Acumen

D5.2 ACUMEN TOOLS INTEGRATION

Publishing Date: 30/11/2024

Lead Beneficiary: LIST

Due Date: 30/11/2024

Funded by the European Union

Legal Disclaimer

Funded by the European Union. Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union or the European Climate, Infrastructure and Environment Executive Agency (CINEA). Neither the European Union nor the granting authority can be held responsible for them.

General Information	
Grant Agreement Nr	101103808
Action Acronym	ACUMEN
Action Title	Ai- aided deCision tool for seamless mUltiModal nEtwork and traffic managemeNt
Type of Action	HORIZON - RIA
Call	HORIZON-CL5-2022-D6-02
Start Date and Duration	1st June 2023 – 31st May 2026
Due Date of the Deliverable	M18
Submission Date	**30/11/2024**
Lead Beneficiary	LIST
Deliverable Dissemination Level	Public

Authors in alphabetical order		
Name	**Beneficiary**	**pMail**
Nicolas Damien	LIST	damien.nicolas@list.lu
Feltus Christophe	LIST	christophe.feltus@list.lu
Ruben Marquez	LIST	ruben.marquez@list.lu

Document reviewers in alphabetical order		
Name	**Beneficiary**	**EMail**
Ludovic Leclercq	UGE	ludovic.leclercq@univ-eiffel.fr

Revision History		
Version	**Date**	**Comments**
V0.1	30/03/2024	initial table of content
V0.2	12/07/2024	Chapter 2 updates
V0.3	20/09/2024	Chapter 3 & 4 updates
V0.4	10/11/2025	Chapter 5 updates
V0.5	25/11/2025	Format check
V0.6	04/07/2025	Emblem and disclaimer updates

Abstract

This document presents the integration principles that will be put in place for enabling the integration of the different microservices that composed the ACUMEN platform.

Table of Contents

List of Figures

Acronyms	
Acronym	**Meaning**
API	**Application Programming Interface**
DT	**Digital Twin**
M-DT	**Mobility Digital Twin**
UML	**Unified Modelling Language**
SYSML	**System Modeling Language**
HTTP[1]	**Hypertext Transfer Protocol**
gRPC[2]	**(Google) Remote Procedure Call**

[1] https://httpwg.org/specs/
[2] https://grpc.io/

Executive Summary

This document presents the intermediary technical integration of mobility tools developed in the ACUMEN project.

ACUMEN platform is a micro-services based architecture enabling modular composition, loosely-coopled and independent deployment of services according to the needs. A set of services compose the “core” of the platform providing mandatory functionalities such as services registration, authentication and authorization mechanism, data management and services orchestration. Other services are considered as “external” services and should provide APIs to be integrated in the ACUMEN and also to consume provided services.

After presenting an overview of the ACUMEN platform, the micro-services architecture will be described including the main principles and patterns used.

Then integration patterns are detailed according to their responsabilies:

- Services-to-services communication protocoles
- Messages-based architecture for asynchronous communication between services
- API gateway for centralizing clients communication with multiple backends micro-services
- Services orchestration for supporting business/technical workflows
- Security integration covering authentication and authorization

1. Introduction

As software systems become increasingly complex, the shift toward microservices architecture provides a way to build scalable, resilient, and maintainable platforms. The ACUMEN platform embraces this architectural style to ensure modular development and efficient service interaction. Central to this approach is the application of well-defined integration principles that allow independently developed services to function cohesively.

This document outlines the core elements of the ACUMEN system and the foundational strategies used to manage service integration. Chapter 2 introduces the ACUMEN platform and its overall structure. Chapter 3 explores the microservices architecture, focusing on how services communicate and collaborate. Chapter 4 explains the integration patterns applied to enable reliable and flexible data flow. Finally, Chapter 5 addresses the security mechanisms that support secure authentication and authorization across the platform.

Together, these chapters provide a comprehensive view of how ACUMEN leverages integration best practices—such as API-based communication, event-driven workflows, and decentralized data ownership—to meet both technical and business demands.

2. Document summary

2.1. ACUMEN project overview

ACUMEN (Ai-aided deCision tool for seamless mUltiModal nEtwork and traffic managemeNt) is a Research and Innovation Action (RIA) receiving funding from the European Union's Horizon Europe. ACUMEN aims to establish a technological and methodological framework that integrates state-of-the-art mobility modelling, data processing, prediction, and advanced visualization capabilities. By embracing the concept of Digital Twins and adopting a hybrid intelligence paradigm, ACUMEN aims to develop intuitive decision-support tools for smart cities, facilitating effective collaboration between humans and AI, and ultimately delivering innovative services that benefit citizens and stakeholders.

2.2. Purpose and audience of the document

Task 5.2 "Integration of ACUMEN AI-Aided Management and Decision Tools" aims at producing API necessary to connect the different modules (pieces of code) developed in the scientific work packages (WP2, WP3 and WP4) enabling access to data and resources through the ACUMEN platform. According to the micro-services-based architecture described in D5.1 "ACUMEN Reference Architecture", each service is considered as an autonomous service providing API to interact with the others, following defined rules.

