flex": [],
      "window_prediction": []
  },
  "version": "1.0"
}
```


Figure 23 - Sample payload for IS1 output

```
{
  "pilot": "gr1",
  "workflow_id": "S1.0_run_2025_04_16_00",
  "service": "S1.0",
  "source": "IS3",
  "timestamp": "2025-04-16T00:00:25Z",
  "output_data": {
    "building_id": "certh",
    "demand_forecast_kwh": [],
    "MAPE": 0.12
  },
  "version": "1.0"
}
```


Figure 24 - Sample payload for IS3


```
{
  "pilot": "gr1",
  "workflow_id": "S1.0_run_2025_04_16_00",
  "service": "S1.0",
  "source": "IS4",
  "timestamp": "2025-04-16T00:00:30Z",
  "output_data": {
    "building_id": "certh",
    "flex_potential_kw": [],
    "time_slots": []
  },
  "version": "1.0"
}
```


Figure 25 - Sample payload for IS4

6. **Orchestrator Collects Results and Stores in DB**
   - The orchestrator listens to all relevant output topics (IS1, IS3, IS4, IS5)
   - Once all results are received, it aggregates them and stores them under the current workflow_id in the Service Layer Database
   - Metadata (e.g., timestamps, pilot, service_id, etc.) is attached to each document

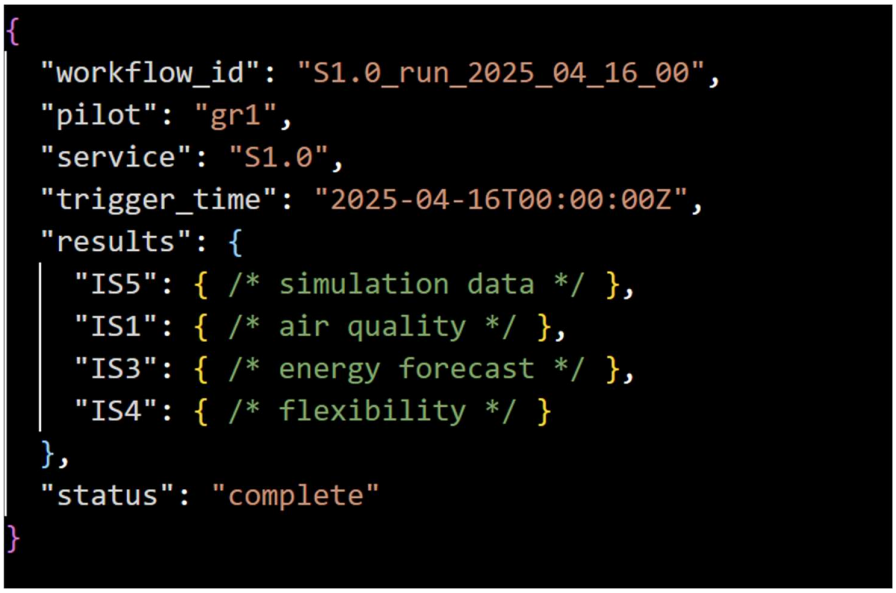


Figure 26 - Service Database entry for successful S1.0 workflow execution

7. **Service Results Exposed to Frontend**
   - The orchestrator exposes the full output set via an endpoint for workflow results
   - The Stakeholder Platform can retrieve the results using the workflow_id
8. **User Accesses Results via Stakeholder Platform**
   - Results are visualized as charts and recommendations for engineers/operators
   - Data can be used directly or routed to subsequent services (e.g., S1.1)

5.3 S1.1 - Day-Ahead Building Optimization – Equipment Control

This section describes a complete execution of a B2G service that is scheduled to run daily. The service selected is Day-Ahead Building Optimization - Control Actions, implemented in the first version using IS containers and the full-Service Layer stack. The workflow leverages all core components (API Gateway, Orchestrator, Kafka Broker, and multiple ISs) to coordinate a multi-step optimization and simulation flow, returning results to the user and storing them for future reference.

The high-level Use Case for this service can be found in Table 15:

Table 15 – S1.1 - Day-Ahead Building Optimization – Equipment Control

Service Name	**5.3 S1.1 - Day-Ahead Building Optimization – Equipment Control**
Type	B2G
Architectural Pattern	Scheduled
Actors Involved	**Main Actor:** Building Owner, Tenant, User, Facility Manager **Secondary Actor:** Building Engineers
Brief Description	1. Every day this service is automatically triggered by the daily forecasts and simulations produced by **S1.1**. 2. The forecasted values trigger the **Continuous Energy Performance Management (IS10)** to retrieve the already existing IoT data from the IoT topic. 3. Utilizing the IoT Data and the forecasts from S1.1, **IS10** generates indoor trends and an optimal schedule for the operation of the HVAC equipment for the next day. 4. The indoor trends are forwarded to the end-user via the Stakeholder Platform. 5. The control set-points are sent to the HVAC equipment via the Middleware to be implemented the next day.
Alternate Flow/ exceptions	If by 01:00 there are no new S1.1 results or IoT data to automatically trigger **IS10**, the orchestrator pulls data from the previous day to provide to the tool.
Assumptions and Pre-conditions	• The building model has to be initialized and updated • The building needs to have the appropriate control equipment for its equipment and devices.
Trigger	• Daily Schedule • S1.1 Results
Goal (Successful End Condition)	• The end user is informed for the actions that needs to be implemented to operate efficiently their building. The equipment with control capabilities operate according to the set-points generated by the EVELIXIA platform.
Pre-Conditions (UCs)	• S1.1. Day-Ahead Building Forecasting
Post-Conditions(UCs)	• S1.3 - Day-Ahead Building Optimization - Recommendations

For the purposes of this deliverable and to stay within its scope, the following assumptions were made:

- Since the integration between the Service Layer and Middleware will be implemented in T5.1, we have not received any real IoT Pilot data for this workflow. Instead, a test stream has been utilized for this implementation.
- Additionally, for the same reason, the actuation signal that this service will produce (control actions), will not be provided to the Middleware at this version.
- The user does not need to initiate the analysis through the Stakeholder Platform since this is a scheduled workflow that will begin automatically after the completion of S1.1 or at a scheduled time (01:00). We have chosen this time in order to produce 24-hour timeseries for the whole day ahead.
- We also assume that the S1.1 has been finalized successfully and the results of IS5, IS1, IS3 and IS4, that are required for the execution of S1.2, are already existing in the Message Broker. In case of failed S1.1 workflow, the previous day results will be utilized, but this is not a scenario we will not expand in this in this section.

The involved components are:

- **Stakeholder Platform** (User Input / UI)
- **API Gateway**
- **Middleware** (test stream)
- **Orchestration Layer**
- **Message Broker**
- **IS10**: Continuous Energy Performance Management (CEPM)
- **Service Layer Database**

Figure 27 illustrates the cross-functional flowchart of the service's workflow:

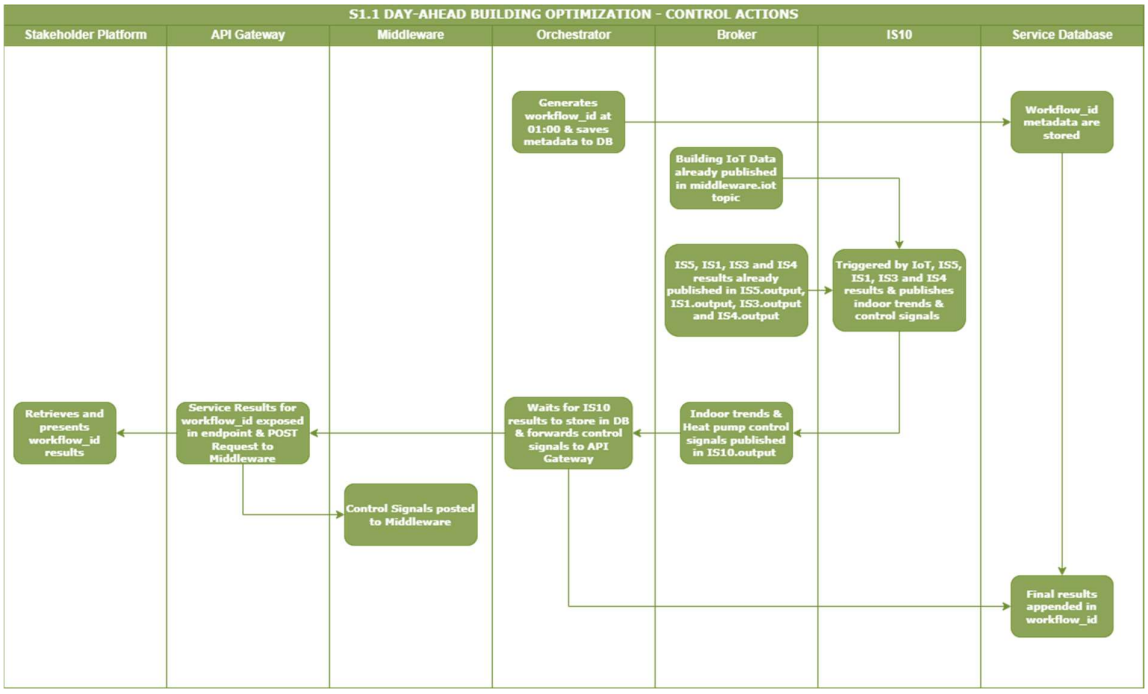
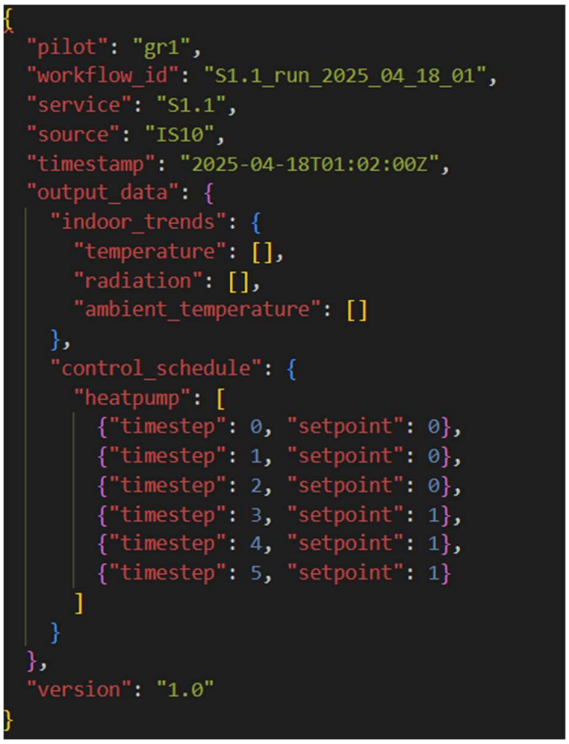


Figure 27 - S1.1 - Day-Ahead Building Optimization – Equipment Control Workflow

1. **Daily Trigger and Metadata Generation**: At 01:00, the orchestrator generates a new workflow_id and stores initial metadata in MongoDB, marking the start of the control action service.
2. **IoT Data and Forecast Availability**: At this point, building IoT data has already been published to the topic gr1.middleware.iot by the Middleware. Similarly, the results from the forecasting ISs (IS5, IS1, IS3, IS4) are already present in the topics:
   - gr1.IS5.output (baseline simulation)
   - gr1.IS1.output (indoor air quality forecast)
   - gr1.IS3.output (demand forecast)
   - gr1.IS4.output (flex forecast)
3. **IS10 Triggered by Data Availability**: The Continuous Energy Performance Management IS (IS10) is triggered when it detects that the required data from the IoT stream and the forecasting ISs are available. It retrieves this information from the broker and performs its optimization logic.
4. **IS10 Publishes Control Actions**: IS10 generates:
   - Indoor environmental trends (e.g., temperature, comfort levels)
   - HVAC control schedules (e.g., hourly set-points for the following day)

- It publishes these in the topic gr1.IS10.output.



