Research and Innovation Action
HORIZON-CL6-2021-FARM2FORK-01
TOPIC: HORIZON-CL6-2021-FARM2FORK-01-18

NESTLER
oNe hEalth SusTainabiLity partnership between
EU-AFRICA for food sEcuRity

Deliverable D4.3
NESTLER backend implementation of AI algorithms and agricultural services

Authors	P. Athanasoulis, N. Arvanitis, Th. Zahariadis, A. Tomaras, G. Athanasiou, A. Skias, G. Pantelide, A. Nchange Kouotou, T. Odedeyi
Nature	*Demonstrator*
Dissemination	PUBLIC
Version	1.0
Status	FINAL
Delivery Date (DoA)	M36
Actual Delivery Date	03/10/2025

Keywords	Federated Machine Learning, Artificial intelligence, Machine Learning, Risk Assessment, Impact Assessment, Drought/Flood, Prediction, Smart Irrigation, Pest Detection, Weather, Climate Data, Satellite Data, GIS, Crop Yield, Livestock, Insects, NESTLER Architecture.
Abstract	Deliverable D4.3 constitutes an updated version of deliverable D4.1 and outlines the backend implementation of the NESTLER AI algorithms and agricultural services. Especially, it presents the high-level architecture of the NESTLER platform as well as the NESTLER AI framework. Then, it presents a set of backend agricultural services and available interfaces. Finally, the NESTLER Integration Framework and Tools are defined and described each of which is able to cater to all of NESTLER's subcomponents.

© Copyright by the NESTLER Consortium
Funded by the European Union under Grant Agreement no.101060762

DISCLAIMER

This document is a deliverable of the NESTLER project funded by the European Union under Grant Agreement no.101060762. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Executive Agency, while neither the European Union nor the granting authority can be held responsible for any use of this content.
This document may contain material, which is the copyright of certain NESTLER consortium parties, and may not be reproduced or copied without permission. All NESTLER consortium parties have agreed to the full publication of this document. The commercial use of any information contained in this document may require a license from the proprietor of that information.
Neither the NESTLER consortium as a whole, nor a certain party of the NESTLER consortium warrant that the information contained in this document is capable of use, nor that use of the information is free from risk and does not accept any liability for loss or damage suffered using this information.

	Participant organisation name	Short	Country
01	SYNELIXIS SOLUTIONS S.A.	SYN	EL
02	CloudEO AG (Terminated)	CEO	DE
03	RINIGARD DOO ZA USLUGE	RINI	HR
04	EBOS TECHNOLOGIES LIMITED	eBOS	CY
05	STICHTING IDH SUSTAINABLE TRADE INITIATIVE	IDH	NL
06	ZANASI ALESSANDRO SRL	Z&P	IT
07	AGRIX TECH SARL	AGRI	CM
08	CONSERVATION THROUGH PUBLIC HEALTH	CTPH	UG
09	THE INTERNATIONAL CENTRE OF INSECT PHYSIOLOGY AND ECOLOGY LBG	ICIPE	KE
10	ETHIOPIAN INSTITUTE OF AGRICULTURAL RESEARCH	EIAR	ET
11	RWANDA AGRICULTURE AND ANIMAL RESOURCES DEVELOPMENT BOARD	RAB	RW
12	INTERNATIONAL INSTITUTE OF TROPICAL AGRICULTURE	IITA	NG
13	MANA BIOSYSTEMS LIMITED	MANA	UK
14	UNIVERSITY COLLEGE LONDON	UCL	UK
15	RINISOFT LTD	RINIS	BG
16	ADAPT IT	ADA	DE

ACKNOWLEDGEMENT

This document is a deliverable of the NESTLER project. This project has received funding from the European Union's Horizon Research and innovation programme under grant agreement Nº 101060762. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Executive Agency, while neither the European Union nor the granting authority can be held responsible for any use that may be made of the information it contains.

