D3.1

Architecture and data sharing ecosystem

UNDER REVIEW

www.verneproject.eu
info@verneproject.eu

#VERNEPROJECT

Funded by the European Union

D3.1	Architecture and data sharing ecosystem
DELIVERABLE TYPE	Report
MONTH AND DATE OF DELIVERABLE	April, 30/04/2026
WORK PACKAGE	WP3
LEADER	UNISYSTEMS
DISSEMINATION LEVEL	PU - Public
PROGRAMME	HORIZON EUROPE
GRANT AGREEMENT	101181941
START	1 November 2024
DURATION	36 Months

VERNE has received funding from the EU's Horizon Europe research and innovation programme under Grant Agreement no. 101181941.

Contributors

NAME	ORGANISATION
Giannis Gialelis	Unisystems
Joseph Psistakis	Unisystems
Ilias Aliferis	Unisystems
Grigorios Protopsaltis	Unisystems
Chris Mountzouris	Unisystems
John Prokopakis	Unisystems
Bill Gialelis	Unisystems

Peer Reviews

NAME	ORGANISATION
Tuomas Kari	LUKE
Pau Fonseca	INLAB
Janja Viher	ITC

Document Log

DATE (DD/MM/YYYY)	ORGANISATION	CONTRIBUTORS	MODIFICATIONS
14/04/2026	Unisystems	John Gialelis, Joseph Psistakis	First Version
17/04/2026	Unisystems	Ilias Aleiferis & Thodoris Mitsis (Updated Section 4.6)	Second Version
28/04/2026	Unisystems	John Gialelis & Ilias Aliferis (Integrated Final Version)	Third Version
29/04/2026	PNO	Carolina Salas	Final format check before submission

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 European Commission. Neither the European Union nor the granting authority can be held responsible for them.

Index of Contents

1 Executive Summary 11
2 Introduction 13
2.1 Purpose and Scope of this Document 13
2.1.1 Purpose of D3.1 13
2.1.2 Scope of D3.1 13
2.2 Relationship with Deliverables of other WPs 14
2.3 VERNE Ecosystem Vision 15
2.4 Ecosystem Overview 15
3 High Level Platform Architecture 18
3.1 Architectural Principles 18
3.1.1 Design methodology 18
3.1.2 The ISO/IEC/IEEE 42010:2022 Standard 20
3.1.3 The 4+1 Architectural View Model 22
3.1.4 Architectural & Ecosystem considerations 23
3.2 Goals of the VERNE Platform 24
3.2.1 Scope and Context 24
3.2.2 Platform clients 26
3.2.3 Concerns & Requirements 27
3.2.4 Agents 30
3.3 Architecture Views 31
3.3.1 Platform Boundary 31
3.3.2 Key required components 32
3.3.3 Logical View 34
3.3.4 Deployment View 37
3.3.5 Platform Integration – CI/CD 40
3.3.6 Process View 55
3.3.7 Development View 59
3.3.8 Scenarios 60
4 Pilot system architectures and data integration points 62
4.1 Pilot 1 – Bornholm (Food Experience and Circular Tourism) 62

4.1.1 AS-IS situation 62
4.1.2 Digital infrastructure 64
4.1.3 Pilot objectives & Use Cases 64
4.1.4 Architecture mapping & integration points 65
4.1.5 Data specifications 66
4.1.6 Circularity indicators 67
4.1.7 VERNE platform digital tools 68
4.2 Pilot 2 – Sønderjylland-Schleswig (Sustainable Corporate Heritage 68
4.2.1 AS-IS situation 68
4.2.2 Digital infrastructure 69
4.2.3 Pilot objectives & use cases 69
4.2.4 Architecture mapping & integration points 70
4.2.5 Data specifications 71
4.2.6 Circularity indicators 72
4.2.7 VERNE platform digital tools 72
4.3 Pilot 3 – Santorini (Impact Hotel) 72
4.3.1 AS-IS Situation 72
4.3.2 Digital and sensing infrastructure 74
4.3.3 Pilot objectives & use cases 75
4.3.4 Architecture mapping & integration points 76
4.3.5 Data specifications 77
4.3.6 Circularity indicators 77
4.3.7 VERNE platform digital tools 78
4.4 Pilot 4 – Uusimaa (Virtual Nature-based Tourism) 79
4.4.1 AS-IS situation 79
4.4.2 Digital infrastructure 79
4.4.3 Pilot objectives & Use Cases 80
4.4.4 Architecture mapping & integration points 81
4.4.5 Data specifications 82
4.4.6 Circularity indicators 83
4.4.7 VERNE platform digital tools 83
4.5 Pilot 5 – Pomurje (Regional ZeroNET Food System) 83
4.5.1 AS-IS situation 83

4.5.2 Digital infrastructure 84
4.5.3 Pilot objectives & Use Cases 85
4.5.4 Architecture mapping & integration points 86
4.5.5 Data specifications 88
4.5.6 Circularity indicators 89
4.5.7 VERNE platform digital tools 90
4.6 VERNE Marketplace 91
4.6.1 Integration Flow 91
4.6.2 VERNE Marketplace – Technical Viewpoints 92
4.6.3 VERNE Trustworthiness Marketplace 104
5 Conclusions 110
6 References 111

Index of Tables

Table 1. Relationship with Deliverables of other WPs....14
Table 2. Data Formats & Sampling Strategies....23
Table 3. List of Stakeholders....27
Table 4. List of Design Concerns....29
Table 5. Alternative CI/CD solutions....41
Table 6. Alternatives to Grafana....41
Table 7. Alternatives to Prometheus....42
Table 8. Alternatives to Loki....43
Table 9. Alternatives to Jaeger....44
Table 10. Alternatives to Ansible....45
Table 11. Alternatives to RabbitMQ....46
Table 12. Alternatives to Debezium....47
Table 13. Alternative time-series databases....50
Table 14. Alternatives to Keycloak....53
Table 15. Alternatives to APISIX....54
Table 16: WordPress-based stack....93
Table 17: Event Type & Related APIs....94
Table 18: Related Use Case with solution component....109

Index of Figures

Figure 1. VERNE Ecosystem Overview....16
Figure 2. Architecture Concepts & their relationships in ISO/IEC/IEEE 42010....21
Figure 3. The VERNE Ecosystem....26
Figure 4. Key Required Components....32
Figure 5. Logical View....34
Figure 6. Deployment View....38
Figure 7. Sensor data flow....55
Figure 8. Agent Computation Process....56
Figure 9. Agent lifecycle....57
Figure 10. Agent Initialization and Registration....59
Figure 11. Development process....59
Figure 12. Data Aggregation to agents....60
Figure 13. Configuration Scenario....61
Figure 14. Developer access....62
Figure 15. ZeroNET Food System Data integration with VERNE ecosystem....88
Figure 16: Vendor Sales Cycle....96
Figure 17 Consumer Purchasing Cycle (Points Economy)....98

Figure 18: Register Screen 99
Figure 19: Vendor Dashboard 99
Figure 20: Add New Product 100
Figure 21: Product Management 101
Figure 22: Order View 102
Figure 23: Product View & Add to Cart 102
Figure 24: Checkout and Order Confirmation 103
Figure 25: Order History 104
Figure 26: Order Details & Status 104
Figure 27: Trustworthiness Market & Decentralized Ledger 105
Figure 28: Trustworthy Zero-Waste Food Transaction in VERNE Marketplace 108
Figure 29: Tokenisation API & Ledger flow 110

Partners short names

PNO	PNO INNOVATION SL
RETHINK	ASSOCIACION RE THINK TOURISM
CETT	ESTUDIS D ' HOTELERIA I TURISME CETT SA
INLAB	UNIVERSITAT POLITECNICA DE CATALUNYA
ITC	ITC - INOVACIJSKO TEHNOLOSKI GROZD MURSKA SOBOTA
SDU	SYDDANSK UNIVERSITET
CSCP	COLLABORATING CENTRE ON SUSTAINABLE CONSUMPTION AND PRODUCTION GGMBH
LUKE	LUONNONVARAKESKUS
UNI	UNI SYSTEMS SYSTIMATA PLIROFORIKIS MONOPROSOPI ANONYMI EMPORIKI ETAIRIA
ANRI	ALCHEMIA-NOVA RESEARCH & INNOVATION GEMEINNUTZIGE GMBH
IHA	IMPACT HUB LABS
APRE	AGENZIA PER LA PROMOZIONE DELLA RICERCA EUROPEA
BUU	BURSA ULUDAG UNIVERSITESI
ATU	ATLANTIC TECHNOLOGICAL UNIVERSITY
LU	LUNDS UNIVERSITET

Abbreviations

0FLW	Zero Food Loss and Waste
ADA	Agent Data Access
API	Application Programming Interface
APISIX	Apache APISIX (API Gateway)
CDC	Change Data Capture
CI	Continuous Integration
CI/CD	Continuous Integration / Continuous Delivery
CNCF	Cloud Native Computing Foundation
CSV	Comma-Separated Values
D3.1	Deliverable 3.1
DADS	DIH Agrifood Data Space
DIH	Digital Innovation Hub
EDA	External Data Aggregator
ELT	Extract, Load, Transform
ERP	Enterprise Resource Planning
EV	Electric Vehicle
FR	Functional Requirement
GHG	Greenhouse Gas
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure

IAM	Identity and Access Management
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
JWT	JSON Web Token
KPI	Key Performance Indicator
MQTT	Message Queuing Telemetry Transport
NBT	Nature-Based Tourism
NFR	Non-Functional Requirement
OAuth	Open Authorization
OCI	Open Container Initiative
REST	Representational State Transfer
SDA	Sensor Data Aggregator
SQL	Structured Query Language
SSO	Single Sign-On
TLS	Transport Layer Security
UC	Use Case
UI	User Interface
UIA	User Input Aggregator
URL	Uniform Resource Locator
VR	Virtual Reality

1 Executive Summary

VERNE brings together a set of diverse pilots to demonstrate circular economy practices in tourism-related contexts. Because these pilots operate in different domains and maturity levels, VERNE requires a modular digital ecosystem that can ingest heterogeneous data, support pilot-specific digital tools, and provide consistent ways to compute and communicate circularity evidence.

Deliverable D3.1 is the main output of Task 3.1 (Data requirements and architecture) and provides the architectural and data foundations needed to implement WP3 digital tools. It defines an open, secure, and extensible high-level architecture for the VERNE ecosystem and outlines how data will be collected, structured, and managed so that different digital solutions (services, tools, modules) can be developed and integrated over time. The deliverable is written for public dissemination and therefore focuses on the architecture and data handling approach at a high level, avoiding operational or sensitive implementation details.

At platform level, D3.1 proposes a modular, component-based architecture suitable for a multi-pilot environment. The approach follows recognized architecture description practices (including a viewpoint-based description aligned with established standards) and organizes the platform into functional building blocks that cover:

- Data ingestion from multiple channels, including sensor/IoT streams, user-submitted information (forms/uploads), and external/third-party sources (e.g., reference datasets).
- Orchestration and scheduling, enabling automated workflows for data collection, processing, and indicator generation across pilots.
- Scalable storage appropriate for the expected data types (e.g., time-series for monitoring data, state and configuration storage, and support for structured/graph-like representations where needed).
- Security and access management, enabling controlled access across multiple stakeholder roles.
- Operational integration practices, such as containerized deployment and CI/CD concepts, supporting maintainability and future scalability.

From a data perspective, D3.1 establishes the basis for handling heterogeneous pilot datasets by proposing a data collection and management approach that supports:

- definition of pilot data categories (what is collected),
- specification of formats and exchange mechanisms (e.g., APIs where available, MQTT/streaming for telemetry, and periodic structured exports where necessary),

- harmonization principles to support interoperability and consistent interpretation, and
- reliable information-sharing processes that can scale as pilots evolve and new services are introduced.

For each pilot, D3.1 summarizes (i) the AS-IS situation, (ii) the pilot objectives and use cases, (iii) expected data categories and data sources, and (iv) integration points required to connect the pilot to the broader VERNE ecosystem. This supports a realistic implementation pathway, acknowledging that pilots differ substantially: some provide direct operational/transactional datasets from existing systems, others rely primarily on structured documentation and surveys, and others provide real-time monitoring streams. KPIs are referenced, providing valuable inputs to be integrated within the different layers.

Overall, D3.1 delivers a consolidated view of how VERNE can support multiple circular tourism scenarios through a common architectural backbone and a pragmatic data strategy, enabling (a) pilot-specific digital tools, and (b) consistent evidence generation and communication across the ecosystem.

Next steps include refining pilot inputs as they mature, and iteratively validating the architecture and data exchange approach through WP3 implementation and integration activities.

2 Introduction

2.1 Purpose and Scope of this Document

2.1.1 Purpose of D3.1

D3.1 defines the architectural and data foundation of the VERNE ecosystem so that WP3 can implement digital tools in a consistent way across diverse pilots. Specifically, it:

- Specifies a modular, secure, open, and extensible high-level platform architecture capable of hosting multiple digital solutions (services, tools, modules) and supporting future scalability.
- Establishes a common architectural language and set of views (including a methodology aligned with architecture description standards) to ensure a shared understanding among stakeholders.
- Maps components, interactions, and communication needs—including alignment with the SGAM framework, as described in Task 3.1—to support interoperability and acceptability.
- Defines the initial data requirements from pilots and the corresponding platform-side capabilities needed to ingest, manage, and utilize these data to enable circularity-related digital services.

2.1.2 Scope of D3.1

D3.1 covers:

- The VERNE platform architecture (core building blocks, interactions, and integration logic).
- Pilot system architectures and data integration points (what each pilot can provide and how it connects).
- A data collection and management approach, including:
    - data categories and indicative fields,
    - exchange mechanisms (e.g., APIs where available, MQTT/streaming where relevant, periodic structured exports),
    - harmonization/interoperability considerations,
    - scalable storage approaches (relational/NoSQL/graph as needed) and data flow adjustments.
- Guidelines and constraints that steer implementation of WP3 tools and future integrations.

2.2 Relationship with Deliverables of other WPs

WP	Related deliverable(s)	Relationship to D3.1
WP1	D1.2 (Multidimensional transition framework)	D3.1 builds on the KPI framework developed in WP1 (T1.2, integrated in D1.2) by translating the validated KPIs into concrete data requirements, data exchange specifications, and architectural components required for their computation and presentation
WP2	D2.1 (Stakeholder map)	D3.1 builds on the stakeholder matrices produced in WP2 (T2.1, D2.1) by translating stakeholder needs, influence, and early-adopter priorities into platform requirements, role-based access considerations, and the functional scope of WP3 digital tools across pilots.
WP3	D3.3 & D3.4 (VERNE Ecosystem releases)	The work plan explicitly states that the integrated ecosystem releases are based on the T3.1 architecture; therefore, D3.1 defines the reference architecture and data standards that D3.3/D3.4 implement and deploy.
WP3	D3.6 (VERNE Marketplace)	D3.1 underpins T3.5 by defining the VERNE ecosystem architecture and data integration approach required for an incentives marketplace, including the ingestion and harmonization of pilots' data and modular interfaces that enable cross-pilot marketplace operation and scalability.
WP4	D4.2 (Progress report of pilots & CoPs activities)	T4.2 (WP4) operationalizes the VERNE pilots and monitors their progress, while D3.1 (WP3) provides the modular system architecture, integration points, and data requirements that enable the pilots' digital tools and data streams to be collected, harmonized, and used for demonstration monitoring and circularity performance assessment.
WP5	D5.5 (One-Stop-Shop for sustainable tourism)	D3.1 provides the architectural and data integration foundation that enables the implementation of the One-Stop-Shop developed under T5.4 (D5.5). In particular, D3.1 defines how pilot data and results can be collected, harmonized, stored, and accessed through standardized integration points, so that the One-Stop-Shop can reliably retrieve and present VERNE outputs (methods, guidelines, tools, success cases, and pilot evidence) to different stakeholder segments.

Table 1. Relationship with Deliverables of other WPs

2.3 VERNE Ecosystem Vision

VERNE's mission is to accelerate the transition of European tourism destinations from linear to circular and sustainable models by turning circular economy principles into practical, measurable, and widely adoptable solutions. VERNE does this by demonstrating systemic interventions in real tourism contexts—covering resources, services, and local value chains—so that destinations and businesses can reduce waste, improve resource efficiency, and strengthen resilience without compromising the quality of visitor experiences or local livelihoods. The project emphasizes approaches that are affordable, scalable, and replicable, ensuring that lessons learned and proven practices can be transferred beyond the pilots to other regions across Europe.

To support this transformation, VERNE establishes a unified, open, and secure digital ecosystem that enables diverse actors—public authorities, tourism businesses, local communities, researchers, and visitors—to engage with circular solutions in an evidence-based way. The ecosystem connects stakeholders around shared goals, consolidates relevant data and knowledge, and makes circularity performance visible through meaningful indicators and narratives. By translating pilot outcomes into accessible tools, guidelines, and decision-support assets, VERNE helps stakeholders understand what works, why it works, and how it can be implemented in their own settings. Ultimately, VERNE aims to build a lasting foundation for circular tourism by enabling destinations to plan, adopt, monitor, and continuously improve circular practices across the tourism value chain.

2.4 Ecosystem Overview

The VERNE Ecosystem brings together a diverse set of actors in the private and public sector, across 5 distinct geographic areas, in an effort to promote circularity and sustainable tourism through technological innovation. The Project aims to provide an integrated technological framework to this diverse set of actors. These actors, which can be public authorities, local communities or groups of businesses, will in turn apply this technological framework in pilot projects (the Pilots) as a case study, benefitting from the technologies and providing early adopter feedback.

As such, at the most abstract level, the ecosystem is comprised by two major groups of stakeholders: (1) the Technology providers and (2) the Early Adopters, which will be introducing custom solutions based on the technology. In the following paragraphs we present the VERNE Ecosystem in higher detail, from the Functional and Business standpoints.

Figure 1 below provides an overview of the VERNE Ecosystem at the functional level. As illustrated, the VERNE Project consists of a technological solution which provides functionality to different communities across Europe – each community is a separate Case Study, which uses a subset of the technological solution for its own specific use-case.

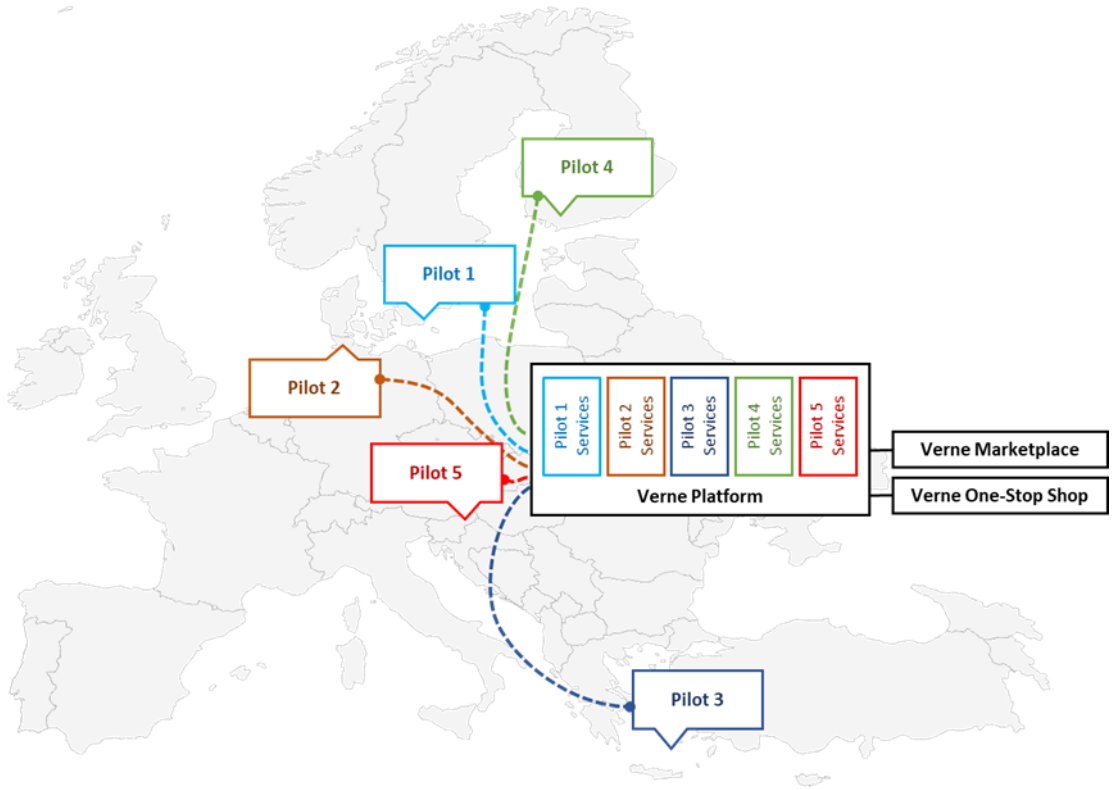


Figure 1. VERNE Ecosystem Overview

More Specifically, the key parts of the VERNE technological framework are the following:

- The VERNE Platform is a software platform that manages the flow of information, aggregating data from Points-Of-Interest and running automated computational workloads on that data, on behalf of its users. The computational workloads can be computational models (digital twins), machine learning systems for forecasting, what-if-analysis systems or even simple KPI (Key Performance Indicator) calculations.
- The VERNE One-Stop-Shop is an enhanced knowledge base that catalogues and documents the sustainability and circularity actions taken in the Project, promoting the relevant best practices and supplying the relevant performance data related to those practices, as they have been measured in the project.
- The VERNE Marketplace is an award-based market, designed to offer rewards as incentives to positive behaviours and environmental action. The idea behind the Marketplace is to provide a tool for communities to incentivize positive actions for their members, be it by rewarding recycling and upcycling or by providing a token-based platform for the exchange of goods.

On the other end, 5 distinct communities, the VERNE Pilots, will be integrating parts of this technological framework, to enhance their operation. These Pilots are:

1. **Bornholm (DK).** Bornholm is a Danish island in the Baltic Sea, off the coast of Sweden. A tourism-focused consortium aims to measure the food consumption and production aspects of the local food-tourism sector, introducing sustainability initiatives.
2. **Sønderjylland-Schleswig (DK-DE)**. The region is host to many corporate heritage museums. The local tourism agencies aim to introduce new sustainability initiatives that leverage the significant cultural heritage and boost local tourism, also helping visitors discover the many museums in the area.
3. **Santorini (EL)**. As with many Aegean islands, Santorini sees a significant increase in population every summer, which in turn leads to increased demand for water and other resources. A local consortium of NGOs and property owners are introducing circularity and sustainability in the tourism industry, digitizing the water management systems and providing greywater recycling facilities to the local agriculture. The goal is to significantly reduce water wastage, while at the same time providing an irrigation boost during the summer months, when it is most needed.
4. **Uusimaa (FI):** Uusimaa is the southernmost and most populous region of Finland, holding the capital city, Helsinki. The pilot uses LUKE's VirtuLab research infrastructure to create and test virtual nature-based tourism experiences (from Nuuksio national park), aiming to complement or partially substitute physical visits, reduce travel-related impacts, and improve access for users who face barriers to travel. Evidence is collected mainly through researcher-led surveys and observations (usage, experience types, beneficiary groups, and indicative avoided GHG emissions).
5. **Pomurje (SL)**: Pomurje is an agriculturally rich area in Slovenia, known for its thermal and mineral waters. A local consortium aims to introduce sustainability and circularity actions by minimizing food and fuel waste in the distribution of local food for Tourism and Agriculture purposes. The pilot develops an innovative system that not only incentivizes producers to provide surplus food back in circulation, but also helps minimize emissions through smart food delivery.

The Pilots above will be connecting to the VERNE Platform in various ways, according to the most fitting mode of operation. The core concept, however, remains the same with every Pilot: The VERNE Platform will be collecting sustainability and circularity data, either through sensors or by user submission and will be providing key insight through various computational workloads, i.e. via the calculation of KPIs (Key Performance Indicators) or by providing a digital model or ML forecasts of elements of the system.

Administrators will be able to set up sensor streams, KPIs, Points-of-Interest, event-processing chains and measurables that will help them measure their circularity actions, make decisions regarding their environmental impact and ultimately track their progress as they implement business actions towards sustainability. They will also be able to leverage the VERNE Platform and the Marketplace to incentivize visitors and community members.

In the meantime, the VERNE governing body, in conjunction with the Pilots, will be developing a knowledge base of best-practices, documenting the Circularity and Sustainability efforts so that other regions can establish similar actions in the future.

3 High Level Platform Architecture

This chapter aims to provide an overview of the VERNE Platform – the part of the ecosystem developed as part of the VERNE project that acts as a central hub of information and processing apparatus between the various stakeholders and Pilot applications.

The VERNE Platform aims to be the central information hub between the various Pilot projects, streaming information from Pilot sites, providing Digital-Twin based calculations and simulations where necessary and ultimately distilling all Pilot-relevant Key Performance Indicators in an easy-to-read, actionable form.

Together with an introduction to the VERNE Platform and its place in the VERNE Ecosystem, this chapter discusses the core principles, requirements and goals set for the Platform, as well as the architecture that covers those requirements. It presents the design methodology, informed by the ISO/IEC/IEEE 42010:2022 Standard [1] and maps requirements to design decisions, also detailing key architectural features.

3.1 Architectural Principles

This section discusses the key architectural principles behind the VERNE Platform, namely the mode of operation when designing a central hub for a multi-client ecosystem with multiple stakeholders, each with their own requirements and pre-existing functionalities.
It briefly discusses the overall design methodology implemented, the ISO/IEC/IEEE 42010 standard for architectural design, its application in the popular 4+1 Architectural View Model, and the project-specific considerations that further inform design decisions.

3.1.1 Design methodology

The VERNE Project aims to provide a central multi-functionality data hub to facilitate and measure large scale initiatives in ecology, sustainability and circularity actions, across multiple domains – the VERNE Pilots.

Each of the 5 distinct VERNE Pilots has its own unique requirements and challenges, as it applies to a specific region, industry and set of stakeholders and has a pre-existing set of technologies of varying maturity and technological readiness: In summary, two of the Pilots have a Food Sustainability component, four of the Pilots would benefit from an Experience / Service discovery component and at least one Pilot requires a real-time resource monitoring component.

A central Platform to support multiple such Pilots would need to address the common modalities, if any, then implement Pilot-specific functionalities. It would also need to allow for future extensibility, both vertically (i.e. with the inclusion of new solutions in existing Pilots) and horizontally (i.e. with the expansion to new regions).

Our approach to designing the Architecture was to apply the ISO/IEC/IEEE 42010:2022 Standard, optimizing for the common requirements that exist across Pilots, while also providing a foundation for expandability using industry-standard practices and solutions.

A popular architectural design process, compatible with the Standard, was formulated by Rozanski and Woods [2], in which "Architecture Definition is a process by which stakeholder needs and concerns are captured, an architecture to meet these needs is designed, and the architecture is clearly and unambiguously described via an architectural description." In short, this entails the following actions:

1. Define the Initial Scope and Context
2. Engage Stakeholders
3. Capture First Cut *Concerns*[1] (in conjunction with the Stakeholders)
4. Use the *Concerns*[1], Scope and Context to Define the Architecture, producing an Architectural Description and a set of Guidelines and Constraints.
5. Create a Skeleton System

In practice, the scale of the VERNE Project, with multiple Pilots and Use-Cases, necessitated a refinement of that action plan, where each Pilot would constitute a group of Stakeholders and would be considered separately. Then, the common requirements would be grouped when considering Scope and Context: the intent was to design a flexible platform that addresses the common needs across the Pilots, while allowing specific solutions to be implemented and new regions to be connected in the future.

In summary, we followed the following workflow:

1. Define the Initial Scope and Context of the VERNE Platform.
2. Engage Stakeholders for each Pilot,
   a. Identify common requirements shared between the Pilots
   b. Identify necessary Pilot-specific functionalities that need to be supported
3. Catalogue unique requirements, Functional and Non-Functional, across all Stakeholders
4. Use this information to derive an architecture that maximally covers Pilot requirements while minimizing extraneous components
5. Create a Skeleton System

[1] A *Concern*, per the ISO/IEC/IEEE 42010 nomenclature, is an interest that is important to one or more stakeholders. This can be a requirement, either functional or non-functional, or an objective that a stakeholder has with respect to the system.

3.1.2 The ISO/IEC/IEEE 42010:2022 Standard

The current standard for architecture design, ISO/IEC/IEEE 42010:2022 [1], is an evolution of previous standards, notably ANSI/IEEE 1471 [3], published in 2000 as the first prevailing approach to software architecture design. It provides a structured methodology for creating architectural descriptions (ADs) for software-intensive systems, including a step-by-step workflow.

The workflow includes identifying and documenting stakeholders and their architecture-related concerns, selecting and detailing architecture viewpoints to address these concerns, generating architecture views with corresponding models, ensuring view consistency, and documenting the rationale for architectural decisions.

The Standard relies heavily on viewpoints and views to capture different facets of a software system, allowing focused examination of specific concerns and issues while maintaining overall architectural consistency. This approach has proven successful in several cases, delivering higher quality artifacts and deliverables than less structured methods that attempt to address all issues in a single pass.

Viewpoints are essentially collections of patterns, templates, and conventions tailored to construct specific types of views. For example, the functional viewpoint contains all the necessary functions for architecture description.

Originally, the precursor standard of ANSI/IEEE 1471 aimed to establish best practices for the software and systems communities, addressing the fragmented landscape of architectural practices and description languages prevalent at the time. The wide adoption of ANSI/IEEE 1471 highlighted benefits of having a common conceptual foundation and terminology within the software architecture community.

Over decades of use within the industry, academia and the government sectors, the Standard evolved even further. The 2011 version introduced additional features such as modelling architecture decisions, managing correspondence between views, and supporting Architecture Description Languages (ADLs) and architecture frameworks. It remains the benchmark standard for software architecture design. The 2022 version further expanded on the standard.

In its current iteration, the standard defines a comprehensive methodology for the architectural description of software-intensive systems that includes a workflow with steps such as identifying stakeholders, identifying architecture-related concerns, selecting architecture viewpoints, creating architecture views with models, and analysing their consistency. The standard continues to use viewpoints and views to document different aspects of a software system, highlighting specific concerns and issues while ensuring a coherent architectural design.

Rozanski and Woods revised the ISO/IEC/IEEE 42010:2011, where these key definitions are introduced. In Figure 2 the relationships between these concepts are graphically represented.