```
{
  "pilot": "gr1",
  "workflow_id": "S1.1_run_2025_04_18_01",
  "service": "S1.1",
  "source": "IS10",
  "timestamp": "2025-04-18T01:02:00Z",
  "output_data": {
    "indoor_trends": {
      "temperature": [],
      "radiation": [],
      "ambient_temperature": []
    },
    "control_schedule": {
      "heatpump": [
        {"timestep": 0, "setpoint": 0},
        {"timestep": 1, "setpoint": 0},
        {"timestep": 2, "setpoint": 0},
        {"timestep": 3, "setpoint": 1},
        {"timestep": 4, "setpoint": 1},
        {"timestep": 5, "setpoint": 1}
      ]
    }
  },
  "version": "1.0"
}
```


Figure 28 - Example payload of IS10

5. **Orchestrator Receives and Stores IS10 Results:** The orchestrator, subscribed to gr1.IS10.output, consumes the results, attaches them to the workflow metadata, and stores everything in the Service Layer database.
6. **Control Signal Transmission via API Gateway**: The orchestrator posts the control signals to the Middleware via the API Gateway for actuation. This step is mocked in the MVP and will be handled via a secure HTTP endpoint in T5.1.

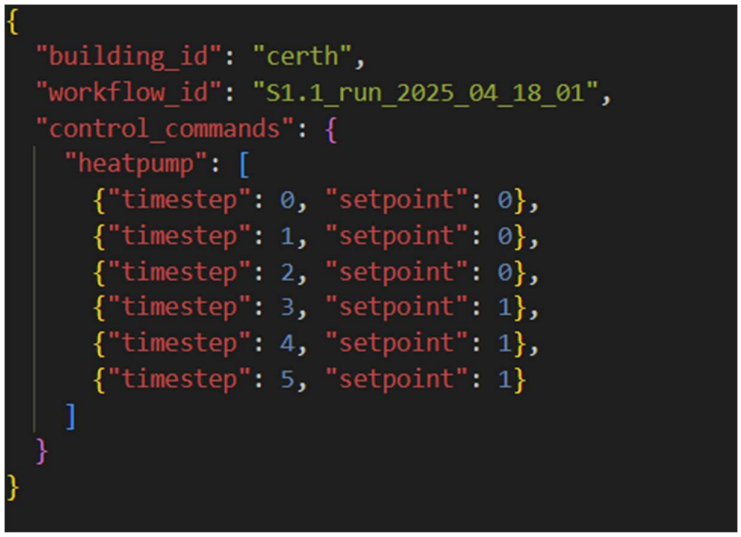


Figure 29 - Example payload for the control signal provided to the middleware (to be implemented in T5.1)

7. **Results Made Available to UI**: The results are exposed via the orchestrator's results endpoint /api/service/status/{workflow_id} and made available to the Stakeholder Platform.

5.4 S1.2 - Day-Ahead Building Optimization - User Recommendations

This section describes a complete execution of a B2G service that is scheduled to run daily. The service selected is Day-Ahead Building Optimization - Recommendations, implemented in the first version using IS containers and the full-Service Layer stack. The workflow leverages all core components (API Gateway, Orchestrator, Kafka Broker, and multiple ISs) to coordinate a multi-step optimization and simulation flow, returning results to the user and storing them for future reference.

The high-level Use Case for this service can be found in the Table 16:

Table 16 - Day-Ahead Building Optimization – User Recommendations

Service Name	**S1.2. Day-Ahead Building Optimization – User Recommendations**
Type	B2G
Architectural Pattern	On-demand
Actors Involved	**Main Actor:** Building Owner, Tenant, User, Facility Manager **Secondary Actor:** Building Engineers
Brief Description	1. Every day this service is automatically triggered by the results produced by **S1.2**. 2. The S1.2 results trigger the **Proactive Demand Planning (IS9)** to retrieve the already existing IoT data from the IoT topic, as well as, the forecasted values from IS1, IS3 & IS4 and the HVAC schedule from IS10 that exist in their own topics. 3. **IS9** generates recommendations for the optimization of the operation of the building's flexible loads for the next day. 4. The recommendations are forwarder to the end-user via the Stakeholder Platform.
Alternate Flow/ exceptions	• If by 01:30 there are no new S1.2 and S1.1 results or IoT data to automatically trigger **IS9**, the orchestrator pulls data from the previous day to provide to the tool.
Assumptions and Pre-conditions	• The building model has to be initialized and updated
Trigger	• Daily Schedule • S1.1 Results
Goal	The end user is informed for the actions that needs to be implemented to operate efficiently their building.
Pre-Conditions (UCs)	• S1.0. Day-Ahead Building Forecasting • S1.2 - Day-Ahead Building Optimization - Control Actions
Post-Conditions (UCs)	• None.

For the purposes of this deliverable and to stay within its scope, the following assumptions were made:

- Since the integration between the Service Layer and Middleware will be implemented in T5.1, we have not received any real IoT Pilot data for this workflow. Instead, a test stream has been utilized for this implementation.
- The user does not need to initiate the analysis through the Stakeholder Platform since this is a scheduled workflow that will begin automatically after the completion of S1.2 or at a scheduled time (01:30). We have chosen this time according to the approximate execution time of S1.2 in order to produce 24-hour timeseries for the whole day ahead.
- We also assume that both S1.1 and S1.2 have been finalized successfully and the results of IS5, IS1, IS3, IS4 and IS10, that are required for the execution of S1.3, are already existing in the Message Broker. In case of failed S1.1 or S1.2 workflow, the previous day results will be utilized, but this is not a scenario we will not expand in this in this section.

The involved components are:

- **Stakeholder Platform** (User Input / UI)
- **API Gateway**
- **Middleware** (test stream)
- **Orchestration Layer**
- **Message Broker**
- **IS9**: Proactive Demand Planning
- **Service Layer Database**

Figure 30 illustrates the cross-functional flowchart of the service's workflow:

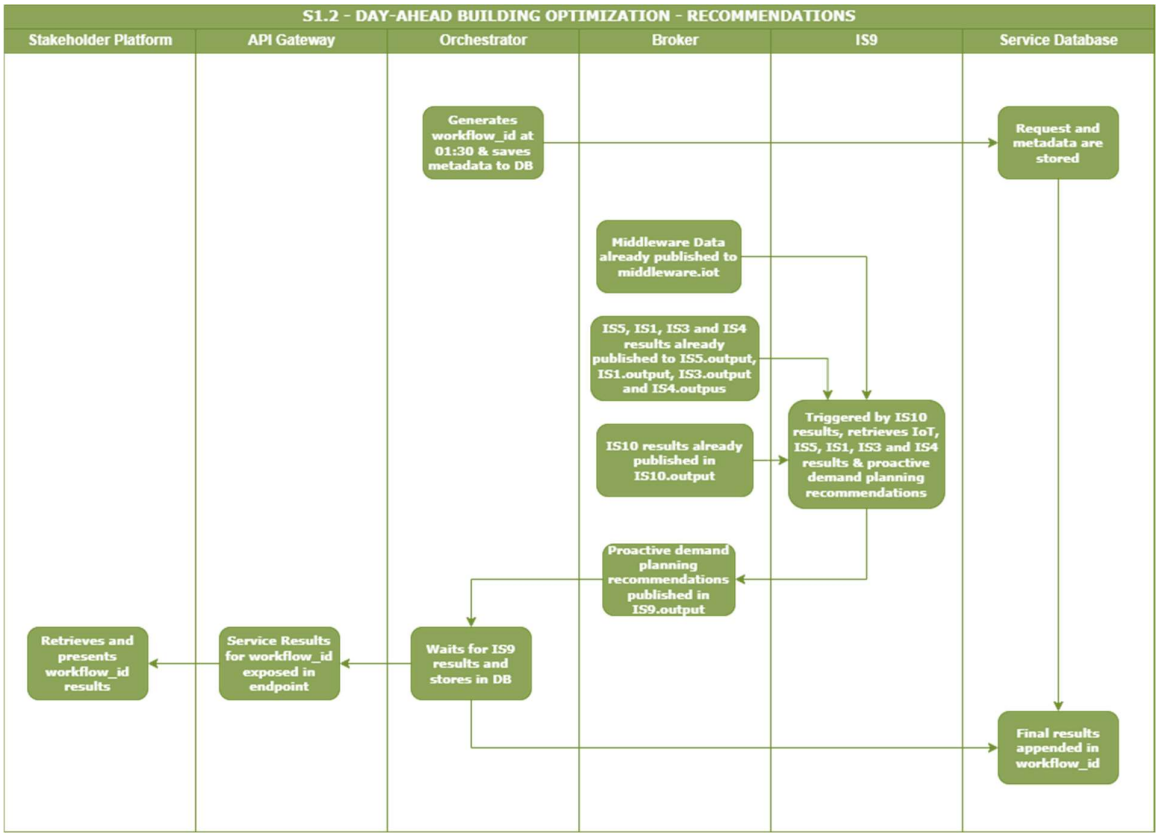


Figure 30 - Day-Ahead Building Optimization – User Recommendations Workflow

1. **Scheduled Trigger and Metadata Initialization**: At 01:30, the orchestrator assigns a new workflow_id, stores metadata in the database, and marks the workflow as ready. It is assumed that results from previous services (IS5, IS1, IS3, IS4, and IS10) are already present in their respective topics.
2. **IS9 Triggered by Prior Results**: The Proactive Demand Planning module (IS9) is triggered by the availability of IS10's control signals (from gr1.IS10.output). Upon trigger, IS9 retrieves:
   - IoT Data from gr1.middleware.iot
   - Forecasting Data from gr1.IS1.output, gr1.IS3.output, and gr1.IS4.output
   - HVAC Control Signals from gr1.IS10.output
3. **IS9 Publishes Building Optimization Recommendations**: IS9 processes all retrieved information and generates a set of proactive planning recommendations for flexible load optimization, which are published to the topic gr1.IS9.output.