The target audience is the development teams and potentially system architects.

2.3. Document structure

This deliverable is structured as follows: Chapter 2 presents the overall ACUMEN system overview, while Chapter 3 describes the micro-services architecture with the main interaction between the different services. Chapter 4 details the integration patterns put in place and finally Chapter 5 gives information about the security mechanisms for authentication and authorization aspects.

2.4. Dependencies and supporting documents

This document has been produced based on the outputs of D5.1 "ACUMEN Reference Architecture" and D6.1 "Specification of data sources and existing models for each pilot".

3. System Overview

ACUMEN reference Architecture is composed of 4 logical layers and around 20 main components in the first iteration. By component, we mean microservice. As defined by IBM, a microservice has its own technology stack including its own database and data management, communicates with other microservices through defined communication protocols such as HTTP/S or gRPC, event streaming and messages brokers and is organized by business or technical capability.

Figure 1 presents these 4 layers with the services attached.

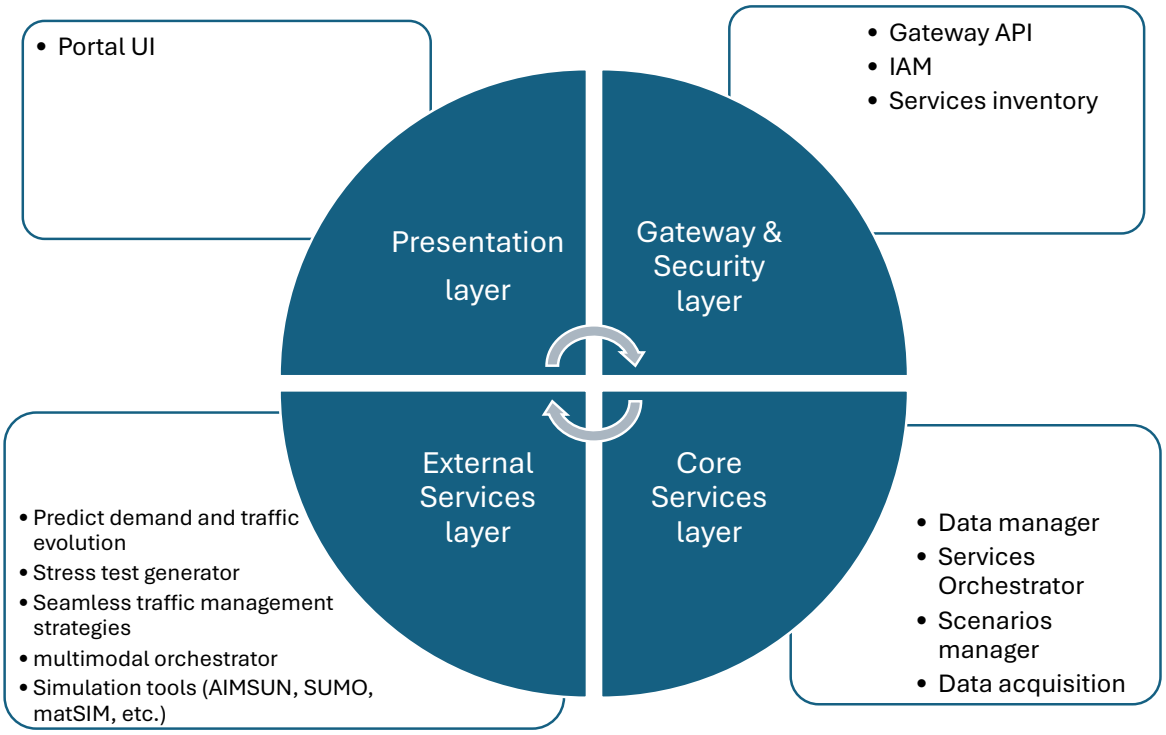

Figure 1: ACUMEN platform logical layers

The presentation layer has the responsibility to display the portal with the services accessible by the logged user. This layer comprises the following services:

- Login/registration graphical user interface: this graphical interface presents to a user, through a web browser, the login/password page and the registration page.
- Portal user interface: this graphical user interface displays to a user, through a web browser, the dashboard (home page) according to the user rights to access to services.

The gateway and security layer comprises the following services:

- Gateway API component: this component acts as a single-entry point for the different clients to access, according to specific access rules, various backend services provided by the ACUMEN platform.
- Identification and Authorization Management component: this component handles user accounts and provides authentication token shared with microservices
- Services inventory component: this component registers the services into the platform and controls the different access rights for each service.