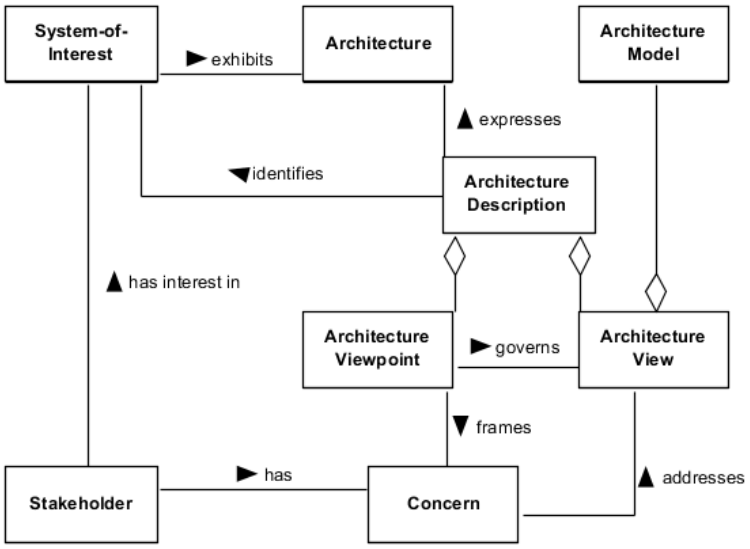


Figure 2. Architecture Concepts & their relationships in ISO/IEC/IEEE 42010

- **System of interest:** the system whose architecture is being studied or considered.
- **Stakeholder:** an individual, group, organization, or similar entity with a vested interest in the realization of the system.
- **Architecture:** the basic ideas or characteristics of a system within its environment, expressed through its components, connections, and the principles that guide its design and growth. Essentially, it's how a system is structured, its properties, how it interacts with its environment, and so on.
- **Architectural Description:** a document or output used to convey the structure of an architecture, which may include diagrams such as component diagrams or data flow diagrams.
- **Concern:** an interest in a system that is important to one or more stakeholders. This can be a requirement, either functional or non-functional, or an objective that a stakeholder has with respect to the system.
- **View:** a collection of models and explanations that represent a system or part of it, focusing on a particular set of concerns.
- **Viewpoint:** a set of patterns, templates, and conventions that guide the creation of a particular type of view.
- **Model:** a simplified representation of a facet of the architecture, often in the form of a UML diagram[2].

[2] In the context of this deliverable, it was preferred not to use UML diagrams, in order to provide an easily digestible, simplified high-level overview. With that in mind, minor or irrelevant technical details may be omitted in favor of simplicity.

As ISO/IEC/IEEE 42010 is designed to be maximally generalizable, simpler case-specific specializations such as the "4+1 Architectural View Model" [4] and the "C4 model" [5] have seen extensive use in the industry. The 4+1 Architectural View Model strikes an optimal balance between simplicity and representational expressivity for a project in the scale of the VERNE Platform, so it was chosen for the Architectural Descriptions.

3.1.3 The 4+1 Architectural View Model

The 4+1 model [4] is an application of the larger ISO/IEC/IEEE 42010 Standard. It aims to address all *stakeholder concerns* (vis-à-vis the 42010 nomenclature), providing an easy-to-understand model of the architecture, described in 4+1 views. The "+1" in the model represents the "Scenario" view, which complements the four traditional architectural views by providing specific use cases or scenarios that demonstrate a system's behaviour in various situations.

More specifically, the 4+1 model defines the following views:

- **The Logical View** (Functional Viewpoint). This view focuses on the functional requirements of the system and describes the system in terms of key abstractions or objects. It outlines the interactions, responsibilities, and relationships between these objects. The main purpose is to capture the high-level design of the system's functionality.
- **The Physical View** (Deployment Viewpoint). The physical view describes the deployment and distribution of the system components across the hardware. It includes details about the network connections, and how software components are mapped to the physical environment. The goal is to specify the system's deployment architecture, addressing hardware and network considerations.
- **The Process View.** This view addresses the runtime aspects of the system. It illustrates how the system will be structured to handle concurrent execution, including processes, threads, and their interactions. The objective is to show how the system components will execute concurrently and how they will coordinate and communicate.
- **The Development View.** This view focuses on the software development aspects and provides guidance on how to organize the software development process. It includes information about coding standards, development tools, and version control. It is designed to help developers implement the system and ensure consistency and efficiency.
- **The Scenarios View.** The scenarios, or use case view, illustrates how the system behaves in response to different scenarios or use cases. It highlights the interaction between the system and external entities under various conditions. Its purpose is to provide a realistic, user-centric perspective on the system's operation.

3.1.4 Architectural & Ecosystem considerations

The VERNE Ecosystem is comprised of multiple Pilots, each with their own use case for a specific industry and a specific region. This heterogeneity of the ecosystem warrants special consideration, especially since some Pilots may require real-time, or near-real-time data processing while others may not have the capacity to submit real-time data and will be using file-based data exchange methods.

To support such a heterogeneous ecosystem, the Platform will need to be flexible enough to allow multi-modal integrations: via APIs or via manual data entry and file transfers when necessary. This horizontal requirement essentially stems from the definition of the project.

We worked closely with the Pilots to identify the data requirements and the data entry modes of operation, including the sampling frequency for the input data, which ultimately affects the performance requirements for the platform. Depending on the use case, Pilot data can be reports (automatically or manually submitted), user-submitted and curated data or live sensor data streams.

The various possible data formats and sampling frequencies are summarized in Table 2 below.

Type of Data	Data Format	Sampling Period	Sampling strategy
Sensor Data	MQTT Streams	15s – 1h	Push
	API-based	1h – 24h	Pull (polling)
Structured Reports	CSV/JSON files	Daily – Annually	Manual file input
	API-submitted CSV/JSON data	Daily – Annually	Push
Third-party data	API-based	6h – Annually	Pull (polling)
KPI components	Interactive form data	Daily – Annually	Manual UI input

Table 2. Data Formats & Sampling Strategies

The table describes data of various types and formats:

1. **Sensor Data** are data originating from various sensors, which can be used in conjunction with Digital Twins and procedural predictors to model a system and calculate various parameters. This can be used both to model the current state of a physical system and in what-if analysis.
2. **KPI Components** are individual variables that can be used to calculate a Key Performance Indicator. Manual input of such variables could be required because KPIs may rely on information that is not readily available and cannot be obtained procedurally – for example, the total area of irrigated farmland, or the total tonnage of recyclable material collected in an area. Such information can be collected and submitted manually, generating a datapoint and prompting recalculation of a KPI by the system. Those can be typically submitted through a form-based User Interface.

3. **Structured Reports** are documents prepared procedurally or manually that contain datapoints for various variables pertinent to a system – mostly compilations of KPI components or similar operational data.
4. **Third-party Data** are data that can be obtained via API calls to third party data sources, for use in a digital twin or other procedural model. For example, weather forecasts can be used to predict Photovoltaic panel performance and heating requirements. Traffic data can be used as an indicator of fuel expenditure in a shipping / logistics model.

3.2 Goals of the VERNE Platform

This section discusses the Goals and Requirements of the VERNE Platform. As with any software platform, understanding the Requirements is essential.

The ISO/IEC/IEEE 42010 standard uses the term "concerns" to describe any requirement or goal from the viewpoint of a Stakeholder. This is to delineate between explicit requirements and other goals and constraints. All such Stakeholder concerns are equally considered by the Standard. Furthermore, concerns are prioritized and those with significant impact on the design of an architecture are characterised as Architecturally Significant (ASCs).

Unless otherwise noted, the terms "Requirements" and "Concerns" are used interchangeably in this section.

As discussed previously, the steps to produce an Architectural Description include identifying the Scope and Context of the system; identifying the Stakeholders; Cataloguing their Concerns and using this information as *inputs* to produce the various Architectural Views. This section focuses on the inputs of the Architecture, while the various Views will be presented in the next section.

3.2.1 Scope and Context

To identify the Scope and Context of the VERNE Platform, we considered the VERNE Ecosystem and its goals. The Platform is the part of the Ecosystem that connects to existing infrastructure in the Pilots and provides the foundation for the deployment of VERNE-related digital services.

More specifically, The VERNE Ecosystem is comprised of the following:

1. **The Pilots**, each with their own:
    a. administrative authority, which is their liaison to VERNE
    b. network of associates, in a specific region and industry of interest
    c. set of industry-related metrics that can be measured, either by physical measurement or by statistical calculations.
    d. set of Key Performance Indicators which are performance metrics of interest that require calculation.

2. **The VERNE Platform**, which is envisioned as a central data hub that enables VERNE related services.

3. **The VERNE Services**, which are digital systems connected to the Platform, each implementing specific VERNE-related functionality, to be made available to the Pilots. Some of the Services include:
    a. **The VERNE Marketplace** - a digital marketplace that can be used by VERNE Pilots
    b. **The VERNE One Stop Shop** - a web-based central Q&A library that can inform web visitors about the project.
4. **The communication (network) infrastructure**

Figure 3 below provides a visualization of this model of the VERNE Ecosystem: the VERNE Platform receives measurements from the Pilots and provides the necessary infrastructure to implement VERNE-related services, including the VERNE Marketplace, the One-Stop Shop, the calculation and dissemination of the Pilot KPIs, as well as Pilot-specific services.

Based on the description above, one can deduce the minimal functionality that the Platform will need to provide: it will handle the input and storage of Pilot-related data, and it will facilitate the calculation of per-Pilot Services and KPI calculations. It will also provide an Application Programming Interface (API) to support the VERNE Marketplace and the VERNE One-Stop Shop. It is also implied that the Administrative Authorities of each Pilot will be given the necessary tools and privileges to configure and control their respective Pilot's data in the Platform. As such, the platform will have multiple tiers of users, each with their own privileges to view and/or edit content.

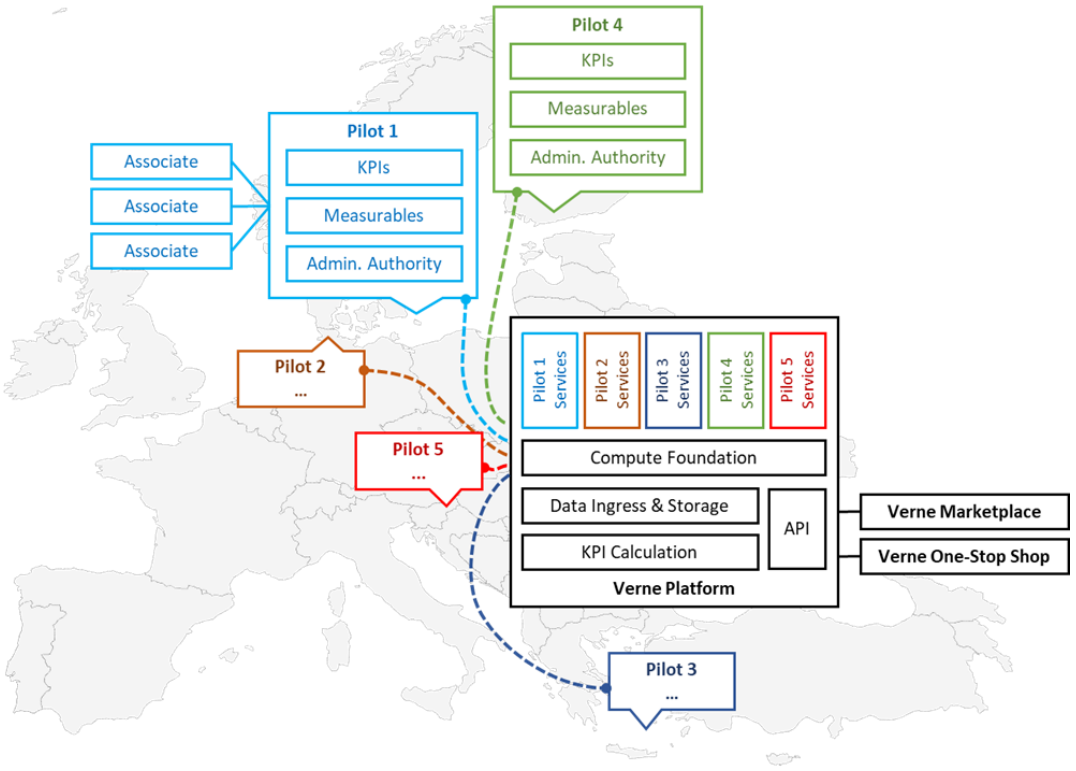


Figure 3. The VERNE Ecosystem

3.2.2 Platform clients

Per the ISO/IEC/IEEE 42010 Standard, when identifying stakeholders, the following are considered and included when applicable: users; operators; acquirers; owners; suppliers; developers; builders and maintainers of the system – any person interacting with the system at any point of the system's lifecycle.

Identifying a system's Stakeholders is important, because it informs both the Concerns (i.e. the Goals and Requirements) for that system and it allows to visualize possible use cases and modes of operation for each group of Interest.

Table 3 below summarizes the key Stakeholders involved in the use of the Platform.

Stakeholder	Role
Pilot Leader	Specifies KPIs and measurables for a specific Pilot with respect to the VERNE Project during the design phase.
Pilot Engineer	Specifies sensors, protocols and metrics used to obtain measurements – manages installation where required.
Pilot Administrator	Manages the configuration of the Pilot in the Platform.
Pilot Data Operator	Inputs data to the platform with respect to a specific Pilot.
Pilot Service Developer	Develops Pilot-related Services that are Platform clients, attached to the platform.
Third-Party (API) Developer	Uses a publicly available API to interface with the Platform

Stakeholder	Role
Consumer / Data Analyst	Collects and Views anonymized aggregate public data relevant to the Platform. Interacts with all publicly available parts of the platform and the One-Stop-Shop
Platform Developer	Develops and maintains Platform features
Platform Administrator	Configures & maintains all aspects of the Platform
Platform Operator	Configures & maintains parts of the Platform

Table 3. List of Stakeholders

3.2.3 Concerns & Requirements

Per the ISO/IEC/IEEE 42010 Standard, the following can be considered and included as **Stakeholder Concerns** when applicable: the purposes of the system; the suitability of the architecture for achieving the system's purposes; the feasibility of constructing and deploying the system; the potential risks and impacts of the system to its stakeholders throughout its life cycle; maintainability and evolvability of the system.

The term "Concerns" is an umbrella term for any requirement, aspiration or goal of the Stakeholders that drives system design. In the State of The Art, there is often a distinction between Functional and Non-Functional requirements: Functional Requirements specify a system's behaviour (i.e. What the system does), while Non-Functional Requirements specify broader quality traits and constraints (i.e. How the system behaves). This dichotomy has lately been criticized [6] [7] [8], as it can hinder prioritization during a system's phase and it has been observed that Non-Functional Requirements can often be re-interpreted as more specific, more actionable Functional Requirements. For example, the common Non-Functional Requirement "the system should be secure" can be translated into multiple explicit Functional Requirements for modules and security processes (Authorization, Authentication, Data Encryption, Audit Systems etc), which can produce a more testable and actionable architecture.

The following Table 4 presents all Stakeholder Concerns in a uniform list:

ID	Type	Description
C001	FR	The Platform's objective is to calculate Performance Metrics for pre-defined domains. Such Performance Metrics are named KPIs - Key Performance Indicators.
C002	FR	The Platform shall present the KPIs to users with view authority. Presentation includes current value, past values, as well as the historic record of any input variables used to calculate the KPI.
C003	FR	The Platform shall provide data-entry mechanisms for the submission of input data relevant to the calculation of KPIs, by users with data-entry authority. The Platform shall implement a form-based Graphical User Interface and a CSV-file submission feature to allow for data entry.

ID	Type	Description
C004	FR	The Platform shall provide programmatic data-submission (push) mechanisms, for the submission of input data relevant to the calculation of KPIs, by web services with data-entry authority.
C005	FR	The Platform shall provide programmatic data-retrieval (pull) mechanisms, for the retrieval of input data relevant to the calculation of KPIs, from appropriate web services.
C006	FR	The Platform shall allow properly authorised users or developers to define KPIs for that domain, specifying the KPI's name, the input data necessary for the calculation of said KPI and the mode of obtaining said input data.
C007	FR	The Platform shall allow properly authorised users or developers to define and schedule KPI-calculation work-flows for that domain.
C008	FR	The Platform shall allow platform developers to provide bespoke programs (domain-modules) for the calculation of KPIs.
C009	FR	The Platform shall allow platform developers to provide bespoke programs (retrieval-modules) for the retrieval of the input data necessary for the calculation of KPIs.
C010	FR	The Platform shall implement programmatic data access end-points, in order to facilitate communication between domain-modules, retrieval-modules and third-parties.
C011	FR	The Platform shall keep historic record of any data entered via the various data entry and data retrieval methods.
C012	FR	The Platform shall keep historic record of any calculated KPIs and any output data generated and marked "for-storage" by all domain-modules.
C013	FR	The Platform shall make available to domain-modules all necessary historic records for their respective domain.
C014	FR	The Platform shall allow users with domain-administrative authority to specify a domain-specific presentation page.
C015	FR	The Platform shall allow users with domain-administrative authority to specify multiple domain-specific Points of Interest (POIs), together with relevant information, including optionally a POI tag-name, geolocation, a number of associated data streams and a short presentation text.
C016	FR	The Platform shall allow users with view-authority to view a domain's POIs on a map, together with current values of associated data streams.
C017	FR	The Platform will allow users with global-administration-authority to view all data streams and all calculation work-flows, as well as the underlying modules for the purposes of management and fault identification.
C018	FR	The Platform will implement monitoring and inspection mechanisms to assist administrators and developers in ensuring the proper operation of the Platform and its modules.

ID	Type	Description
C019	FR	The Platform will implement and enforce authorization mechanisms on the user-level and the client software-level to ensure that data and functionality will be made available only to users and software services with explicit authorisation for that data and functionality.
C020	NFR	The Platform shall implement graceful failure mechanisms, allowing operations to be resumed and retried from a failed state.
C021	NFR	The Platform shall implement idempotency mechanisms, allowing operations to be safely retried without unintended side effects.
C022	NFR	The Platform shall implement redundant storage mechanisms to increase the reliability of its data stores.
C023	NFR	The Platform will implement horizontal scalability mechanisms, allowing platform developers to distribute computational loads between multiple execution engines, in order to respond to increased computational requirements.
C024	NFR	The Platform may implement throttling mechanisms to regulate the volume of user requests in order to manage computational resources.

Table 4. List of Design Concerns

In simple terms, the table above depicts in generic terms the entire functionality that the platform will need to implement based on the initial discussions with the various Project Stakeholders. Namely:

- The Pilots require a platform to calculate and display various Key Performance Indicators (C001) as well as the source data used for their calculation when possible (C002). Those source data can be obtained either through manual data entry (C003), automated, API-based data entry (C004) or by the Platform querying other platform APIs (C005). We anticipate specific modes of data ingestion, as discussed in Table 1: data can be ingested through streams of sensor data, API calls, files or by form-based manual entry.

- Pilot administrators or developers will be able to define KPIs for their respective Pilots, provide data sources and calculation formulas (C006), as well as work-flows, which are scenarios to trigger a calculation (C007). Such calculations happen in bespoke modules that run within the Platform (C008 – domain-modules) and may use customized data retrieval

mechanisms (C009 – retrieval-modules). The "domain-modules" in this nomenclature provide the processing apparatus for calculating the KPIs, while the "retrieval-modules" extend the platform, allowing it to interface with third party APIs when necessary. In the following, we will be using the term **"Agent Modules"** to refer to groups of modules that provide domain-specific computational and data retrieval functionalities, i.e. an Agent is nothing more than a set of computational modules and retrieval modules configured to work together, to access specific data and act on those data. All such extensions will have gated access to the Platform via an API (C010).

- As is often the case, the Platform will have historic record for data entered (C011) and for data calculated (C012). It is possible that the calculation of a KPI is dependent on the calculation of another KPI – as such the chain (workflow) of calculating all KPIs may require previous calculations (C013).

- In addition to KPI calculation, a set of common presentation features will be made available to all Pilots: Administrators will be able to provide general information about their Pilot (C014) as well as about Points of Interest (POIs) within the Pilot (C015) – For example, the Santorini pilot can have a hotel, a water reservoir and a plantation as separate Points of Interest – moreover, specific data sources can be associated to POIs for ease of use.

- Services such as the VERNE Marketplace and the VERNE One-Stop-Shop can be connected to the platform through modules, allowing information exchange via API when necessary.

- Some users will have full administrative privileges on the Platform to ensure it is well maintained and to identify any operational issues (C017). They will be utilizing Platform-wide monitoring and inspection services (C018). Authorization mechanisms will be in place, both for interactive users and for software-based clients (C019).

- The platform will implement State of The Art reliability techniques to provide increased reliability and fault tolerance (C020, C021, C022), and address escalating computational requirements through both a horizontally scalable design as well as a throttling system design (C023, C024)

As is customary in consortium-led projects, the list of Concerns may change as features are implemented and Stakeholder requirements/Concerns materialize with the project maturing.

3.2.4 Agents

It would be beneficial to address the concept of an Agent, as there will be reference to Agents in the following chapters. For the purposes of the Platform, an Agent is just a well-defined bundle of computational modules, grouped together in a meaningful way. An Agent includes:

- A set of computational workflows which run on a schedule, or when triggered, to perform a series of calculations.
- A set of data access methods which can be utilized by the External Data Aggregator to fetch any necessary data for the computational workloads. This functionality is for accessing external, third-party APIs – sensor data and file data are aggregated automatically.
- A set of configuration parameters that describe the necessary data.
- A set of User Interface components that help present Agent-related data to the user

Agents encapsulate modules that perform Pilot relevant calculations, per the Design Concern 008 (C008 – domain-modules) and relevant data retrieval mechanisms (C009 – retrieval-modules). Because all such calculations require appropriate data, it makes sense to bundle and configure the computational and the data-retrieval parts together, as they synergize towards a common goal.

Agents are implemented as docker containers presenting a thin API to the platform for control and data exchange. They have 6 distinct states in their lifecycle:

1. **Disabled**: the Agent container is not running
2. **Spin-up**: the Agent container is launching and initialization processes are running
3. **Registration**: the Agent connects with the Platform, registering functionality and exchanging configuration data
4. **Ready**: the Agent has been properly set up and is ready to receive signals
5. **Inference**: the Agent processes some data, per the Platform's request
6. **Deregistration**: the Agent is about to shut down and enter the disabled state – all Agent-related functionality is removed from the Platform

Note that while there is an "inference" state, there is no "training" process. The Platform expects Machine Learning based Agents to be pretrained. A semi-dynamic model can be implemented by uploading an updated ML model as part of the Agent configuration data.

3.3 Architecture Views

In this chapter we present the various Architectural Views describing the Platform, describing key components and their function in the Architecture. We also touch upon the process of deriving the Architecture based on the description of the project and the various requirements, as specified in the previous chapter.

3.3.1 Platform Boundary

Figure 4 provides an overview of the VERNE ecosystem and the placement of the VERNE Platform as a central data hub. As the Platform is designed for the general case, it is designed to enable functionality across all VERNE Pilots.

Notably, some Pilots require use of sensor streams and can benefit from the definition of Points of Interest, while others may rely on file-based or form-based data entry. This is a matter of configuration for each Pilot. In the following we assume that each pilot can have multiple Points of Interest, each associated with multiple sensor streams.

A simplified view of the Platform is that its goal is to aggregate data from various data sources, provide it to Agents to perform calculations and to present their calculations to their users.

3.3.2 Key required components

Given the requirements and ecosystem description, it is now possible to describe the key components of a VERNE Platform, based on the minimal feature-set to be implemented.
The various components are illustrated in Figure 3 below, while their functionality is discussed later in this chapter.

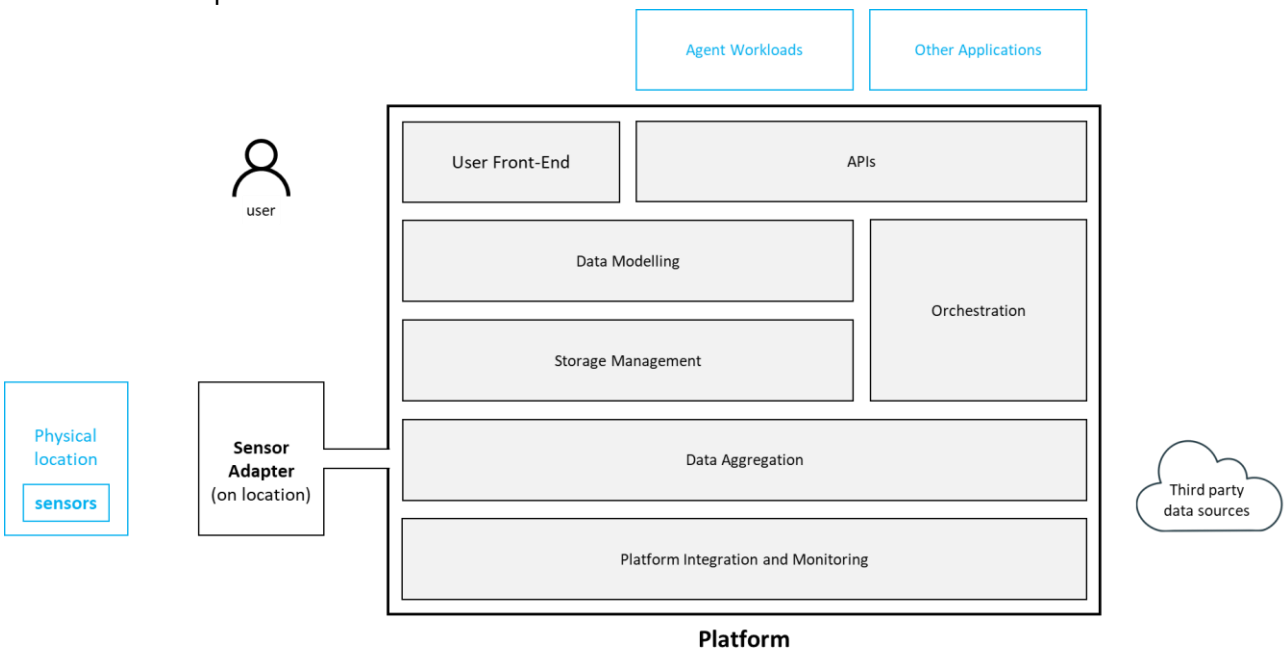


Figure 4. Key Required Components

As the platform will be required to ingest data from various data sources, a **Data Aggregation** layer is provisioned. This is a system that interfaces with the various data sources and manages data ingress and transformation (in a fashion similar to the ELT layer of the Data Lake model). The Data Aggregation Layer will be required to support multiple data transmission formats and modes of operation, from push-based methods like MQTT (popular with sensors and IoT) to scheduled pull-based and polling methods for third party data, to CSV-file uploads. Note that a **Sensor Adapter** is a system that may be used as a data relay station, on the field, near a group of sensors, to facilitate data transmission. Its use is optional, but it is included in the Figure for completeness.

As data are ingested, the Platform will need to store them in properly designed systems. A **Storage Management** layer should store data in a secure and high availability system that also allows quick access and enables search and filtering functionality when feasible. As most of the data are timecoded, this system may utilize Time-Series Databases to store historic data. Redundant storage, horizontal scaling, caching, anonymization and other tasks are managed by this layer.

A **User Front-end** layer models the user and provides a web-based user interface, providing browsing and search functionality as well as the configuration and management toolset. It also manages user authentication.

As the Platform will be executing Pilot-specific code (C008 – the domain-modules and C009 – the retrieval-modules), an **Application Programming Interface** layer will be implemented: this will provide gated access to computational workloads and to the various data interface algorithms, ensuring that each domain-specific workload will have access to its domain-specific data. In reality, the API layer is a set of APIs, designed to enable different types of functionalities:

- The **Agent** API, which focuses on enabling **Agent Modules.** Agents are sets of domain-modules and retrieval-modules, which synergize to provide domain-specific data access and computational functionality. Because Agents may require fine-grain, privileged access to Pilot-specific configurations and data, as well as a user interface for form-based input, a special Agent API will be provisioned.
- The **Platform** API, which enables integration with third party systems.
- A **User Interface** API is also used internally and provides common Platform functionality to the user interface.

To provide actionable functionality on the data, a **Data Modelling** layer may interact with the data to translate raw values into meaningful data types, usable by the various components of the Platform. The Data Modelling layer essentially allows for schema-like validation of data units as well as a set of minimal but useful data transforms to be used by agents on the upper levels.

Requirement C007 mentions the idea of work-flows in the Platform. Work-flows are schedules and processing pipelines that can be triggered on specific events (for example, when new input data become available, or when a specific value crosses a certain threshold) or at regular intervals (for example, monthly calculations). Because there may be interdependencies between KPIs (one KPI may use the output of another as its input), it is important that the evaluation of the various KPIs happens in a well-defined process, to eliminate errors. This process is essentially the work-flow, and the Platform that manages such workflows, as well as the overall execution of platform components is the **Orchestration** Layer**.**

Finally, a set of foundational tools in the **Platform Integration and Monitoring** layer manage and monitor the execution of the platform in a scalable environment, implementing the

principle of containerization: all Platform components are developed as containers, allowing for easier monitoring, hardware independent execution, fault tolerance, scalability, elasticity and redundancy when required.

3.3.3 Logical View

The primary objective of the Logical View is to illustrate the functionality that the Platform provides to its users at a logical level. This functionality is closely related to the components and modules that make up the architecture.

The Logical View presents a more detailed picture of the Platform components described in Figure 3, expanding on key modules and describing their interactions and interdependencies. As the components have been presented in the previous chapter, here the focus shifts on the underlying logical modules that implement each component's functionality.