```
{
  "pilot": "gr1",
  "workflow_id": "S1.2_run_2025_04_18_01",
  "service": "S1.2",
  "source": "IS9",
  "timestamp": "2025-04-18T01:32:00Z",
  "building_id": "certh",
  "output_data": {
    "recommendations": {
      "decision": [],
      "baseline": [],
      "down": [],
      "up": [],
      "price": []
    },

    "performance": {
      "residual_energy": 1.31,
      "monetary_cost": 2.51,
      "thermal_deviation": 0,
      "unit": "%"
    }
  },
  "version": "1.0"
}
```


Figure 31 - Sample payload for the outputs of IS9

4. **Orchestrator Collects and Persists Results**: The orchestrator, subscribed to gr1.IS9.output, gathers the recommendation output and stores it into MongoDB under the corresponding workflow_id.
5. **Results Made Available to Frontend**: The orchestrator exposes the results under /api/service/status/{workflow_id}. The Stakeholder Platform retrieves and visualizes the final results.

5.5 S2.1 – Building Investment Planning - SRI Advisor

This section describes a complete execution of a B2G service that is triggered on-demand by the user. The service selected is SRI Advisor, implemented in the first version using IS containers and the full-Service Layer stack. The workflow leverages all core components (API Gateway, Orchestrator, Kafka Broker, and multiple ISs) to coordinate a multi-step analytical flow, returning results to the user and storing them for future reference.

The high-level Use Case for this service can be found in Table 17:

Table 17 - S2.1 Building Investment Planning - SRI Advisor

Service Name	S2.1 Building Investment Planning - SRI Advisor
Type	B2G
Architectural Pattern	On-demand
Actors Involved	**Main Actor:** Engineers, Building designers **Secondary Actor:** Building user / Building Operator
Brief Description	1. The building engineer inserts the EVELIXIA Building Investment Planning Toolbox and requests to use the service**.** 2. The user inputs a set of parameters related to the existing SRI assessment of the building, the characteristics of the building and the preferences for the generated set of recommendations. 3. The engineer uploads the data from the SRI assessment and the preferences that the building owner has indicated. The building characteristics can be retrieved from the existing information on the EVELIXIA platform or added by the engineer via a dedicated user interface. 4. The **SRI Advisor Tool (IS7)** is triggered by the user inputs**,** and generates a set of upgrades, tailored to the user preferences. 5. The generated report is stored to the platform and forwarded to the building engineer.
Assumptions and Pre-conditions	• An SRI assessment should exist.
Alternate Flows	• None
Trigger	• The building engineer requests to use the service.
Goal	• The building engineer gets a stepwise set of upgrades that indicates the installation of the equipment to upgrade the smartness of their building.
Pre-Conditions (UCs)	• None
Post-Conditions (UCs)	• None

For the purposes of this deliverable and to stay within its scope, the following assumptions were made:

- Static building information (e.g., BIM) are, for this version, stored in the Service Database. We will also explore storing this information on the Digital Building Logbook (IS16).
- The user initiates the SRI by inputting a set of parameters through a questionnaire form in the Stakeholder Platform and hitting “Run”.

The involved components are:

- **Stakeholder Platform** (User Input / UI)
- **API Gateway**
- **Middleware** (test stream)
- **Orchestration Layer**
- **Message Broker**
- **IS7**: SRI Advisor
- **Service Layer Database**

Figure 32 illustrates the cross-functional flowchart of the service's workflow:

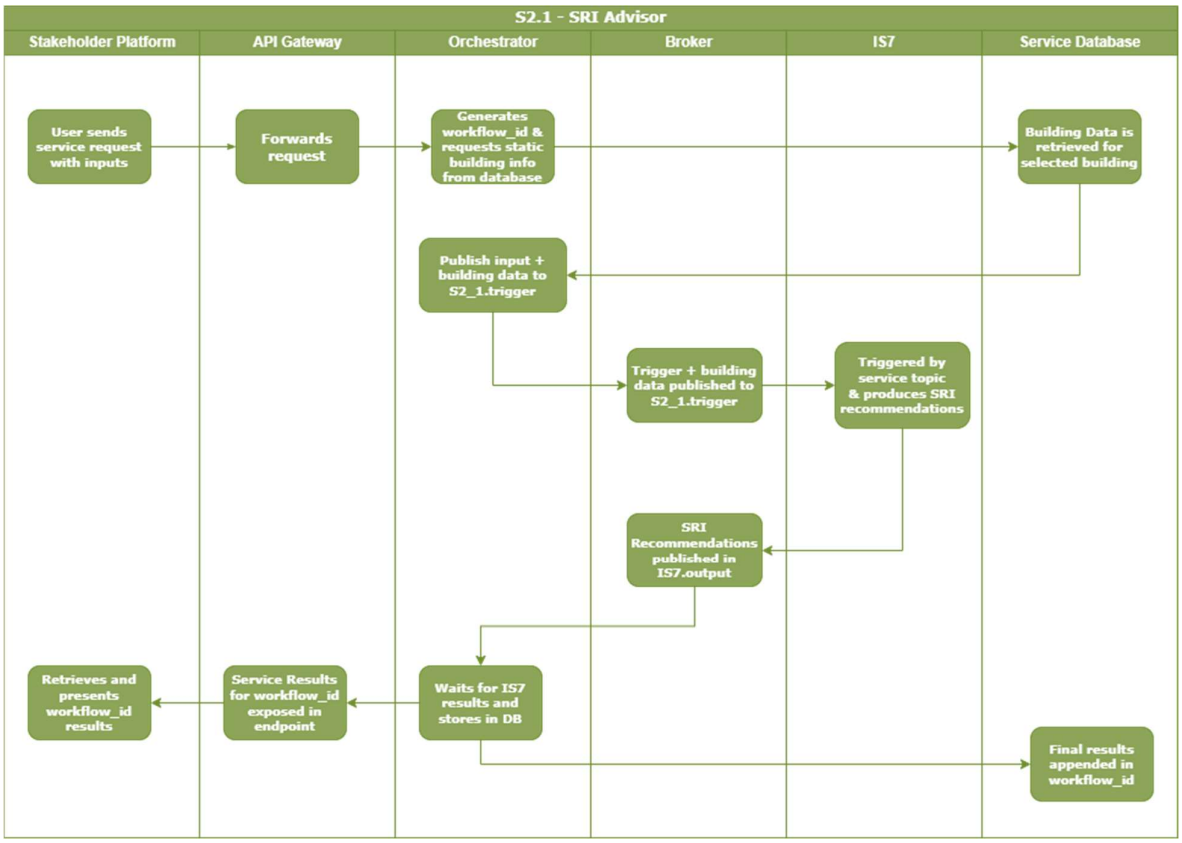


Figure 32 - Building Investment Planning - SRI Advisor Workflow

1. User Sends Request: The building engineer accesses the EVELIXIA Building Investment Planning toolbox and fills out a questionnaire with parameters derived from an SRI assessment, preferences, and optionally selects a building from the platform.

2. API Gateway Routes Request: The request is posted to /api/service/execute/s2_1 with user input. The API Gateway validates and forwards the request internally to the Orchestrator.

3. Orchestrator Generates Workflow and Fetches Metadata: Generates a unique workflow_id for the session.

- Pulls static building metadata from the database based on the selected building ID.
- Aggregates user input and building metadata.
- Publishes the structured trigger payload to Kafka topic: gr1.s2_1.trigger.


```
{
  "pilot": "gr1",
  "workflow_id": "S2.1_run_2025_04_18_01",
  "service": "S2.1",
  "building_id": "certh",
  "timestamp": "2025-04-18T08:00:00Z",
  "user_inputs": {
    "user_preference": "flexibility",
    "assessment_preference": "sri_1"
  },
  "building_static": {
    "country": "greece",
    "sector": "non-residential",
    "heating_fields": [],
    "cooling_fields": [],
    "ventilation_fields": [],
    "water_fields": [],
    "lighting_fields": [],
    "electricity_fields": [],
    "ev_fields": []
  },
  "version": "1.0"
}
```


Figure 33 - Sample S2.1 trigger payload

4. Broker Receives Trigger Message: Kafka receives the message and stores it on the s2_1.trigger topic, available to IS7.

5. IS7 Consumes Trigger and Executes Analysis: IS7, listening on s2_1.trigger, retrieves the payload, processes the building metadata and SRI parameters, and computes a tailored upgrade plan. Results are published to the s2_1.output topic.

```
{
  "pilot": "gr1",
  "workflow_id": "S2.1_run_2025_04_18_01",
  "service": "S2.1",
  "source": "IS7",
  "timestamp": "2025-04-18T08:00:05Z",
  "building_id": "certh",
  "output_data": {
    "sri_scores": {
        "h-1a": 8.6,
        "h-1c": 10.4,
        "h-2a": 3.8,
        "h-3": 6.6,
        "dhw-1b": 5.3,
        "dhw-3": 2.3,
        "v-1a": 7.2,
        "v-6": 5
    }
  },
  "version": "1.0"
}
```


Figure 34 - Sample payload for the results of IS7

6. Orchestrator Gathers Results and Stores: The orchestrator consumes results from s2_1.output, validates their format, and stores them in the Results collection under the generated workflow_id.