The core services layer comprises the following services:

- Data manager component: this component is responsible to manage the stored data.
- Services orchestrator component: this component handles the orchestration (workflows) of the services.
- Scenarios manager component: this component manages the different scenarios of usage of the ACUMEN platform.
- Data acquisition component: this component manages the gathering of external data in an automatic way (batch or stream) from external sources such as IoT devices or hub, databases sources or files.
- Maps (2D/3D) visualization components: this component visualizes, through a map in 2D or 3D, the different results of the scenarios (simulations).

The external services layer is composed of:

- Predict demand and traffic evolution
- Stress test generator
- Seamless traffic management strategies
- Multimodal orchestrator
- Simulation tools supporting a set of simulations linked to mobility aspects targeted by the ACUMEN project.

At the time of writing this document, some tools are still in early development, implying that their detailed and complete integration is not yet done.

4. Mobility Digital Twin Microservices Architecture

ACUMEN platform architecture is microservices based enabling the following main characteristics:

- Modularity and service independent

Each microservice is designed to perform a single, specific function (such as user account management). Services are loosely coupled, meaning they can operate, deploy, and scale independently without impacting others. Each service has its own codebase, which allows teams to develop, deploy, and maintain them separately.

- Decentralized data management

Each microservice manages its own database, which aligns with the service's specific data needs and prevents data-sharing bottlenecks. This design promotes a decentralized data management approach, where each microservice controls its own data and database, eliminating dependencies on a single, shared database.

- Scalability

Microservices can be scaled independently based on demand, which improves resource efficiency. For example, if one service experiences high traffic, only that specific service can be scaled, rather than the entire application. This approach enables granular control over resources and is especially useful for cloud-based or containerized environments where on-demand scaling is required.

- Resilience and fault isolation

Since each service is independent, failures in one microservice don't directly impact others. This fault isolation improves overall system resilience and simplifies troubleshooting. Developers can implement mechanisms like circuit breakers and retries for better fault tolerance, ensuring that other parts of the system can continue to function even if one service fails.

- API-driven communication

Microservices typically communicate with each other using lightweight protocols such as HTTP/REST or gRPC and can use message brokers for asynchronous communication. APIs enable clear, standard interfaces for each service, allowing teams to modify, replace, or scale services without breaking the application's overall functionality. The communication protocol used, in this preliminary version of the platform, is secured HTTP with REST APIs.

Figure 2 presents the high-level of the ACUMEN architecture including interfaces and relationships between components:

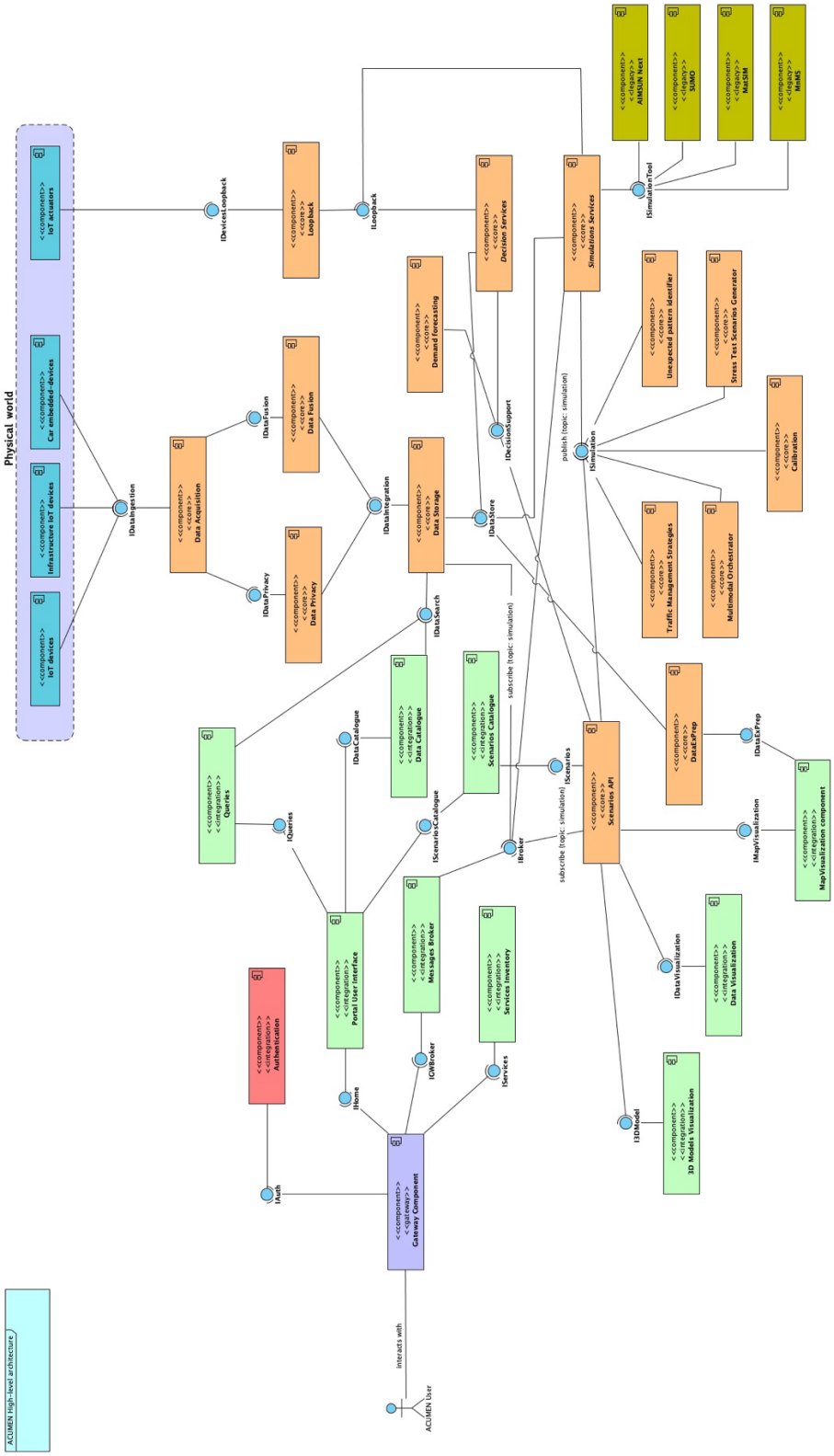

Figure 2: ACUMEN microservices-based architecture

5. Integration patterns

This section presents the patterns used to assure the integration of the several existing or developed services.

5.1. Service-to-Service communication

In ACUMEN platform, all the services provide or consume REST API endpoints. A REST API endpoint is a specific URL that a client (service) can use to access a particular resource or execute a specific action on a RESTful web service. In this context, an endpoint represents a "location" on a server where the data or functionality can be accessed via a particular HTTP method.

The characteristics of REST endpoints are the following:

- Stateless: means each request to the endpoint contains all the information needed to process it. The server doesn't store any session information related to the request.
- Uniform Resource Naming: designed to be intuitive and consistent making it easy for clients to understand the structure of the API
- Resource-Oriented: each endpoint is associated with a resource, allowing clients to interact with resources using standard HTTP operations.

A typical REST API endpoint is composed of several key parts:

- Base URL: this is the main URL that identifies the web service. When a client wants to access to a service of the ACUMEN platform, it will call something like that:

https://api.acumen.org

https is the secured communication protocol, and api.acumen.org is the base URL.

- Resource path: this part of the endpoint specifies the particular resource the client wants to access. In a REST API, resources are usually nouns, representing entities within the service. For example, to access to a dataset in the data manager service, a client needs to use the following endpoint:

https://api.acumen.org/data-manager/datasets/

- Parameters: parameters in the URL or as a query string allow for filtering, specifying a specific resource or modifying the behavior of the request. For example, if the client wants to retrieve the dataset with the id=123, it needs to use the following endpoint:

https://api.acumen.org/data-manager/datasets/123

- HTTP method: the HTTP method defines the action to be taken on the resource. The main HTTP methods used in REST API are:
    - GET: retrieve data from the endpoint
    - POST: submit new data to create new resource
    - PUT: update or replace an existing data on the server

- DELETE: remove a resource from the server

For example, when the client wants to retrieve the dataset with the id=123, the method to use is GET as following:

GET https://api.acumen.org/data-manager/datasets/123

Then the server will return the corresponding dataset or an error if a dataset with id=123 doesn't exist.

5.2. API Gateway Pattern

The API gateway component is critical in modern architecture particularly in distributed and microservices architectures. Clients make requests to the API gateway, which then routes the requests to the appropriate backend service based on predefined rules and configurations. In ACUMEN platform, this component is coupled with the Services inventory component enabling the registration and unregistration of services.

The following Figure 3 shows the interfaces provided and required by this component:

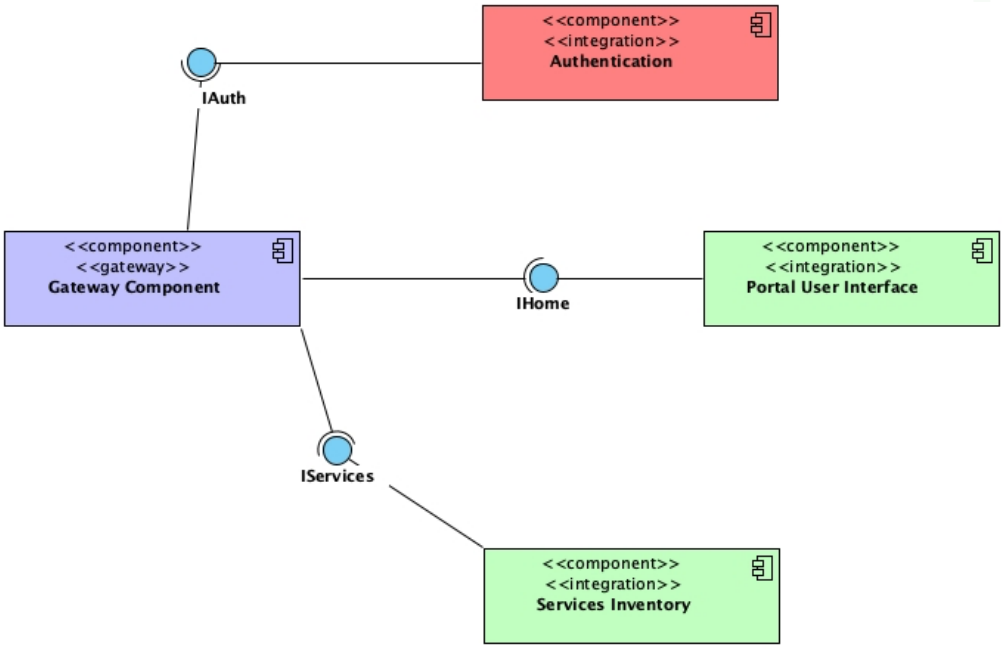

Figure 3: API Gateway component

When a new service should be registered, it has to give the following information:

- route id: unique id for the service. Should be a String
- endpoint name: unique endpoint name. Should be a String
- service name: service name. Should be a String
- description: description. Should be a String
- is user interface: this service is a graphical user interface. Should be a Boolean

- public endpoint: the public endpoint for the service. Should be a String
- pretty name: the name of the service. Should be a String
- JSON template: the JSON template linked to the service to be used in the Scenarios service. Should be in JSON format
- Methods allowed: HTTP method(s) allowed to access to this service. Should be a valid HTTP method such as GET, POST, PUT, DELETE
- URI: the private URI of the service. Only known by the API Gateway. Should be a String
- private endpoint: private resource path. Only known by the API Gateway. Should be a String

For example, the Service inventory service is registered as follow in the gateway:

Route Id	Endpoint Name	Service Name	Description	Is User Interface	Public Endpoint	Pretty Name	Json Template	Methods	Uri	Private Endpoint
services_inventory	Services Inventory Endpoint	Services Inventory Service	This endpoint permits to manage the routes and ser...	false	/admin/services			GET POST PUT DELETE OPTIONS	http://acumen-api-gateway:81	/services

Figure 4: Services inventory service information

Each service that can be accessed through the ACUMEN platform should be registered following the above registration.

Hereinafter the graphical interface of the API gateway with the registered services at the writing time of the deliverable.

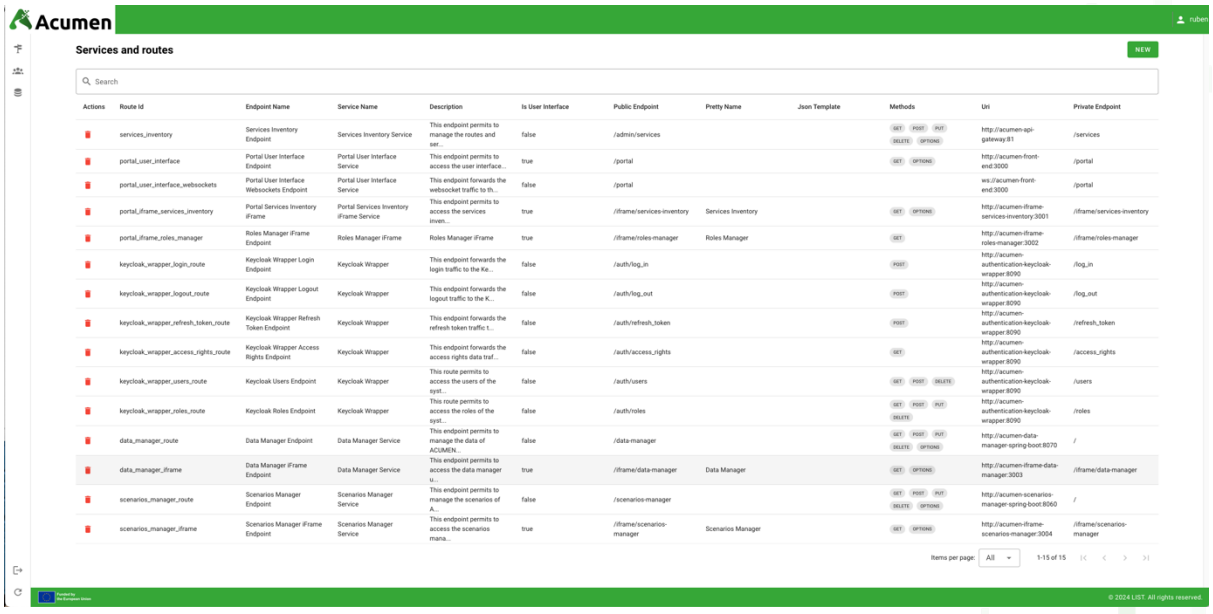

Figure 5: Services and routes graphical interface

5.3. Services Orchestration

Services orchestration is crucial for managing complex distributed systems built using microservices architecture. Orchestration involves coordinating and managing the interactions between multiples microservices to execute technical or business processes and workflows efficiency.