The Logical View is presented in Figure 5 below:

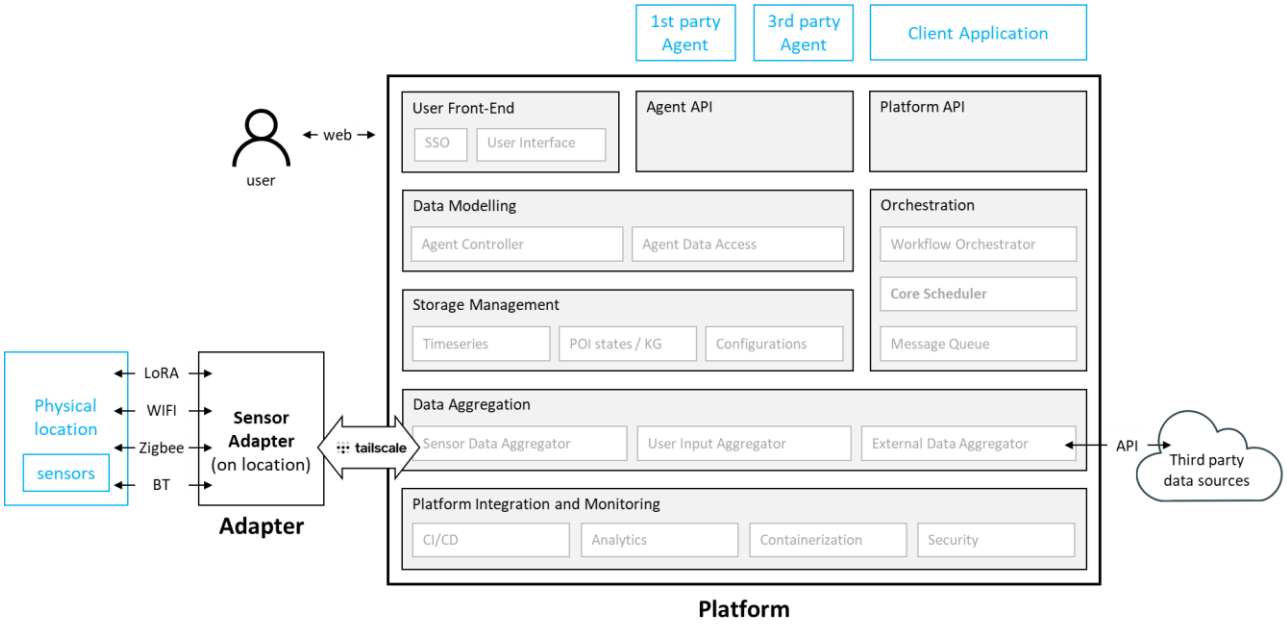


Figure 5. Logical View

The Platform Components are as follows:

1. **Platform Integration & Monitoring:** This component manages the deployment, execution and monitoring of Platform Software on a properly configured machine or set of machines. It utilizes containerization principles to deploy source code at scale and integrate changes to that code when required. This is the software foundation level, divided in 4 key modules:
   a. **Containerization:** The Platform is developed as a set of containers, executed within a set of Virtual Machines in properly configured hardware (a computer or cluster of computers). This is to ensure scalability and redundancy. The Containerization module

is the execution and instrumentation environment that runs those containers on configured systems. In the state-of-the-art this environment is often based in Docker, Kubernetes or both.

b. **Security:** A set of modules for user authentication, Single-Sign-On, Virtual Private Networking and similar technologies

c. **CI/CD:** This is shorthand for "Continuous Integration / Continuous Deployment" and is a module that continuously monitors a Code Repository for code updates. Then, when required, it tests that update and automatically integrates it to Production. In reality, this code integration process is not fully automated for security and code quality reasons, but the CI/CD process helps automate deployments.

d. **Analytics:** A set of analytics tools like Grafana and Prometheus monitor the hardware and the containers during both development and production, allowing developers and administrators to recognize issues, software errors or performance bottlenecks and react accordingly. The same analytics services can be utilized in higher architectural levels to monitor Agent execution, identify potential execution errors.

2. **Data Aggregation:** The data aggregation layer implements all data aggregation functionality in an extensible manner. It allows developers to create API endpoints or access API endpoints, schedule polling services and ingest file-based and form-based data. A Key feature of the Data Aggregation layer is that it can be extensible allowing agent developers to create bespoke functionality to access a specific third-party API and gather data required for KPI calculations. Its modules are:

   a. **Sensor Data Aggregator.** The SDA module addresses real-time or near-real-time sensor data streams, through known IoT protocols such as MQTT. It provides an MQTT broker and can be programmed to accept streams of sensor or IoT data. Because of hardware constraints we do not anticipate high sampling rates – as mentioned in Table 1, sampling periods smaller than 15seconds are not to be expected in the context of this project.

   b. **User Input Aggregator.** The UIA module handles user-based inputs, including file-based and UI (form) based input. In addition to storing the user-input data, it provides minimal validation, anonymization and error checking services and also adds relevant metadata to the stored file (i.e. submitting user, time of submission etc)

   c. **External Data Aggregator.** The EDA module handles access to third-party APIs. It can be programmed to fetch specific data sets from specific API end-points and store their replies for future use. Due to security and containerization policy constraints, Agents do not have direct access to third party web domains and have to request data-sets from the Platform. Because such requests can be scheduled, the Platform can prefetch those data-sets and have them ready for calculation. This process happens through the EDA. A common example of this is the access of weather forecasts for a region of interest. When an agent requires weather data for some calculation, an EDA request is scheduled and EDA prefetches the data. It can also ensure that multiple retries take

place in case of a third-party API error. The data will be ready and waiting on the Platform before any calculation takes place. This bolsters security, optimizes performance and minimizes errors due to transient network issues.

3. **Storage Management.** This component provides storage services to its client layers. All ingested data, once validated, are stored in an appropriate module of the Storage Management layer, which also handles storage redundancy. Data are stored in 3 different types of data stores.
   a. A **Time-Series database** contains all timestamped data for use by the various Agents. In addition to the reference time, the Domain (i.e. the corresponding Pilot) and the associated Point of Interest are also included in the record.
   b. An Object Storage service stores **POI-State** dynamic data and other information related to the "current state" of all Domains (Pilots). This is used across the platform as an object cache for dynamic data, mainly for use with the Orchestration layer.
   c. An Object Storage service stores **Configuration data**, i.e. semi-static data related to the configuration of the Domains (Pilots), their Points of Interest, their presence in the Platform and their associated work-flows. Those data are not expected to change often and changes to them may trigger multiple computation or reconfiguration events across the platform. As such, the Configuration data store has different characteristics and performance requirements than the POI-State data store.
4. **Orchestration.** This layer manages the execution of Domain (Pilot) related workflows and their respective Agents. A work-flow may require that certain actions be taken on schedule or when triggered by an external factor (for example, when new data become available or when the state of a variable has changed). In addition, a work-flow may specify that a KPI needs to be calculated first, as it will be used as input to the calculation of another KPI. The Orchestration layer manages such requirements, orchestrating the execution of the various platform Components to the successful fulfilment of a work-flow. It is comprised of
   a. A **Message Queue** – this is a component that enables asynchronous communication between different modules of the Platform. It decouples components by storing messages (data / tasks) sent by a producer module and allowing consumer modules to process them later. This simplifies development, ensures reliability and reduces computational spikes. Popular Message Queues include RabbitMQ and Apache Kafka
   b. The **Workflow Orchestrator** - this is the logic component that analyses the Platform configuration data and accordingly configures the various scheduling and event-based components of the Platform, orchestrating their execution based on the configuration data, to achieve scheduled or event-based KPI calculation.
   c. The **Core Scheduler** – This is a programmable scheduler that manages scheduled tasks and can manage event-based execution of processes. It can be thought of as a component similar to *cron* but scaled for a containerized environment and making use of the Message Queue services to communicate with all the other Platform subsystems.

5. The **Data Modelling** layer. This layer provides gated access to the Stored data, performing data transformations when required (in a model similar to a Data Lake's ELT), for use within the Agents (and User Front End). Instead of accessing the Storage layer directly, all higher-level client systems will be required to use an API, which is essentially the endpoint of the Data Modelling layer. This layer is comprised of two components.
    a. **Agent Controller**: This is the part of the layer that manages layer execution, providing endpoints for configuring and executing the Agents. It manages Agent initialization, configuration and monitors their execution. It also registers Agent-specific User Interface components, to be used with the User Front End when required.
    b. **Agent Data Access**: This is the part of the layer that manages access to the data storage, validating requests and caching the results when appropriate.
6. **APIs.** A set of API endpoints is made available through the Data Modelling layer:
    a. the **Agent API** allows for the development of Agent services, i.e. of systems that can be used for the computation of KPIs and for the presentation of those KPIs in the user interface. Since the Agents can have privileged access to Platform data, all Agents, including third-party developed ones, are deployed as containers within the Platform. The Agent API is thus not a public API.
    b. The **Platform API** is a generic API for use by Third-Party Applications. While the Agents can have privileged access to data, the Platform API is envisioned as a public API for statistics and KPI results, with a minimally exposed surface.
7. **User front-end**. This is the part of the Platform that renders a web-based user interface and allows user sign-on. Its task is to allow for authenticated users to input data, configure the Platform, interface with the Pilot Agents and generally interact with the platform. It has two key components:
    a. **User Interface.** This is the web-based user interface renderer.
    b. **SSO – Single Sign On.** This is the part of the platform that manages the user credentials and the security tokens related to their browsing session, implementing state of the art Authentication strategies. It can act as an Agent to the system, requesting access to data and altering the state of components in the Data Modelling layer, which in turn causes ripple changes in the Storage, Orchestration and Data Aggregation components of the Platform.

3.3.4 Deployment View

It is worth noting that the entire software stack of the Platform will be developed and deployed using container technologies. This is not only advisable considering the State of the Art [9] but also a necessary provision to fulfil scalability and extensibility requirements.

Container technologies allow a hardware-platform-agnostic approach to design and development, allowing the dynamic provisioning of hardware when required: In principle,

services run in a Kubernetes (or similar) hardware cluster, and computing nodes can be assigned dynamically when required.

In addition to providing scalability benefits, this decoupling between hardware and software addresses development considerations as well: parts of the system can be written in whatever programming language is most convenient and containerized – the underlying technologies address most computing environment compatibility issues, allowing seamless execution, even if the hardware or operating system changes. For example, core components where performance is critical can be developed in languages such as Go, C++ or even node.js, while Machine Learning components can use Python if required. Each component runs in their own, appropriately configured execution context, within their container.

Another benefit of containers is that off-the-shelf technologies such as Apache Kafka, PostgreSQL or TailscaleVPN can be easily integrated via their own, vendor-supplied containers, minimizing human involvement in the installation process, which is key when considering adaptability and technology updates requirements within a product's lifecycle.

Figure 6 below provides an overview of the proposed technologies utilized within each component.

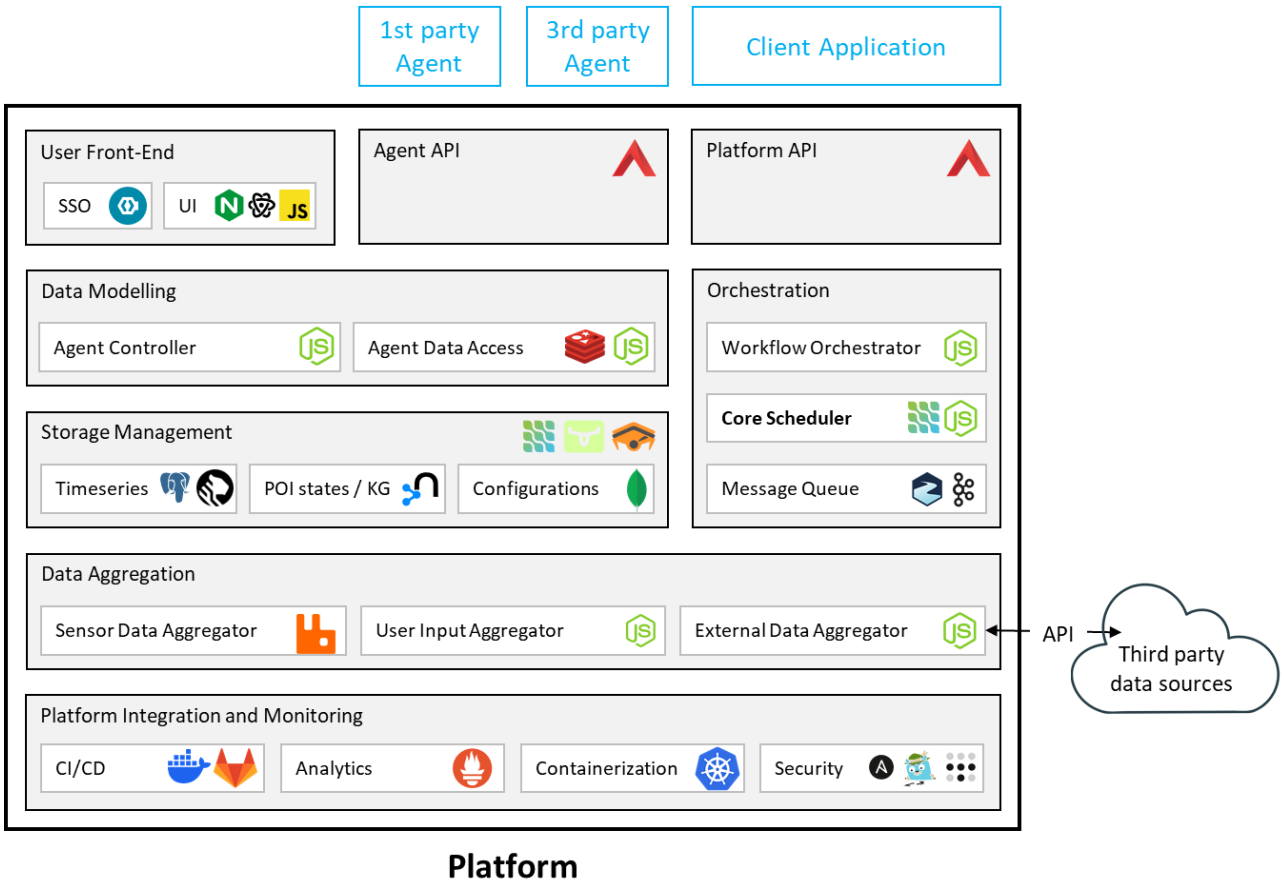


Figure 6. Deployment View

Each module in the Logical View corresponds to a set of containers that are executed within a containerization framework, namely a Kubernetes Cluster. Because the Hardware is abstracted, the View focuses on the software and the technologies used within each module.

The function of each component has been discussed in section 3.3.3. The Deployment View proposes a set of technologies for the implementation of each module, as signified by the respective icon. Those technologies, together with possible alternatives and the rationale behind their use are discussed in the following paragraphs.

3.3.4.1 Platform Integration – Containerization

Starting from the foundational components in the Platform and Integration Monitoring, the industry standard technologies of Docker and Kubernetes are used.

Docker is the industry standard solution for the creation, distribution and deployment of containers. A docker container is a virtual machine with its own operating system, virtual networking infrastructure and execution environment. It allows developers to create their own containers, either by creating them from scratch or by extending existing off-the-shelf containers. Those containers can then be published to container repositories for integration into a CI/CD pipeline, in a process similar to how Git pushes changes to code repositories. Docker is built on an open foundation, leveraging components such as runc, an implementation of the Open Container Initiative (OCI) specifications and a high-level container runtime. This integration provides a robust framework for encapsulating applications and ensuring consistent deployment. In addition, Docker provides developer-friendly features such as a CLI for container lifecycle management and access to a public container registry. Higher level tools such as Kubernetes and Docker Compose simplify and automate deployments, addressing a vast spectrum of deployment needs, from a developer machine to a network-of-clusters.

Kubernetes is the industry standard solution for the management and horizontal scaling of container-based workloads in a distributed manner. It allows the deployment of container applications in self-adaptive computer networks, or clusters, automating deployment, scaling and operational tasks while abstracting from the hardware resources. It is not a docker-replacement, but rather a production-proven orchestration system for effectively managing multiple container instances in a multi-node cluster. Using a declarative configuration approach, it seamlessly integrates Docker containers as its primary packaging format. Kubernetes uses various service components, including Kubernetes Service, kube-proxy, and Ingress controller, to distribute loads efficiently across services in a cluster. Kubernetes Service offers a dependable endpoint for load balancing, kube-proxy maintains network rules for routing traffic to container instances (pods), and the Ingress controller manages external access and routing strategies.

3.3.5 Platform Integration – CI/CD

For **Continuous Integration / Continuous Deployment** tasks, **GitLab** will be used in conjunction with Docker. GitLab is a platform similar to Github and Bitbucket, with the difference that it allows self-hosting: Developers can host their own instance of Gitlab on their own server. The free Community Edition is open source with the permissive MIT License. Because GitLab allows secure and private workflows that do not require uploading source code to a third-party cloud, it is used by security-sensitive teams [10], including NASA [11], CERN, Thales, Northrop Grumman and Lockheed Martin.

In addition to offering a git-based source control repository, GitLab also offers Docker container registry functionality. This allows developers to submit entire containers that can then be deployed as-is in the production server, with their own preset execution environment. This solution is beneficial when required to execute third-party-managed workloads, especially Python-based ones, as Python code requires careful set up of its execution environment.

Table 5 discusses GitLab alternatives for CI/CD:

Tool	Description
GitHub	GitHub is a leading code hosting platform with powerful collaboration features and built-in CI/CD through GitHub Actions, making it the best choice for version control and automated workflows. The downside of GitHub is that it cannot be self-hosted. URL: github.com
Bitbucket	Bitbucket by Atlassian supports Git and Mercurial repositories. It offers robust code collaboration features and integrated CI/CD with Bitbucket Pipelines, providing a complete solution for development teams. Bitbucket offers a commercial self-hosted version, Bitbucket Server. URL: bitbucket.org
Jenkins	Jenkins is a popular open-source automation server. While not a repository, it integrates seamlessly with Git and other VCSs to provide rich CI/CD capabilities for automating software development processes. Jenkins can be easily self-hosted inside an existing infrastructure. URL: jenkins.io

Tool	Description
Travis CI	Travis CI is a CI/CD platform built for GitHub repositories. Configured through a simple YAML file, it automates builds and tests, supporting seamless integration with a range of programming languages. Travis CI is commercially available. URL: travis-ci.com

Table 5. Alternative CI/CD solutions

3.3.5.1 Platform Integration – Analytics

For **Analytics** and supervision tasks, Prometheus, Loki and Grafana will be used.

Grafana is an open-source analytics and monitoring platform that excels in visualizing time-series data. It integrates with various data sources, including popular databases, cloud services, and monitoring systems, enabling users to create interactive and customizable dashboards. Grafana's rich set of features includes support for different chart types, alerting, and the ability to share dashboards, making it a widely adopted tool for monitoring and observability. Grafana alternatives are presented in Table 6 below.

Tool	Description
Kibana	Kibana is an open-source data visualization platform primarily used with the Elasticsearch stack. It provides powerful analytics and visualization capabilities. URL: www.elastic.co/kibana
Chronograf	Chronograf is part of the TICK stack (Telegraf, InfluxDB, Chronograf, Kapacitor) and is designed to visualize and explore time series data. URL: www.influxdata.com/time-series-platform/chronograf
Metabase	Metabase is an open-source business intelligence tool that allows users to create charts and dashboards and provides an easy-to-use interface for exploring data. URL: www.metabase.com
Apache Superset	Apache Superset is an open-source business intelligence web application that supports data exploration, visualization, and interactive dashboards. URL: superset.apache.org

Table 6. Alternatives to Grafana

Prometheus is a cloud-native, open-source monitoring and alerting solution. Originally built by SoundCloud, it is maintained independently and a member of the Cloud Native Computing Foundation. Prometheus offers a self-hosted platform for collecting, storing and processing multi-dimensional time series, identified by metric name and key/value pairs. In simple terms, it is a very efficient, easy to configure system that intercepts and stores numerical

measurements in a timeline. It offers a query language, PromQL and it does not rely on distributed storage; single server nodes are autonomous. It excels as a reliable, high availability monitoring solution but, as it aggregates data, it does not offer 100% accuracy. Alternatives are presented in Table 7 below:

Tool	Description
VictoriaMetrics	VictoriaMetrics is a fast, cost-effective, and scalable solution for monitoring and managing time series data. It delivers high performance and reliability, making it an ideal choice for businesses of all sizes. It has been used in Wix, Roblox and Spotify. It offers long-term storage and can even be used as a drop-in replacement for Prometheus. URL: victoriametrics.com
OpenTelemetry	is an open-source observability framework for cloud native software. It provides a single set of APIs, libraries, agents, and collector services to capture distributed traces and metrics from your application. While Prometheus offers a pull (scraping) data collection strategy, OpenTelemetry offers both a pull and a push strategy. Because OpenTelemetry is not a direct alternative to Prometheus, as it does not offer storage and query functions, but it can be used in conjunction with Grafana Mimir or even Prometheus to provide a richer feature set. URL: opentelemetry.io
Mimir	Mimir is a long-term storage solution by Grafana Labs. It integrates with Grafana and can be used in conjunction with OpenTelemetry or Prometheus. URL: grafana.com/oss/mimir
Datadog	Datadog is a SaaS-based observability and security platform for cloud-scale applications, providing monitoring of servers, databases, tools, and services. It can visualize infrastructure, monitor application performance, manage logs, and detect threats in real-time, all through a unified, integrated dashboard with multiple integrations. It offers a very rich, highly scalable toolkit, but is SaaS based. URL: www.datadoghq.com

Table 7. Alternatives to Prometheus

Loki is an open-source, horizontally scalable log aggregation system, part of the Cloud Native Computing Foundation (CNCF) landscape. It is designed to handle and store log data generated

by various applications and services in a distributed and efficient manner. It is maintained by Grafana Labs and therefore it is easy to integrate with Grafana. Loki offers a service similar to Prometheus, but for server logs. Loki alternatives are presented in Table 8 below:

Tool	Description
Fluentd	Fluentd is an open-source data collector that efficiently aggregates logs from multiple sources, parses them, and forwards them to different destinations using a unified logging approach. Its lightweight architecture and extensive plugin support enhance flexibility in log processing workflows. Fluentd is a graduated project of the Cloud Native Computing Foundation. URL: www.fluentd.org
Logstash	Logstash is a tool part of the ELK stack. Hence, it is an Elastic product for log processing. It allows you to collect, parse, and transform logs before passing them to Elasticsearch. It supports a wide variety of input sources and output destinations. URL: www.elastic.co/logstash
Apache Flume	Apache Flume is a distributed and reliable service designed for collecting, aggregating, and transporting large amounts of log data. It is commonly used in conjunction with Apache Hadoop, making it suitable for big data environments. It is written in Java, and the latest version, 1.11.0, was released on October 10th, 2022. URL: flume.apache.org

Table 8. Alternatives to Loki

Both Loki and Prometheus can be used as Grafana Data sources, this enables a foundational monitoring service that can display both performance analytics and respective warning and error logs in time-based graphs, allowing developers to easily spot performance issues, even in the event of a service failure at the higher-levels. This trio can further be complemented by **Jaeger's** tracing capabilities to further assist in development and maintenance operations.

3.3.5.2 Platform Integration – Security

Security is a horizontal consideration across all components of the Platform. However, a set of foundational tools are deployed to monitor functionality, trace errors, secure communications and provide automation features to administrators and developers.

Jaeger is an open-source distributed tracing system designed to monitor and troubleshoot microservices architectures. It collects, visualises and analyses trace data to help developers understand latency issues and pinpoint performance bottlenecks. Jaeger's comprehensive feature set includes support for root cause analysis, service dependency mapping, and performance monitoring, making it an essential tool for ensuring observability in complex systems. Alternatives are presented in Table 9 below:

Tool	Description
Zipkin	Zipkin is an open-source distributed tracing system that collects and analyses trace data to identify latency problems in microservice environments. URL: zipkin.io
Open Telemetry	OpenTelemetry is a vendor-neutral observability framework that offers APIs, libraries, and agents for capturing distributed traces and metrics across services. URL: opentelemetry.io
AWS X-Ray	AWS X-Ray is a distributed tracing service provided by Amazon Web Services that helps developers analyze and debug microservices-based applications in production. URL: aws.amazon.com/it/xray
Elastic APM	Elastic APM is an application performance monitoring solution within the Elastic Stack that delivers distributed tracing and real-time analytics for enhanced system observability. URL: elastic.co

Table 9. Alternatives to Jaeger

Ansible is a free, open-source IT automation engine used for software provisioning, configuration management, and application deployment. It operates agentlessly via SSH, utilizing YAML-based "playbooks" to automate tasks across IT infrastructure. Red Hat sponsors the project, offering both a free community version and an enterprise version. Ansible can be used to automate and issue commands across cluster nodes. Ansible alternatives are presented in Table 10 below.

Tool	Description
SaltStack	SaltStack is a high-speed, agent-based (or agentless) alternative often preferred for large-scale, complex infrastructure requiring rapid execution. URL: saltproject.io

Tool	Description
Puppet	Puppet is a mature, model-driven, agent-based tool that excels at enforcing desired state and handling complex dependencies, often favored for enterprise environments. URL: puppet.com
Terraform	Terraform is an infrastructure provisioning solution, operating across cloud providers using a declarative language. It is often used alongside Ansible rather than a direct replacement.URL: terraform.io

Table 10. Alternatives to Ansible

3.3.5.3 Data Aggregation - Sensor Data Aggregator

The Sensor Data Aggregator (SDA) module interfaces with sensors and other IoT devices in the field, capturing data streams through a public interface. Most, if not all, of the sensors provisioned in VERNE have some compatibility with the MQTT protocol. To act as a liaison between the sensors and the Platform, the SDA processes the Pilot configuration data, configures an MQTT compatible service and captures the incoming data using the Storage Management layer. The proposed MQTT solution is RabbitMQ, which is a popular self-hosted Message Queue service that supports the MQTT protocol natively. In addition to offering an MQTT implementation, RabbitMQ offers a reliable Message Queue service – it can thus be considered a viable alternative to Apache Kafka, albeit without message retention. RabbitMQ alternatives are considered in Table 11 below.

Tool	Description
Mosquitto	Mosquitto is an open-source MQTT broker that enables efficient communication especially for the Internet of Things. It supports the MQTT protocol, providing a publish-subscribe model for messaging. Mosquitto is simple, resource-efficient, and easy to integrate into IoT scenarios. URL: mosquitto.org
Zenoh	Zenoh is an IoT middleware that prioritizes efficient and decentralized communication. It supports various communication patterns, such as publish-subscribe and request-reply, and seamlessly integrates multiple protocols like MQTT, CoAP, and WebSocket. Zenoh's protocol-agnostic approach provides flexibility and scalability in dynamic and resource-constrained environments. URL: zenoh.io

Tool	Description
ActiveMQ	Apache ActiveMQ is a popular, open-source, Java-based message broker and message-oriented middleware (MOM) developed by the Apache Software Foundation. It acts as a bridge between distributed applications, facilitating reliable, asynchronous, and loosely coupled communication using standard protocols like JMS 1.1, AMQP, STOMP, and MQTT.URL: activemq.apache.org

Table 11. Alternatives to RabbitMQ

3.3.5.4 Data Aggregation - User Input Aggregator

The User Input Aggregator (UIA) module is an ingestion point for user-submitted data. It provides user-customizable input forms and drag-and-drop file upload services, based on the Pilot configuration. Once user data have been submitted, the UIA performs a rudimentary validation, stores the data with the capture timestamp and triggers the appropriate data-updated events, for Pilot workflows affected by the ingested data. The UIA is currently developed in node.js.

3.3.5.5 Data Aggregation - External Data Aggregator

The External Data Aggregator (EDA) module is responsible for fetching data from third-party data sources. It allows developers and administrators to specify specific API endpoints and queries that can be run, name them and schedule their execution. Once the data are fetched, the EDA stores both the request and the API response in a database. EDA is extensible, allowing developers to write worker scripts that perform API requests for additional APIs. It is also configurable, allowing administrators to configure those workers with parameters, for each named query schedule.

3.3.5.6 Orchestration - Workflow Orchestrator

The orchestration layer schedules and synchronizes the execution of computational workloads and work-flow chains, as defined by the Pilot Developers and Administrators. It utilizes Message Queues and a scheduler service to launch computational workflows and manage workflow chains. The Orchestrator service is event-based: it works by triggering events in the system and responding to events. It is also configurable: it uses a set of rules (the workflows) within each Pilot configuration as a guide to configure the event pipeline and to respond to events.

The Workflow Orchestrator primarily interacts with and configures the Core Scheduler, which implements the event triggering functionality.

To illustrate this interaction, consider this exemplary workflow: the Pilot Administrator wants to “trigger GasTank.alarm when the GasTank.pressureSensor exceeds 20psi”. The Workflow Orchestrator is the part of the system that parses the (JSON) configuration file and recognizes

the sensors, thresholds and results in the system. The Core Scheduler is the part of the system that listens for changes in the GasTank.pressureSensor and triggers the appropriate events related to the GasTank.alarm.

3.3.5.7 Orchestration - Core Scheduler

While the Workflow Orchestrator processes the Pilot Configuration data to schedule, trigger and respond to events, the Core Scheduler is the part of the orchestration layer that provides the event triggering functionality. It is a programmable system for triggering events, either through time schedules or by sensing changes to items of interest in the storage management layer.

To implement a seamless and extensible event triggering pipeline, the Scheduler integrates with a Change Data Capture solution, like **Debezium.** Debezium is an open-source distributed platform for change data capture. It monitors the Platform's databases and captures any changes as they happen. This can then be used to respond to those changes, i.e. by triggering events in the system. Alternatives to Debezium are presented in Table 12 below:

Tool	Description
Airbyte	Airbyte is an open-source ELT platform that is much easier to set up than Debezium, ideal for moving data into warehouses with moderate latency. URL: airbyte.com
PostgreSQL	The PostgreSQL database offers a logical replication feature which can simulate CDC. It can be used to setup a publisher / subscriber workflow, but it is mostly intended for database migrations. URL: postgresql.org
Conduit	Conduit is an open-source data streaming tool that uses "CDC-first" connectors to capture real-time database changes (inserts, updates, deletes) by reading transaction logs. It enables streaming data from sources like Postgres, MySQL, and MongoDB to destinations without full table scans, reducing load on source systems. Conduit is modern, lightweight and open-source, written in Go. It is a simpler alternative to Kafka Connect setups. URL: conduit.io

Table 12. Alternatives to Debezium

Most CDC solutions introduce a pub/sub pattern, where a publisher notifies the subscribers for any changes to a resource. This pattern naturally benefits from - it often requires - a message queue. Debezium, for example, relies on Kafka Connect, the industry standard solution for cloud-native distributed message queues with built-in persistence features. In addition to CDC,

a message queue is required because the orchestration layer will need to coordinate with multiple computational workloads through events.