```
{
  "workflow_id": "S2.1_run_2025_04_18_01",
  "pilot": "gr1",
  "service": "S2.1",
  "building_id": "certh",
  "trigger_time": "2025-04-18T08:00:00Z",
  "user_id": "evelixia_user_13",
  "user_inputs": {
    "user_preference": "flexibility",
    "assessment_preference": "sri_1"
  },
  "building_static": {
    "country": "greece",
    "sector": "non-residential",
    "heating_fields": [],
    "cooling_fields": [],
    "ventilation_fields": [],
    "water_fields": [],
    "lighting_fields": [],
    "electricity_fields": [],
    "ev_fields": []
  },
  "results": {
    "IS7": { /* results from IS7*/ }
  },
  "status": "complete",
  "version": "1.0"
}
```


Figure 35 - Sample payload for the database entry for S2.1

7. API Gateway Serves Results: Upon user request (GET /api/service/results/{workflow_id}), the gateway returns the structured SRI recommendations to the Stakeholder Platform.

8. User Views the Upgrade Report: The building engineer views a set of ranked equipment upgrade suggestions, including estimated costs, impact on smartness, and expected return.

5.6 S2.2 – Building Investment Planning - VERIFY

This section describes a complete execution of a B2G service that is triggered on-demand by the user. The service selected is the Building Investment Planning Assistant, implemented in the first version using IS containers and the full-Service Layer stack. The workflow leverages all core components (API Gateway, Orchestrator, Kafka Broker, and multiple ISs) to coordinate a multi-step analytical flow, returning results to the user and storing them for future reference.

The high-level Use Case for this service can be found in Table 18:

For the purposes of this deliverable and to stay within its scope, the following assumptions were made:

- Static building information (e.g., BIM) are, for this version, stored in the Service Database. We will also explore storing this information on the Digital Building Logbook (IS16).
- Since the integration between the Service Layer and Middleware will be implemented in T5.1, we have not received any real IoT Pilot data for this workflow. Instead, a test stream has been utilized for this implementation.
- The user initiates the service by inputting a set of pre-defined upgrade scenarios that they want to investigate and compare through a questionnaire form in the Stakeholder Platform and hitting “Run”.

The involved components are:

- **Stakeholder Platform** (User Input / UI)
- **API Gateway**
- **Middleware** (test stream)
- **Orchestration Layer**
- **Message Broker**
- **IS6**: Building Investment Planning Assistant
- **Service Layer Database**

Table 18: Building Investment Planning - VERIFY

Service Name	S2.2 Building Investment Planning - VERIFY
Type	B2G
Architectural Pattern	On-demand
Actors Involved	**Main Actor:** Engineers, Building designers **Secondary Actor:** Building user / Building Operator
Brief Description	1. The building engineer inserts the EVELIXIA Building Investment Planning Toolbox and requests to use the service. 2. The engineer develops the model of the building by inserting a set of related documentation. 3. This information can be retrieved from the existing information on the EVELIXIA platform or added by the engineer via a dedicated user interface. 4. After having constructed the building's model, the engineer sets a series of retrofitting scenarios that they want to investigate and the **Investment Planning Assistant (IS6)** performs the LCC analysis, calculates financial KPIs and performs a comparative analysis between the scenarios for the selected analysis period-timespan and generates a report. 5. The generated report is stored to the platform and forwarded to the building engineer.
Assumptions and Pre-conditions	• The required input data to setup the building model, and operational data regarding its systems, components and energy performance should be provided in advanced. The sensor data should be coherent and accurate. • The engineer should provide a set of pre-defined upgrade scenarios that wants to investigate and compare.
Alternate Flows	• None
Trigger	• The building engineer requests to use the service.
Goal	• The building engineer gets a report with the LCA/LCC analysis and the resulting quantitative metrics for each scenario.
Pre-Conditions (UCs)	• None
Post-Conditions (UCs)	• None

Figure 36 illustrates the cross-functional flowchart of the service's workflow:

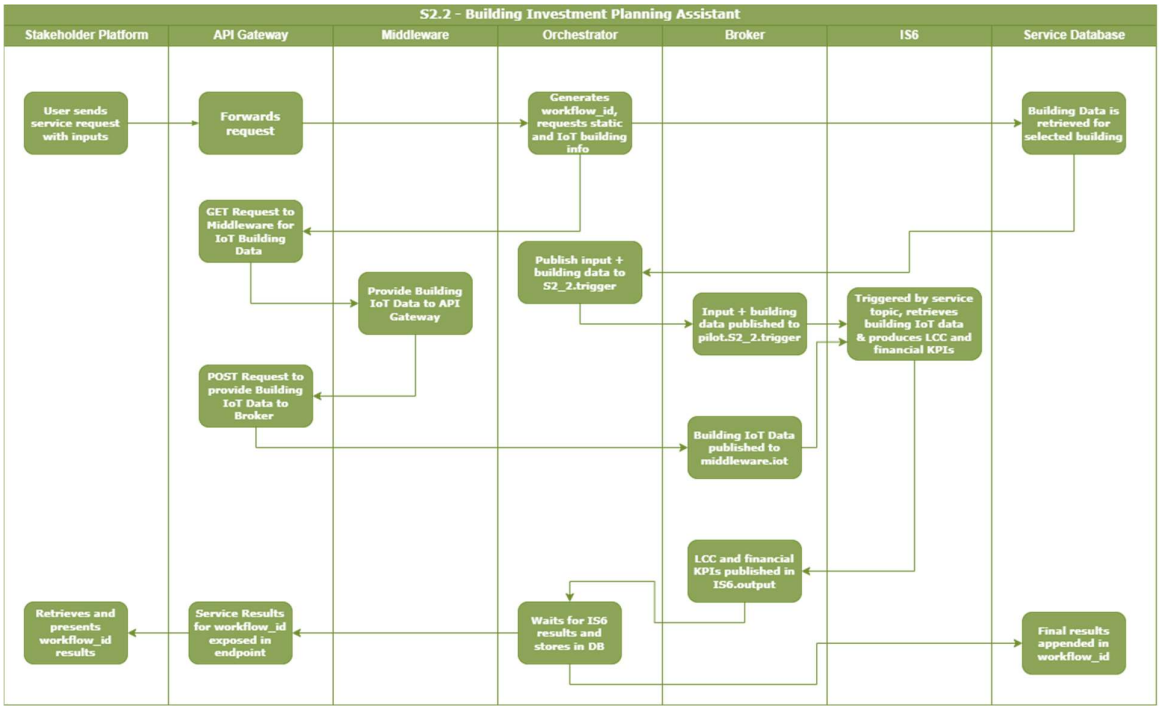


Figure 36 - Building Investment Planning - VERIFY Workflow

1. **User Sends Request via Stakeholder Platform**: The user (e.g., building engineer or designer) accesses the Stakeholder Platform and selects the "Building Investment Planning Assistant" service. They choose from a set of predefined upgrade scenarios.
2. **API Gateway Forwards the Request**: The Stakeholder Platform sends an HTTP POST request to /api/service/execute/building_investment_assistant. The API Gateway verifies the request and routes it to the Orchestrator.
3. **Orchestrator Generates Workflow & Prepares Trigger Payload** The orchestrator:
   - Generates a unique workflow_id
   - Retrieves static building information from the Service Database
   - Collects dynamic building data from the Middleware (simulated for MVP)
   - Composes a structured workflow trigger payload
   - Publishes this payload to the Kafka topic gr1.S2_2.trigger

```
{
  "pilot": "gr1",
  "workflow_id": "S2.2_run_2025_04_18_01",
  "service": "S2.2",
  "building_id": "certh",
  "timestamp": "2025-04-18T09:00:00Z",
  "user_inputs": {
    "selected_scenarios": ["baseline", "upgrade"]
  },
  "building_static": {
      "building_area": 1895,
      "floor_height": 3,
      "external_wall_surface": 638,
      "floors": 2,
      "common_area_surface": 3350,
      "external_Windows": 40,
  },
  "version": "1.0"
}
```


Figure 37 - Sample payload for the trigger message of S2.2

4. **Kafka Broker Publishes Trigger to Subscribed ISs**: The broker receives the payload and makes it available on the topic. IS6 is subscribed to this topic and is triggered.
5. **IS6 Executes LCC & Financial KPI Computation**: The Building Investment Planning Assistant (IS6) receives:
   - User inputs (scenarios, timespan)
   - Building characteristics (static metadata)
   - IoT data (if available).

   It performs financial simulations and publishes the results to the gr1.IS6.output topic.

```
{
  "pilot": "gr1",
  "workflow_id": "S2.2_run_2025_04_18_01",
  "service": "S2.2",
  "source": "IS6",
  "timestamp": "2025-04-18T09:00:12Z",
  "building_id": "certh",
  "output_data": {
    "scenarios": {
      "baseline": {
        "environmental": {
          "primary_energy_demand_kwh": 21773889,
          "global_warming_potential_kgco2eq": 4056913
        },
        "economic": {
          "lifecycle_costs_eur": 6204466,
          "payback_time_years": null,
        }
      },
      "upgrade": {
        "environmental": {
          "primary_energy_demand_kwh": 22066752,
          "global_warming_potential_kgco2eq": 4189196
        },
        "economic": {
          "lifecycle_costs_eur": 5899259,
          "payback_time_years": 1.618,
        }
      }
    },
  },
  "version": "1.0"
}
```


Figure 38 - Sample payload for IS6 outputs

6. **Orchestrator Aggregates and Stores Results**: Subscribed to the output topic, the Orchestrator collects the final KPIs and recommended investment plan and stores them in the Service Database under the same workflow_id.
7. **Results Are Made Available to the Frontend**: A new API endpoint (e.g., /api/service/results/:workflow_id) exposes the results. The Stakeholder Platform retrieves and presents them to the user in a visual format (e.g., ROI bar chart, NPV comparison, etc.).

5.7 S3 - Building Equipment Maintenance

This section describes a complete execution of a B2G service that is triggered on-demand by the user. The service selected is the Building Equipment Maintenance, implemented in the first version using IS containers and the full-Service Layer stack. The workflow leverages all core components (API Gateway, Orchestrator, Kafka Broker, and multiple ISs) to coordinate a multi-step analytical flow, returning results to the user and storing them for future reference.
The high-level Use Case for this service can be found in Table 19:

Table 19 - Building Equipment Maintenance

Service Name	**S3 Building Equipment Maintenance**
Type	B2G
Architectural Pattern	On-demand
Actors Involved	**Main Actor:** Building Operator/ Facility Manager **Secondary Actor:** Building Owner, User
Brief Description	1. The building operator logs into the EVELIXIA Platform and subscribes to the EVELIXIA Building Maintenance Toolbox and gains access to the **Energy Assets Maintenance (IS2)** tool. 2. The platform requests documentation related to the **battery storage units and their cooling system**, as well as the **Domestic Hot Water Storage tank**. It also requests interconnection with sensors monitoring the equipment status. This information can be retrieved from the platform if it already exists; otherwise, the end-user can provide it. 3. Once it has access to the necessary information, the **Energy Assets Maintenance (IS2)** module conducts periodic tests to monitor and check the equipment's status. 4. If the module detects a malfunction, it sends a notification to the end-user to resolve the issue by the following day.
Assumptions and Pre-conditions	• The building must be equipped with battery storage units and a battery cooling system, as well as a Domestic Hot Water Storage tank. Additionally, these systems should be outfitted with sensors to monitor their health status.
Trigger	• The building operator requests to use the building Equipment Maintenance toolbox.
Goal	• The building operator is always informed about the health status of the equipment and receives on-time notification in case of a malfunction.
Pre-Conditions (UCs)	• None
Post-Conditions (UCs)	• None

For the purposes of this deliverable and to stay within its scope, the following assumptions were made:

- Since the integration between the Service Layer and Middleware will be implemented in T5.1, we have not received any real IoT Pilot data for this workflow. Instead, a test stream has been utilized for this implementation.
- The user does not need to initiate the analysis through the Stakeholder Platform since this is a scheduled workflow that will begin automatically at a scheduled time (00:00). We have chosen this time in order to produce 24-hour timeseries for the whole day ahead.

The involved components are:

- **Stakeholder Platform** (User Input / UI)
- **API Gateway**
- **Orchestration Layer**
- **Message Broker**
- **IS2**: Energy Assets Maintenance
- **Service Layer Database**

Figure 39 illustrates the cross-functional flowchart of the service's workflow:

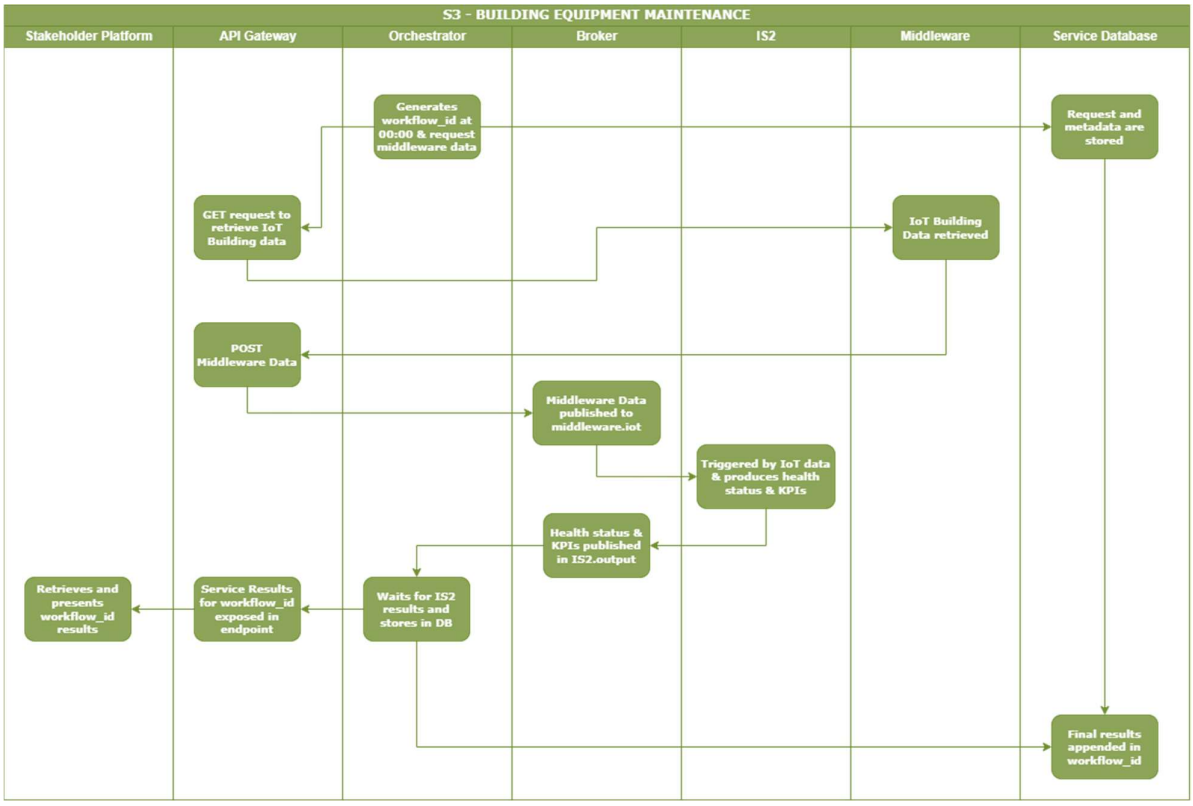


Figure 39 - Building Equipment Maintenance workflow

1. **Workflow Initialization by Orchestrator**: At 00:00, the orchestrator generates a new workflow_id. It records metadata in the database (workflow_id, timestamp, pilot). A request is sent to retrieve the latest IoT data for the building via the Middleware adapter.
2. **IoT Data Ingested via Middleware**
   - Middleware returns recent IoT data from relevant building sensors (temperature, charge cycles, pressure, etc.).
   - Middleware pushes the data to the Kafka topic: gr1.middleware.iot.
3. **IS2 Triggered by IoT Data**
   - The Energy Assets Maintenance module (IS2) is subscribed to gr1.middleware.iot.
   - Upon receiving the data, it processes the equipment health checks and computes the relevant KPIs.