Document History

Version	Date	Contributor(s)	Description
v0.1	24/07/2025	SYN	Finalise Table of Contents
v0.2	12/09/2025	AGRI, eBOS	Input for weather impact assessment services and economic risk models
v0.3	14/09/2025	SYN	Input for NAIF
v0.4	19/09/2025	SYN	Input for backend services and interfaces
v0.5	29/09/2025	UCL, eBOS	Peer review
V0.6	02/10/2025	SYN	Minor changes in entire document
V1.0	03/10/2025	SYN	Final Quality check

Document Reviewers

Date	Reviewer's name	Affiliation
29/09/2025	Temitope Odedeyi	UCL
29/09/2025	Georgia Pantelide	eBOS

Definitions, Acronyms and Abbreviations

AI	Artificial Intelligence
BSF	Black Soldier Fly
BSFL	Black Soldier Fly Larvae
BSFLM	Black Soldier Fly Larvae Meal
CI	Continuous Integration
CD	Continuous Delivery
CNN	Convolutional Neural Network
CSI	Cubic Spline Interpolation
EO	Earth Observation
EPPO	European and Mediterranean Plant Protection Organization
EU	European Union
FAO	Food and Agriculture Organization
FL	Federated Learning
GDPR	EU General Data Protection Regulation no. 2016/679
GIS	Geographic Information System
HE	Homomorphic Encryption
IAM	Identity and Access Manager
IoT	Internet of Things
IQR	Interquartile Range
KPI	Key Performance Indicator
LSA	Least Square Approximation
ML	Machine Learning
MA	Moving Average
MAE	Mean Absolute Error
NESTLER	oNe hEalth SusTainabiLity partnership between EU-AFRICA for food sEcuRity
OIDC	OpenID Connect
OGC	Open Geospatial Consortium
One Health	Integrated approach to human, animal, and environmental health

OS	Operating System
PV	Persistent Volume
PVC	Persistent Volume Claim
PCA	Principal Component Analysis
PM	Project Manager
PMT	Project Management Team
PVC	Persistent Volume Claim
PQC	Post-Quantum Cryptography
QA	Quality Assurance
SCM	Source Code Management
SPEI	Standardised Precipitation Evapotranspiration Index
SSO	Single-Sign-On
SVR	Support Vector Regression
TL	Task Leader
TM	Technical Manager
UPOV	Union for the Protection of New Varieties of Plants
ViT	Vision Transformer
WFS	Web Feature Service
WMS	Web Map Service
WMTS	Web Maps Tile Service
WP	Work Package
WPL	Work Package Lead
WSGI	Web Server Gateway Interface
ZTFL	Zero Trust Federated Learning

Table of Contents

List of Figures

List of Tables

Executive Summary

The NESTLER consortium presents the high-level system architecture of the project, encompassing data sources as well as backend and frontend services through which stakeholders can access the platform.

This deliverable focuses mainly on the backend implementation of the AI algorithms and agricultural services. To set up the NESTLER AI platform, NESTLER has developed NAIF which is an FML framework that combines the functionalities of the best frameworks in their respective fields. Thanks to an appropriate choice and combination of frameworks, the NESTLER AI platform is both reliable and flexible.

Also, in line with the identified user requirements, NESTLER AI platform is designed to host AI algorithms for:

- weather impact assessment.
- animal health surveillance.
- agricultural crop surveillance and management.
- pest infestation monitoring.

All these algorithms are based on IoT sensor data, drone data, camera data, microphone data and EO data that can be saved on the platform via dedicated interfaces.

Finally, a structured yet adaptable integration framework for NESTLER is defined, introducing the DevSecOps approach—encompassing development, security, and operations—and outlining the initial framework components such as Source Code Management, CI/CD, Issue Tracking, and Containerization.

1 Introduction

The deliverable D4.3 is the third deliverable in the WP4, and it constitutes an update of deliverable D4.1 [1]. It contains reports of work already achieved and related to the following NESTLER work package tasks:

- T4.1: AI algorithms for external weather impact assessment upon agriculture farming.
- T4.2: Automated monitoring services for crop yield quality, livestock wellbeing and insect population.
- T4.3: Economic risk assessment models for predicting the yield quality.
- T4.5: NESTLER platform architecture design and integration.

This document is organized into six sections, as it follows:

- Section 1 lists the NESTLER's work package tasks related to D4.3 and presents an overview of the document's content.
- Section 2 provides an updated version of the NESTLER reference architecture that is mentioned in deliverable D4.1.
- Section 3 provides an update on NESTLER AI framework (NAIF) aiming to allow a flexible deployment of the Federated Learning frameworks for the Federated Learning based model training to be implemented by NESTLER.
- Section 4 presents agricultural services that are developed and integrated by NESTLER partners.
- *Section 5* provides an update of the NESTLER integration framework and tools aiming to ensure a structured and flexible integration of all NESTLER's subcomponents.
- Finally, Section 6 concludes this report and draws the perspectives of works to come.