In ACUMEN platform, the execution workflow is the following:

- The orchestrator receives a task or a transaction request
- It breaks down the task into smaller subtasks if needed
- These subtasks are assigned to relevant microservices
- The orchestrator collects responses from the microservices
- Finally, it compiles the results and sends them back to the client

The orchestrator ensures that each microservice performs its specific functions at the right time and in the correct order.

The benefits to implementing microservices orchestration are the following:

- Enhanced coordination
- Scalability
- Reduced coupling

Two main patterns exist for implementing orchestration:

- Event-driven orchestration for asynchronous workflow: the orchestrator maintains a record of events/messages issues to services and updates its internal state based on responses. This pattern is resilient to service failures, as messages can be stored in a queue for later processing.
- Direct call orchestration for synchronous workflow: the orchestrator uses a request-response pattern, synchronously sending requests to microservices and awaiting responses. This approach allows for immediate state updates but may be less resilient to service failure. This approach is implemented in the preliminary version of the ACUMEN platform.

Conductor[3], an open-source services orchestrator, developed originally by Netflix and perfectly adapted for microservices architecture has been chosen as main services orchestrator for the ACUMEN platform.

Conductor uses a JSON-based domain-specific language (DSL) to define workflows: workflows are composed of tasks, which can be system tasks or worker tasks. System tasks are executed within the Conductor server, while worker tasks run on external systems. Workers poll the Conductor server for tasks and execute them independently. Conductor supports both synchronous and asynchronous task execution models. The platform offers client SDKs in multiple languages for easy integration and provides out-of-the-box APIs for workflow and task operations, enabling custom automations.

It also provides a distributed queue for task management and execution.

Workflows can include complex control flows like decision trees, fork-join patterns, and sub-workflows. Conductor offers features like workflow versioning, parameterization, and dynamic task generation.

Conductor uses a persistence layer (e.g., Redis, Cassandra) to store workflow and task states.

It supports event-based triggers for starting workflows and inter-task communication.

It includes features for error handling, retries, and timeouts to ensure workflow resilience.

The platform is designed to be horizontally scalable to handle millions of concurrent workflows.

Conductor allows for dynamic workflow creation and modification at runtime.

It supports integration with various messaging systems and databases for flexibility in deployment.

A workflow is composed of the following elements:

- Workflow Definition: This is the blueprint for the workflow, specified in JSON format. It contains the following key elements:
    - Name and description of the workflow
    - Version number
    - Input parameters

[3] https://docs.conductor-oss.org/

  - Output parameters
  - Tasks array

- Tasks: These are the building blocks of a workflow. There are two main types of tasks:
  - System Tasks: Predefined tasks executed by Conductor itself, such as HTTP requests, conditional logic, forks, and joins.
  - Worker Tasks: Custom tasks executed by external workers, implementing specific business logic.
  - Task Configurations: Each task in the workflow is defined by a configuration object that includes:
  - Name of the task
  - Task reference name (unique within the workflow)
  - Type of task (SIMPLE for worker tasks, or a system task type)
  - Optional parameters like description and error handling
- Operators: These are control flow primitives used to manage the execution of tasks within a workflow. Examples include:
  - Switch (for conditional execution)
  - Do While (for looping)
  - Fork and Join (for parallel execution)
  - Sub-workflows
- Input and Output Parameters: These define the data that flows into and out of the workflow.
- Workflow Execution: This is an instance of a workflow definition, representing the actual work being done step by step4.
- Workers: These are external services or processes that poll for and execute worker tasks

5.4. Data integration

In the preliminary version of the ACUMEN platform, the service Data manager is responsible to handle data related to mobility such as car trajectories. This component is accessible through a graphical interface or through a dedicated API provided by the service.

Figure 6 shows its current graphical interface.