```
"pilot": "gr1",
"workflow_id": "S3_run_2025_04_18_01",
"service": "S3",
"building_id": "certh",
"source": "IS2",
"timestamp": "2025-04-18T00:00:15Z",
"output_data": {
  "health_status": {
    "battery_cooling": {
      "measurement": [],
      "performance": [],
      "EER": []
    },
    "limescale_deposits": {
      "hw_discharge": [],
      "base_hx": [],
      "temperature": [],
      "supplied_heat": []
    },
    "battery_ageing": {
      "rm_results_bess1": [],
      "rm_results_bess2": [],
      "rm_results_bess3": [],
      "bms_bess1": [],
      "bms_bess2": [],
      "bms_bess3": [],

    }
},
"version": "1.0"
```


Figure 40 - Sample payload for the output of IS2

4. **IS2 Produces Diagnostic Results**
   - IS2 publishes results to gr1.IS2.output.
   - The output includes health KPIs, diagnostic messages, and flags for components needing attention.
5. **Orchestrator Stores Final Results**
   - The orchestrator consumes the results from IS2.
   - It logs them to the database under the current workflow_id and marks the workflow as complete.
6. **Frontend Retrieves and Presents Results:** When the user queries the endpoint for workflow_id, results are presented via the Stakeholder Platform.

5.8 S4.1 - Grid Congestion Management

This section describes a complete execution of a G2B service triggered on-demand by the user. The service selected is Grid Congestion Management, implemented in the first version using IS containers and the full-Service Layer stack. The workflow leverages all core components (API Gateway, Orchestrator, Kafka Broker, and multiple ISs) to coordinate a multi-step optimization and simulation flow, returning results to the user and storing them for future reference.

The high-level Use Case for this service can be found in Table 20:

Table 20 - S4.1 Grid Congestion Management

Service Name	S4.1 Grid Congestion Management
Type	G2B
Architectural Pattern	On-demand /Scheduled
Actors Involved	**Main Actor:** Grid Operator (DSO) **Secondary Actor:**
Brief Description	1. The Aggregator logs in the EVELIXIA platform and requests the activation of the Grid congestion management service. 2. The initial step involves gathering and organizing energy profile forecasts for the day ahead from all assets in the energy network. 3. This information is then sent to the **Multi-Vector Grids Energy Modelling & Simulation (IS15)** to simulate the operation ov grid level and detect potential congestion instances. 4. The outcomes from these simulations are forwarded to the **Multi-Vector Network Manager (IS12)**, which employs an optimization algorithm to define the optimal dispatch schedule of the large DER resources on the grid (generation and storage). 5. The optimization results are sent back to the **Multi-Vector Grids Energy Modelling & Simulation (IS15)** to simulate the optimized power flow analysis. 6. Finally, the optimized results are communicated back to the DSO who issues the Day- Ahead DER Dispatch schedule.
Assumptions and Pre-conditions	- Information/ models for DER units → the energy profile forecasts generated by the **Flex Forecast (IS4)**
Trigger	The request of the aggregator for the calculation of a dispatch schedule.
Goal	The Aggregator possesses a day-ahead dispatch schedule that serves as a guideline for managing the grid scale energy resources efficiently.
Pre-Conditions (UCs)	S1 – Building Management Optimization
Post-Conditions (UCs)	None

For the purposes of this deliverable and to stay within its scope, the following assumptions were made:

- Since the integration between the Service Layer and Middleware will be implemented in T5.1, we consider that the District Digital Twin (IS15) has been initialized previously with the required grid topology and static metadata from the Pilot Site. The initial Pilot Site for this current execution is the Greek Pilot Site (PS5).

- The Flex Forecasting IS (IS4) has published next-day energy forecasts for each node/building into its output topic (gr1.IS4.output) at 00:00.
- The user initiates the analysis by inputting flexibility and curtailment cost parameters through the Stakeholder Platform and hitting "Run".
- A scheduled version of this service is also applicable, but in this section we will focus on the on-demand version.

The involved components are:

- **Stakeholder Platform** (User Input / UI)
- **API Gateway**
- **Orchestration Layer**
- **Message Broker**
- **IS15**: Multi-Vector Grids Energy Modelling & Simulation
- **IS12**: Multi-vector Network Manager
- **Service Layer Database**

Figure 41 illustrates the cross-functional flowchart of the service's workflow:

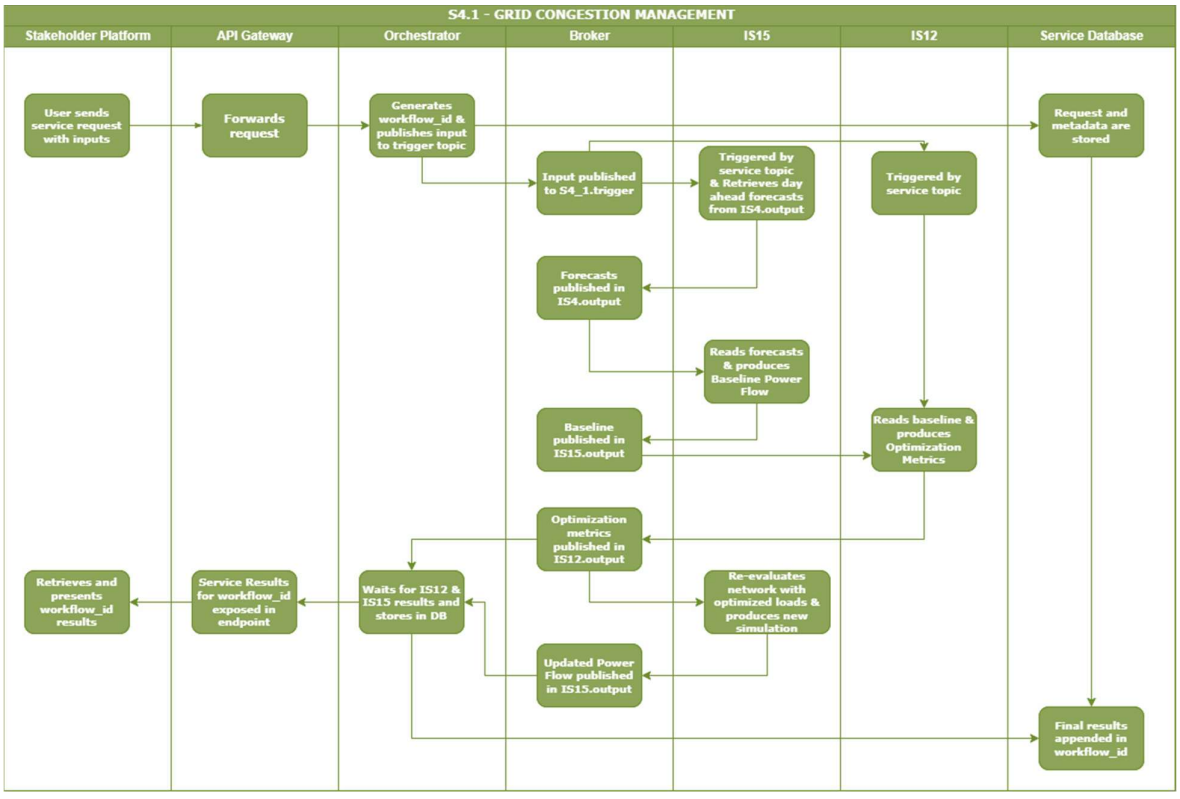


Figure 41 Grid Congestion Management Workflow (on-demand version)

1. User Sends Request: The user enters parameters (e.g., upward/downward flexibility penalties, generation curtailment cost) in the Stakeholder Platform and triggers the workflow.

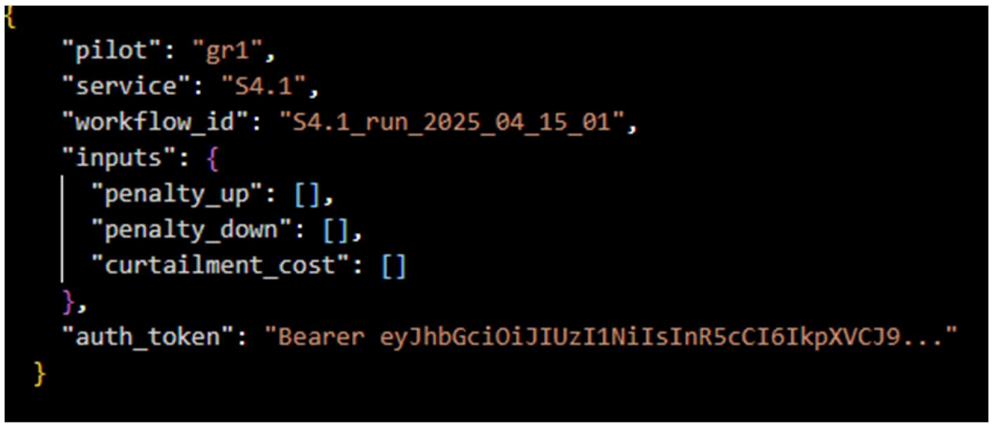


Figure 42 - User Trigger for the initialization of the Grid Congestion Management Service

2. API Gateway Routes Request: The request is sent via HTTP to the endpoint /api/service/execute/grid_congestion_management. The API Gateway forwards it internally to the Orchestrator's planning endpoint.

3. Orchestrator Generates Workflow and Publishes Trigger:

- Assigns a new *workflow_id*
- Stores initial metadata in MongoDB (*workflow_id, user, timestamp*, parameters)
- Publishes a structured JSON message to the service trigger topic gr1. *grid_congestion_management.trigger*. Payload includes *workflow_id*, *user inputs, timestamp*

Figure 43 - Generated Orchestrator Payload for the Grid Congestion Management Service

4. Broker Receives Trigger Message: Kafka receives the message and makes it available to any component subscribed to the service topic.

5. IS15 and IS12 Consume the Trigger

- IS15 consumes the trigger from the topic, checks for the latest forecasts in gr1.IS4.output, and starts a baseline simulation.
- IS12 also consumes the trigger and waits for results from IS15 before starting its optimization.

6. IS15 Produces Baseline Simulation Results: IS15 publishes power flow simulation results to its output topic *gr1.IS15.output*. These include values for active/reactive power, voltage magnitude, and angle for each node.