3.3.5.8 Orchestration - Message Queue

In addition to being a required component for the Core Scheduler and its CDC (Change Data Capture) functionality, the Message Queue is a core building block for the entire platform. It is the communication backbone between systems, used for data exchange and synchronization. Furthermore, it is used to decouple chains of interconnected systems.

Conventional processing pipelines end up having long chains of processing nodes, where each node calls some function of the next node in the chain. This is suboptimal, because delays in one node can cause ripple delays down the pipeline. A message queue solves this issue by acting both as a buffer between nodes and as an execution pipeline. As soon as they finish processing, nodes store their results in the message queue, as messages to be forwarded to the next node in the pipeline. As soon as the next node in the chain becomes available, the Message Queue forwards the message, with its required actions and their inputs embedded within, so that the chain of computations can resume. Because now the modules are decoupled, bottlenecks are less likely to happen. In addition, if a certain process takes too much time, it is now easier to include multiple instances of the respective module, speeding up the pipeline overall. Scaling becomes a matter of identifying potential bottlenecks and including more nodes that can act in parallel.

Most large-scale projects rely on **Apache Kafka** as a message queue broker. Apache Kafka is an open-source, distributed event streaming platform that is renowned for its ability to handle real-time streaming data. It has a central broker that oversees the organization of data streams into individual records encapsulated in topics. This design ensures data persistence across multiple server instances, which increases reliability. Kafka's main components consist of producers, which generate and publish data, topics that categorize data streams, consumers that subscribe to and process these streams, and brokers that facilitate communication between producers and consumers. This framework is designed for speed, scalability, and efficient distributed operations, making it a robust solution for real-time data streaming applications.

Kafka was designed as a high-scalability, high reliability event broker, not a message broker. Events are normally very small messages, designed to communicate a single activity associated to a specific component. This distinction is important, because, in contrast to traditional message queues, Kafka features event persistency: all events are stored in a Log, even after they have been consumed (processed) by the intended recipients. This bolsters reliability and allows for advanced replication and auditing actions, but places constraints to the size of messages contained within the log. To address this, a caching system can be utilized in tandem with Kafka: large-size data can be stored separately as transient files while events point to those files.

Both **Kafka** and **RabbitMQ** (discussed in section 3.3.5.3) can function as message queue brokers, and alternative solutions to both are presented in Table 10 above. In addition to message brokerage, Kafka also provides a persistent storage and a large array of integrations. Thus, while RabbitMQ is used for its MQTT implementation, Kafka, in conjunction with **Debezium**, are preferred for the event brokerage and orchestration modules.

To facilitate Kafka's integration with Kubernetes, **Strimzi** can be used. Strimzi is an open-source CNCF project that simplifies running Apache Kafka clusters on Kubernetes or OpenShift. It provides specialized operators to automate the deployment, management, and scaling of Kafka brokers, topics, users, Kafka Connect, and MirrorMaker using native Kubernetes API resources. It uses a Cluster Operator for managing Kafka clusters, a Topic Operator for managing topics, and a User Operator for managing user credentials and access control. It extends Kubernetes with CRDs, allowing you to define and manage Kafka components using YAML files. It also supports TLS encryption, certificate management, and authentication for secure client connections. Most importantly, it provides flexible deployment options, including support for KRaft mode (replacing ZooKeeper) and persistent storage. This simplifies Kafka execution within the Kubernetes cluster, automating operations and reducing the need for manual operations.

3.3.5.9 Storage Management

The Storage Management layer provides a cluster-compatible storage layer to the Platform. In that storage layer, various specialized storage solutions are implemented and made available to their relevant clients. The layer also provides endpoints for Change Data Capture (CDC) schemes, which can be used to generate events and alerts. Solutions such as Apache Longhorn and Garage can provide cluster-wide block (file) storage and object storage.

3.3.5.10 Storage Management – Timeseries

It is anticipated that most of the data aggregated to the VERNE Platform will have a time aspect. This includes sensor data, third-party API data (i.e. weather, traffic, market, energy mixture etc) computed KPIs. It is also beneficial for most use cases to process and present data based on a time window. It would thus be optimal to store such data in a time-series database.

Several database solutions exist for time-series data, many of which are designed around analytics management and very high data rates. Considering the VERNE Project requirements, we propose an open source, self-hosted general-purpose database which can connect to Debezium for event generation, with data throughput being of lower priority. PostgreSQL is an enterprise-level database that fits that profile. It is open-source and reliable, supports both SQL and JSON, it can be extended to support both spatial queries (with postGIS) and time-series queries (with timescaleDB). It can also connect to Debezium via the Debezium PostgreSQL connector, implementing Change Data Capture for the Orchestration layer.

Some of the numerous alternatives to Postgres are presented in Table 13 below:

Tool	Description
InfluxDB	InfluxDB is a purpose-built time-series database, optimized for high-throughput ingestion and real-time monitoring. It uses a schema-less design, making it easy to add new measurements and tags on the fly. The latest version (3.0) is built in Rust with columnar storage for high-cardinality data. It supports SQL and InfluxQL for querying and provides a robust UI for dashboards and alerting. Common use cases include DevOps monitoring, IoT sensor data, and application performance metrics. It is open-source, can be self-hosted and offers a free tier. It can also support CDC via decodable. URL: influxdata.com
QuestDB	QuestDB is a high-performance SQL time-series database engineered for ultra-low latency and very high ingestion throughput. It uses a column-oriented storage engine with memory-mapped files and vectorized execution to handle millions of rows per second. QuestDB is written from scratch in Java and C++ with zero garbage collection to ensure predictable tail latency. It extends standard SQL with time-series operators like ASOF JOIN and SAMPLE BY. It is popular in finance, trading floors, and mission-critical IoT applications. It has a free and open-source core. It can also act as a CDC provider. URL: questdb.io
GreptimeDB	Greptime is a cloud-native, open-source time-series database written in Rust that unifies metrics, logs, and events. It is designed for high-velocity data ingestion and efficient storage on low-cost object storage like S3. GreptimeDB supports multiple protocols, including SQL, PromQL, and InfluxDB Line Protocol, for maximum flexibility. Its architecture is built for horizontal scalability and distributed processing in cloud environments. It aims to reduce the "learning cost" by offering familiar query languages for diverse data types. It offers self-hosting and has an open-source component. It can also provide CDC features. URL: greptime.com
TDengine	TDengine is purpose-built for the Internet of Things (IoT) and Industrial 4.0, featuring a unique "one table per device" architecture. It Includes built-in caching, stream processing, and data subscription (similar to Kafka) out of the box. TDengine includes built-in caching, stream processing, and data subscription features out of the box. It is highly efficient, often reducing storage costs by over 90% through specialized compression algorithms. It is ideal for smart meters, connected cars, and large-scale environmental monitoring. URL: tdengine.com

Table 13. Alternative time-series databases

3.3.5.11 Storage Management – POI states

The POI states module is a caching module for providing the "current state" of all the Points of Interest of VERNE Pilots. Since the "current state" of a POI is of particular interest when rendering a website and when inferring its status programmatically, it is regularly cached for fast and efficient retrieval.

In the event that a RAG or LLM model needs to be implemented, the POI state information can also be stored in a RAG scheme, with a graph database such as **Neo4j**. Neo4j is a graph database, offering vector search and graph traversal features, which are useful when building LLM-based RAG (Retrieval Augmented Generation) schemes. Neo4j Community Edition is free for self-hosting scenarios.

3.3.5.12 Storage Management - Configurations

While the POI state data update regularly and need to be made available across the Platform, Pilot and Platform configurations do not normally change often. As such, a different storage scheme such as MongoDB would be preferred. The Configuration Storage provides gated access to configuration data for the Platform, the Pilots and their POIs. It also manages changes to those configurations, launching or resetting processing pipelines when required.

3.3.5.13 Data Modelling – Agent Data Access

For security and reliability reasons, Agents, including UI (user) agents, are not permitted direct access to the layers below, including the Storage Management and the Orchestration layers. To retrieve and store data, they must use the agent API, which provides gated data access when requested. The Agent Data Access (ADA) module acts on behalf of the Agent, fetching or storing data to their appropriate store, providing a single point of control and helping avoid race conditions. It is also possible that an agent-facing cache can be used to serve often-read data, thus reducing pressure to the storage modules. This can be done by integrating a service like Redis at this layer.

3.3.5.14 Data Modelling – Agent Controller

The Agent controller is a module that controls the execution of Agents in the system. Since, for security and extensibility reasons, Agents are considered third-party code, they have limited access to the Platform resources and need a registration process to interconnect with the platform. More specifically,

- Agents may need to supply custom code for accessing third party APIs through the External Data Aggregator service
- Agents may need to notify of the KPIs they calculate and the KPI inputs required, which are in turn used in the User Input Aggregator service
- Agents may need to provide a custom user interface, to properly visualize their inputs and outputs in the (web-based) user interface for the platform.

- Agents need to be informed when spinning up (after startup) or spinning down (before shutdown). They also need to provide endpoints for launching a computation.

The agent controller module handles all this functionality, exposing the necessary endpoints to the Agent API of the platform. In addition, it monitors Agents, taking action to deregister and re-initialize stalled agents and provides Agent-operation statistics to the system.

3.3.5.15 User Front-End – UI

The User Interface component is the web server that handles the interaction with the user, including User Interface rendering, Authentication etc. It is a modular system using standard web technologies such as HTML and Javascript.

3.3.5.16 User Front-End - SSO

The Single Sign One component implements a SSO service. Single Sign-On (SSO) is an authentication method allowing users to securely access multiple applications, websites, or services using a single set of credentials (username and password). By centralizing authentication through an identity provider, it streamlines user login, enhances security, and improves efficiency for both users and IT departments. To implement SSO, the SSO module relies on an off-the-shelf authentication service similar to Keycloak or Zitadel.

Keycloak is an open-source Identity and Access Management (IAM) solution, developed by Red Hat. It provides single sign-on (SSO), user authentication, authorization, and social login capabilities. Designed to secure applications and services, Keycloak supports multiple protocols, including OAuth 2.0 and OpenID Connect. This makes Keycloak suitable for centralized IAM. Alternatives to Keycloak are presented in Table 14 below.

Tool	Description
thentik	Authentik is a modern, Python-based identity provider that features a visual flow builder for designing custom authentication stages. It is highly flexible and includes a built-in user dashboard and "outposts" to easily secure legacy applications. URL: goauthentik.io
Zitadel	Zitadel is a Go-based, API-first identity system specifically built to handle multi-tenant B2B SaaS architectures with ease. it provides a cleaner organizational hierarchy than Keycloak and includes robust audit trails for enterprise-grade compliance. It is lightweight and provides OAuth 2.0, JWT and OpenID Connect implementations. URL: zitadel.com

Tool	Description
Authelia	Authelia is a lightweight authentication server that works with reverse proxies like NGINX or Traefik to provide a secure login layer. It is the ideal minimalist choice for adding MFA and hardware key support to internal tools and homelab services. URL: authelia.com
SuperTokens	SuperTokens is a developer-first alternative that uses modular SDKs to let you build custom login UIs directly within your own application. It focuses on providing a seamless user experience and secure session management without forcing external redirects. URL: supertokens.com

Table 14. Alternatives to Keycloak

For enhanced security between the various microservices, more advanced frameworks like SPIFFE (Secure Production Identity Framework for Everyone) and SPIRE (SPIFFE Runtime Environment) can be used. SPIRE is a specialized framework focused on providing secure identities for microservices in dynamic and distributed environments. Considering the current project requirements, we do not currently anticipate the need to deploy a SPIFFE/SPIRE solution.

3.3.5.17 Agent and Platform APIs

The Platform provides two public APIs:

1. The **Agent API**, which is a set of endpoints to register and use Agent services
2. The **Platform API**, which is a set of simple methods to read basic information in the Platform

Kubernetes features Ingress and Egress API objects for managing external access. Inside a Kubernetes environment, The Ingress controller manages external access to services within the cluster, providing HTTP and HTTPS routing. It acts as an entry point, handling traffic routing, load balancing, and SSL termination. Conversely, the Egress controller governs outgoing traffic, controlling pod access to external services or the internet. Ingress and Egress together regulate how external requests enter the cluster and how internal requests leave, contributing to secure and controlled communication for Kubernetes applications.

More advanced features like dynamic load balancing and dynamic configuration can be handled by an API Gateway, like Apache APISIX. Apache APISIX is a dynamic, ultra-high-performance, cloud-native API gateway designed for microservices. Based on NGINX and etcd, it manages, secures, and routes traffic for APIs, supporting features like load balancing, canary releases, and dynamic plugin loading without restarts. It also provides specialized features for AI applications, including token-based rate limiting and LLM load balancing. It can be extended

via plug-ins, written in multiple languages, including Java, Go and Python. APISIX is widely used for API management and as a Kubernetes Ingress controller. Alternative solutions to Apache APISIX are presented in Table 15 below:

Tool	Description
Kong	Kong Gateway is a widely-adopted, open-source API gateway built on NGINX that excels in high-traffic environments. It provides a massive ecosystem of plugins for authentication, monitoring, and traffic control across multi-cloud setups. The platform is highly extensible, allowing developers to customize workflows using Lua or through a sidecar deployment. It is particularly favored by enterprise teams for its battle-tested reliability and robust Kubernetes integration. URL: konghq.com.
Tyk	Tyk is a powerful, open-source API management platform written in Go that prioritizes performance and developer experience. It includes a built-in gateway, dashboard, and developer portal to manage the full API lifecycle right out of the box. The architecture is designed to be lightweight, making it an excellent choice for microservices and containerized applications. It supports a variety of protocols including REST, GraphQL, and gRPC with granular security controls. URL: tyk.io.
KrakenD	KrakenD is a high-performance, stateless API gateway designed specifically for microservices and "Backend for Frontend" patterns. Because it is stateless, it requires no database to run, allowing it to scale linearly and handle massive throughput with minimal latency. It focuses on aggregating multiple backend responses into a single, optimized result for the client. The configuration is handled through simple JSON or YAML files, making it ideal for CI/CD pipelines. URL: krakend.io.
Traefik	Traefik is a modern, HTTP reverse proxy and load balancer designed to deploy microservices with ease. It integrates natively with Docker, Kubernetes, and other orchestrators to automatically discover and update your routing configuration in real-time. This "zero-config" approach removes the need to manually manage configuration files when services are added or removed. It includes a built-in web UI for monitoring and supports Let's Encrypt for automated SSL certificate management. URL: https://traefik.io.

Table 15. Alternatives to APISIX

3.3.6 Process View

The Process View focuses on a system's runtime behaviours and the interaction among parts of the system. While modern cloud-native and distributed systems include multiple subsystems, interacting in complex ways, it is relatively straightforward to describe the core concepts, namely the user-facing interactions and the core data flows, possibly hiding the intricate details behind abstraction layers.

Figure 7 below illustrates the general data flow from sensors to Agents. The grey area illustrates the Platform boundary at a logical level. Note that the Agents are also hosted in the same Kubernetes cluster and have privileged access through a specialized API. For the design purposes of the Platform however, an Agent is a client system, since in principle the Platform developers have no visibility or control on the intrinsic processes of the Agent

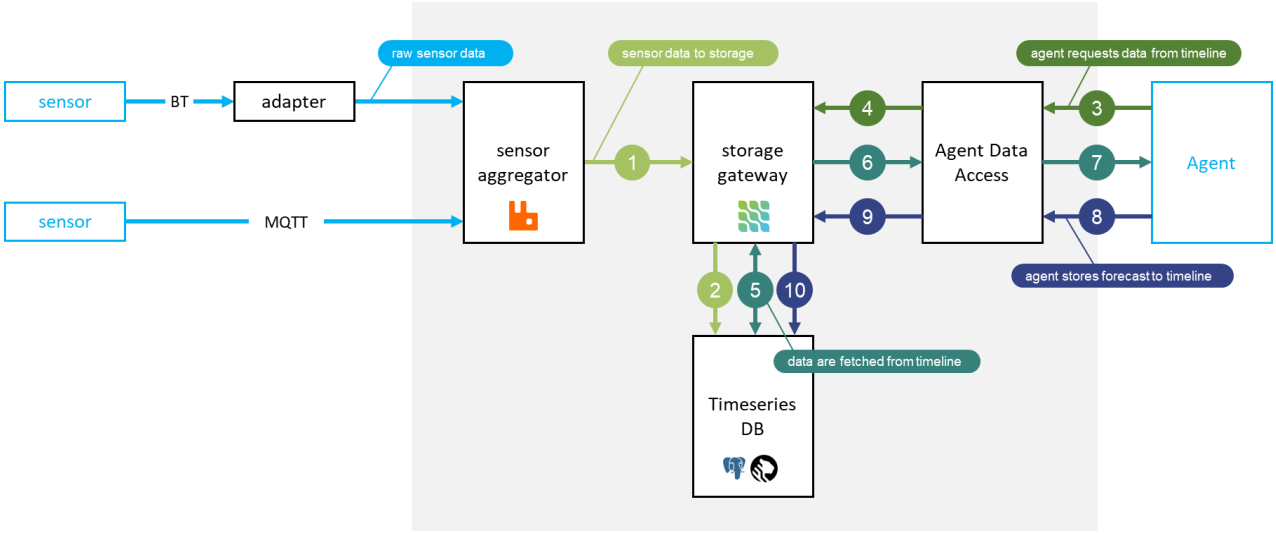


Figure 7. Sensor data flow

Sensor data are set-up (through a configuration component not included in the figure) to reach the Sensor Aggregator, either directly through MQTT or via an adapter. The Aggregator service then submits the data, along with metadata describing the relevant POI tag, timestamps and sensor names, to the storage gateway. Since sensor data are timestamped, they are stored in a Time-series database of the Storage Layer. When an Agent requires the sensor data, they will submit a data retrieval request to the Agent Data Access layer (via the Agent API), which, if properly formed, will be forwarded to the storage gateway. From there, the data will be retrieved from the time-series database and passed to the Agent. The computational results of the Agent will be then submitted back to the storage gateway via the Agent Data Access layer and stored in the time-series database, so that other Agents can have access to that computation. Note that, because of the Change Data Capture (CDC) characteristics of the storage gateway, it is possible to sense when data of a specific source (sensor, agent, external

API) are submitted to the Storage Layer and events can be fired conditionally to launch additional computations.

3.3.6.1 Agent Inference Process

The sequence diagram in Figure 8 below illustrates the process of performing an Agent Computation.

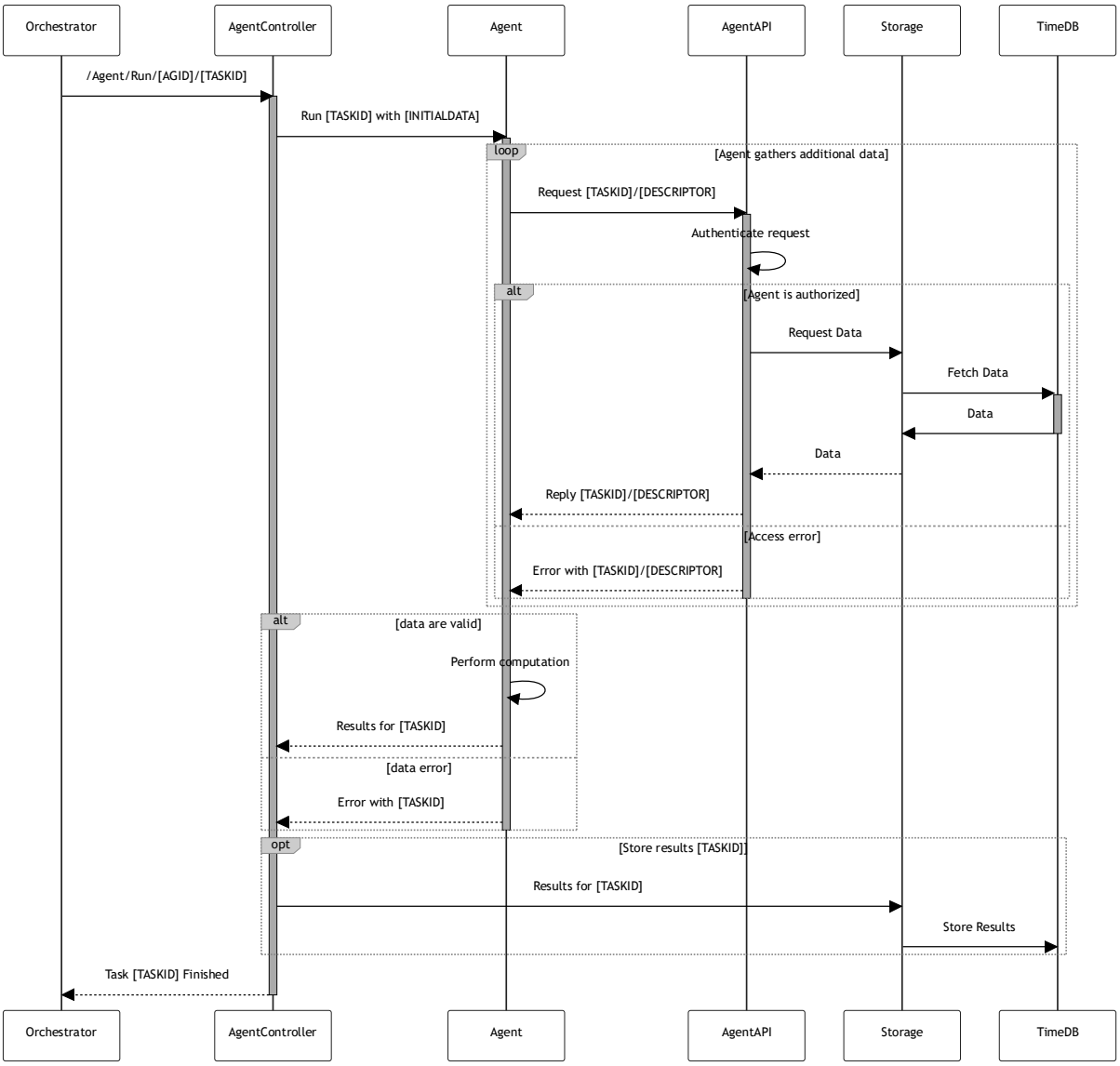


Figure 8. Agent Computation Process

The Orchestration layer launches the process by requesting a job to be run by the Agent Controller, also providing a task-id as reference to this specific request. The task-id is utilized to ensure idempotency and synchronize all agent-related actions, since it is possible that multiple instances of the agent are still running, performing calculations for other tasks. The

Agent Controller then launches the Agent, supplying an initial bundle of data relevant to the execution of that agent. The Agent may request additional data using the Agent API, which will intern access the data via the Storage Management Layer. The results are then returned to the Agent Controller together with the task-id, marking the task as complete in the system. The controller may store the results in the Storage Layer.

3.3.6.2 Agent registration process

Agents, by definition, operate in the background, without user intervention. However, they may need to present a user interface to the user, as part of the platform. More specifically, they may need to provide interfaces for the configuration of the agent and for the visualization of the agent computation results.

In short, the Platform may need to be extended in order to handle the data that the Agents require. To implement this, there is a registration process that all Agents undergo when initialized.

During the registration process, the Agents describe the data they require and provide the necessary functionality to capture the data from any third-party sources and to present their output to the user. The state diagram in Figure 9 presents the Agent lifecycle and the Registration process.

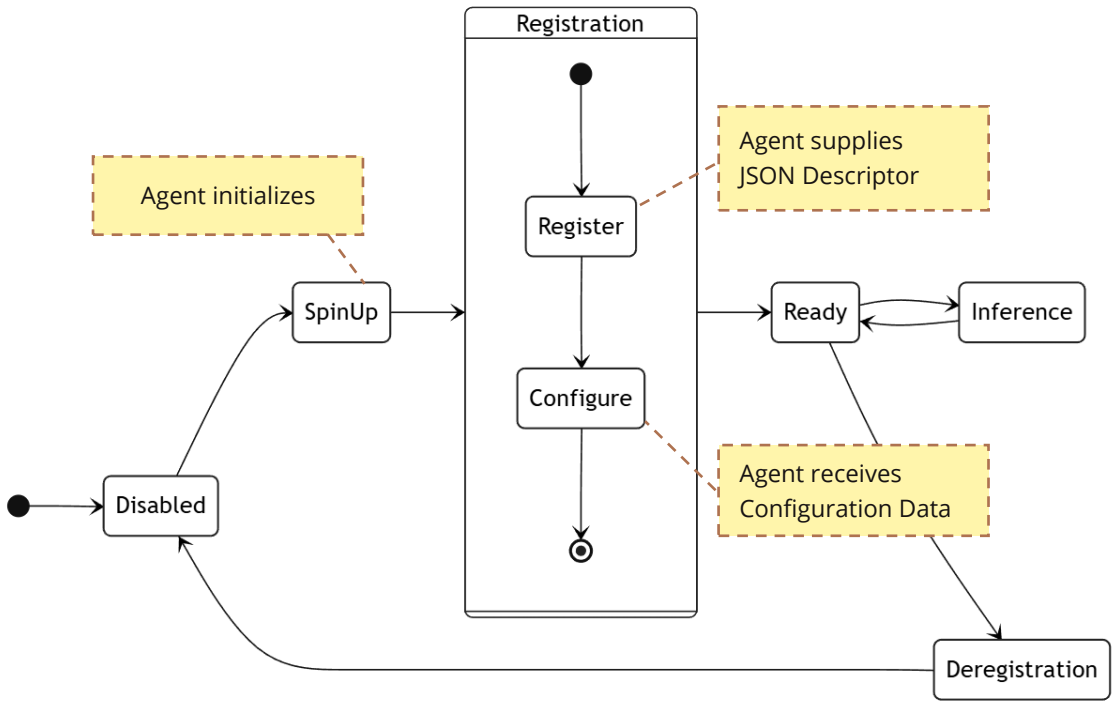


Figure 9. Agent lifecycle

During registration, the Agent must provide a JSON Descriptor object, which describes:

- The configuration data required during initialization, also including types and default values.
- The standard input data that will be required during inference
- The endpoints for accessing extra functionality:
- The methods and endpoints that will be used by the External Data Aggregator to capture and aggregate data from external APIs
- The necessary UI components, i.e. user-facing (frontend) scripts that assist with the visualization of stored data

The Figure 10 below summarizes the Initialization and Registration process.

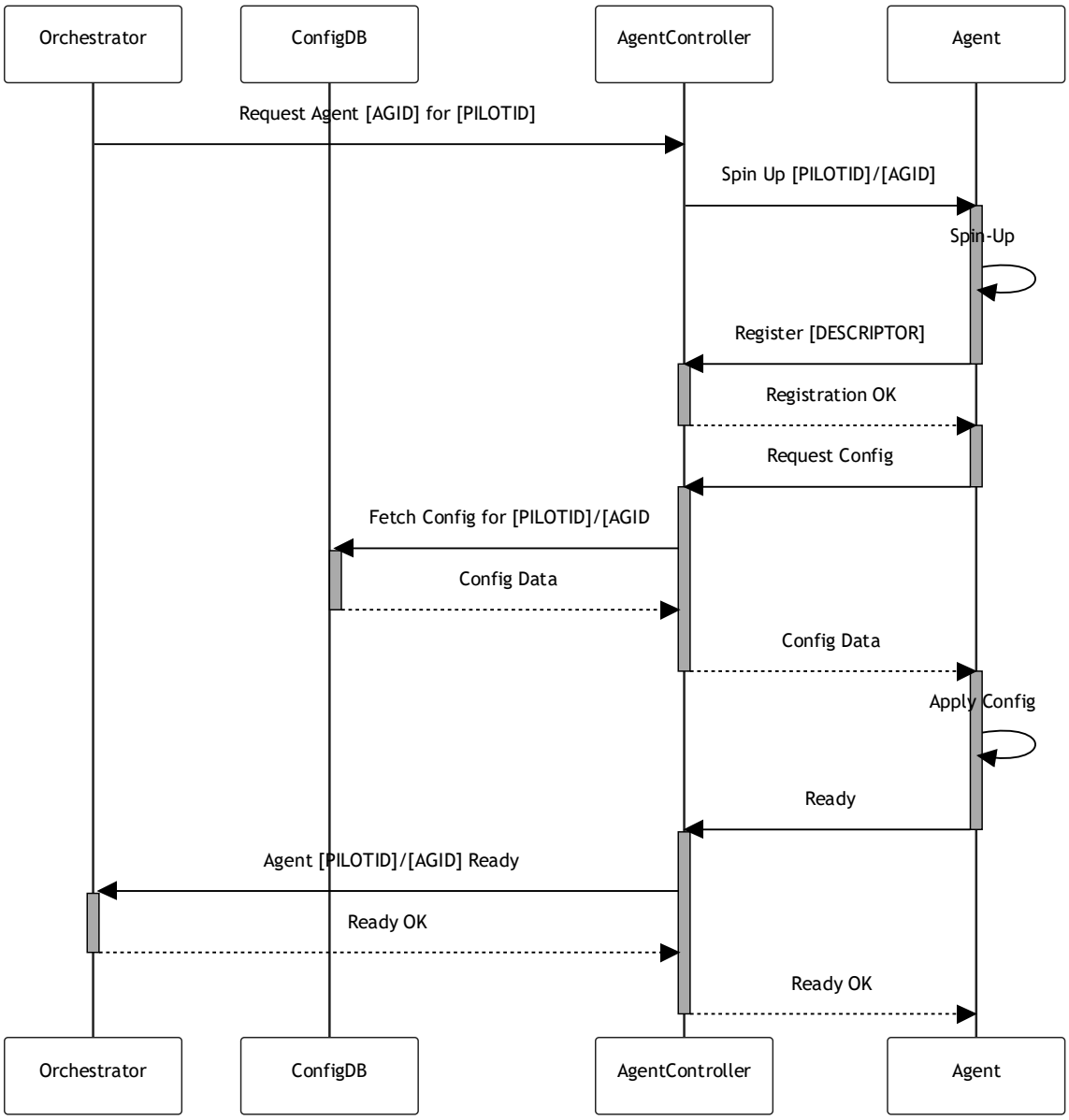


Figure 10. Agent Initialization and Registration

Accordingly, the Agent Controller will be using the JSON descriptor to present configuration options in the User Interface directed to Agent Administrators, allowing them to change configuration parameters. The altered configuration object is stored in the configuration database and is sent back to the Agent during the Agent-registration process.

Note that before executing an Agent calculation/inference, the Agent Controller uses the JSON descriptor to identify what data will be required, collect and submit them together with the inference request. This is done to minimize the API calls between Agent and Platform. The inference results are timestamped and stored, to be retrieved either for presentation or for further computations.