The final results of WP4 activities will be provided in Deliverable D4.4, entitled "NESTLER Integrated Platform Release".

2 NESTLER Platform Architecture

This section provides the updated version of the NESTLER reference architecture as it has been evolved from the beginning of the project. Its development has been guided by the system's intended functionality and the identified user requirements. The proposed architecture is designed to be flexible, allowing for adjustments and updates, and scalable, ensuring it can adapt to increasing demands and evolving technologies.

2.1 High-level Platform Architecture

The high-level system architecture of NESTLER platform is structured into different layers, each one of which is responsible for specific functionalities within the platform. Figure 1 depicts the updated version of the high-level NESTLER reference architecture.

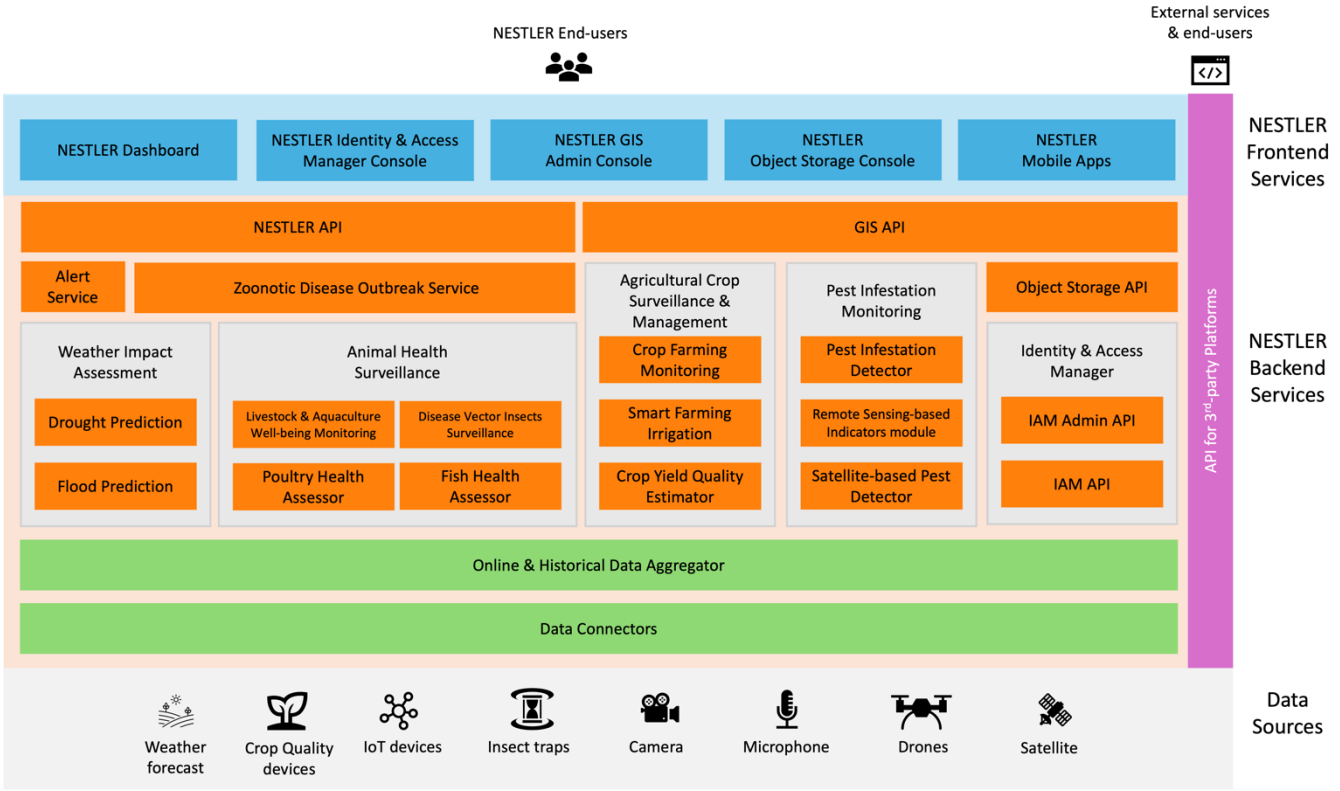

Figure 1: High-level reference architecture overview

At the foundation of the system lie diverse data sources, such as IoT devices and sensors, drones, cameras, microphones, insect traps, satellite Earth observation, and weather forecast data. The platform is capable of integrating and analyzing this wide spectrum of inputs from heterogeneous technologies, thereby endowing the system with multi-modality characteristics (see section §2.2).