```
{
  "pilot": "gr1",
  "workflow_id": "S4.1_run_2025_04_15_01",
  "service": "S4.1",
  "source": "IS15",
  "timestamp": "2025-04-15T08:00:05Z",
  "output_data": {
    "simulation_type": "baseline",
    "nodes": [
      {
        "id": "Node2",
        "active_power_kw": [],
        "reactive_power_kvar": [],
        "voltage_pu": []
      },
      {
        "id": "Node3",
        "active_power_kw": [],
        "reactive_power_kvar": [],
        "voltage_pu": []
      }
    ]
  },
  "version": "1.0"
}
```


Figure 44 - Initial Baseline results payload from IS15

7. IS12 Consumes IS15 Results and Executes Optimization: IS12 reads the gr1.IS15.output and performs optimization based on:

- Baseline grid status
- Flexibility requirements
- User-defined penalties

IS12 then publishes the optimization results to gr1.IS12.output, containing:

- Allocated flexibility per node
- Curtailment values
- Operational metrics and cost impacts

```
{
  "pilot": "gr1",
  "workflow_id": "S4.1_run_2025_04_15_01",
  "service": "S4.1",
  "source": "IS12",
  "timestamp": "2025-04-15T08:00:12Z",
  "output_data": {
    "optimization": {
      "excess_res_generation_kwh": [],
      "utilized_flex_kwh": [],
      "remaining_flex_kwh": [],
      "operational_costs": {
        "total": [],
        "breakdown": {
          "flexibility_costs": [],
          "curtailment_costs": []
        }
      }
    },
    "node_allocations": [
      {
        "node": "Node2",
        "flex_allocated_kwh": [],
        "metrics": { }
      },
      {
        "node": "Node3",
        "flex_allocated_kwh": 50,
        "metrics": { }
      }
    ]
  },
  "version": "1.0"
}
```


Figure 45 - Initial IS12 Payload

8. IS15 Consumes IS12 Output and Re-runs Simulation: IS15 reads the updated flexibility profile from IS12 and executes a post-optimization power flow simulation, again producing power-related metrics. Results are published again to *gr1.IS15.output*, tagged as post-optimization.

```
{
  "pilot": "gr1",
  "workflow_id": "S4.1_run_2025_04_15_01",
  "service": "S4.1",
  "source": "IS15",
  "timestamp": "2025-04-15T08:00:20Z",
  "output_data": {
    "simulation_type": "optimized",
    "nodes": [
      {
        "id": "Node2",
        "active_power_kw": [],
        "reactive_power_kvar": [],
        "voltage_pu": []
      },
      {
        "id": "Node3",
        "active_power_kw": [],
        "reactive_power_kvar": [],
        "voltage_pu": []
      }
    ]
  },
  "version": "1.0"
}
```


Figure 46 - Optimized IS15 Payload after IS12 optimization

9. Orchestrator Collects Final Results: The Orchestrator, subscribed to both IS12 and IS15 output topics, collects:

- The baseline and post-optimization simulation results
- Optimization decisions and metrics

These results are aggregated into a structured result object and inserted into the MongoDB Results collection.

```
{
  "workflow_id": "S4.1_run_2025_04_15_01",
  "pilot": "gr1",
  "service": "S4.1",
  "trigger_time": "2025-04-15T08:00:00Z",
  "user_id": "evelixia_user_1",
  "inputs": {
    "penalty_up": [],
    "penalty_down": [],
    "curtailment_cost": []
  },
  "results": {
    "IS15_baseline": { /* baseline results from IS15 */ },
    "IS12_optimization": { /* results from IS12 */ },
    "IS15_optimized": { /* post-optimization results */ }
  },
  "status": "complete"
}
```


Figure 47 - Service Database Grid Congestion Management payload

10. Results Are Made Available to the Frontend: The Orchestrator responds via the API with the results, now accessible to the Stakeholder Platform. These include:

- Baseline and optimized power flow states
- Node-level flexibility allocations
- Cost metrics and investment suggestions

11. User Views Results: The platform visualizes the results using charts or tables for the operator — who now receives a report showing the system's performance with and without optimization.

5.9 S4.2 - Portfolio Management

This section describes a complete execution of a G2B service triggered automatically by the EVELIXIA Ecosystem. The service selected is Portfolio management services, implemented in the first version using IS containers and the full-Service Layer stack. The workflow leverages all core components (API Gateway, Orchestrator, Kafka Broker, and multiple ISs) to coordinate a multi-step optimization and simulation flow, returning results to the user and storing them for future reference.

The high-level Use Case for this service can be found in Table 21:

Table 21: Portfolio Management

Service Name	**S4.2 Portfolio management**
Type	G2B
Architectural Pattern	Scheduled
Actors Involved	**Main Actor:** Aggregator **Secondary Actor:** DSO, Grid Customer (Large Industrial customers, Pooled non-industrial customers)
Brief Description	1. Every day this service is automatically triggered by the results produced by S4.1 and S1.0. 2. The generated energy profiles for the day-ahead behaviour of all the assets managed by the Aggregator (from S1.0) and the Day-Ahead Dispatch schedule (from S 4.1) are forwarded to the **Aggregated Demand Portfolio Manager (IS13)**, which employs an optimization algorithm to refine the energy schedules per managed asset. 3. The generated results are forwarded to the end-user via the Stakeholder Platform.
Assumptions and Pre-conditions	All the assets managed by the aggregator have to be subscribed to the S1 Building Management Optimization so the Forecasted energy behaviour can be generated for each building.
Trigger	• DER Dispatch Schedule • Day- Ahead Energy Consumption/ Generation Forecasting
Goal	The Aggregator possesses an optimization schedule that serves as a guideline for portfolio management of their assets.

Pre-Conditions (UCs)	• S1.0. Day-Ahead Building Forecasting • S4.1 -Grid Congestion Management
Post-Conditions (UCs)	• None.

For the purposes of this deliverable and to stay within its scope, the following assumptions were made:

- Since the integration between the Service Layer and Middleware will be implemented in T5.1, the real-time exchange with flexibility market platforms or physical DERs is not active in this version. Instead, test forecasts and schedules generated by S1.0 and S4.1 have been used.
- It is assumed that all assets managed by the aggregator are already registered in the EVELIXIA platform and properly linked to S1 and S4 services, enabling the automatic retrieval of required forecasting and dispatching data.
- The S1.0 Building Forecasting Service and S4.1 Grid Congestion Management Service have already been completed for the current day before S4.2 is executed. Their output data are published in the broker and accessible for consumption by IS13.
- For this version, the Aggregator's asset list and ownership structure are assumed to be static and preconfigured in the Service Layer Database. Future versions will allow dynamic configuration of portfolios via the Stakeholder Platform.
- The optimization output is intended as a recommendation layer, and no automatic control actions are dispatched to real systems in this version. Dispatch enforcement or signal translation to DERs is planned for later stages.

The involved components are:

- **Stakeholder Platform** (User Input / UI)
- **API Gateway**
- **Orchestration Layer**
- **Message Broker**
- **IS13**: Aggregated Demand Portfolio Manager
- **Service Layer Database**

Figure 48 illustrates the cross-functional flowchart of the service's workflow:

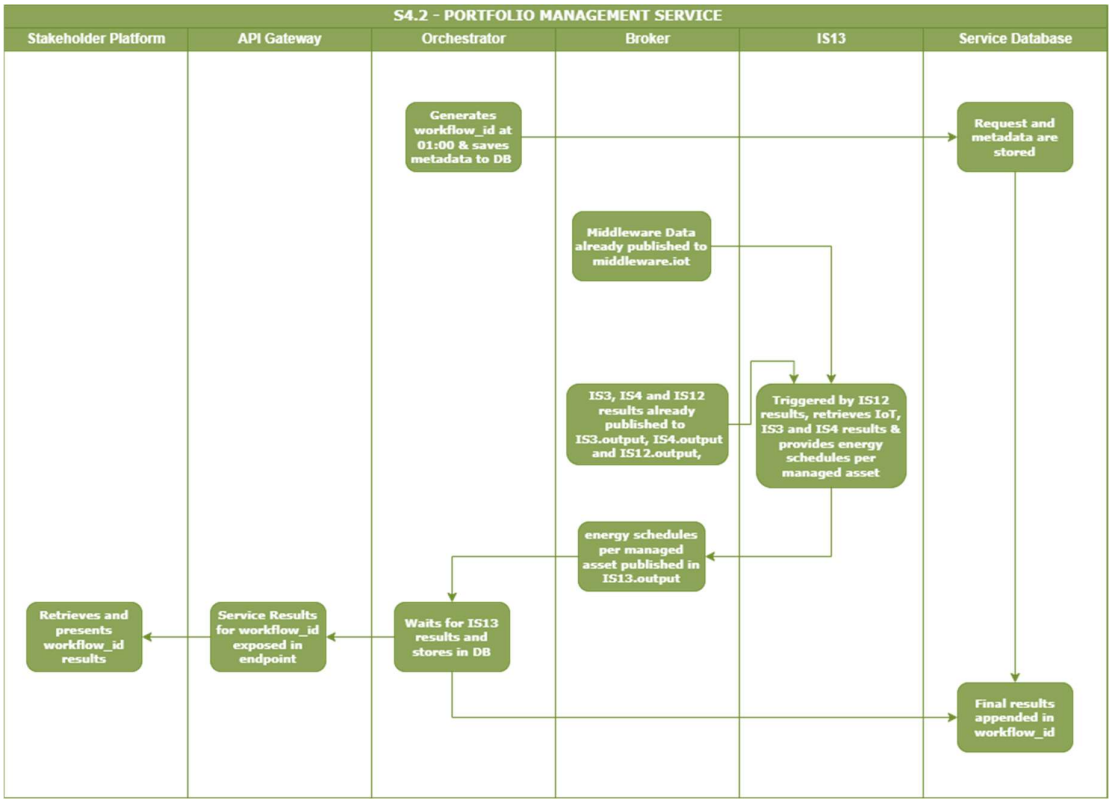


Figure 48 - Portfolio Management Workflow

- **Orchestrator Generates Workflow ID and Initializes Metadata**: The Orchestrator automatically triggers the Portfolio Management service at 01:00. It generates a unique workflow_id, stores initial metadata into the database, and begins monitoring the relevant topics.
- **Forecasts and Dispatch Results Are Published by Previous Services**: IS3 (Demand Forecasting), IS4 (Flex Forecasting), and IS12 (Dispatch Scheduling) have already produced their outputs to Kafka topics (IS3.output, IS4.output, and IS12.output). These contain per-building energy forecasts and dispatch schedules for the upcoming day.
- **IS13 Consumes Inputs and Executes Optimization**: The Portfolio Management Module (IS13) retrieves the relevant input data for each managed building or asset and computes optimized day-ahead energy schedules per asset, aiming to improve operational efficiency or cost-effectiveness.