To present the Agent results, the Agent Controller retrieves the previously stored and timestamped inference from the Storage layer, then presents them to the user, using the Agent-supplied UI components. Note that those components will be used only for data that cannot be presented using common UI components. For example, a time-series of scalar measurements would not require a special UI component for display. On the other hand, a time-series of 3D Vectors might need a special UI component, otherwise they would be displayed as 3 independent curves. The registration process allows the Agent to provide customized visualization functionality for their data.

3.3.7 Development View

The development view depicts a system from the standpoint of the developers and is concerned with software administration. As the previous section presented several core architectural concepts, this section will focus on code integration and development – related topics.

As the system is implemented as a Kubernetes cluster, developers will be either submitting code to be executed within a container (Kubernetes Pods), or an entire container, interfacing with the rest of the Platform via a well-established interface. The Figure 11 below illustrates the coding and continuous integration process across domains:

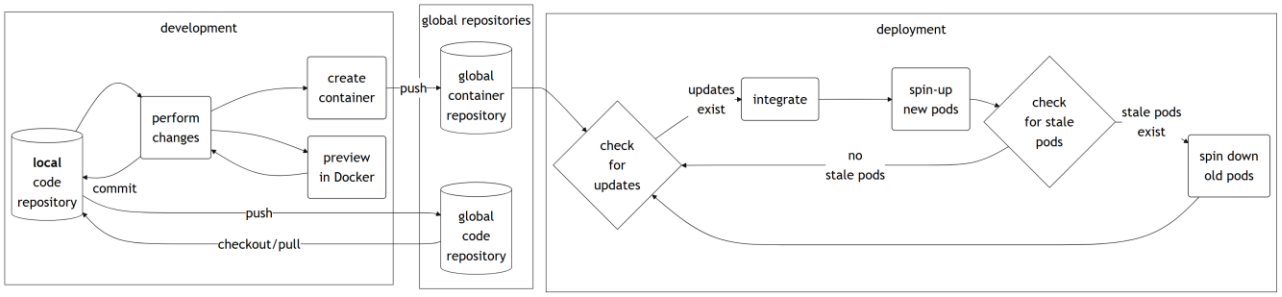


Figure 11. Development process

The process illustrated in Figure 10 reflects the common practice regarding container-based development. Developers can develop code in their local environment, utilizing containers to preview their code. A development instance of the system is provisioned for testing purposes where required. To deploy their changes, developers have two options, depending on the component under development.

1. commit changes to the common code repository, using industry-standard CVS concepts like commit, push and pull requests. This option applies to internal Platform components.
2. create (or update) a container image, which can be published in the common container repository (container registry), which will then be integrated to the platform via CI/CD processes. This option applies to Agent development and specific user-facing Platform components, namely User Interfaces.

To simplify Agent development, a template with boilerplate code will be provided, allowing developers to quickly create Agents by extending its functionality – essentially by defining a set of functions and configuration objects, while the template handles initialization and communication with the platform.

3.3.8 Scenarios

Kruchten's 4+1 Architectural view model postulates the use of a set of scenarios, which illustrate the use of the architecture. Scenarios describe the interactions of the various objects and processes within the architecture to perform exemplary use cases. Some foundational use cases have been described in the previous chapters and will be expanded upon here.

3.3.8.1 Scenario 1: Platform as an aggregator

As demonstrated in Figure 5 and Figure 6, the VERNE ecosystem utilizes various data sources, (sensors, forms, files, third-party APIs) as inputs to Agents, which can in turn be used to model resources, make predictions or calculate performance indicators. The Figure 12 below illustrates the process.

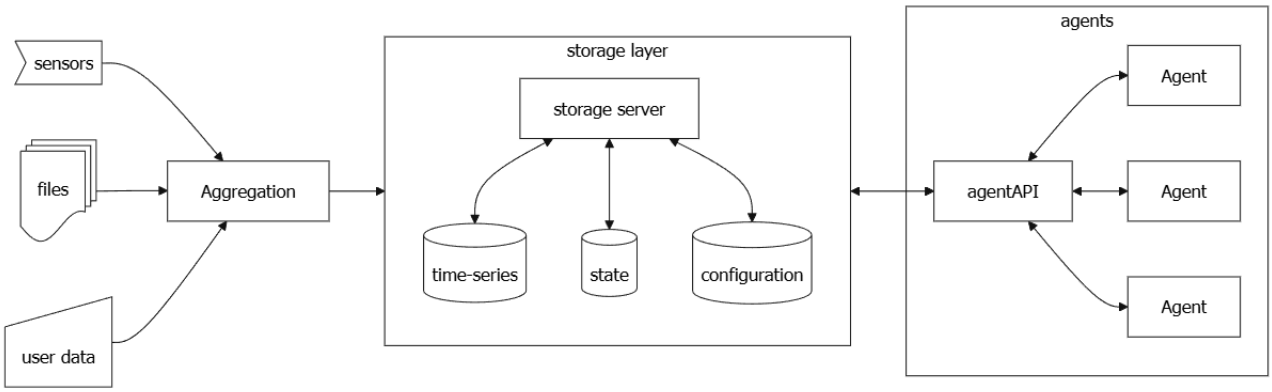


Figure 12. Data Aggregation to agents

This is an automated process that takes place according to configurations set by the Pilot administrators, which is part of another platform scenario.

3.3.8.2 Scenario 2: configuration by Pilot Administrators

The Platform allows Pilot Administrators to configure certain parameters of the system, including the sensor streams that will be used as input to the Agents, as well as the order in which calculations take place, the workflows. Those configuration files are used by the Orchestration layer to schedule execution of the computational workloads (the Agents), which, in turn, store their data in the time-series database. Figure 13 below demonstrates this process.

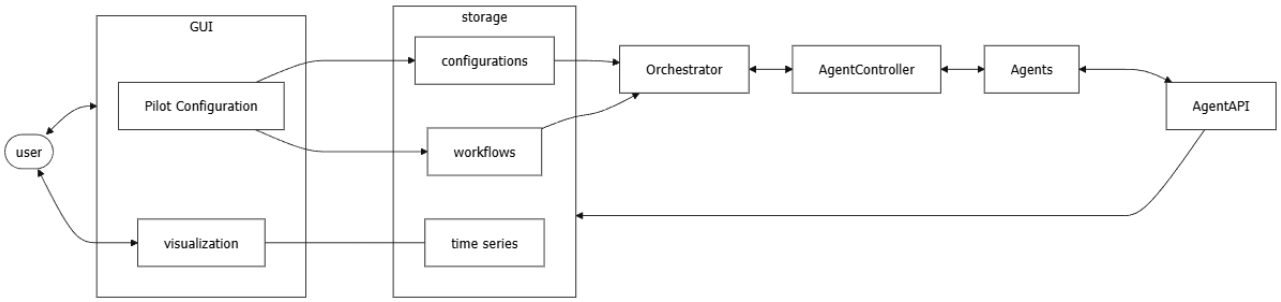


Figure 13. Configuration Scenario

The Graphical User interface (GUI) is a web-based system that allows configuration, data entry, as well as visualization of the various KPIs.

3.3.8.3 Scenario 3: development process

Another scenario of interest is the development process, i.e. the interaction of the Platform with the developers. At least two categories of developers are implied in this case, as Figure 14 depicts:

1. The Platform developers. This includes all individuals who create parts of the platform.
2. The Agent developers. Agent developers, by definition, are interested in developing an Agent, i.e. a computational unit that performs calculations using data from the Platform.

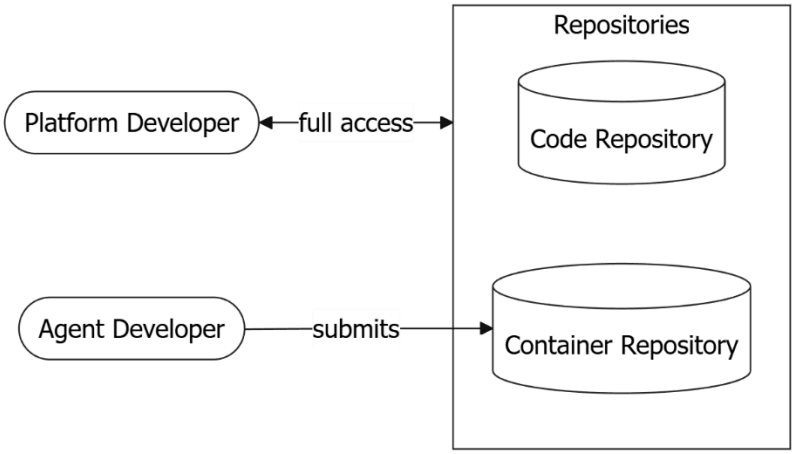


Figure 14. Developer access

Agent Developers do not have access to the code of the platform. However, they interact with the Platform by creating deployable containers and submitting them for execution within the Platform. Modern code repositories, including GitLab, allow different classes of developers and feature container repositories.

4 Pilot system architectures and data integration points

4.1 Pilot 1 – Bornholm (Food Experience and Circular Tourism)

4.1.1 AS-IS situation

The Bornholm pilot is situated on an island in the Baltic Sea with a strong strategic focus on tourism through local food experiences and gastronomy, supported by established policy instruments (e.g., *Foodstuff strategy for Bornholm 2017–2025* and *Development plan for tourism on Bornholm toward 2030*). These strategies emphasize food product innovation, small batch production, food culture, off-season tourism, art and culture, and climate and environment as important intangibles and strongholds of the island.

The local economy is strongly reliant on tourism and features a dense food ecosystem with a number of consumer touchpoints, including:

- farmers' markets, curb-side shops, food experience tours and farm shops
- a broad range of high-end restaurants
- Denmark's biggest annual cooking contest "Sol Over Gudhjem"
- local food producers and small-batch processors
- hospitality and tourism service providers (restaurants, hotels, events)

- active networks of stakeholders working to position Bornholm as a leading destination for food experiences

Bornholm also actively works to combine its food strategy with three broader strategic greening pillars:

- Energy island ambitions, supporting renewable energy use and energy transition
- Zero waste strategy, supporting circular resource management
- Nature tourism and re-wilding as part of the recent national Tri-Partite deal

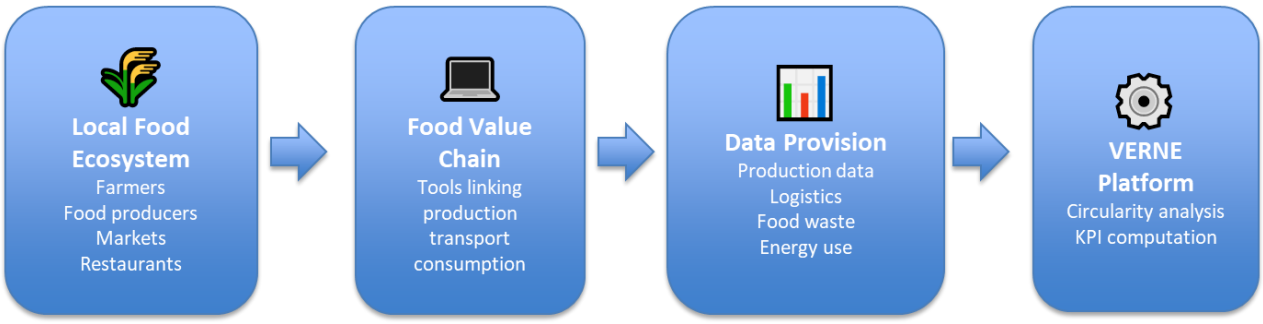
An important part of the Bornholm approach is to integrate these pillars as active components of a sustainable and circular tourism strategy, supported by strong stakeholder networks and communication of sustainable practices to end consumers.

In circular economy terms, the innovation goal focuses on integrating and optimizing sustainable practices across the entire food value chain, including:

- supporting local sourcing, self-sufficiency, resilience and reduced transport distances
- reducing food waste mainly through labelling and awareness raising actions
- promoting renewable energy use and low-impact operations
- strengthening transparency and communication of sustainability activities to consumers

The interaction between the Bornholm pilot and the VERNE platform can be summarized as follows:

- The local food tourism ecosystem develops and implements circular practices across the food value chain (production, distribution, consumption, and waste reduction).
- The pilot's tools and data sources provide operational data to the VERNE platform for the area's energy, waste and land use.
- The VERNE platform harmonizes and interprets these data to compute and showcase circularity performance indicators for Bornholm's sustainable and circular food tourism strategy.



Bornholm pilot: Data flow to the VERNE platform

4.1.2 Digital infrastructure

Below are indicatives examples regarding digital tools that exist in the context of Bornholm pilot:

- The “Mad Bornholm” is a digital tool developed and offered by Gourmet Bornholm which promotes B2B producer and restaurateur trade https://bornholm.info/gourmet-bornholm-regional-madkultur/
- Digital tools provided by Destination Bornholm are supporting tourism engagement (e.g., tourism portals, mapping and planning services, booking interfaces) https://bornholm.info/en/
- Data sources supporting sustainability and waste-related monitoring are provided by Statistics Denmark (e.g., waste logs, waste management reporting) https://www.dst.dk/en
- Data sources supporting energy / renewable integration (e.g., aggregated energy indicators) is provided by DTU Power lab https://bornholm.powerlab.dk/
- General statistics on tourism is provided by Visit Denmark https://www.visitdenmark.com/

4.1.3 Pilot objectives & Use Cases

Strategic objective

The primary objective of the Bornholm pilot is to strengthen sustainable and circular tourism by enabling visibility, optimization and communication of circular practices across the local food value chain.

The pilot aims to:

- promote sustainable food behaviours and practices by tourists and travelers
- increase the uptake of local foods in the tourism eco system
- reduce food waste and improve circular resource utilization within food experiences
- improve traceability and transparency of local food sourcing and sustainable practices
- support the integration of renewable energy and reduced-waste practices into tourism operations
- generate measurable indicators demonstrating circular tourism performance in the food ecosystem

Core Use Cases

UC-1: Circular local food value chain visibility

Provide visibility on local food production, distribution and consumption patterns (as available), enabling better understanding of flows across producers and tourism actors.

This use case supports:

- transparency for consumers of local sourcing and food supply networks

- visibility of food product availability and local offerings
- evidence of local value-chain activity supporting circular food tourism

UC-2: Food waste reduction and circular practices monitoring

Support monitoring of food waste-relevant data (where available) and circular practices applied by food and tourism actors (e.g., reuse, redistribution, labelling, awareness raising, improved planning).

This use case supports:

- quantification of waste reduction efforts
- tracking of surplus handling practices (where applicable)
- evidence of circular resource management in tourism food services

UC-3: Circularity performance verification and communication

Generate circularity indicators demonstrating how Bornholm's food tourism ecosystem performs against circular economy goals (e.g., reduced waste, local sourcing, low-impact practices).

This use case supports:

- measurable circularity evidence for stakeholders
- communication of sustainable practices to visitors
- reporting and benchmarking over time

UC-4: Sustainable experience guidance (consumer-facing enablement)

Enable visitors to discover and select food experiences aligned with sustainability and circularity principles (e.g., local sourcing, low waste, renewable-powered operations, seasonal offerings).

This use case supports:

- sustainable consumer decision-making
- improved visibility of circular food actors
- strengthened positioning of Bornholm as a circular food tourism destination

4.1.4 Architecture mapping & integration points

Circular food tourism ecosystem architecture

The Bornholm pilot architecture is centered around a local food and food tourism ecosystem, where circularity is pursued for improved coordination, transparency and optimization across the food value chain. The system architecture consists of the following subsystems:

- Food production & processing: Local producers, small-batch processors, farm shops and food innovation actors.
- Tourism consumption: Restaurants, hotels, food related and other events, food tours, tourism services and consumer touchpoints.

- Circular practices: Practices and mechanisms supporting reduction of food waste, reuse/recovery where applicable, and circular resource management.
- Digital data: Digital tools and data sources used to link production and consumption and to document sustainability practices and to convey transparency and visibility towards consumers.

Key integration points include:

Integration point	Purpose
Local food/tourism sources	Operational data and content related to producers, offerings, and tourism actors
Sustainability & waste-related sources	Data supporting circularity indicators
Renewable energy - related sources	Data supporting sustainability indicators

Through these integration points, data from the Bornholm pilot will be analyzed to derive circularity and sustainability indicators describing the performance and evolution of the circular food tourism ecosystem.

4.1.5 Data specifications

Data categories

The Bornholm pilot is expected to generate data related to:

- local food actors (producers, processors, food service providers)
- local food offerings and experiences (products, events, seasonal availability)
- tourism-relevant transactions or engagement signals
- waste-reduction and circular practice indicators including food waste
- energy/renewable integration indicators
- land use indicators for various food and meal types

These datasets may be accessed and integrated into the VERNE platform data layer for circularity analysis mainly via periodic structured exports and via APIs if available. The payload format is expected to be CSV/Excel (for exports), containing time-stamped records and relevant metadata (ID, location, category, etc.) and JSON (for any available APIs).

Data category	Data fields	Frequency	Granularity	Purpose
Producer / supplier registry	ID, type, location, product category	On change / periodic	Per actor	Map ecosystem actors
Circular practices	Low-waste practices, reuse/recovery practices (if logged)	Monthly / quarterly	Per actor	Circularity evidence
Waste-related	Food waste quantities or proxies	Weekly / monthly	Per site / time period	Waste reduction monitoring
Energy / renewables	Renewable share	Monthly	Aggregated	Link energy to tourism

4.1.6 Circularity indicators

Indicative circularity indicators for Bornholm pilot may include:

- % of local food sourcing
- Tons and % of food waste reduction
- % of sustainable transport options used to reach food experiences and cultural attractions
- Scale of visitor satisfaction with or recognition of local food experiences
- % of tourists aware on sustainable food practices and experiences
- % or number of circular and sustainable practices by food experience suppliers
- % of local food system businesses involved in circular and sustainable food tourism experiences
- Number of jobs created in the local food tourism sector
- Number of different digital solutions applied to foster sustainable food experiences

4.1.7 VERNE platform digital tools

Bornholm circular food experience explorer

A variation of an interactive map and narrative visualization of Bornholm's local food ecosystem, connecting consumers with local producers, showcasing food experiences, and presenting sustainability/circularity evidence.

Bornholm circularity KPI dashboard

A KPI dashboard that visualizes circularity indicators over time and supports communication of Bornholm's progress toward circular tourism goals (food waste reduction, local value-chain strengthening, renewable-friendly practices).

4.2 Pilot 2 – Sønderjylland-Schleswig (Sustainable Corporate Heritage

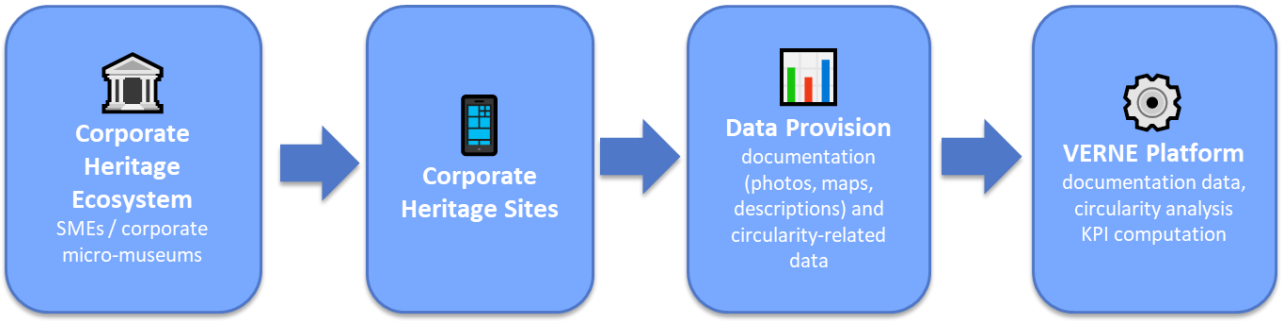
4.2.1 AS-IS situation

The Sønderjylland–Schleswig pilot is implemented in the Danish–German border region, characterized by stable governance and strong cross-border that regards the border region as a tourist destination. The region shows increasing technological capacity, including a recent trend of start-ups locating in cities such as Flensburg and Sønderborg. While tourism is an important economic driver in the region, and numerous large- and small-scale projects have been successfully launched in the recent past, there is cross-border coordination with a view to sustainable tourism formats.

The pilot focuses on sustainable corporate heritage tourism, where local and regional SMEs leverage their own history, identity, and stakeholder relations as cultural attractions. A key pilot output is the mapping of corporate heritage tourism destinations, highlighting cultural richness that is often overlooked in rural areas. Circularity is implemented through the micro-museum (MM) concept, which prioritizes the reuse of existing spaces and the use of reused/recycled/upcycled materials for exhibition elements and installations. This approach benefits both visitors and local communities by promoting decentralized cultural tourism, reducing the need for travel to urban centres, and supporting rural socio-economic resilience. The pilot involves a multi-stakeholder ecosystem, including public administrations, destination management and private services, citizens and communities, visitors/tourists. The interaction between the Sønderjylland–Schleswig pilot and the VERNE platform is summarized as follows:

- The corporate heritage tourism ecosystem implements circular practices through micro-museums, prioritizing the reuse of existing spaces and the use of reused/recycled/upcycled materials across rural heritage sites.

- There is a web-app prototype scoped as a discovery and documentation tool (photos, maps, descriptions) for a network of the corporate heritage micro-museums.
- Circularity-related data shall be collected through complementary pilot-side documentation/reporting mechanisms
- The VERNE platform harmonizes with the web-app documentation data (e.g., photos, maps, descriptions) to support the communication of the Sønderjylland–Schleswig corporate heritage tourism network.
- Circularity performance indicators will be derived from related data.



Sønderjylland–Schleswig pilot: Data flow to the VERNE platform

4.2.2 Digital infrastructure

A digital tool supports corporate heritage site visibility, visitor access, and cross-border tourism engagement through a single web app that showcases corporate museums and provides an interactive site map.

4.2.3 Pilot objectives & use cases

Strategic Objective

The strategic role of the Sønderjylland–Schleswig pilot is to demonstrate a cultural heritage ecosystem applying circularity through the use of existing, recycled, and upcycled resources in museum and heritage-related activities.

The pilot aims to:

- Identify and map corporate heritage tourism destinations and corporate (micro-) museums sites across the cross-border region
- Support SMEs and local actors in implementing corporate (micro-) museums interventions using existing, recycled, and upcycled materials
- Create a network / community of practice of stakeholders of heritage and cultural tourism
- Provide information to support communication of circular practices

Core use cases

UC-1: Cross-border site mapping

Identify, document, and maintain a structured inventory of corporate heritage tourism destinations and micro-museum sites across the Sønderjylland–Schleswig cross-border region.
This use case supports:

- creation and update of a site overview (site profiles, locations, categories)
- thematic clustering (e.g., potentially routes, categories of heritage)

UC-2: Micro-museum intervention documentation (reuse/upcycling)
Capture micro-museum interventions implemented by SMEs and local actors, with specific focus on the use of existing, recycled, and upcycled resources.
This use case supports:

- recording of intervention (type, scope, materials)

UC-3: Circularity performance
Measurable circularity indicators that quantify the extent and effectiveness of reuse/upcycling practices at pilot sites.
This use case supports:

- measurable circularity evidence for stakeholders
- communication of sustainable practices to visitors

4.2.4 Architecture mapping & integration points

Cultural heritage circularity ecosystem architecture
The Sønderjylland–Schleswig pilot is structured around a cross-border corporate heritage tourism ecosystem that makes use of circularity through micro-museum interventions and the reuse of existing, recycled, and upcycled resources. The envisaged system architecture consists of the following subsystems:

- Corporate heritage site ecosystem: Local and regional SMEs and corporate heritage site owners implementing corporate (micro-) museum interventions and providing site-level information (e.g., site attributes, interventions, circularity practices).
- Site registry and mapping subsystem: A registry of corporate heritage tourism destinations and corporate (micro-) museum sites, including location and categorization information to support mapping and discovery.
- Micro-museum documentation subsystem: Documenting micro-museum interventions, with a specific focus on materials and reuse/upcycling characteristics. (e.g., reporting templates, forms, photos, etc.).

Key integration points include:

Integration Point	Purpose
Site registry / mapping data interface	Provide site inventory and location data enabling mapping and discovery
Circularity reporting data sources	provide Information to support communication of circular practices

Through these integration points, the VERNE platform will interpret data from the Sønderjylland–Schleswig pilot and generate circularity indicators describing the reuse/upcycling practices within the corporate heritage tourism ecosystem.

4.2.5 Data specifications

Data categories

- Heritage data, including site identity, type, location, and thematic focus
- Accessibility, opening hours
- Form of visitor engagement
- Exhibition and infrastructure data, related to exhibition elements, and reuse of existing components
- Private or state-funded site
- Contact information
- Exhibition examples (5 pictures at least)
- Exhibition and infrastructure data, related to exhibition elements, and reuse of existing components

These datasets may be accessed and integrated into the VERNE platform data layer for circularity analysis mainly via periodic structured exports and via APIs if available. The payload format is expected to be CSV/Excel (for exports), containing time-stamped records and relevant metadata (ID, location, category, etc.) and JSON (for any available APIs).

Data category	Data fields	Frequency	Granularity	Purpose
Heritage supplier registry	ID, type, location, product category	On change / periodic	Per actor	Map ecosystem actors
Circular practices	Reuse/recovery practices	Semestral	Per actor	Circularity evidence

Data category	Data fields	Frequency	Granularity	Purpose
Waste-related	waste quantities or proxies	Annually	Per site / time period	Waste reduction monitoring
Energy / renewables	Renewable share	Annually	Aggregated	Link energy to tourism

4.2.6 Circularity indicators

Indicative circularity indicators for Sønderjylland–Schleswig pilot may include:

- % of MMs using RES
- Waste generation at MMs in Kg
- % of recycled materials in exhibitions
- % of MMs accessible for people with disabilities
- % of visitors aware on local cultural heritage and sustainability practices
- Number of local SMEs involved in heritage tourism
- Number of cultural heritage attractions created and identified

4.2.7 VERNE platform digital tools

Sønderjylland–Schleswig corporate heritage explorer
A VERNE platform tool that presents a map and narrative visualization of corporate heritage tourism destinations and micro-museum sites across the Sønderjylland–Schleswig cross-border region. The tool presents site profiles, potentially routes, and circularity evidence (e.g., reuse/upcycling highlights, material reuse indicators), supporting both stakeholder understanding and visitor-oriented discovery.

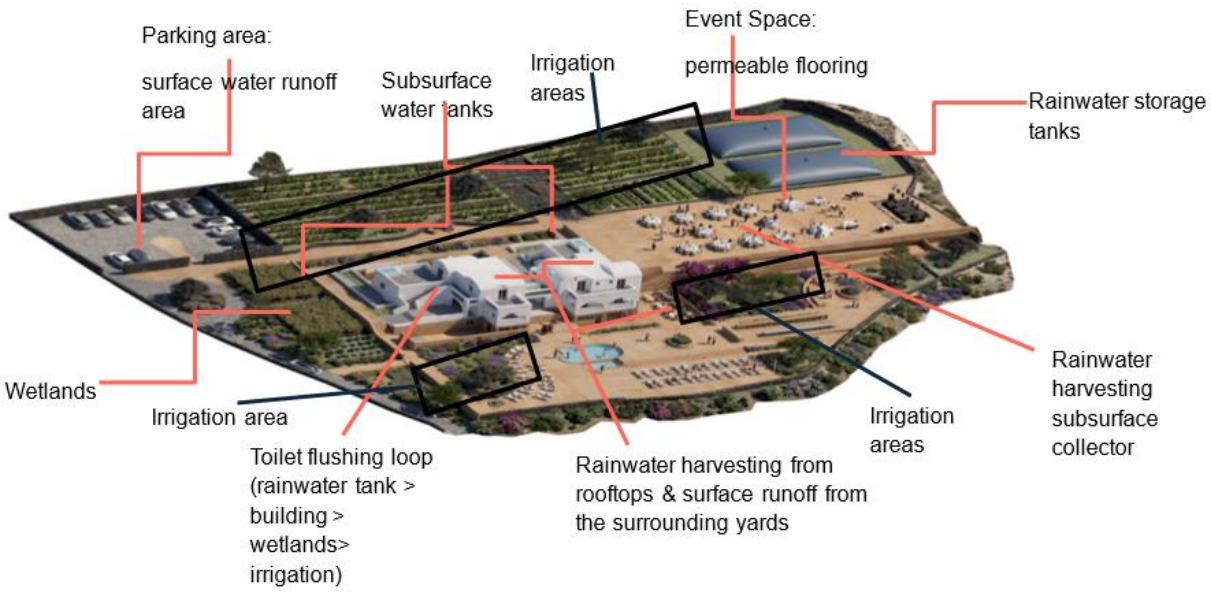
Sønderjylland–Schleswig circularity KPI dashboard
A KPI dashboard within the VERNE platform that visualizes circularity indicators over time and supports evidence-based communication of the pilot's circular heritage performance.

4.3 Pilot 3 – Santorini (Impact Hotel)

4.3.1 AS-IS Situation

The Santorini pilot addresses water scarcity and seasonal demand peaks typical of island tourism contexts by deploying an on-site circular water infrastructure that reduces

dependence on external freshwater supply. The Impact Hotel complex is treated as an integrated "site-scale" system, where rainwater capture, storage, reuse, and natural treatment processes are combined into a single operational loop. This approach enables continuous circulation of water resources across the facility, coupling engineered components (collection surfaces, subsurface aggregation and storage, flushing reuse) with nature-based treatment (constructed wetlands) and final reuse in irrigation.



As a result, the pilot operates as a closed, measurable hydraulic loop that can be monitored in terms of inflows, storage dynamics, reuse volumes, treatment performance, and redistribution flows, forming the baseline system upon which digital monitoring and circularity assessment functionalities can be built. The figure below illustrates the integrated system configuration at the Impact Hotel complex.