On top of the data sources, the platform incorporates building blocks that consume and process that data which constitutes the backend services. The backend services include both components that are

implementing during of the project and open-source services that have been configured and integrated in the platform (see section §2.3).

End-users are able to interact with the NESTLER platform through its frontend services that include web Graphical User Interfaces (GUI) as well as mobile applications (see section §2.4). Finally, external Services API have been implemented to facilitate the connection between the NESTLER platform and external platform services.

2.2 Data sources

The description of the data, its acquisition methods through the NESTLER platform, and the platform's multimodal data aggregation characteristics are thoroughly detailed in D3.1 "Remote Sensing Technologies and Multimodal Data Aggregation Protocols" [2]. This section aims to offer a holistic overview of the high-level architecture of the NESTLER platform by outlining the various data sources that provide data to the system.

- *IoT* devices are devices within the Internet of Things (IoT) ecosystem, designed to gather diverse types of data, which can be used for real-time monitoring and decision-making in services. In the NESTLER project, data is collected using the SynField platform [3], which supports connections with the following devices:
  - *Weather station,* that offers remote monitoring of environmental weather factors.
  - *Soil sensors*, gathering data related to soil conditions.
  - *SynWater*, which provides water quality monitoring.
  - *SynAir*, which provides air quality monitoring.
  - *Valves*, which used to irrigate or not the crop
- *Quality Measurement Devices* extracts measurements related to crop quality characteristics. In the NESTLER project, those devices are used to gather information about the quality of cassava crops.
- *Insect Trap* are traps that provide remote monitoring of fly (e.g. Glossina, Phlebotomus, Simulium, Musca, Calliphoridae, Sarcophagidae, Stomoxys calcitrans and Cordylobia anthropophaga species) and mosquito (e.g. Aedes, Anapheles, Culex and Mansonia genus) insects in selected locations of the pilots.
- *Drone* is a dynamic data source within the system, providing high-resolution aerial imagery and geographic information. In the NESTLER project, drones will be utilized to collect data, providing insights about the biotic stress of crops.
- *Camera* captures visual information in the form of images and video. In the NESTLER project, camera is used to gather video data of livestock and aquaculture well-being as well as images of crops.
- *Microphone* captures surrounding information from the surrounding environment. In NESTLER, microphones are employed to monitor acoustic signals within livestock farming.
- *Weather forecast* includes data that is both processed by backend services and visualized in the NESTLER dashboard.
- *Satellite Earth Observation data* provide insights into environmental and agricultural conditions by capturing information across large geographical areas with high temporal frequency. These

datasets, derived from remote sensing technologies, enable the monitoring of key variables such as land cover, vegetation indices, soil moisture, and weather patterns.

2.3 Backend Services

As mentioned earlier, several open-source tools were configured and integrated into the backend services. The open-source tool *Keycloak* [4] is used in the NESTLER platform and is employed in the NESTLER platform as the central *Identity and Access Manager* (IAM). IAM supports standard protocols such as OpenID Connect [5], OAuth 2.0 [6], and SAML [7], while also providing fine-grained authorization services, identity brokering and social login features, as well as single-sign-on (SSO) functionality across the NESTLER platform's services. IAM includes realms, where each realm constitutes the top-level container and each one manages a distinct set of the users, group of users, roles, permissions, sessions and clients (backend and frontend services) independently. To be IAM persistent, it keeps any kind of data in a PostgreSQL database.

NESTLER platform uses the open-source tool *Geoserver* [8] that acts as *GIS API* to share geospatial data being available in the NESTLER databases (PostgreSQL/PostGIS) and Satellite Earth Observation (EO) data within the NESTLER components. *Geoserver* publishes this data through Open Geospatial Consortium (OGC) standards such Web Feature Service (WFS), Web Map Service (WMS) and Web Maps Tile Service (WMTS). WFS provides raw vector data (e.g. GeoJSON, GML) while WMS provides rendered map images (e.g. PNG, PNG8). WMTS provides pre-rendered and cached map tiles in the requested SRS and zoom level. GIS API is planned to be integrated with *IAM* until the end of the NESTLER project.