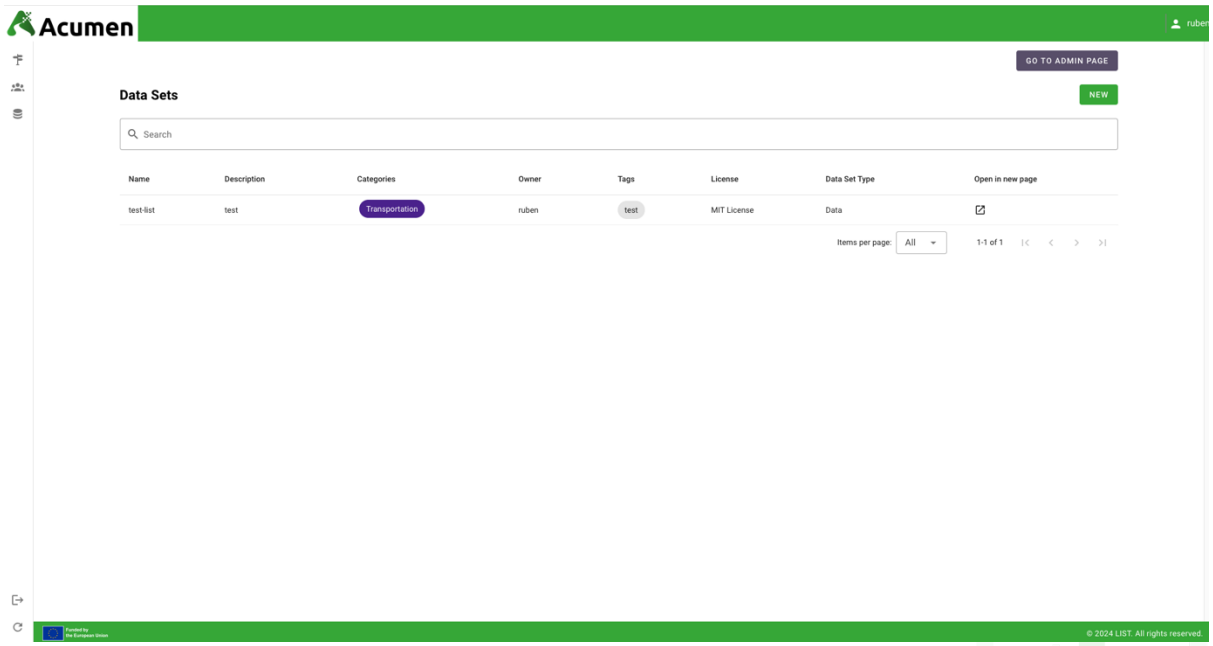

Figure 6: Data manager graphical interface (home)

For each dataset, different versions are available containing a set of files, as shown by Figure 7.

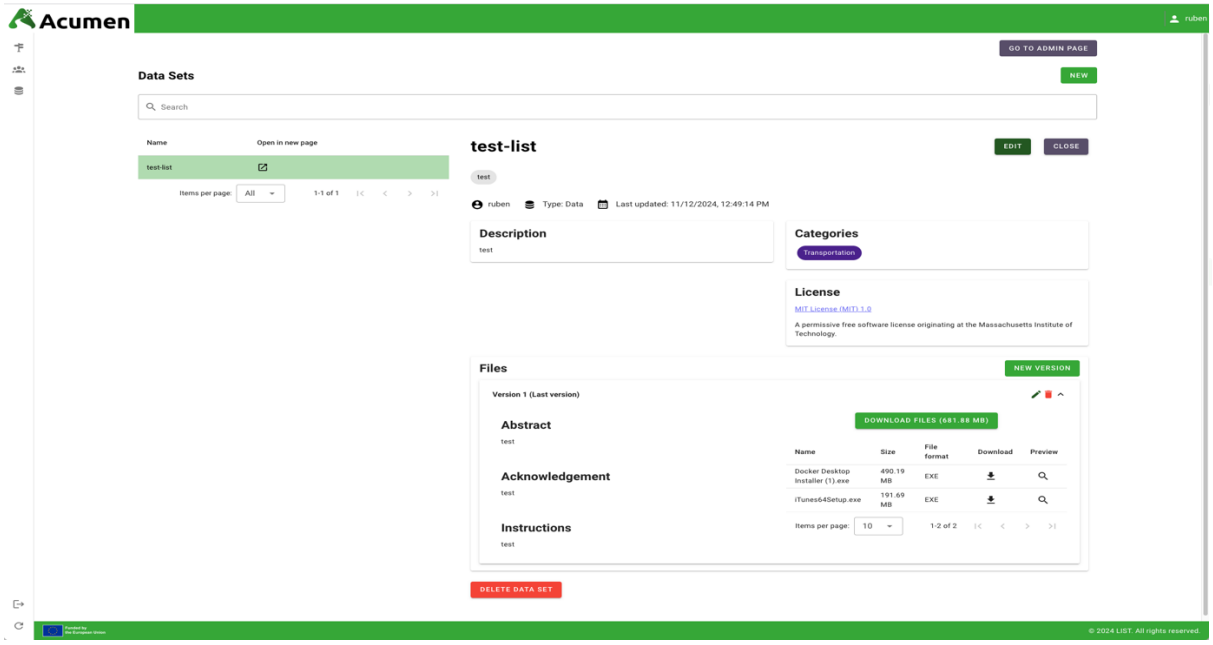

Figure 7: Dataset information

Each dataset could have the following information:

- Name: the name for this dataset. Should be a String
- Description: a description for this dataset. Should be a String

- Tags: associated tag(s) for this dataset.
- Categories: associated category for this dataset.
- Type: type for this dataset. Should be a String
- License: associated license for this dataset. Should be a String

Each version of a dataset contains a set of files and the following information:

- Abstract: abstract for the current version. Should be a String
- Acknowledgment: information about the owner of this dataset
- Instructions: information about how to use this dataset. Should be a String

6. Security Integration

This section presents the user account management, and the authentication and authorization mechanism used in the preliminary version of the ACUMEN platform.