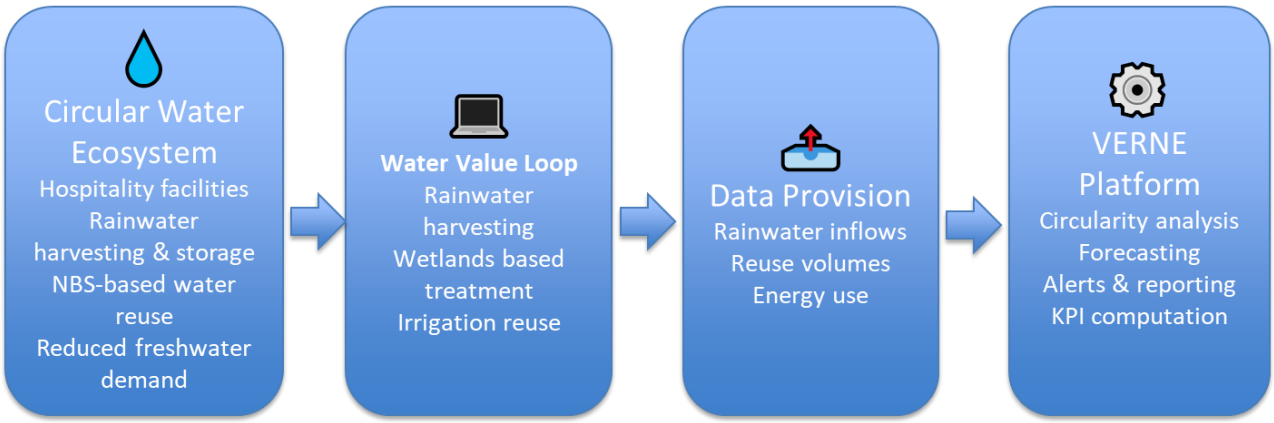
The physical layers include:

- Rainwater collection surfaces
  - Roof surfaces (primary collectors)
  - Permeable event space
  - Parking area runoff surfaces
- Hydraulic aggregation & storage
  - Central subsurface collector
  - Inflatable storage tanks
- Reuse & treatment loop
  - Water flushing reuse loop
  - Constructed wetlands for natural greywater treatment
  - Irrigation zones for treated water reuse

The system operates as a closed and measurable hydraulic loop, enabling traceable inflow, storage, reuse, treatment, and redistribution flows.

The interaction between the Santorini pilot and the VERNE platform can be summarized as follows:

- The circular water system at the hospitality complex generates primary measurement data through its monitoring infrastructure (e.g., flow/level sensors and environmental measurements).
- These data streams are transmitted to the VERNE platform (e.g., via the MQTT broker) for ingestion and further processing.
- The VERNE platform harmonizes and interprets the incoming data to compute and showcase circularity performance indicators, supporting monitoring of the water reuse loop and communication of achieved circularity outcomes.



Santorini pilot: Data flow to the VERNE platform

4.3.2 Digital and sensing infrastructure

The circular water system is supported by an IoT sensing substrate, forming the physical sensor layer, as shown in detail below, from which data streams are generated for processing and analysis.

Category	Type	Measured variable	Deployment location
Water	Flow meters	Inflow – outflow / tank levels	Collector, reuse loop / storage
Soil	Soil moisture & temp probes	Soil saturation	Irrigation zones
Environment	Temperature sensor	Ambient temperature	Outdoor area
Environment	Humidity sensor	Ambient Relative humidity	Outdoor area

Category	Type	Measured variable	Deployment location
Environment	Pressure sensor	Atmospheric	Outdoor area
Environment	Rain Gauges	Precipitation (rainfall depth)	Outdoor area
Energy	Energy meters	Water system energy consumption	Pumping & water treatment system
Energy	Energy meters	Building energy consumption	Building
Communication	Gateways	Data transfer	Site

4.3.3 Pilot objectives & use cases

Strategic objective

The primary objective of the Santorini pilot is to maintain stable, resilient circular water operation in a water-stressed, seasonally variable hospitality context by enabling predictive monitoring and evidence-based control of the closed-loop water system. By continuously assessing inflows, storage autonomy, reuse volumes, and treatment performance through flow-balance analytics and forecasting, the pilot aims to detect emerging imbalances early and support timely corrective actions. This prevents silent degradation of the loop (e.g., storage depletion or reduced wetland treatment efficiency) during peak demand periods and ensures continuous verification of circularity performance.

Core use cases

UC-1: Predictive storage depletion management

- Forecast storage autonomy
- Combine inflow, usage patterns, and weather forecasts
- Enable pre-emptive corrective actions

UC-2: Treatment loop stability monitoring

- Monitor inflow–outflow balance
- Detect gradual efficiency degradation
- Trigger alerts before threshold violation

UC-3: Circularity performance verification

- Produce time-stamped circularity metrics
- Enable performance verification
- Support long-term circular operation

UC-4: Visualization model

- Engineering view
- Visitor narrative view (soft indicators & participation footprint)

4.3.4 Architecture mapping & integration points

Circular water system architecture

The Santorini pilot is built around a circular water recycling system. The system architecture concerns exclusively the hydraulic and operational structure of the water reuse loop and it is structured as follows:

1. Water Collection Subsystem

Rainwater harvesting from roof surfaces

Surface runoff collection from permeable event and parking areas

Routing to central subsurface collector

2. Storage Subsystem

Central underground collector

Inflatable storage tanks

Hydraulic routing mechanisms

3. Reuse Subsystem

Toilet flushing loop supplied from storage tanks

Controlled distribution to building facilities

4. Treatment & Recovery Subsystem

Greywater routing to constructed wetlands

Natural treatment process

Redistribution to irrigation areas

Key integration points include:

Integration Point	Purpose
Circular water system sources	Operational data related to all circular aspects
Environment related sources	Forecasting
Soil conditions related sources	Irrigation optimization
Energy related sources	sustainability indicators

Through these integration points, data from the Santorini pilot will be analyzed to derive circularity and sustainability indicators describing the performance and evolution of the circular water recycling system.

4.3.5 Data specifications

Data categories

The Santorini pilot is expected to generate data related to:

- Rainwater harvesting and inflows, including collected rainfall volumes and inflow rates to the storage system
- Storage dynamics, including tank/collector water levels, storage volumes
- Water reuse and consumption, including overall reuse rates
- Greywater treatment performance, including wetland inflow/outflow balance
- Soil and irrigation monitoring data, including volumes supplied for irrigation and soil moisture conditions in irrigated zones
- Meteorological conditions, including temperature, humidity, pressure, and rainfall measurements
- Energy use, including electricity consumption of pumping and treatment processes

These datasets may be accessed and integrated into the VERNE platform data layer for circularity analysis mainly via an MQTT broker using a publish–subscribe model. The payload format is structured as JSON-encoded time-series data, containing timestamped sensor measurements with associated metadata (sensor ID, unit, location).

Data category	Data fields	Frequency	Granularity	Purpose
Circular practices	Reuse loop, reuse volume, rainwater capture	Real time	Continuous	Balance computation / Storage prediction
Waste related	Greywater inflow–outflow	Real time	Continuous	Treatment performance, irrigation optimization
Environmental	Meteorological conditions	Real time	Continuous	Context modelling
Energy	Energy consumption	Real time	Continuous	Energy-water nexus

4.3.6 Circularity indicators

Indicative circularity indicators for Santorini pilot may include:

- Energy efficiency in accommodation facilities
- % of Water reused for non-potable purposes
- Volume of water recollected or water import reduction

- Number of biodiversity enrichment
- % of visitor aware on local biodiversity conservation efforts
- % of revenue spent on sustainable products and services
- Investment in sustainable infrastructure and practices

4.3.7 VERNE platform digital tools

Santorini circular water performance explorer

A data enabled Digital Twin of the Santorini pilot is conceived, acting as a platform-level analytical application within the VERNE platform. It leverages real-time data streams originating from the pilot's monitoring infrastructure and operates as part of the VERNE ecosystem, ensuring:

- Cross-pilot harmonization of circularity
- Centralized governance of analytics models and data processing workflows
- Interoperability with service providers

It adopts a four-layer system architecture as follows:

1. Data acquisition layer
    - Sensors (flow, level, environmental, soil, energy)
2. Integration layer
    - Heterogeneous data ingestion
    - Time-series storage
3. Analytics layer
    - Flow balance computation
    - Circularity calculation
    - Predictive storage models
4. Visualization & interaction layer
    - System monitoring interface
    - Visitor interface

Santorini circularity KPI dashboard

A KPI dashboard that visualizes circularity indicators over time and supports communication of Santorini's progress toward circular goals (rainwater harvesting and autonomy forecast, greywater treatment performance, water reuse volumes and reuse rate, irrigation reuse performance, and energy efficiency of pumping/treatment processes).

4.4 Pilot 4 – Uusimaa (Virtual Nature-based Tourism)

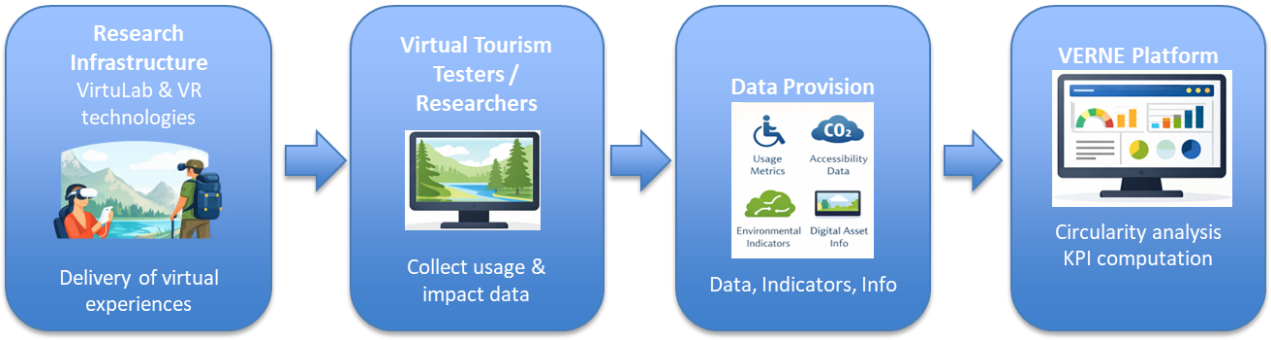
4.4.1 AS-IS situation

The Uusimaa pilot explores innovative approaches to sustainable tourism through the development of virtual nature-based tourism (NBT) experiences using the VirtuLab research infrastructure. This infrastructure enables the development, functionality testing, and assessment of market appeal of virtual nature-based tourism experiences, in collaboration with entrepreneurs, with the aim of early adopters (nature entrepreneurs, VR experience providers, tech sector developers, etc.) commercializing these products. Currently, virtual nature experiences are being piloted on a small scale in hotels and in social and healthcare sector services around the world. Developed immersive digital environments allow users to experience natural sites such as the Nuuksio national park without the need for physical travel. The Uusimaa pilot contributes to circular tourism by demonstrating how digital technologies can complement or substitute physical tourism activities, thereby reducing environmental pressure on natural areas and expanding accessibility. Through surveys and observation, the pilot can provide the following data:

- Usage statistics
- VR experiences offered
- Beneficiary target groups
- Information that allows the VERNE platform to estimate the GHG emissions avoided due to virtual experiences

The interaction between the Uusimaa pilot and the VERNE platform can be summarized as follows:

- The research infrastructure provides the technology to deliver the virtual experience.
- The researchers provide usage and impact data relevant for the Uusimaa Pilot (pilot testing phase).
- The VERNE platform interprets the data and demonstrates circularity performance.



Uusimaa pilot: Data flow to the VERNE platform

4.4.2 Digital infrastructure

The digital tools that exist in the context of Uusimaa pilot are described below:

VirtuLab, located in Viikki, Helsinki, is an immersive virtual nature research space that recreates the natural environment indoors through a 360-degree audiovisual experience. The space engages different senses through audiovisual content (using projectors and ambisonic sound system), scent diffusion, interactive walls with the ThingLink platform, VR headsets, and also offers the possibility for heart rate measurements. The environments are created by using a 360-degree camera and microphone. VirtuLab serves as a research platform to study how virtual nature experiences affect human psychophysiology, providing a model for future restorative virtual environments. VirtuLab also drives research and innovation on solutions that aim to make nature accessible to everyone, addressing barriers such as income, age, and mobility that can limit access to real outdoor settings. It can also be used as a testing environment for immersive content. Data collection in VirtuLab takes place through surveys built with Webropol or similar survey tools. Observation is also a possibility. Physiological data, when needed, can be collected by using Polar H10 heart rate sensors (currently not planned to be used in VERNE). ThingLink platform can store simple session metadata (e.g. number of tag selections) about the use of interactive content that are run through the ThingLink platform. However, in VERNE, ThingLink will likely not be used to a wide extent due to its limitations on content quality.

4.4.3 Pilot objectives & Use Cases

Strategic objective

The primary objective of the Uusimaa pilot is to explore innovative approaches to sustainable tourism through the development of virtual nature-based tourism (NBT) experiences. The pilot aims to:

- explore the usability and potential of virtual nature-based tourism in Uusimaa context (pilot testing phase),
- examine the environmental friendliness and inclusivity of digital tourism services,
- showcase improvements in accessibility and inclusiveness,
- provide data for the VERNE platform for estimating the ability of virtual experiences in reducing environmental impacts associated with physical travel
- study the appeal of virtual nature-based travel pilot product for customers across diverse markets.

Core Use Cases

UC-1: Virtual nature experience delivery

Provide multi-sensory immersive virtual nature experiences through either screen-based, VR headset-based, or immersive multi-sensory room-based experiences, allowing users to experience natural environments remotely. Users can remotely experience representations of natural sites in the Uusimaa region (e.g., national parks) through interactive, multi-sensory, and immersive virtual environments. These experiences enable engagement with nature with lesser need for physical travel.

UC-2: Usage and accessibility monitoring
Collect usage and accessibility-related data from users interacting with the virtual tourism experience. The researchers record information such as number of user sessions, session duration, potentially interaction patterns within the experience (not decided yet), and accessibility-related indicators (to be decided) by using suitable tools, such as surveys. These data allow the assessment of the accessibility and inclusiveness of virtual tourism services.

UC-3: Environmental impact reduction assessment
Provide data (pilot testing phase) for the VERNE platform to evaluate the environmental benefits of virtual tourism compared with conventional physical travel. *Note 1.* We are not offering experiences to the public - but are cyclically developing and refining use cases which will not generate reliable or ongoing data on environmental impact reduction (e.g. GHG emissions avoided). *Note 2.* Product market phase evaluation is not feasible as the data is collected only on the pilot testing phase. *Note 3.* Constraints with marketing testing mean that assessed environmental impacts reductions will not be reflective of commercial use cases.

4.4.4 Architecture mapping & integration points

Virtual nature-based tourism ecosystem architecture
The Uusimaa pilot architecture is centred around a virtual nature-based tourism ecosystem, where circularity is pursued through dematerialization of tourism services and reduced dependence on physical travel, while improving accessibility and inclusiveness. The system architecture consists of the following subsystems:

- Nature experiences content creation: VirtuLab research infrastructure and audio-visual technologies are used to capture, produce, and curate virtual nature experiences representing natural environments.
- Virtual tourism service delivery through which users access the virtual nature experiences.
- Sustainability and inclusion mechanisms: Practices enabling substitution to physical visits, reduction of travel-related impacts, and improved accessibility for groups unable or unwilling to travel on-site.
- Research-based evaluation and evidence collection: Collection of pilot evidence primarily through researcher-led methods (e.g., surveys) to document user experience, perceived benefits, and sustainability-related outcomes.

Key integration points include:

Integration point	**Purpose**
VirtuLab virtual experience	Evidence generated through the pilot's virtual nature experiences

Integration point	Purpose
Research surveys and assessment instruments	Survey results and evaluation outputs supporting circularity and sustainability indicators

Through these integration points, information from the Uusimaa pilot will be analyzed to derive circularity and sustainability indicators describing the performance and evolution of the virtual nature-based tourism approach.

4.4.5 Data specifications

Data categories

The Uusimaa pilot is expected to generate data related to:

- Virtual experience usage
- Accessibility and inclusiveness
- Environmental impact indicators

These datasets may be accessed and integrated into the VERNE platform data layer for circularity analysis via periodic (intermittent) structured exports. The payload format for the exports is expected to be CSV/Excel.

Data category	Data fields	Frequency	Granularity	Purpose
Virtual experience reuse ratio	Number of sessions per unique virtual experience	Semestral	Aggregated	Quantify reuse of created virtual nature experiences
Avoided travel emissions	Estimated CO_2e avoided	Semestral	Aggregated	Estimate environmental benefit of virtual experiences compared to physical travel
Beneficiary target groups	Share of sessions from users unable or unwilling to travel	Semestral	Aggregated	Demonstrate accessibility and inclusiveness

4.4.6 Circularity indicators

Indicative circularity indicators for Uusimaa pilot may include:

- Virtual experience reuse ratio, giving the number of user sessions per virtual experience created (i.e., visitation in virtual experience).
- Categories of beneficiary target groups experiencing constraints to nature access accommodated through virtual experience.
- Estimated CO_2/GHG emissions avoided by substituting physical travel from the pop-up site to the corresponding destination with the virtual tour.

4.4.7 VERNE platform digital tools

Uusimaa circularity KPI dashboard
A KPI dashboard that visualizes circularity indicators and supports communication of Uusimaa's progress toward circular goals (virtual experience reuse ratio, estimated avoided travel emissions (CO_2/GHG), and beneficiary groups reach/accessibility indicators).

4.5 Pilot 5 – Pomurje (Regional ZeroNET Food System)

4.5.1 AS-IS situation

The Pomurje pilot is built around an emerging regional circular food ecosystem, whose objective is to minimize food loss and waste while supporting sustainable tourism and local food production. The system integrates producers, food processors, hospitality businesses, and other stakeholders into a collaborative network designed to maintain food resources within the supply chain for as long as possible. The main operational components which structure the current ecosystem, are:

- The zero-food loss and waste (0FLW) circular supply network
- The sustainable distribution of local food products

Together, these components aim to transform the regional food system into a circular and low-emission food value chain. The Zero Food Loss and Waste (0FLW) network component focuses on preventing food waste through the return and reuse of surplus food products within the supply chain. The initiative builds the ZeroNET community, where food system actors collaborate to redirect surplus food back into productive use. Within this circular model, food products that would otherwise be discarded are reintroduced into the value chain and utilized in several ways, such as upcycling into processed food products, use as animal feed and utilization as feedstock for biogas production. This mechanism ensures that food resources are recovered and valorized, significantly reducing food waste and environmental impact. The

second component of the ecosystem focuses on distribution and logistics of local food products, connecting producers with consumers across the region.

This distribution network enables the delivery of locally produced food products to tourism businesses and other consumers, the collection of surplus food products for redistribution or reuse and the integration of local food supply into the regional tourism economy. The logistics system builds experience gained through the POTI project, which demonstrated the feasibility of local food ordering and delivery services for tourists. Within this system, tourists and residents can order local food products through a digital marketplace, with delivery directly to their place of residence.

A key objective of the initiative is to transition the food distribution system toward zero-emission logistics. This includes the deployment of electric delivery vehicles, testing new delivery services, and optimizing transport routes for food distribution and surplus collection. These measures aim to support the development of a GHG-free regional food logistics system, reducing the environmental footprint of food transport while improving supply chain efficiency.

4.5.2 Digital infrastructure

The Pomurje ecosystem is currently supported with the four main pillars or software solutions:

- Green Point Food Marketplace
- 0FLW Digital Marketplace
- Food Trace
- DADS – DIH Agrifood Data Space

Green Point Food Marketplace

The green point food marketplace enables the distribution and commercialization of locally produced food within the regional short food supply chain. Through this platform, local producers offer their products, while tourism actors and other consumers place orders and receive deliveries. The marketplace supports the integration of local food into the regional tourism ecosystem and complements the ERP system by providing a user-facing interface for ordering and distribution.

Food Trace

The ecosystem also incorporates a blockchain-based traceability system, which records product flows across the supply chain from the field to the shelves in our green point store. This infrastructure enables transparent tracking of product origin, movement, and redistribution within the circular food ecosystem.

0FLW Digital Marketplace

0FLW digital marketplace is a cloud-based platform where various actors can offer surplus food back to the supply chain. Actors participating in this ecosystem include farmers and food producers, food processors, tourism businesses (hotels, restaurants, etc), and public organizations. Through the marketplace, surplus food can be matched with actors capable of

reusing or processing it. This marketplace plays a key role in closing the circular loop by enabling redistribution of surplus food and reducing food waste.

DADS – DIH Agrifood Data Space

DADS represents the data sharing and integration layer testing and is one of the main software pillars of the DIH agrifood ecosystem of agrifood stakeholders. It provides interfaces for data transactions between the Pomurje pilot and other VERNE ecosystem services and applications. DADS enables secure and controlled data exchange between stakeholders through API-based access, identity management, and consent-based data sharing.

A central part of the platform is the custom Consent Management Platform, which enables data providers to manage permissions for sharing datasets and allows data users to request access. It records the purpose, duration, and approval status of each consent, audits data sharing actions and manages API keys linked to specific permissions. Together with the API Market service, DADS lets data providers publish datasets as secure APIs, while data users can discover them through a developer portal, read documentation, test endpoints, and integrate the data into their own applications. Through the DADS API Market, marketplaces, and traceability systems are exposed and integrated into the VERNE platform. The platform follows a data space approach, where data remains under the control of the provider and is shared in a decentralized and interoperable manner.

Overall system integration

Together, these systems form the digital backbone of the Pomurje circular food ecosystem. Green Point Marketplace enables food distribution, the 0FLW marketplace enables surplus redistribution, the blockchain system ensures traceability, and DADS enables secure data sharing and integration with the VERNE platform.

These systems provide the data sources required for monitoring circular food flows and evaluating circularity performance through the VERNE platform.

4.5.3 Pilot objectives & Use Cases

Strategic objective

The primary objective of the Pomurje pilot is to support and optimize the circular operation of the local food ecosystem by improving visibility, efficiency, and sustainability across the entire food distribution loop. The pilot aims to:

- reduce food waste through surplus redistribution
- support circular economic flows between food producers and consumers
- improve logistical efficiency in food distribution and return operations
- generate measurable indicators demonstrating circular food economy performance

Through the integration of digital technologies and data analytics, the pilot aims to transform the current logistics and redistribution process into a data-driven circular food management system.

Core Use Cases

UC-1: Circular food distribution monitoring
Monitor the distribution of food products across the ecosystem, including flows from suppliers to consumers. This use case enables:

- monitoring of product flows across the food ecosystem
- tracking of supplier–consumer transactions
- visibility into distribution volumes across the network of suppliers and consumers

UC-2: Surplus food return and redistribution
Track surplus food returns from consumers back to Green Point and their redistribution through the surplus marketplace. This use case supports:

- quantification of returned surplus goods
- monitoring of redistribution through secondary markets
- reduction of food waste through circular reuse of surplus food resources

UC-3: Circularity performance verification
Generate measurable circularity indicators describing the performance of the food circulation loop. Examples of indicators include:

- volume of food distributed through the system
- volume of surplus food returned
- share of redistributed surplus food
- estimated reduction of food waste

UC-4: Logistics optimization
Improve the efficiency of transport operations connecting suppliers, the Green Point hub, and consumers.

- improved vehicle routing strategies
- reduced transport distances and fuel consumption
- improved operational efficiency of the logistics fleet

4.5.4 Architecture mapping & integration points

Circular food distribution system architecture
The Pomurje pilot architecture is centered around a circular logistics and redistribution system, structured around the Green Point as the central coordination hub. The system architecture consists of the following subsystems:
1. Food supply subsystem

- primary food suppliers
- processed food suppliers
- agricultural producers
- food processing industries

2. Distribution and consumption subsystem
Retail consumers within the ecosystem include:

- hotels
- restaurants
- hospitals
- other food service providers

3. Surplus return subsystem

Surplus food products that remain unused by consumers are returned through the logistics loop to the Green Point, where they are reintroduced into the ecosystem through the surplus food marketplace.

4. Digital data layer

Operational data within the Pomurje pilot are generated through the following digital systems:

- Green Point Food Marketplace
- 0FLW (Zero Food Loss and Waste) digital marketplace
- blockchain-based traceability infrastructure

Together, these systems capture transactional, operational, marketplace, and traceability of information related to the functioning of the circular food ecosystem. The DADS layer enables secure exposure, governance, and sharing of these datasets across the VERNE ecosystem. These data streams constitute the primary digital data sources that can be made available to the VERNE platform for monitoring food circulation and computing circularity indicators.

The data streams are exposed via APIs and made available through the DADS API Market.

Key integration points include:

The Pomurje pilot will provide operational data to the VERNE platform to support the analysis and monitoring of circular food economy activities within the regional food ecosystem. Data generated by the Green Point Marketplace, 0FLW marketplace, and blockchain traceability system are integrated into the VERNE platform through APIs exposed via the DADS API Market as depicted in the following Figure 15.

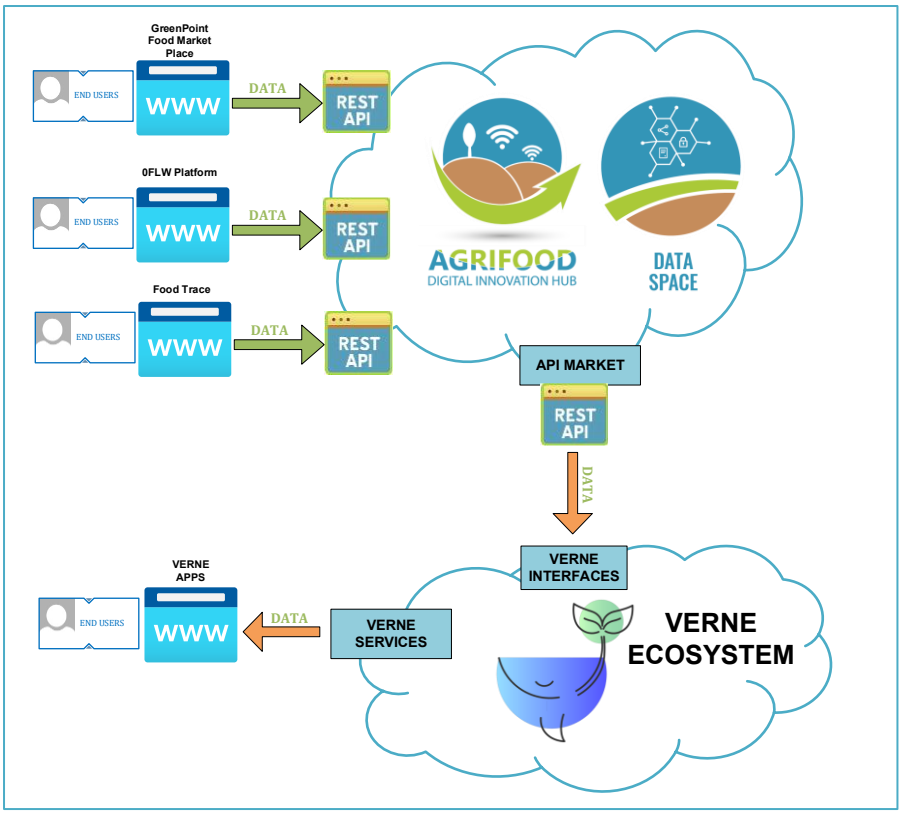


Figure 15. ZeroNET Food System Data integration with VERNE ecosystem

4.5.5 Data specifications

Data categories

The Pomurje pilot generates operational and transactional data through the following digital infrastructures:

- Green Point Food Marketplace, which manages commercial transactions and logistics operations within the regional food ecosystem
- the 0FLW (Zero Food Loss and Waste) digital marketplace, which enables redistribution of surplus food
- the blockchain-based traceability system, which records the provenance and circulation of food products across the supply chain

Marketplaces provide interaction and transaction-level data, and the blockchain system provides traceability data. These systems generate structured datasets that can be accessed via APIs and integrated into the VERNE data layer through the DADS infrastructure.

Data exchange shall follow a pull-based integration model, where the VERNE data ingestion services retrieve structured datasets from the available APIs. The APIs are exposed through the DADS API Market, which enables standardized, secure, and permission-based access to datasets. The data payload format is expected to be structured in JSON format, containing time-stamped records describing transactions, product flows, and supply chain interactions. Data

exchange follows a data space approach, ensuring that data remains under the control of the provider while being shared through interoperable interfaces.

Data Category	Data Fields	Frequency	Granularity	Purpose
Supplier data	supplier type, product category	On change / periodic sync	Per supplier record	Identification of food producers and suppliers
Consumer data	business type (hotel, restaurant, hospital)	On change / periodic sync	Per consumer entity	Identification of retail actors participating in the circular ecosystem
Product data	Product ID, product category, origin	On transaction	Per product batch	Tracking of food product types and origins
Distribution transactions	Quantity distributed	daily	Per delivery transaction	Monitoring food flows across the ecosystem
Surplus return transactions	Quantity returned	daily	Per return transaction	Monitoring circular return flows
Order time series	Order volumes	Daily	Per consumer / time	Demand pattern analysis
Price data	Product price, surplus marketplace price	On transaction	Per product / transaction	Economic analysis of the circular food market
Logistics operations	Delivery events	Daily	Per delivery route	Analysis of transport and distribution operations

4.5.6 Circularity indicators

Indicative circularity indicators for Pomurje pilot may include:

- Kg of carbon footprint reduction from food distribution

- Tons and % of food waste reduction
- Tons and % of upcycled food surplus into new products
- % of electric vehicles used for food distribution
- % of tourists aware on food loss and waste (0FLW) platform
- % of local farmers, producers and distributors involved in circular food tourism initiatives
- Amount of tourist spending on local food products
- Amount of revenue from new surplus food products
- % of circular food initiatives integrated in blockchain technology

4.5.7 VERNE platform digital tools

Circular food flow visualizer

The Circular food flow visualizer is a platform-level digital tool to be implemented within the VERNE platform that illustrates the circular movement of food in the Pomurje ecosystem. It presents interactive diagrams of the supply chain (Suppliers → Green Point → Consumers → Surplus Return) and leverages data from the Food Market Place, Traceability and the 0FLW marketplace to visualize distribution and return flows while displaying key circularity indicators, including surplus recovery rate, volume of redistributed food, number of participating actors, and estimated food waste avoided.

VERNE marketplace

The VERNE marketplace is a platform-integrated digital tool that enables stakeholders to actively participate in the circular food ecosystem demonstrated by the Pomurje pilot. Building on the insights provided by the circular food flow visualizer, users such as hotel managers, restaurants, and other tourism actors can browse and order locally produced food products available within the Pomurje ecosystem through the VERNE marketplace. Participants engaging in circular purchasing activities via the platform may also benefit from incentive mechanisms (e.g., tokens or similar reward schemes) designed to recognize and encourage contributions to circular economy practices. The proposed workflow called "Order Local Food → Return Surplus → View Circularity Impact" is as follows:

Browse & Order (UC1 & UC4)

Customer logs into the VERNE marketplace → selects Pomurje pilot → browses available local products (via Food Market Place) → places order → system optimizes EV route from Green Point.