In the NESTLER platform, open-source tool *MinIO* [9] is employed as a high-performance *Object Storage* solution for managing various types of large objects into buckets, rather than using PostgreSQL databases. *MinIO* provides a RESTful API that is fully compatible with AWS S3 API.

Apart from the open-source tools, NESTLER partners have implemented various backend services. First, given the multi-modal data, *Data Connectors* have been implemented to fetch the data from their origin in case that follow the PULL model. For example, a data connector has been deployed to collect measurements from the SynField, SynAir, and SynWater devices installed at the pilot sites. Another data connector has been deployed to collect measurements from automatic mosquito' traps. The *Online and Historical Data Aggregator* is employed to store all types of data in the NESTLER PostgreSQL database.

The backend also includes the group of services *Weather Impact Assessment* that consists of the *Drought* and *Flood prediction (*also mentioned *Drought* and *Flood forecast)* services. The output of each service is an indicator that forecasts the likelihood of a specific undesired event (drought/flood) in the coming days. If the service predicts that the event will occur, the *Alert Service* is triggered to inform the users about the upcoming event (called "*Alert Module*" in deliverable D4.1). These services keep their datasets in the Object Storage while being integrated with the *IAM*. Section §4.1 provides a description of each service and mentions the API of each service.

Another group of services is the *Animal health surveillance*, which is composed of four services. The first one is the *Livestock & Aquaculture Well-being Monitoring*, which outputs records of environmental conditions of the habitats. The *Poultry and Fish Health Assessors* constitute the other two services, with both delivering a single health index. *Disease Vector Insects Surveillance* constitutes the fourth service of this group and is responsible to monitor the insect traps and the spread of disease vector insects. The description of these services is available in deliverables D3.1 [2] and D3.2 [10]. These services along with

Weather Impact Assessment services provide input in the Zoonotic Disease Outbreak Service is responsible to detect any disease outbreak; this component was designated as the "*Disease outbreaks Module*" in deliverable D4.1.

Agricultural Crop Surveillance & Management, aiming to observe and manage crop cultivation activities and conditions to achieve optimal crop growth. This group of services contains the *Crop Farming monitoring* [10], *Smart Farming Irrigation* (see section 4.2.1) and the *Crop Yield Quality Estimator* (see section 4.4).

The group of services *Pest Infestation Monitoring* includes the *Remote sensing-based indicators module, the Pest infestation detector,* and the *Satellite-based pest detector* (see deliverable D3.2 [10]). It should be mentioned that those services can operate independently. However, the *Remote sensing-based indicators module* and the *Pest infestation detector* have the capacity to be integrated into a single, unified service.

NESTLER API is a RESTful interface that acts as the communication layer among various NESTLER backend services and enables the NESTLER dashboard to exchange data with the NESTLER PostgreSQL. Furthermore, the NESTLER API has been integrated with the IAM service. Moreover, API for 3rd-party services and platforms is being implemented to facilitate the connection between the NESTLER platform and other services/platforms using AIM from Demeter EU project [11].

2.4 Frontend Services

The collected data as well as the backend services' results are made accessible to NESTLER end-users through various frontend services.

NESTLER dashboard provides visualization tool based on Geographic Information System (GIS), designed to provide users with geographically contextualized graphical representation of the collected data as well as services' results, enabling them to gain insights and make decisions about crop conditions and livestock, aquaculture well-being. It is important to emphasize that the NESTLER dashboard has been integrated with the IAM API, the GIS API, the NESTLER API, and the Object Storage API. The initial version of the NESTLER dashboard was described in the deliverable D4.2 [12] while its final version is planned to be reported in deliverable D4.4 (M42). Figure 2 depicts a snapshot of the NESTLER dashboard landing page.

The NESTLER platform enables the farmers or/and agronomists the to conduct an on-field real-time inspection of the crop through the NESTLER *Pest Recognition Mobile Application* to check if pests are present on a crop and infesting it.