```
{
  "pilot": "gr1",
  "workflow_id": "S4.2_run_2025_04_18_01",
  "service": "S4.2",
  "source": "IS13",
  "timestamp": "2025-04-18T01:00:10Z",
  "output_data": {
    "node_id": "Node3",
    "location": "Hospital",
    "signals": {
      "baseline_consumption_kWh": [],
      "aggregated_decision_kWh": [],
      "downward_flexibility_bound_kWh": [],
      "upward_flexibility_bound_kWh": [],
      "dso_trajectory_kWh": [],
      "retail_tariff_eur_per_kWh": [],
      "dso_discounted_tariff_eur_per_kWh": []
    }
  },
  "version": "1.0"
}
```


Figure 49 - Sample payload for the results of IS13

- **Final Results Are Stored and Returned**: The Orchestrator collects the optimized schedules from IS13, appends them under the given workflow_id, stores them in the database, and exposes the results through a dedicated API endpoint. The Stakeholder Platform frontend accesses and presents the results.

5.10 S4.3 - P2P Flexibility Trading

The S4.3 – Peer-to-Peer (P2P) Trading service shares the same architectural pattern, workflow logic, and component involvement as the S4.2 – Portfolio Management Service. It reuses the same underlying execution flow involving IS3, IS4, IS12, and IS13, triggered daily after the availability of flexibility and demand forecasts. However, while the service execution pipeline remains identical, the results produced by IS13 are post-processed to simulate a P2P trading scenario. This includes manipulating the price signals and matching available flexibility between participants to optimize local energy transactions. As such, the reader is referred to to the S4.2 description while noting that S4.3 extends the output logic to include prosumer-prosumer exchanges and trade-based optimization.

5.11 S5 - Grid Investment Planning

This section describes a complete execution of a G2B service triggered on-demand by the user. The service selected is Grid Investment Planning, implemented in the first version using IS containers and the full-Service Layer stack. The workflow leverages all core components (API Gateway, Orchestrator, Kafka Broker, and multiple ISs) to coordinate a multi-step energy and financial analysis flow, returning results to the user and storing them for future reference.

The high-level Use Case for this service can be found in Table 22:

Table 22 - Grid Investment Planning

Service Name	**S5 Grid Investment Planning**
Type	G2B
Architectural Pattern	On-demand
Actors Involved	**Main Actor:** Grid Operator (TSO/DSO) **Secondary Actor:** -
Brief Description	1. The Grid Operator logs in the EVELIXIA platform and enters the Grid Investment Planning service. 2. They are asked to insert a set of information that will initiate the analysis. More specific they are asked to insert a set of scenarios and interventions that will be analyzed and compared by the platform, along with a set of transition goals. 3. The inserted information is forwarder to the **Multi-Vector Grids Energy Modelling & Simulation (IS15)** component, to perform the power flow simulations of the inserted scenarios and calculate a set of energy related KPIs. 4. The results are forwarded to the **Grid Investment Planning (IS 11)** component to perform a cost benefit analysis and sensitivity analysis generates a report. 5. The generated report is stored to the platform and forwarded to the end-user.
Assumptions and Pre-conditions	• The grid has to be initialized in the EVELIXIA platform. If not, the engineer has to do the initialization before proceeding to examining the interventions. • The engineer should have a set of upgrade scenarios that wants to investigate and compare.
Trigger	• The Grid Operator (TSO/ DSO) requests to use the Grid Investment Planning service.
Goal	• The Grid Operator (TSO/ DSO) gets a report with the Grid Investment Planning analysis.
Pre-Conditions (UCs)	• None.
Post-Conditions (UCs)	• None.

For the purposes of this deliverable and to stay within its scope, the following assumptions were made:

- Since the integration between the Service Layer and Middleware will be implemented in T5.1, we consider that the District Digital Twin (IS15) has been initialized previously with the required grid topology and static metadata from the Pilot Site. The initial Pilot Site for this current execution is the Greek Pilot Site (PS5).
- The user initiates the analysis by choosing one of the pre-defined scenarios for investment and inputting objective and scope parameters through the Stakeholder Platform and hitting “Run”.

The involved components are:

- **Stakeholder Platform** (User Input / UI)
- **API Gateway**
- **Orchestration Layer**
- **Message Broker**
- **IS15**: Multi-Vector Grids Energy Modelling & Simulation
- **IS11**: Grid Investment Planning Assistant (GIPA)
- **Service Layer Database**

Figure 50 illustrates the cross-functional flowchart of the service’s workflow:

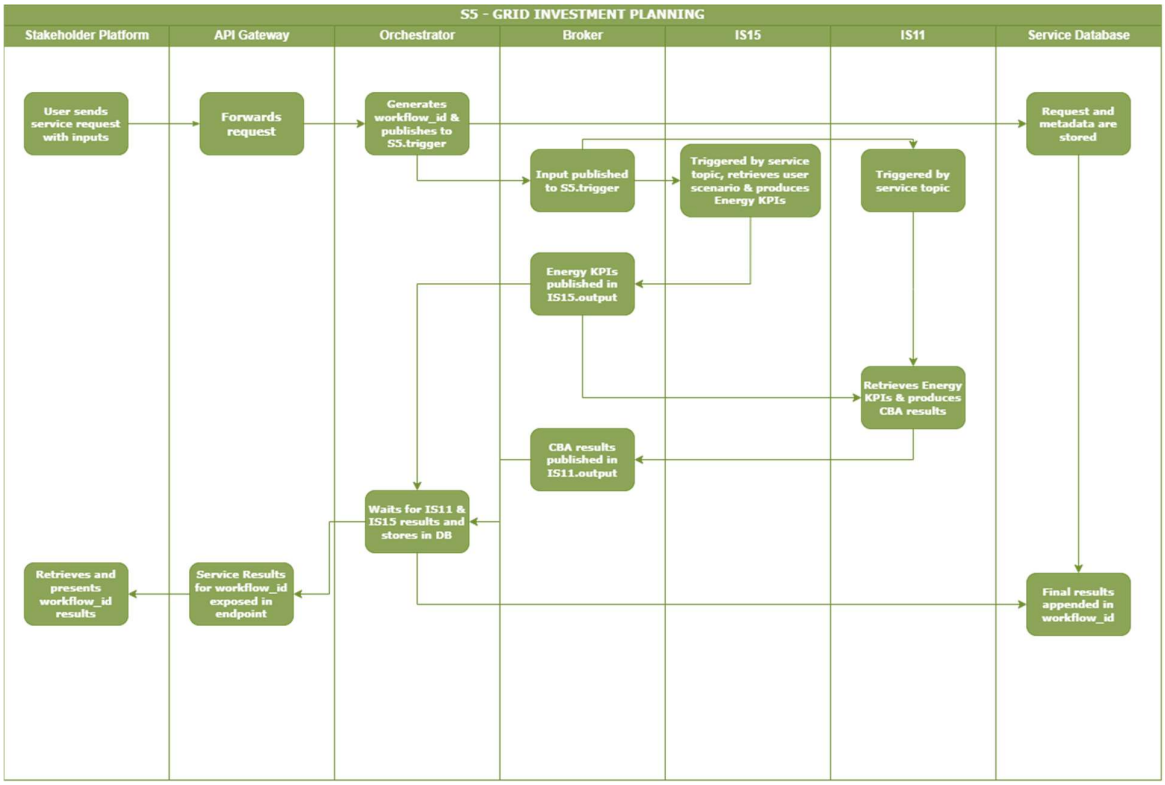


Figure 50 - Grid Investment Planning Workflow

1. **User Sends Service Request with Parameters**: A DSO/TSO operator logs into the Stakeholder Platform, selects the Grid Investment Planning tool, and inputs:
   - One or more scenario IDs
   - Financial expenditures
   - Constraints or priorities (optional)
2. **Request Forwarded via API Gateway**
   - The HTTP POST request is sent to: /api/service/execute/grid_investment_planning.
   - It is routed internally to the orchestrator for processing.
3. **Orchestrator Generates Workflow ID and Publishes Trigger**
   - A workflow_id is generated.
   - User inputs and metadata are saved in MongoDB.
   - The orchestrator publishes a message to the trigger topic gr1.S5.trigger, combining user inputs with relevant grid metadata.

```
{
  "pilot": "gr1",
  "workflow_id": "S5_run_2025_04_18_01",
  "service": "S5",
  "source": "orchestrator",
  "timestamp": "2025-04-18T10:00:03Z",
  "input_parameters": {
    "selected_scenario": "upgrade_2030",
    "capex_eur": 500000,
    "opex_eur": 10000,
    "discount_rate_percent": 5.0
  },
  "version": "1.0"
}
```


Figure 51 - Sample trigger payload for S5

4. **IS15 Consumes Trigger and Produces Energy KPIs**
   - IS15 (Multi-Vector Grid Simulation) retrieves the user scenario, loads the grid model, and runs a simulation.
   - Energy KPIs are computed and published to gr1.IS15.output.