Receive Delivery (UC1)

Delivery arrives; quantity and transaction recorded in Food Market Place.

Identify & Return Surplus (UC2 & UC4)

End of day, customer marks unused items as surplus → lists them on 0FLW marketplace → returns via same EV route.

Redistribution & Traceability (UC2)

Surplus is matched and redistributed (upcycled / animal feed / biogas) → blockchain records every step.

View Impact Dashboard (UC3)
Circularity indicators are updated: surplus redistribution rate, food waste avoided, CO_2 saved.
Repeat & Monitor (all UCs)
Daily/weekly loop with automated alerts when redistribution rate drops below 70 %.

4.6 VERNE Marketplace

The VERNE Marketplace (Task 3.5) is envisaged as an **umbrella platform** (subdomain of existing Accessible circular solutions for sustainable tourism - VERNE) that complements and extends pilots' existing digital tools by enabling trusted transactions, traceable evidence of circular practices, and incentive mechanisms to encourage sustainable choices. The Marketplace is designed to support the publication of pilot offerings (e.g., food experiences), transaction handling, post-transaction verification, and the derivation of KPI-relevant evidence for monitoring and reporting. Key integrations include:

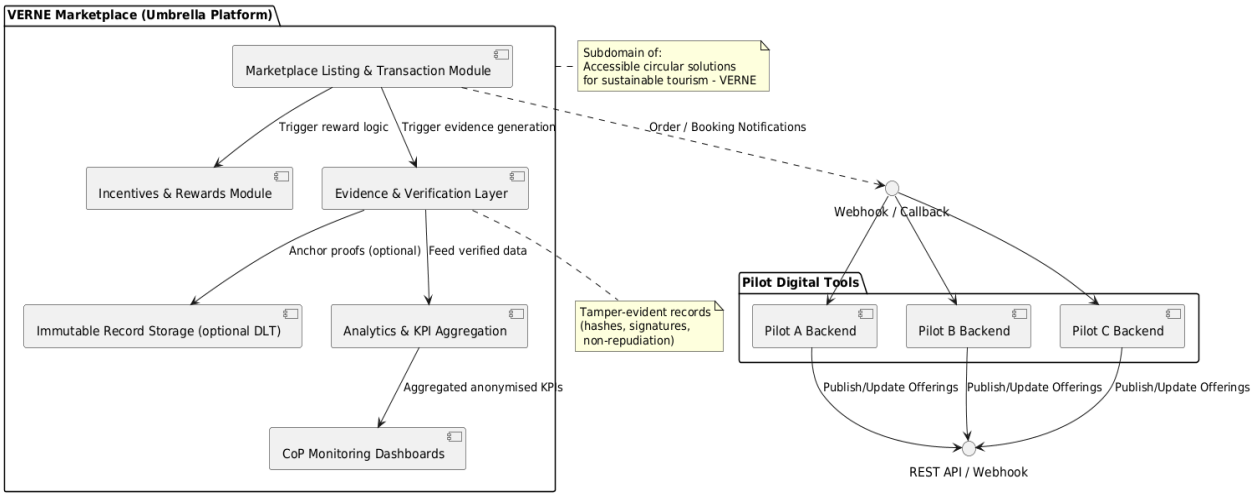
- Marketplace listing and transaction module: central cataloguing and purchasing workflow, with pilots providing offerings via standard interfaces (e.g., REST APIs or Webhook).
- Incentives and rewards module: allocation of points/tokens or equivalent rewards based on verified sustainable actions and outcomes, supporting motivation and engagement.
- Evidence and verification layer: generation of tamper-evident "evidence records" linked to transactions (e.g., hashes/signatures), enabling non-repudiation and traceability of sustainability claims.
- Immutable record storage (optional): anchoring of evidence proofs to a distributed ledger or equivalent immutable store, subject to final design decisions in Task 3.5.
- Analytics and CoP monitoring dashboards: aggregation of verified, anonymised KPI-relevant statistics (e.g., waste reduced, CO_2 saved per category) to support Communities of Practice discussions and iterative improvements.

4.6.1 Integration Flow

The VERNE Marketplace acts as an umbrella platform that integrates with each pilot's existing digital tools while preserving their autonomy. It provides standardized interfaces for publishing offerings, handling transactions, generating verifiable evidence of circular practices, allocating incentives, and feeding anonymized data into monitoring dashboards.

The integration follows a loosely coupled, event-driven architecture that minimizes disruption to pilots. Key principles include:

- Standardized interfaces (REST APIs for synchronous operations + Webhooks for asynchronous notifications)
- Separation of concerns — pilots manage their own inventory and user experience; the Marketplace handles cross-pilot cataloguing, payments/trust, verification, and incentive logic
- Tamper-evident traceability — every sustainable transaction generates cryptographic evidence
- Privacy-first design — only anonymized, aggregated data flows to CoP dashboards and KPI reporting



4.6.2 VERNE Marketplace – Technical Viewpoints

The Verne Marketplace is built on a modern, scalable and secure WordPress-based stack. The architecture follows a classic three-tier model (Presentation – Application – Data) with additional layers for security, backups and gamification.

Table 16: WordPress-based stack

Component	Technology / Plugin	Purpose Key	Benefits
CMS	WordPress	Core content management and extensibility	Flexible, open-source, large community
E-Commerce Core	WooCommerce	Product catalogue, cart, checkout, orders	Industry standard, thousands of extensions
Multi-Vendor	Dokan	Vendor dashboards, commissions, store management	Full frontend vendor control, commission system
Gamification	myCred	Points, badges, rewards for circular behaviors	Encourages sustainable actions
Security	Wordfence	Firewall, malware scan, login protection, 2FA	Real-time threat detection
Component	Technology / Plugin	Purpose Key	Benefits
Backups & Recovery	UpdraftPlus	Automated encrypted backups and restoration	Easy recovery, cloud storage support
Email Delivery	WP Mail SMTP	Reliable transactional and notification emails	Avoids spam folders, better deliverability
Database	MySQL	Storage of all content, users, orders, points	Reliable and scalable

4.6.2.1 Interoperability, REST API and Webhooks

The Verne Marketplace is designed with strong interoperability in mind, leveraging the native WooCommerce REST API and Webhooks capabilities.

WooCommerce REST API

The platform exposes a comprehensive RESTful API that allows external applications to read and write data related to:

- Products and services (including vendor-specific products managed via Dokan)
- Orders and order status updates
- Customers and user profiles
- Coupons, reports, and analytics
- Vendor information and earnings (extended through Dokan compatibility)

Webhooks

WooCommerce Webhooks provide real-time event-driven notifications. Configured webhooks automatically trigger HTTP POST requests to predefined endpoints whenever specific events occur, such as:

- order.created / order.updated / order.completed
- product.created / product.updated

· customer.created

- Vendor-specific events (supported via Dokan extensions)
- These webhooks enable asynchronous integration without constant polling, ensuring:
- Immediate notification of new orders to vendors
- Real-time update of myCred reward points
- Seamless data flow to the VERNE monitoring and impact assessment tools
- Easy connection with external CRM, accounting, or analytics systems

Together, the REST API and Webhooks ensure that the Verne Marketplace remains open, extensible, and future-proof, allowing smooth integration with other digital components of the VERNE project while maintaining high security standards (via Wordfence protection and proper **API authentication.**

4.6.2.2 Examples of Key WooCommerce Events and Use Cases

Table 17: Event Type & Related APIs

Event Type	Webhook / API	Endpoint Use Case in Verne Marketplace	Benefit
Order Created	order.created webhook	Notify vendor + award myCred points	Real-time processing

Event Type	Webhook / API	Endpoint Use Case in Verne Marketplace	Benefit
Order Completed	order.updated / API	Update earnings in Dokan + feed VERNE KPIs	Accurate impact measurement
Product Created/Updated	product.created webhook	Sync with VERNE Solutions Catalog	Centralized knowledge base
Customer Registration	REST API + webhook	Welcome bonus points via myCred	User engagement

4.6.2.2.1 Use Case: Vendor Sales Cycle

The following scenario describes the lifecycle of a transaction from the Vendor's perspective within the VERNE Marketplace, an ecosystem based exclusively on a points economy with zero commission fees.

1. **Secure Access (Wordfence):** The vendor logs into the platform. Wordfence checks their IP in the background, protecting the account from malicious attacks (e.g., bots).

2. **Store Management (Dokan):** Upon successful login, the vendor accesses the Dokan Vendor Dashboard. This is an isolated environment for managing their products, orders, and digital wallet.

3. **Product Listing (WooCommerce & myCred):** The vendor adds new products. Pricing is set exclusively in points through the seamless integration of myCred with WooCommerce.

4. **Order Receipt & Notification (WP Mail SMTP):** As soon as a customer makes a purchase using points, the system generates the order. An instant notification is sent to the vendor, reliably bypassing spam filters thanks to WP Mail SMTP.

5. **Fulfilment & Points Transfer (Zero-Commission):** The vendor fulfils the order. The system automatically credits of the points value to their digital wallet, as the platform retains zero commission.

6. **Automated Backup (UpdraftPlus):** All new data (products, orders, point balances) is automatically backed up to a secure, off-site server at the end of the day to ensure business continuity.

Advanced Interoperability (API & Webhooks) For vendors utilizing external software, the platform supports full automation:

· **REST API:** Bi-directional communication for bulk product imports, inventory updates, and point balance checks, without the need to interact with the marketplace UI.

· **Webhooks:** Real-time event notifications (e.g., pushing new order data directly to the vendor's warehouse management system the exact moment a purchase occurs).

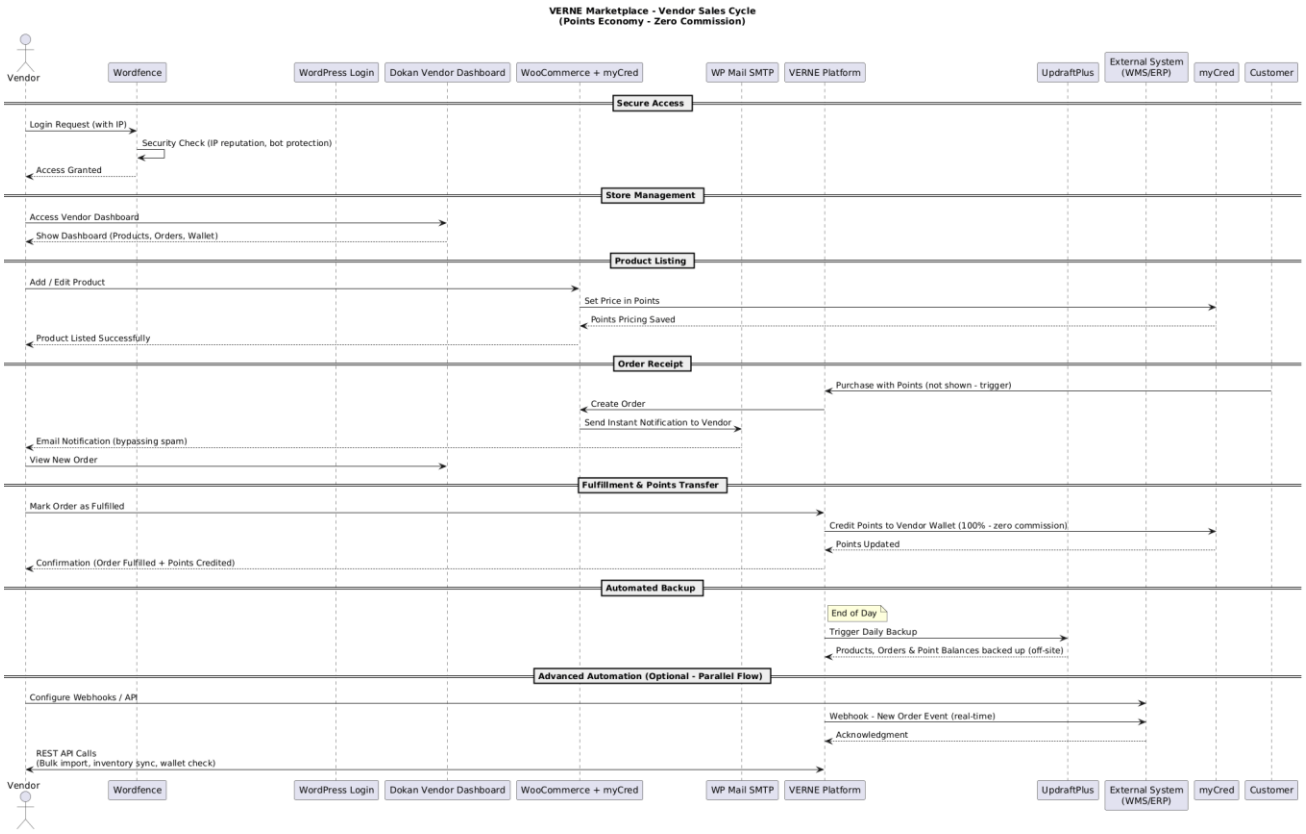


Figure 16: Vendor Sales Cycle

4.6.2.2.2 Use Case: Consumer Purchasing Cycle (Points Economy)

The following scenario describes the experience of a customer (Buyer) in the VERNE Marketplace, focusing on browsing and completing purchases using exclusively the platform's digital currency.

1. **Secure Browsing (Wordfence**): The buyer visits the platform. Wordfence protects their session and personal data in the background by blocking malicious traffic.

2. **Browsing & Cart (WooCommerce & Dokan):** The user browses products from various vendors and adds them to the centralized shopping cart. All product prices are displayed exclusively in points (Verne Credits).

3**. Checkout Process (myCred):** At checkout, no credit card is required. The buyer pays using the available points balance in their digital wallet (myCred). The system automatically deducts the required points.

4. **Instant Confirmation (WP Mail SMTP):** The system processes the order, and the buyer immediately receives a confirmation email with the details. WP Mail SMTP guarantees safe delivery to the Inbox, bypassing Spam filters.

5. **Loyalty Rewards (Gamification):** After receiving or reviewing the product, myCred can automatically credit a small percentage of points back to the user (reward/cashback), incentivizing future purchases.

6. **Data Protection (UpdraftPlus):** The user's order history and updated points balance are securely backed up daily via automated off-site backups.

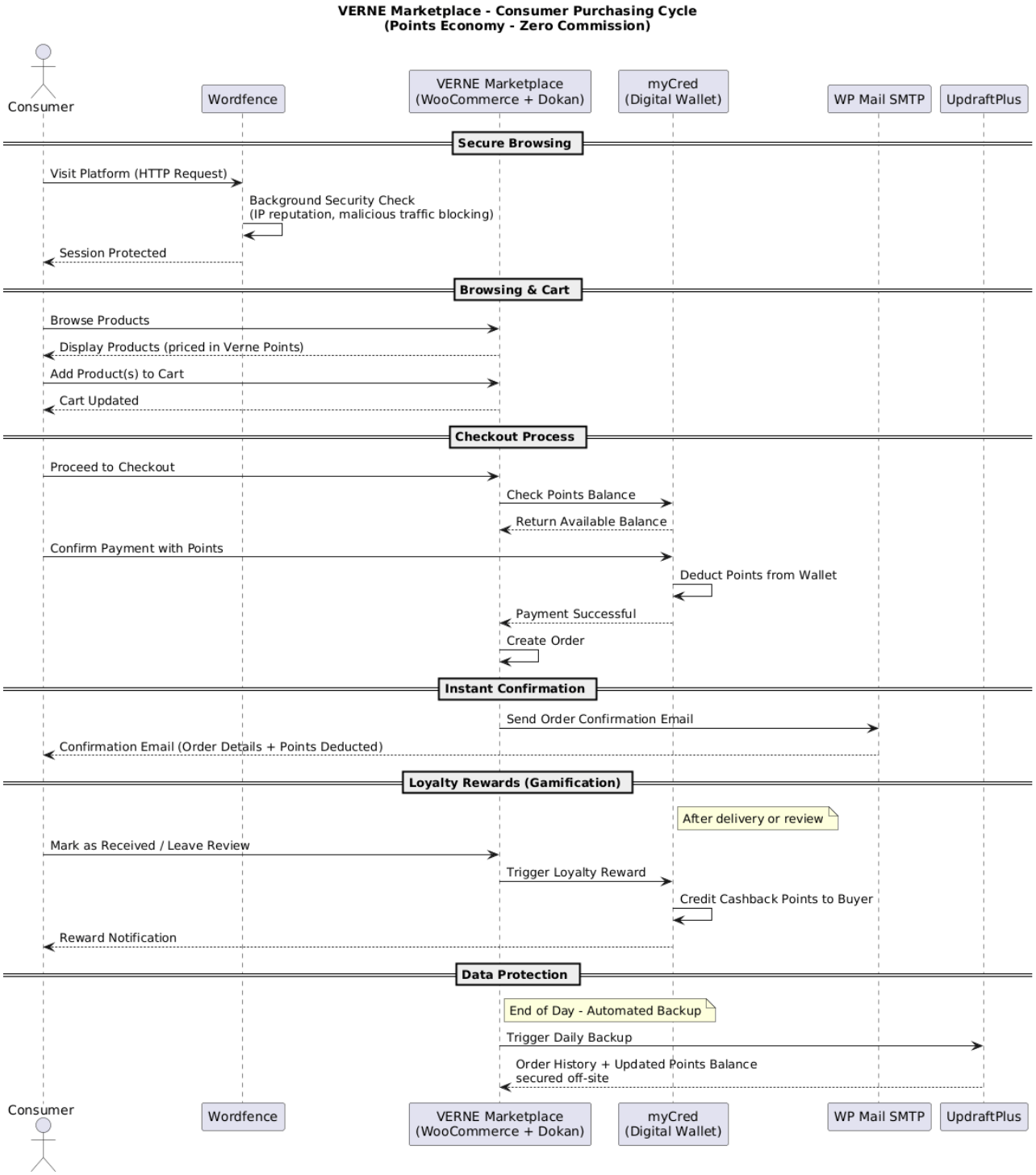


Figure 17 Consumer Purchasing Cycle (Points Economy)

4.6.2.2.3 Login/Register Screen

Login and Registration UI. Existing users access the marketplace through the left-side 'Login' panel, while new users join via the right-side 'Register' section. For registration, users can

select between 'Customer' or 'Vendor' roles (via Dokan integration), activating specific fields like 'Store Name'.

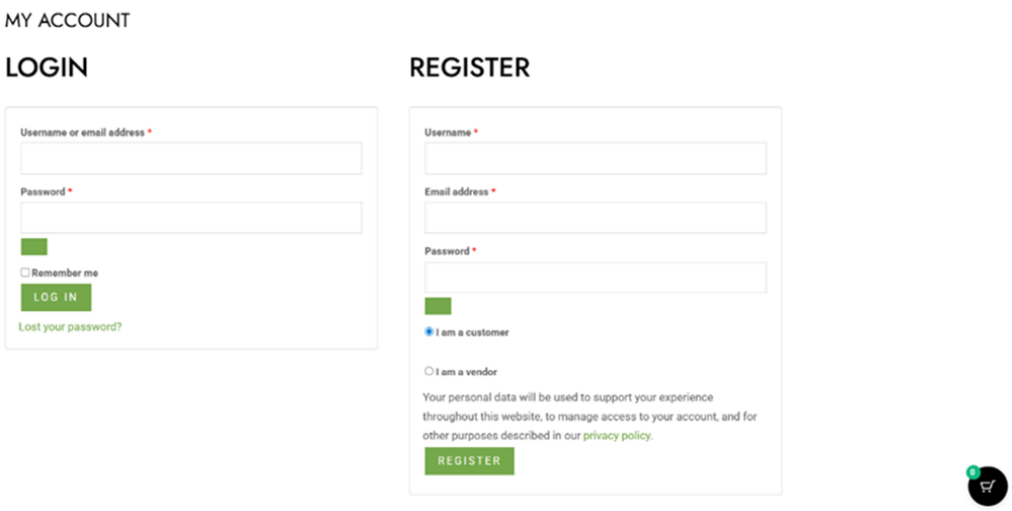


Figure 18: Register Screen

4.6.2.2.4 Vendor Environment (Vendor Dashboard)

This interface provides the vendor with a detailed, real-time view of their store's performance. The reports display daily sales trends, order volumes, and total earnings

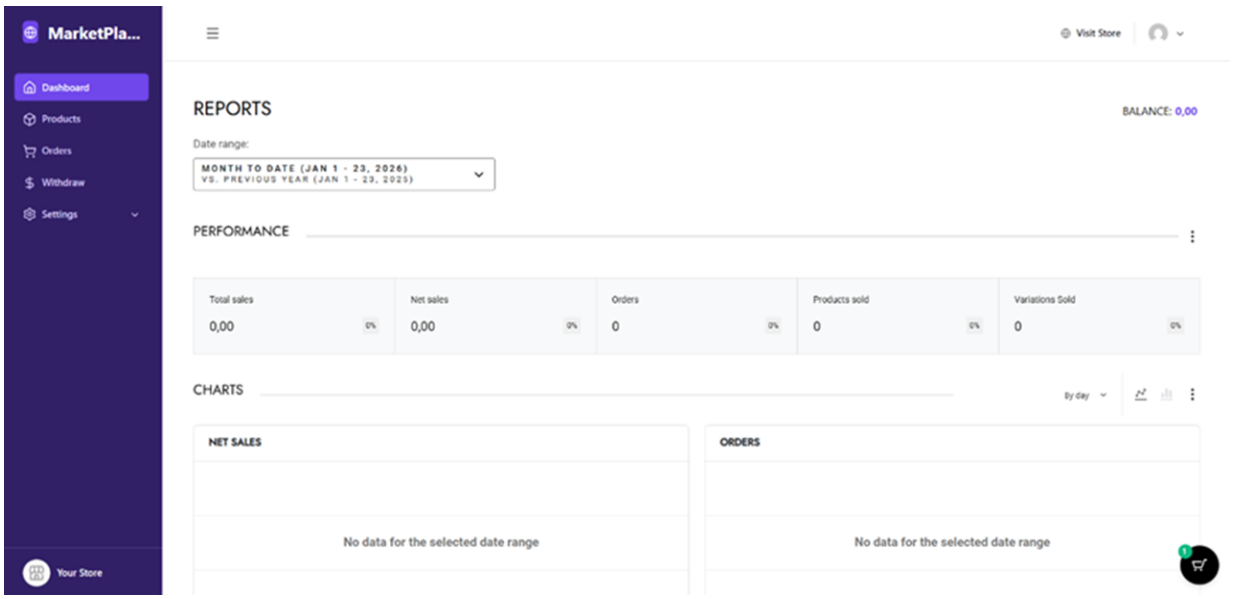


Figure 19: Vendor Dashboard

4.6.2.2.5 Vendor Dashboard – Add New Product.

Through this Dokan interface, vendors can seamlessly upload new offerings (titles, descriptions, and media). A critical feature of the VERNE ecosystem is highlighted here: the product pricing is configured exclusively in the platform's digital currency (Verne Credits) rather than traditional fiat currency, demonstrating the deep integration with the myCred system.

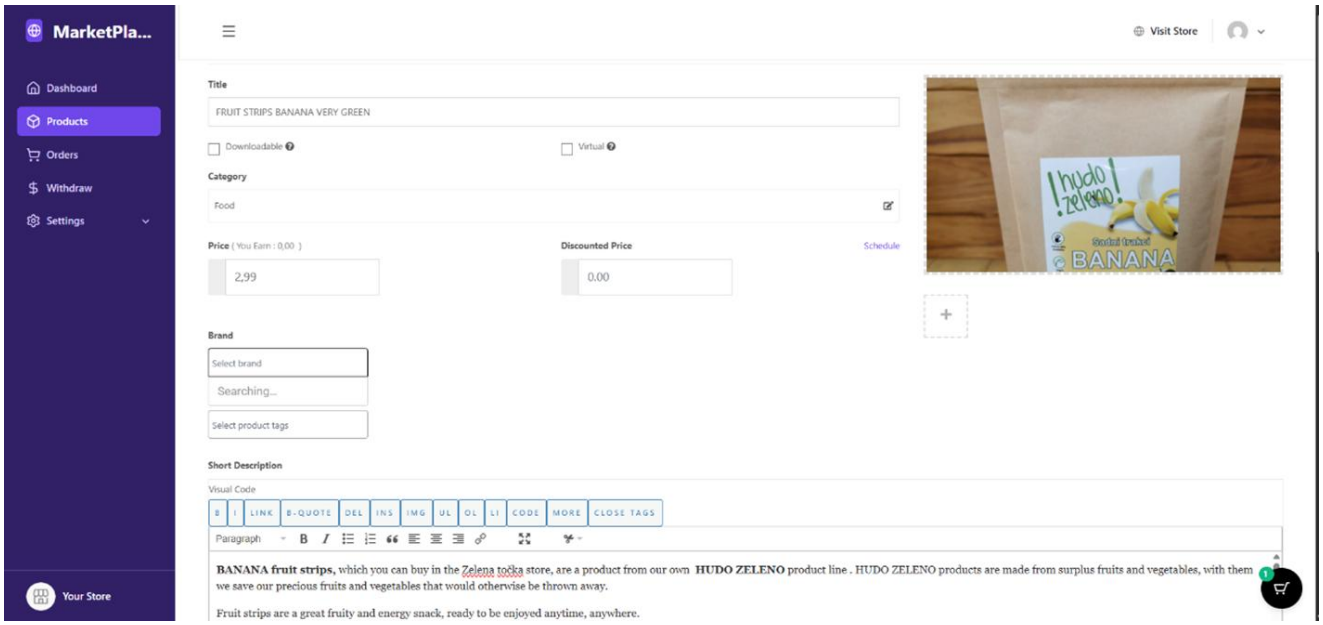


Figure 20: Add New Product

4.6.2.2.6 Vendor Dashboard –Product Management.

This interface provides a comprehensive overview of the vendor's inventory. Sellers can quickly view product status (online, pending, or draft), monitor stock levels, and perform quick edits. Consistent with the marketplace's economic model.

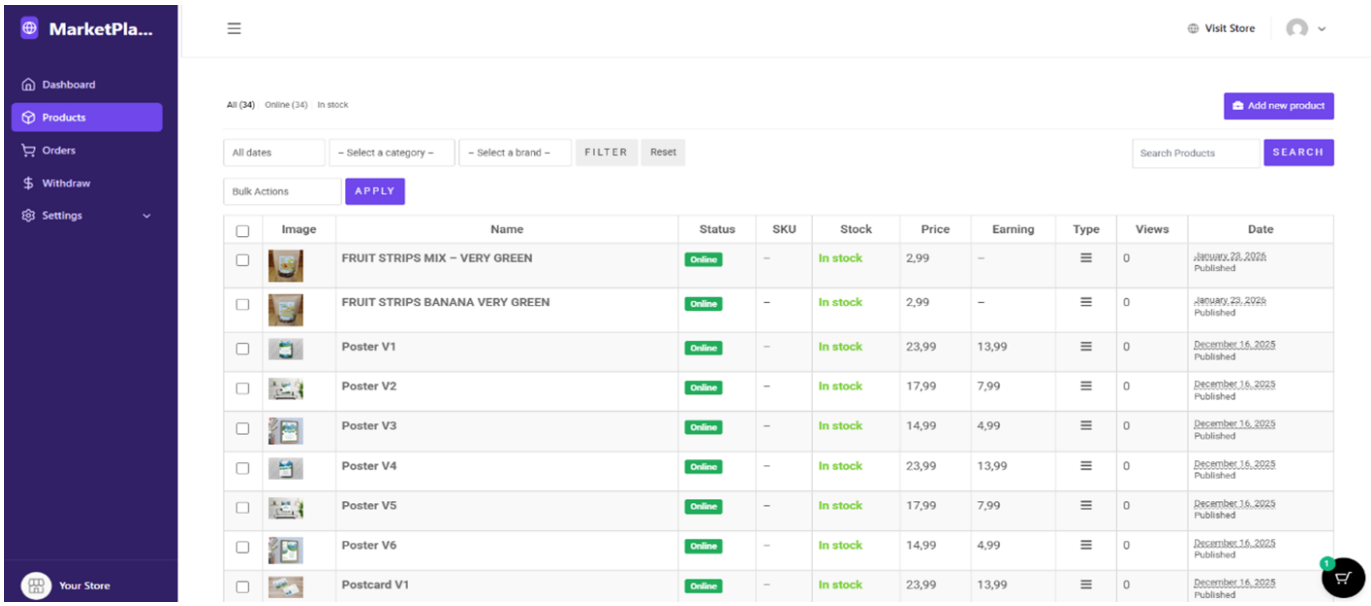


Figure 21: Product Management

4.6.2.2.7 Vendor Dashboard – Order View.

This screen allows the vendor to manage and fulfil an individual order. It displays the customer's shipping details, the items purchased, and the total transaction value paid entirely in Verne Credits. From this interface, the vendor can update the order status (e.g., from 'Processing' to 'Completed').