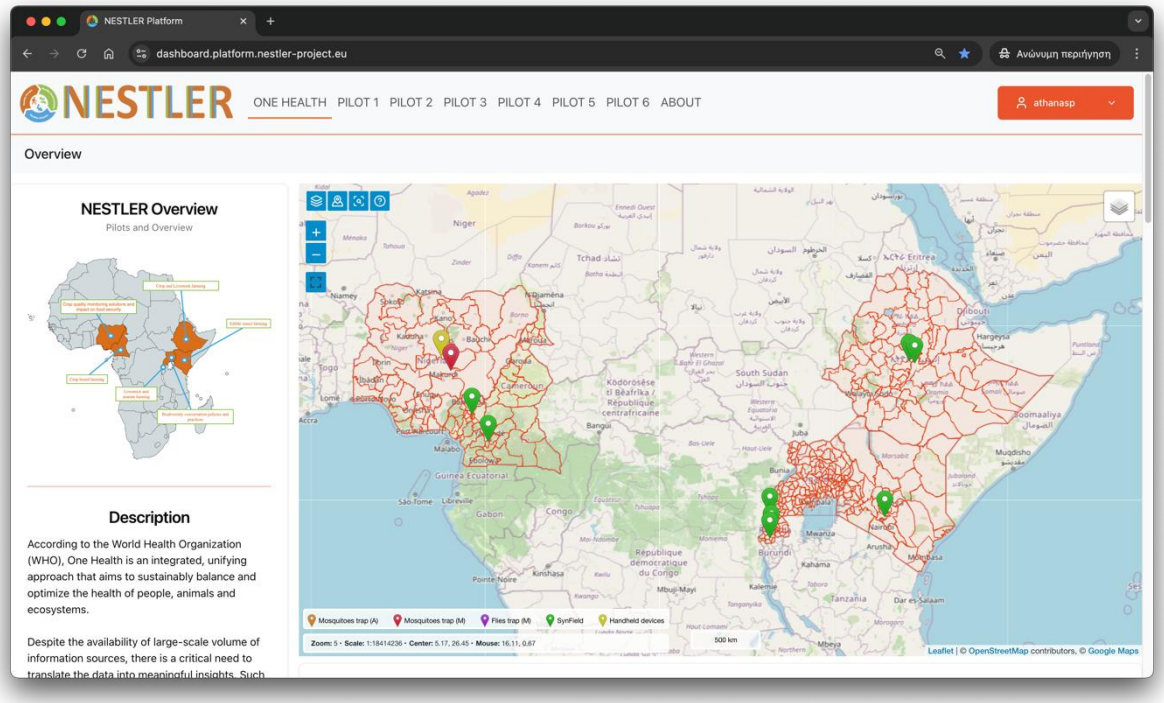

Figure 2: Snapshot of landing page in NESTLER dashboard

As aforementioned, the open-source tools *Geoserver*, *Keycloak* and *MinIO* were configured and integrated into the NESTLER platform. Apart from the backend services, these services provide graphical user interfaces (GUI). *NESTLER Identity & Access Manager Console* allows the administrators of NESTLER platform to manage realms, users, groups, roles, clients, identity providers, authentication and authorization flows, sessions and events. Figure 3 depicts a snapshot of registered services in *IAM Console*.