```
{
  "pilot": "gr1",
  "workflow_id": "S5_run_2025_04_18_01",
  "service": "S5",
  "source": "IS15",
  "timestamp": "2025-04-18T10:01:10Z",
  "output_data": {
    "scenario": "upgrade_2030",
    "energy_kpis": {
      "energy_curtailment_reduction_mwh": 120.5,
      "energy_loss_reduction_mwh": 34.2,
      "system_load_factor_improvement_percent": 5.3,
      "co2_emissions_avoided_tonnes": 42.1,
      "grid_congestion_relief_percent": 18.9,
      "voltage_stability_improvement_index": 0.15
    }
  },
  "version": "1.0"
}
```


Figure 52 - Sample payload of the IS15 results for S5

5. **IS11 Consumes IS15 Output and Runs CBA**
   - IS11 (Grid Investment Planning Assistant) subscribes to the output of IS15.
   - It uses the energy KPIs and scenario metadata to run cost-benefit and sensitivity analyses.
   - Final results are published to gr1.IS11.output.

Figure 53 - Sample payload for the IS11 results

6. **Orchestrator Collects Final Outputs and Stores**
   - Orchestrator waits for outputs from both IS15 and IS11.
   - Final structured results are compiled and stored in the database under the workflow_id.
7. **Frontend Presents Results**
   - When the user queries for results, the system returns:
   - Energy KPIs (IS15)
   - Financial KPIs (IS11)
   - Recommendation summary (if applicable)

5.12 S6 - Grid Maintenance

This section describes a complete execution of a G2B service triggered on-demand by the user. The service selected is Grid Maintenance, implemented in the first version using IS containers and the full-Service Layer stack. The workflow leverages all core components (API Gateway, Orchestrator, Kafka Broker, and multiple ISs) to coordinate a multi-step energy and financial analysis flow, returning results to the user and storing them for future reference.

The high-level Use Case for this service can be found in Table 23:

Table 23: Grid Infrastructure Maintenance

Service Name	S6 Grid Infrastructure Maintenance
Type	G2B
Architectural Pattern	Scheduled / On-Demand
Actors Involved	**Main Actor:** Aggregator / Grid Operator (DSO/ TSO) **Secondary Actor:**
Brief Description	1. Every first day of the month at 00:00, this service is automatically triggered, by requesting new IoT Building & Grid Asset Data from the Middleware. 2. The request is forwarded to the **Multi-Vector Grids Energy Modelling & Simulation (IS15)** to produce a grid-level equipment simulation analysis. 3. **Smart Grid Maintenance (IS14)** receives the needed data to perform assessment of the health/degradation levels of multi-grid related assets (e.g., BESS, MV/LV substations, PVs, HPs, power lines) when connected to electrical networks (main energy carrier), based on black-box data driven models (predictive maintenance and infrastructure ageing models) and co-optimization of the outage scheduling of grid assets with buildings' DERs' generation. 4. The **Smart Grid Maintenance (IS14)** performs an optimization algorithm and generates the maintenance planning of the grid assets based on the state of health of the various grid equipment. The report is a grid maintenance schedule . 5. The generated report is stored to the platform and forwarded to the end-user.
Assumptions and Pre-conditions	• The grid infrastructure must be equipped with sensors to monitor their health status and be interconnected to the EVELIXIA platform.
Trigger	• Monthly Schedule • User request (alternative flow)
Goal	• Report generation of a grid maintenance schedule for the Aggregator/Grid Operator (TSO/DSO)
Pre-Conditions (UCs)	• None.
Post-Conditions (UCs)	• None.

For the purposes of this deliverable and to stay within its scope, the following assumptions were made:

- Since the integration between the Service Layer and Middleware will be implemented in T5.1, we consider that the District Digital Twin (IS15) has been initialized previously with the required grid topology and static metadata

from the Pilot Site. The initial Pilot Site for this current execution is the Greek Pilot Site (PS5).

- The user does not need to initiate the analysis through the Stakeholder Platform since this is a scheduled workflow that will begin automatically at a scheduled time (00:00). We have chosen this time in order to produce a monthly maintenance schedule.
- The service can also be requested on-demand, but in this section we will only present the scheduled version.

The involved components are:

- **Stakeholder Platform** (User Input / UI)
- **API Gateway**
- **Middleware** (test stream)
- **Orchestration Layer**
- **Message Broker**
- **IS15**: Multi-Vector Grids Energy Modelling & Simulation
- **IS14**: Smart Grid Maintenance
- **Service Layer Database**

Figure 54 illustrates the cross-functional flowchart of the service's workflow:

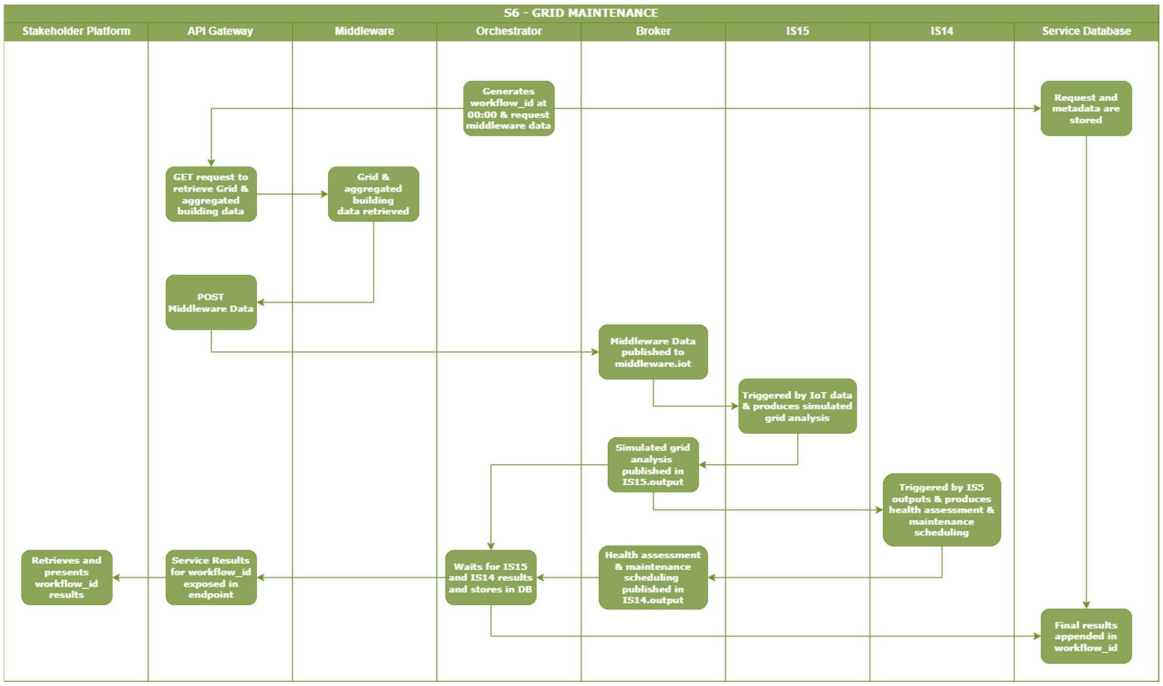


Figure 54 - Grid Maintenance Workflow (scheduled version)

1. **Scheduled Trigger (00:00)**: At the start of each month, the orchestrator schedules the execution of the grid maintenance workflow and generates a new workflow_id.
2. **Middleware Data Request**: The orchestrator sends a request to the Middleware to retrieve the latest IoT building and grid data (e.g., health and operational parameters of grid assets).
3. **Data Delivery to Broker**: The Middleware provides the requested building and grid data and publishes it to the shared Kafka topic: gr1.middleware.iot
4. **IS15 Simulation Initiation**: The Multi-Vector Grid Modelling component (IS15) is triggered by the newly published IoT data and executes grid-level simulations based on the current state.
5. **IS15 Publishes Grid Simulation Output**: IS15 publishes its simulated grid analysis results (voltage profiles, system status etc.) to: gr1.IS15.output

```
{
  "pilot": "gr1",
  "workflow_id": "S6_run_2025_05_01_00",
  "service": "S6",
  "source": "IS15",
  "timestamp": "2025-05-01T00:02:30Z",
  "output_data": {
    "assets": [
      {
        "id": "transformer_2",
        "voltage": [],
        "temperature": [],
        "thd": [],
        "oil_temperature": [],
        "winding_temperature": [],
        "health_index": []
      },
      {
        "id": "transformer_3",
        "voltage": [],
        "temperature": [],
        "thd": [],
        "oil_temperature": [],
        "winding_temperature": [],
        "health_index": []
      }
    ],
    "unit": {
      "voltage": "V",
      "temperature": "C",
      "thd": "%",
      "health_index": "%"
    }
  },
  "version": "1.0"
}
```


Figure 55 - Sample payload for the outputs of IS15 for S6

6. **IS14 Maintenance Assessment Execution**: Smart Grid Maintenance (IS14) is triggered by IS15's outputs. It applies its predictive maintenance and optimization algorithm to assess equipment degradation and propose maintenance schedules.
7. **IS14 Publishes Maintenance Schedule**: IS14 outputs health assessments and suggested maintenance tasks (e.g., inspection of BESS or line replacements) to: gr1.IS14.output

```
{
  "pilot": "gr1",
  "workflow_id": "S6_run_2025_05_01_00",
  "service": "S6",
  "source": "IS14",
  "timestamp": "2025-05-01T00:04:20Z",
  "output_data": {
    "predicted_health_index": {
      "transformer_2": [],
      "transformer_3": []
    },
    "anomalies_detected": [
      {
        "asset_id": "transformer_3",
        "timestamp": "2025-04-26T13:00:00Z"
      }
    ],
    "maintenance_schedule": [
      {
        "asset_id": "transformer_3",
        "recommended_action": "Cooling System Maintenance",
        "scheduled_for": "2025-05-06"
      },
      {
        "asset_id": "transformer_2",
        "recommended_action": "Cooling System Maintenance",
        "scheduled_for": "2025-05-15"
      }
    ]
  },
  "version": "1.0"
}
```


Figure 56 - Sample payload for the results of IS14

8. **Results Aggregation by Orchestrator**: The orchestrator listens to the IS15 and IS14 output topics, gathers the final results, and aggregates them under the same workflow_id. These are stored in the Service Database.
9. **Results Exposure & Delivery**: The results are exposed through the /api/service/results/:workflow_id endpoint and retrieved via the Stakeholder Platform for visualization and review by the user.

6 NEXT STEPS

With the MVP implementation of the EVELIXIA Service Layer successfully implemented and tested in a centralized containerized environment, the next phase of the project will focus on scaling, enriching, and deploying the system using real pilot contexts. These steps aim to validate the architecture in live conditions and progressively evolve the platform into a fully operational backbone for G2B and B2G energy services.

We will explore the transition into per-pilot instances of the Service Layer, maintaining the same architecture and technology stack but with pilot-specific configurations. Each instance could include local building and grid data, site-specific Innovative Solutions (IS), and customized orchestration logic.

To facilitate this, packaging of the Service Layer into reproducible deployments (e.g., Docker Compose bundles) will be performed, allowing deployment either on-premise or on cloud infrastructure. While the MVP used a single Kafka and MongoDB instance, we will assess whether horizontal scaling (e.g., separate Kafka topics per pilot, or Mongo sharding) is required to support high-frequency data flows across multiple pilot sites.

The MVP demonstrated integration with two representative Services. The next step is to integrate the remaining Innovative Solutions defined across the 6 services described in D1.7. Each new IS will require:

- Kafka topic definition (output and optional intermediate topics)
- Configuration of its environment (input/output topics, services, and parameters)
- Integration into orchestrator's workflow logic via updated YAML descriptors

The Middleware Layer developed in parallel work packages will be integrated to replace synthetic or dummy data sources. This includes:

- Subscribing to real-time data (e.g., building sensors, energy meters)
- Dispatching actuation signals to Pilot Infrastructure in the Field Layer

Before public deployment, security must be strengthened:

- API Gateway will be secured via HTTPS
- Role-based access control will be applied per service and user role
- Kafka topics and MongoDB collections will be fully segregated per pilot to ensure data isolation

Although identity federation is out of scope for this task, proper token-based authorization and service identity handling will be enforced to prevent unauthorized access.

To ensure readiness for live pilots:

- Kafka consumer groups will be leveraged to parallelize heavy IS computations
- MongoDB indexing will be optimized for performance queries (by pilot ID, timestamp)
- Logging and monitoring tools (e.g., Grafana, Prometheus) will be configured for observability

Where necessary, multi-instance orchestration or Kafka clustering may be explored. Health checks and retry mechanisms will be validated under stress testing.

As more IS modules are chained together, complex workflows involving multiple decision points will emerge. This includes services that rely on outputs of several IS tools or require conditional execution. Orchestrator logic will be expanded to support multi-branch workflows, timeout handling and composite service orchestration

7 CONCLUSION

This document has provided a comprehensive summary of the EVELIXIA Service Layer, detailing its design rationale, core components, and implementation strategy through a functional Minimum Viable Product (MVP). The Service Layer has been developed as the central backend system that coordinates the execution of energy-related services across the platform, supporting both Building-to-Grid (B2G) and Grid-to-Building (G2B) interactions.

By adopting a modular and event-driven architecture, the Service Layer integrates heterogeneous Innovative Solutions (IS) and manages the communication between them through a message broker. Its architecture comprises key components including the Message Brokerage System, Orchestration Layer, API Gateway, and Service Database. Each of these components plays a distinct role in enabling automated, scalable, and decoupled service execution.

The MVP implementation demonstrated that this architecture can be reliably executed using containerized technologies, specifically Docker Compose, Kafka, MongoDB, and services. Two representative workflows were implemented: one on-demand workflow for Grid Network Maintenance and one continuous workflow for Daily Building Optimization. These examples validated the orchestrator's ability to manage Kafka topics, initiate service logic, collect results, and deliver outputs to both users and downstream systems.

This MVP validates that the core service logic, topic routing, workflow triggering, and output persistence operate as intended in a controlled test environment. It also establishes a strong baseline for the upcoming pilot phase deployments, where the system will interact with real-world data sources, physical infrastructure, and user interfaces.

In the next stages of the project, the Service Layer will be expanded to include the full set of EVELIXIA services, scaled for multiple pilot sites, and integrated with external systems via the Interoperability Layer. The platform will continue to evolve in response to real-world usage, ultimately ensuring that the Service Layer delivers on its role as the operational core of an interactive, efficient, and flexible energy ecosystem.