6.1. Authentication and Authorization

In a microservices architecture, applications are composed of multiple independent services, each potentially requiring authentication. Without centralized authentication, users would need to authenticate separately for each service, leading to a fragmented user experience. Keycloak[4], open-source identity and access management solution that seamlessly integrates with microservices architectures, addresses this issue by offering centralized authentication and Single Sign-On (SSO). Users authenticate once with Keycloak, and their session is propagated across all microservices. This simplifies the user experience and reduces the complexity of handling multiple authentication mechanisms across services.

Keycloak supports standard authentication protocols such as OpenID Connect (OIDC), OAuth 2.0, and SAML, ensuring compatibility with modern systems and applications. These protocols make it easy for microservices to verify user identities via access tokens, eliminating the need for password handling within individual services.

Keycloak issues JSON Web Tokens (JWT) upon successful authentication. These tokens contain user identity and role information, enabling microservices to perform stateless, token-based authentication. Each microservice validates the token against Keycloak, allowing them to enforce fine-grained access control.

Additionally, Keycloak allows defining roles, groups, and permissions, which can be assigned at both global and service-specific levels. For instance, for ACUMEN, we define these roles:

- **Global Roles:** *acumen-admin* role apply across all services.
- **Service-Specific Roles:** *acumen-data-provider* and *acumen-data-scientist* permissions tailored to individual microservices, here for “Data Manager” service.

This flexibility ensures precise control over access while keeping security policies centralized.

Figure 8 shows the first version of the graphical interface developed for managing access rights to specific services for a user.

[4] https://www.keycloak.org/

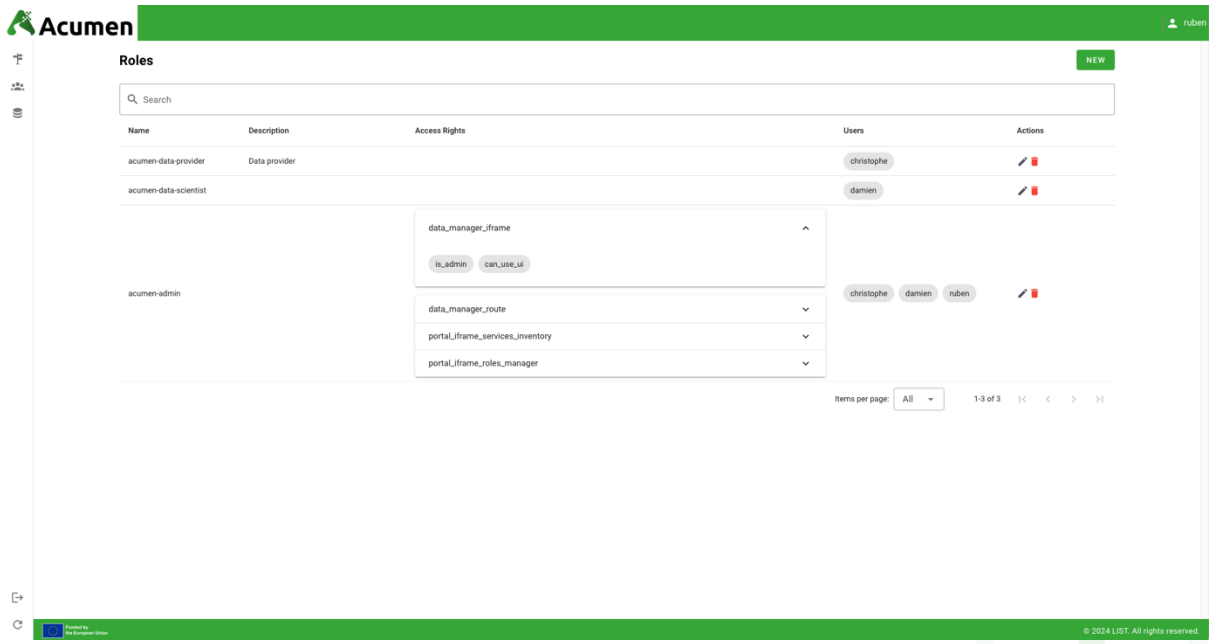

Figure 8: Role's manager interface

Through this interface, the administrator of the ACUMEN platform can manage the different roles and the access rights associated for each user account.

7. Conclusion

The successful implementation of a microservices architecture relies heavily on well-defined and robust integration principles. By utilising decentralised data management, clear ownership of data within bounded contexts, and adopting patterns such as API-based communication and event-driven programming, the ACUMEN platform enables seamless interaction between services.

The integration principles outlined in this deliverable provide a foundational framework to address the complexities of data flow and service interactions in a microservices ecosystem. By adhering to these guidelines, development teams can build tools that are not only technically efficient but also adaptable to evolving business needs, delivering long-term value and operational excellence.