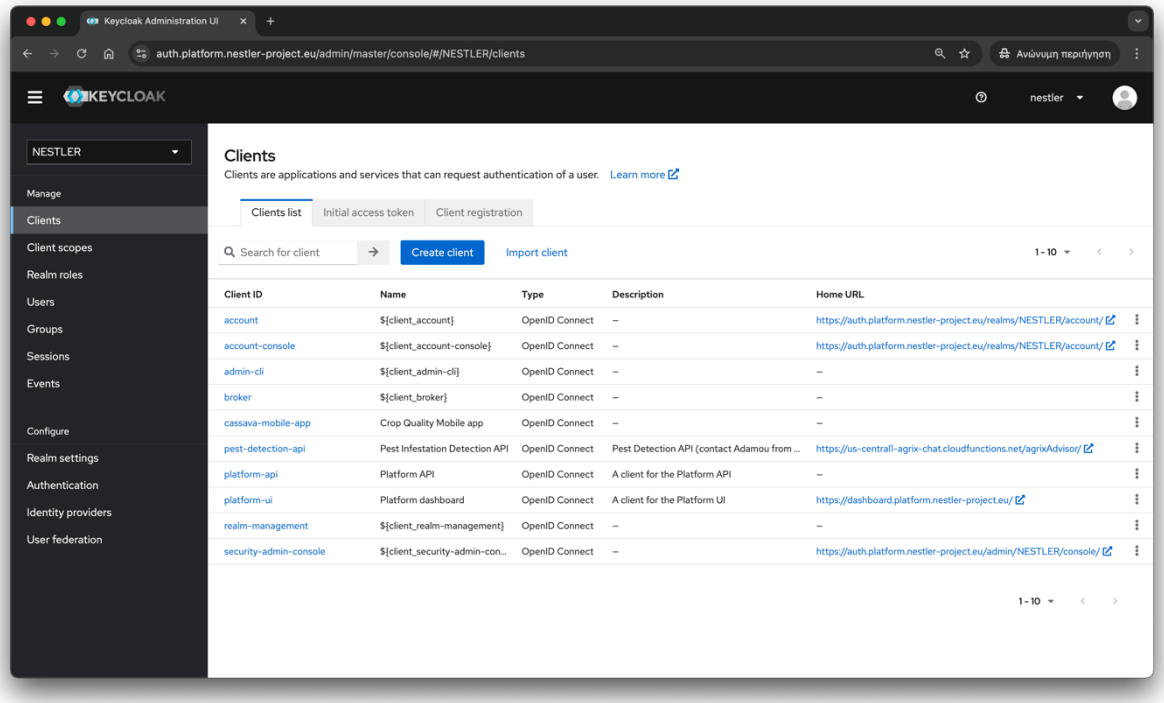

Figure 3: Snapshot of registered services in the Identity & Access Manager Console

NESTLER GIS Admin Console is a web-based GUI that allows administrators to configure and manage all aspects of the *GeoServer* environment. Through the GIS Admin Console, after successful authentication, the admin is able to create and organize workspaces, stores, and layers, enabling data from various sources such as shapefiles, PostGIS, or raster datasets to be published. It supports configuring styles for visualization, managing WMS, WFS, and WMTS services, and setting up layer groups for composite maps. Administrators can also control security settings, including user roles, service access rules, and authentication methods, as well as monitor server status, logs, and performance metrics. In short, the *GIS Admin Console* provides a centralized environment for publishing, styling, securing, and monitoring geospatial data services. Figure 4 depicts a snapshot of the available layers in the GIS Admin *Console*.

NESTLER Object Storage Console is a web-based GUI that allows users to easily interact with the object storage system without relying on command-line tools. Through the GUI, the user is able to create and manage buckets, upload, download, and delete objects, and organize data into folders. It also provides options for generating pre-signed URLs to share files securely, setting object lifecycle policies, and viewing detailed object metadata. Users can monitor storage usage and statistics, manage access keys and secrets (if permissions allow), and configure bucket-level settings such as versioning or encryption. In essence, *NESTLER Object Storage Console* provides an intuitive way for users to perform all common storage operations, manage access, and monitor usage within the object storage environment. Figure 5 depicts a snapshot of the "*nestler*" bucket structure in the *Object Storage Console.*