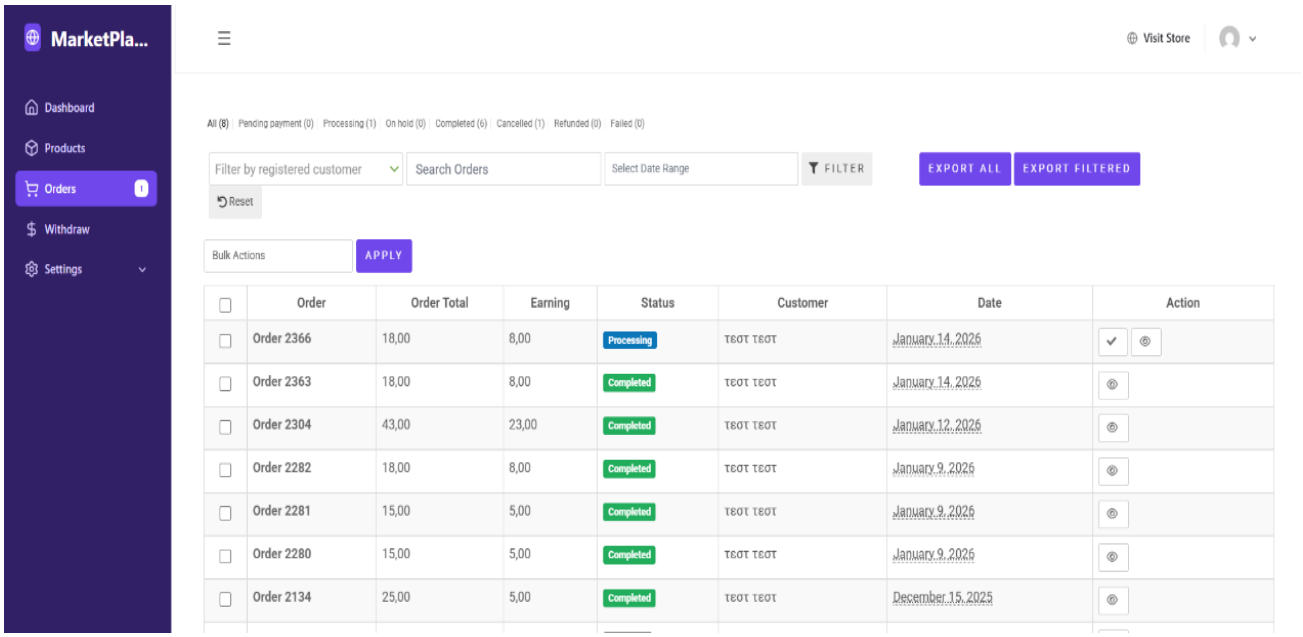


Figure 22: Order View

4.6.2.2.8 Consumer Environment: Product View & Add to Cart

This screen shows the detailed view of a product offered by a marketplace vendor. The price is prominently displayed in the native digital currency (Verne Credits) rather than traditional fiat currency. The consumer can review the product description, check vendor details, and click the 'Add to Cart' button to initiate the purchasing process within the points-based ecosystem.

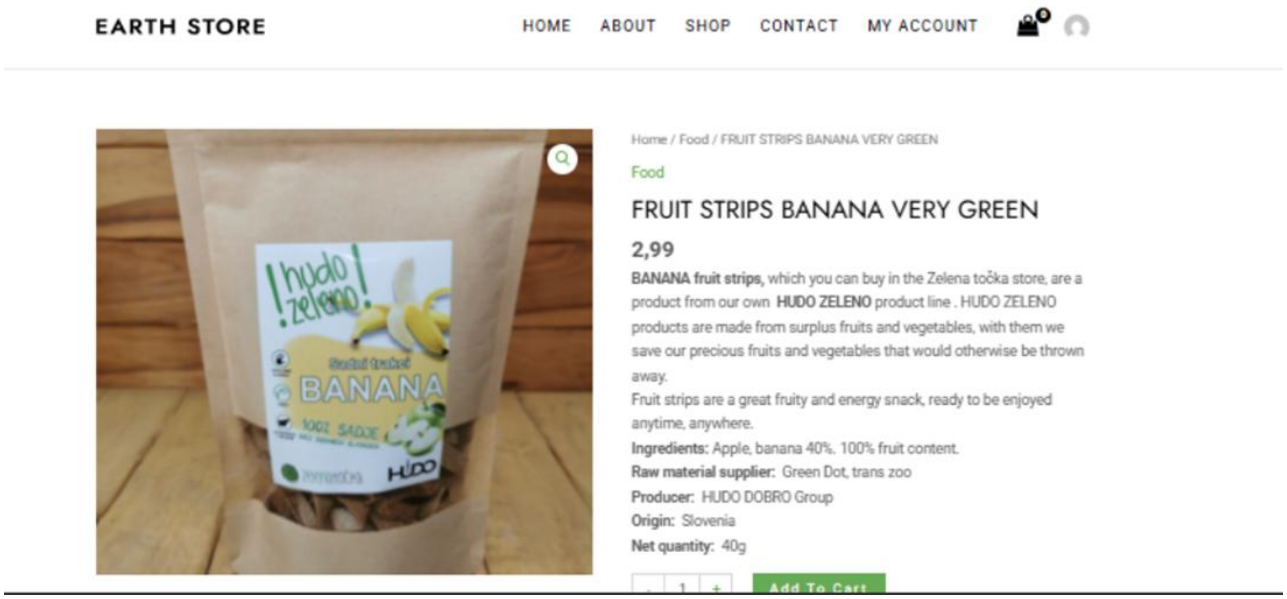


Figure 23: Product View & Add to Cart

4.6.2.2.9 Consumer Environment – Checkout and Order Confirmation.

This interface demonstrates the buyer's checkout process via WooCommerce. Instead of traditional payment gateways (like credit cards or PayPal), the user selects their digital wallet balance to complete the transaction. The order summary clearly displays the total cost exclusively in Verne Credits. Upon confirming the order, the specified points are instantly and securely deducted from the buyer's myCred account.

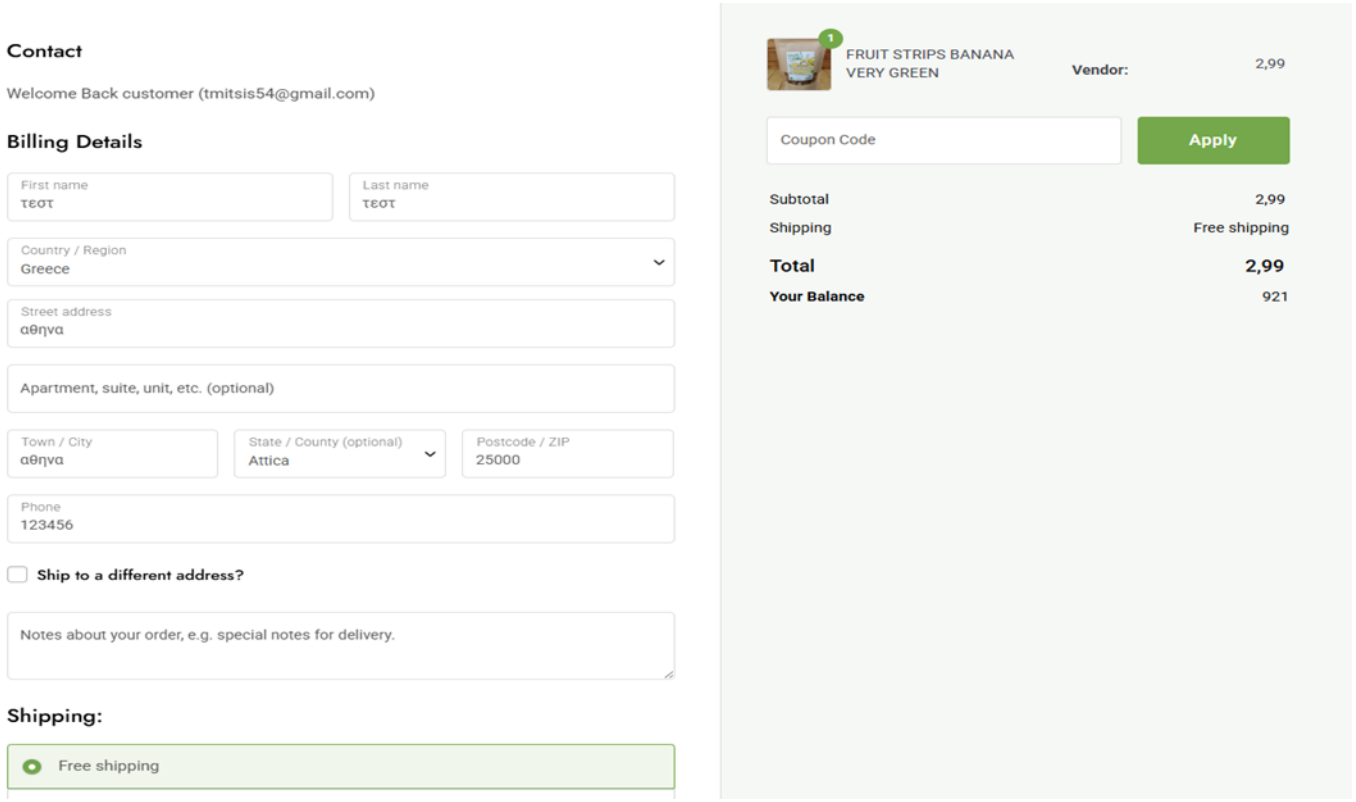


Figure 24: Checkout and Order Confirmation

4.6.2.2.10 Consumer Dashboard – Order History

This interface, powered by WooCommerce, allows the buyer to view their complete transaction history in one centralized location. Users can monitor the current status of their active orders (e.g., 'Processing') and review details of past 'Completed' purchases by clicking the 'View' action.

Figure 25: Order History

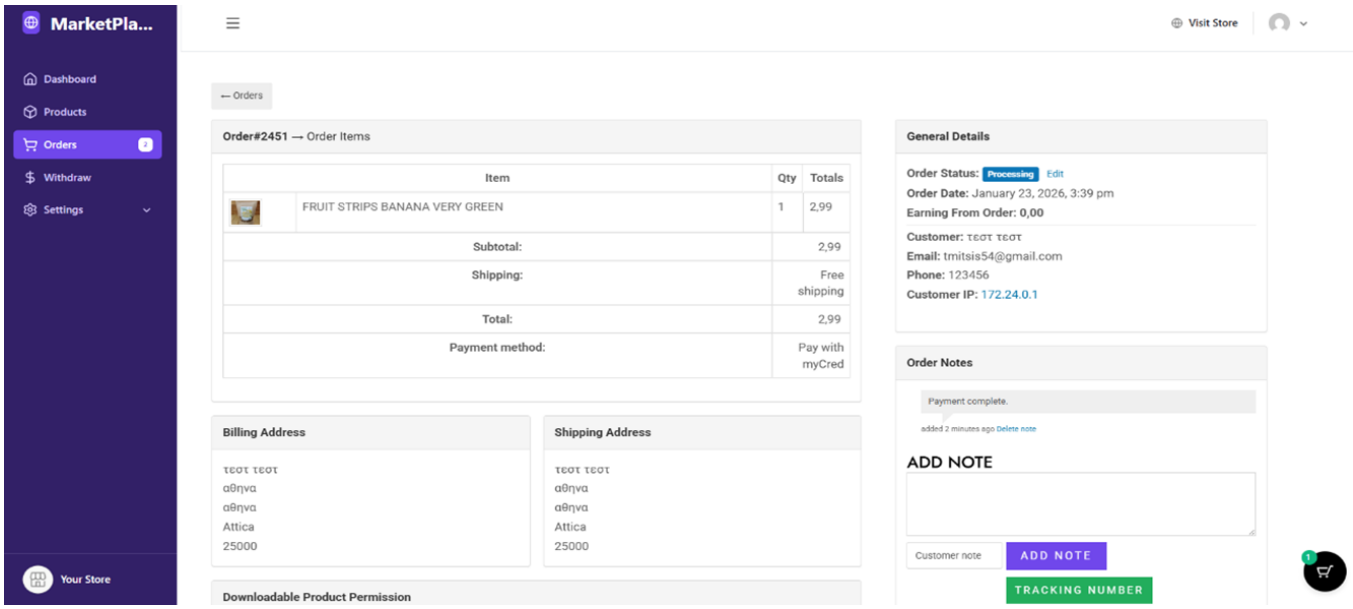


Figure 26: Order Details & Status

4.6.3 VERNE Trustworthiness Marketplace

The VERNE Trustworthiness Marketplace is designed as a hybrid system that combines off-chain efficiency (WooCommerce for UI/order management, MyCred for rewards, ERP webhooks for real-time execution as described on the previous sub-sections) with on-chain trust (Tokenisation API for signing and tokenisation, decentralized ledger for immutable Evidence Profiles). Core trust is achieved through a common evidence_id (traceable across all stages), off-chain SHA-256 hashes (terms, delivery, metrics, credits), and on-chain locking (append-only with Tokenisation API signatures). Security is ensured via HTTPS/TLS, HMAC-signed webhooks, field-level encryption (AES-256), and role-based access control (RBAC). This

solution architecture built on Pilot 5 (Regional ZeroNET Food System in Pomurje, Slovenia) by enabling verifiable zero-emission deliveries. The end-to-end user journey flows through **6** key stages:

1. **Discovery & Onboarding** → Users register, browse listings . Users also get access to the tokenisation service/API.
2. **Product Listing** → Vendors add products (UI/REST API) with metadata.
3. **Purchase & Pre-Commitment** → Consumers checkout, sign termsHash, generate evidence_id.
4. **Order Execution & Delivery** → Vendors process via ERP, calculate co2_savings, push webhook updates.
5. **Integrity Lock & Verification** → Vendor signs/submits Evidence Profile to ledger; Consumer verifies hashes client-side → "Green Light".
6. **Rewards & Redemption** → Points awarded (MyCred); users redeem for discounts, badges, or CoP perks.

This journey ensures trustworthy earnings workflows (ordering → delivery → verification → redemption), with the common evidence_id as the traceability backbone, Tokenisation API for integrity and tokenisation, and CoP integration for social legitimacy — delivering a transparent, scalable, GDPR-compliant solution for circular tourism.

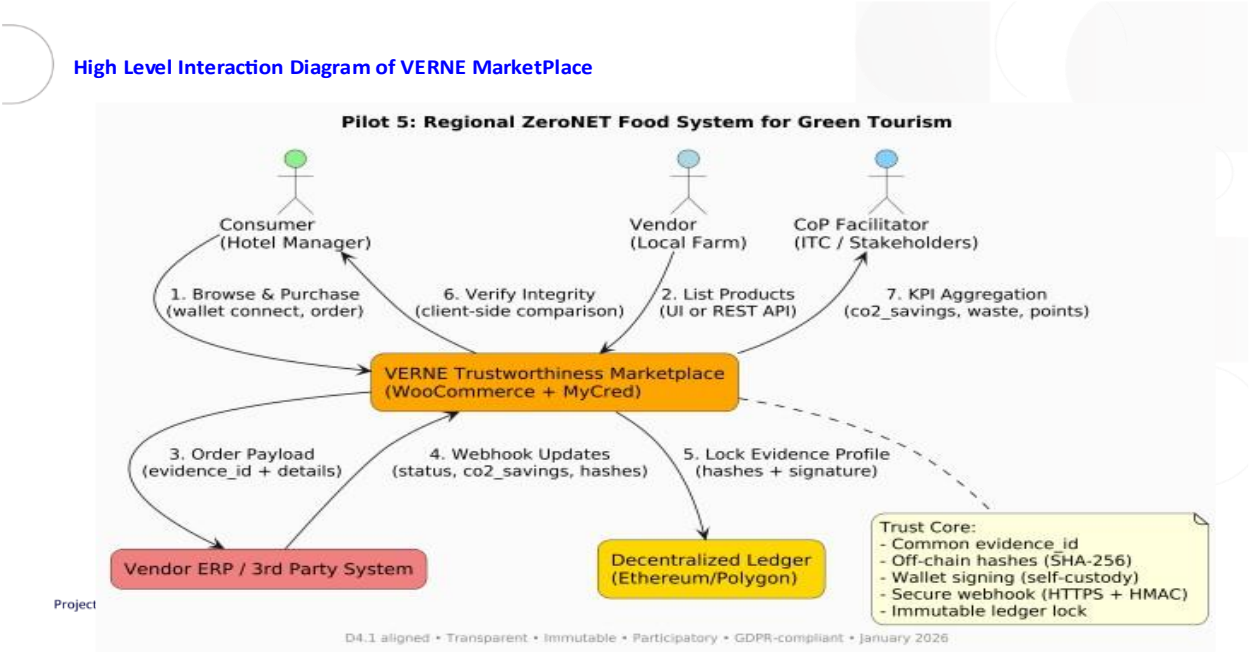


Figure 27: Trustworthiness Market & Decentralized Ledger

4.6.3.1 End-to-End User Journey: Trustworthy Zero-Waste Food Transaction in VERNE Marketplace)

The specific section describes the E2E User Journey of **Trustworthy Zero-Waste Food Transaction in VERNE Marketplace**

Stage	Actor(s)	Key Actions	Pain Points / Requirements	Main Assets/Entities Created/Used	Output / Next Stage Trigger
1. Discovery & Onboarding	Consumer (Hotel) Vendor (Farm)	- Browse listings (products/services) - View Trust Score, badges - Register & link to **Tokenisation API**	- Building initial trust - Easy onboarding without tech friction	- User (role, trust_score) - Product (service_id, category, metadata) - **Tokenisation API** access	- Access to tokenisation platform granted → Proceed to purchase
2. Product Listing	Vendor	- List product (UI or REST API) - Add metadata (quantity, zero-net flag)	- Simple listing without manual repetition - Metadata consistency for KPIs	- Product (service_id, name, price, category, metadata)	- Product live in marketplace → Ready for browsing
3. Purchase & Pre-Commitment	Consumer	- Add to cart, checkout - Pay (escrow + **Tokenisation API**) - Accept terms (pre-commit)	- Fast checkout, no extra steps - Assurance of terms immutability	- Order (order_id, consumer_id, vendor_id, price) - Evidence Profile (evidence_id generated, terms_hash)	- Order created - evidence_id shared → evidence_id to ERP
4. Order Execution & Delivery	Vendor (via ERP)	- Receive payload (evidence_id + details) - Upsycle, deliver (electric vehicle) - Calculate co2_savings & create hashes	- Easy integration with ERP - Accurate KPI calculation	- Delivery (tracking_number, mode, distance_km, load_tons, co2_savings) - Hashes	- Order state = completed - Webhook sent to marketplace → Trigger verification

Stage	Actor(s)	Key Actions	Pain Points / Requirements	Main Assets/Entities Created/Used	Output / Next Stage Trigger
				(delivery_hash, metrics_hash)	
5. Integrity Lock & Verification	Vendor & Consumer	- **Tokenisation API** processes & submits Evidence Profile - Consumer clicks “Verify” in dashboard - Client-side hash comparison	- Easy verification (no tech knowledge) - Immutable proof of integrity	- Evidence Profile (evidence_id, all hashes, signature, timestamp) - Hash (credits_hash for points)	- Profile locked on ledger - “Green Light” certificate - Escrow release & points awarded
6. Rewards & Monitoring	Consumer, Vendor, CoP	- Receive points (MyCred) - View aggregated KPIs in CoP dashboard - Provide feedback for iteration	- Motivational rewards - Transparent KPI visibility for CoPs	- KPI (metric_type, value, evidence_id) - CoP Report (aggregated_kpis, feedback_notes)	- Points redeemable - KPIs feed CoP discussions → Iterative improvement

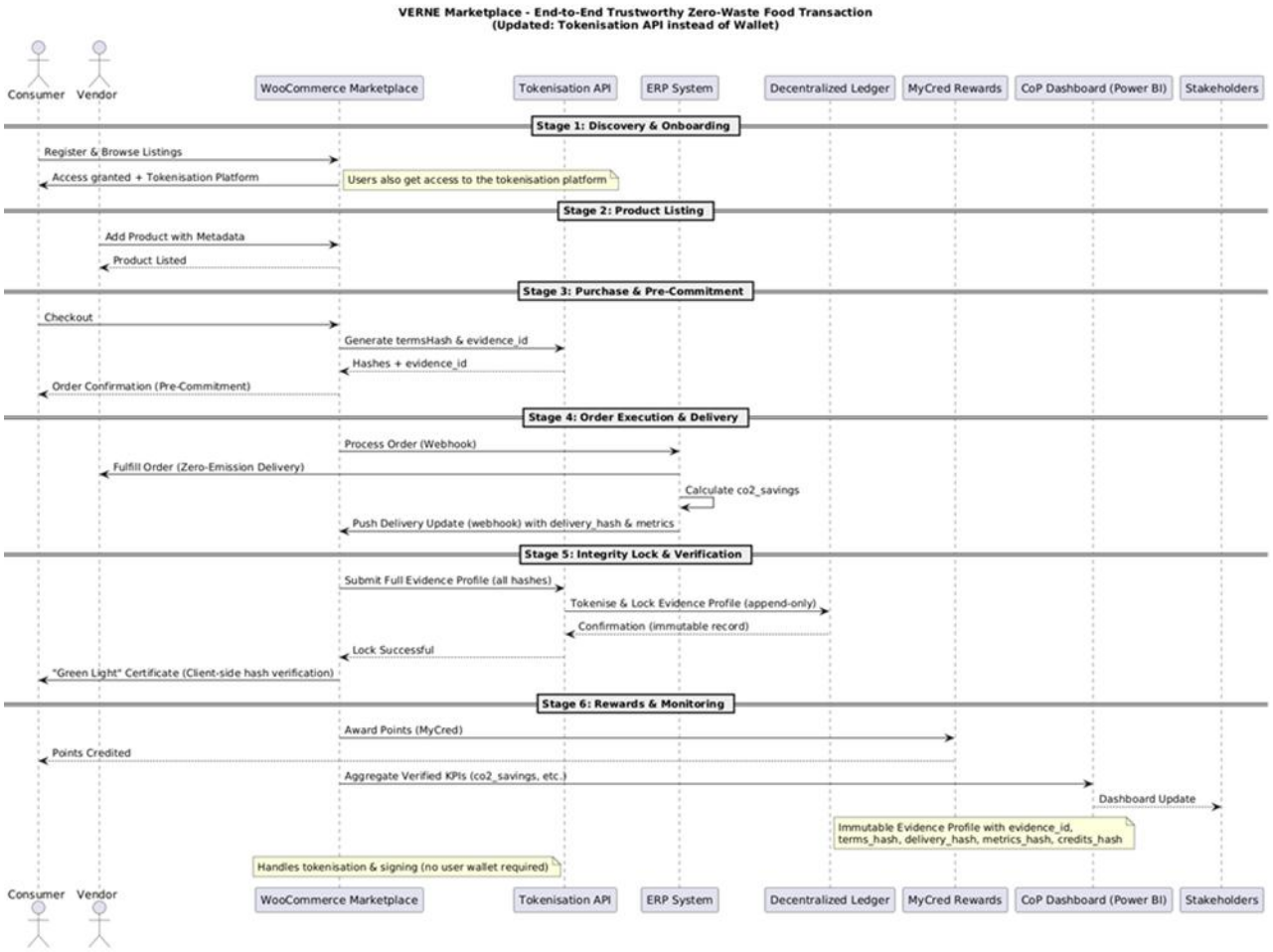


Figure 28: Trustworthy Zero-Waste Food Transaction in VERNE Marketplace

4.6.3.1.1 Summary of Solution Components: Tokenisation API & Decentralized Ledger

In the VERNE Trustworthiness Marketplace architecture (aligned with Pilot 5: Regional ZeroNET Food System), the **Tokenisation API** serves as the secure interface for enabling non-repudiation and verifiability of data via smart contract tokenisation techniques. It handles signing of transaction hashes and Evidence Profiles, ensuring integrity without requiring user-controlled self-custody mechanisms. The Tokenisation API integrates with the WooCommerce marketplace for onboarding, pre-commitment during purchases, and Evidence Profile locking.

The Decentralized Ledger (e.g., Ethereum or Polygon L2) provides immutability, verifiability, and an audit trail by storing Evidence Profiles (including evidence_id, terms_hash, delivery_hash, metrics_hash, credits_hash, and signatures). This enables client-side verification and trustless transparency.

Table 18: Related Use Case with solution component

Use Case	Description	Components Involved	Link to User Journey Stage
Onboarding & Tokenisation API Access	Users (vendors/consumers) register and gain access to the tokenisation platform to view Trust Score and participate in transactions.	Tokenisation API, Marketplace (WooCommerce DB)	Stage 1: Discovery & Onboarding
Purchase & Pre-Commitment	System generates termsHash via Tokenisation API during checkout to create evidence_id and pre-commit to terms.	Tokenisation API, Ledger (optional early lock)	Stage 3: Purchase & Pre-Commitment
Integrity Lock after Delivery	Tokenisation API processes the full Evidence Profile (all hashes) and submits it to the ledger for immutable storage. System-to-system process automation.	Tokenisation API, Ledger (append-only lock)	Stage 5: Integrity Lock & Verification
Client-Side Verification & Rewards	Consumer verifies hashes against ledger to get "Green Light" certificate, releasing escrow and awarding points.	Tokenisation API (public read), Ledger (fetch hashes)	Stage 5: Integrity Lock & Verification; Stage 6: Rewards & Monitoring
CoP KPI Aggregation & Audit	Aggregated KPIs (e.g., co2_savings) from verified ledger data feed into CoP dashboard for stakeholder review.	Ledger (audit trail), CoP Dashboard (Power BI)	Stage 6: Rewards & Monitoring

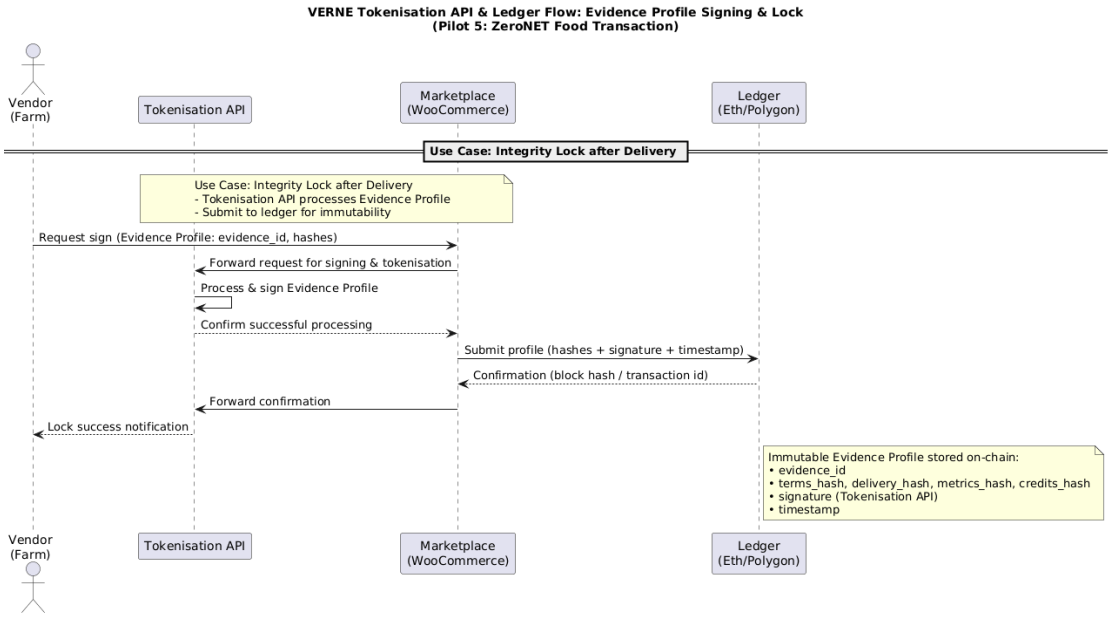


Figure 29: Tokenisation API & Ledger flow

5 Conclusions

Deliverable D3.1: Architecture and data sharing ecosystem, establishes the architectural and data foundations of the VERNE ecosystem required to implement WP3 digital tools consistently across heterogeneous pilots. It defines an open, secure, and modular high-level platform architecture, described through a structured methodology aligned with recognized architecture description practices, and provides a common reference model that supports interoperability, scalability, and future extensibility as additional services and integrations emerge.

At platform level, D3.1 identifies the core building blocks needed to enable end-to-end operation of the ecosystem—data ingestion from multiple channels, orchestration and scheduling, scalable storage, security and access management, and support for analytics and KPI computation. These components are positioned to support diverse operational realities across pilots (sensor-driven, application-driven, and research-driven evidence), while maintaining a unified way to manage data flows, governance, and service integration.

At pilot level, D3.1 consolidates the system context, AS-IS situation, objectives/use cases, envisaged digital tools, and integration points for each pilot. This pilot-by-pilot view makes explicit what data can be provided, how it can be exchanged (e.g., APIs where available or periodic structured exports), and how it can be used to derive circularity and sustainability indicators. Where pilot KPIs are presented, they are treated as indicative and will be aligned

with the final KPI framework developed in WP1, ensuring consistency at ecosystem level without constraining pilots' domain-specific circularity methods.

Overall, D3.1 provides a practical blueprint that supports the implementation path of VERNE: it links stakeholder and use-case needs to platform capabilities, clarifies integration and data requirements, and enables subsequent WP3 developments and releases. The architecture and data specifications presented here will be iteratively refined during implementation and demonstration activities as pilot inputs mature and operational constraints are validated, ensuring that the VERNE ecosystem remains both technically robust and grounded in real pilot conditions.

6 References

[1] ISO/IEC/IEEE, "ISO/IEC/IEEE 42010:2022 Standard," [Online]. Available: https://www.iso.org/standard/74393.html.

[2] N. Rozanski and E. Woods, Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives, Second Edition, O'Reilly, 2011.

[3] M. W. Maier, D. Emery and R. Hilliard, "ANSI/IEEE 1471 and systems engineering," *ACM Systems Engineering,* vol. 7, no. 3, pp. 257-270, 2004.

[4] P. B. Kruchten, "The 4+1 View Model of architecture," *IEEE Software,* vol. 12, no. 6, pp. 42-50, 1995.

[5] S. Brown, "The C4 model for visualising software architecture," [Online]. Available: https://c4model.com/.

[6] R. Gustavsson, J. Lundberg, C. Rindebäck and K. Ådahl, "Functional versus non-functional requirements considered harmful," in *International Conference on Software Engineering Research & Practice*, Las Vegas, Nevada, USA, 2007.

[7] J. Eckhardt, A. Vogelsang and D. M. Fernández, "Are "non-functional" requirements really non-functional?: an investigation of non-functional requirements in practice," in *Proceedings of the 38th International Conference on Software Engineer*, Austin, Texas, USA, 2016.

[8] F. Baskoro, R. A. Andrahsmara, B. R. P. Darnoto and Y. A. Tofan, "A Systematic Comparison of Software Requirements Classification," *IPTEK, Journal for Technology and Science,* vol. 21, no. 3, 2021.

Shaping Sustainable Tourism

www.verneproject.eu

info@verneproject.eu

#VERNEPROJECT

Verne

PNO · RETHINK TOURISM · Barcelona School of Tourism, Hospitality and Gastronomy · Universitat de Barcelona · UPC Universitat Politècnica de Catalunya BarcelonaTech · ITC Inovacijsko tehnološki gruzd Innovation Technology Cluster · AGRIFOOD Digital Innovation Hub · SDU · cscp · Luke · uni.systems · alchemia nova circular by nature · Impact Hub Athens · APRE · Bursa Uludağ Üniversitesi 1975 · ATU · Lund University

Funded by the European Union