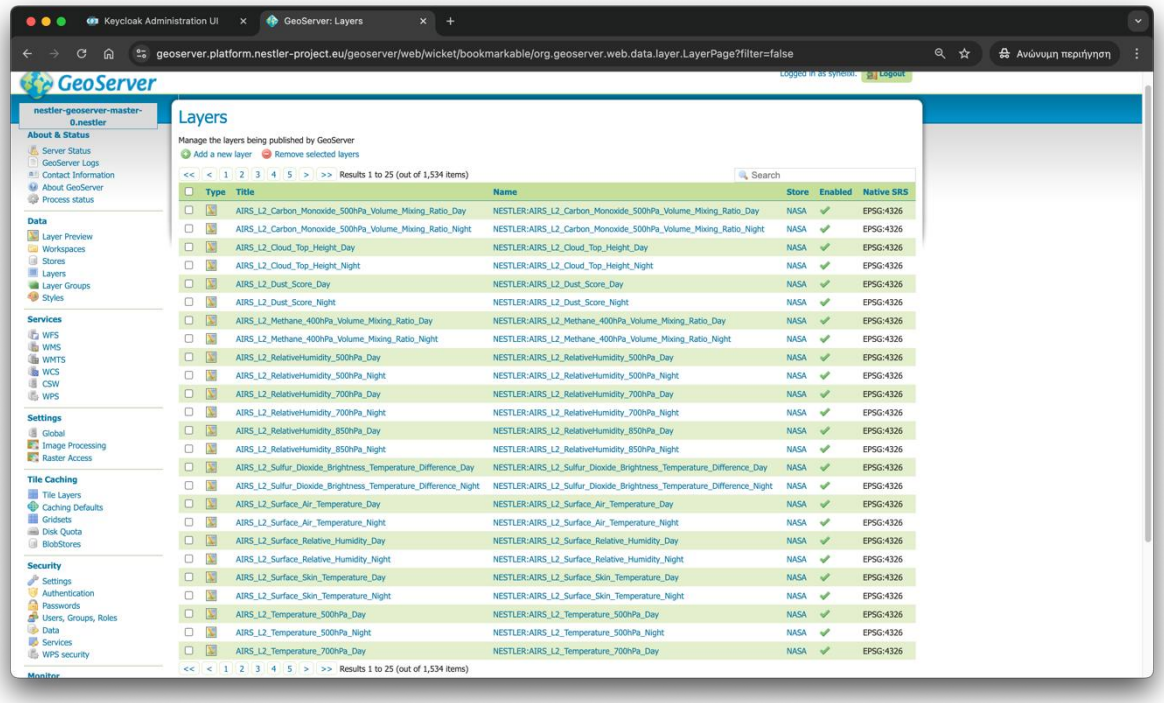

Figure 4: Snapshot of available layers in the GIS Admin Console

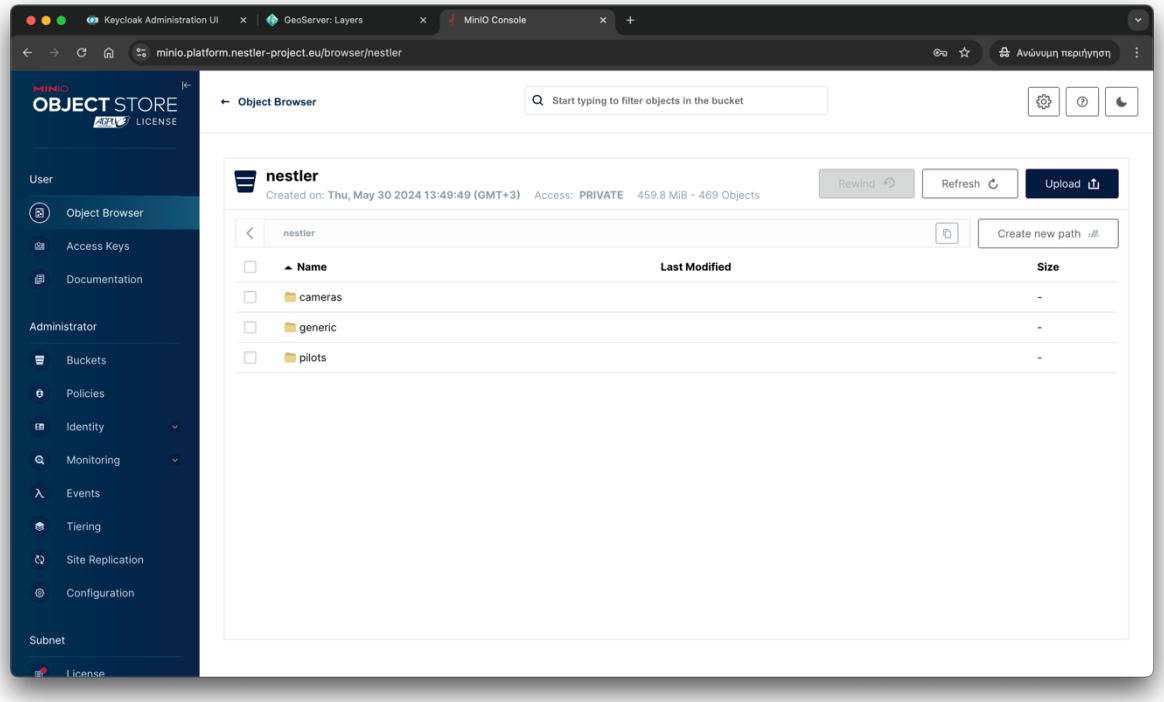

Figure 5: Snapshot of folders structure in the Object Storage Console

3 NESTLER AI Framework (NAIF)

Here the NESTLER AI framework is described, under which the different use cases are supposed to be applied. Compared to NESTLER's deliverable D4.1 [1], there have been some changes regarding the privacy preserving both on the user's local data and the model's weights.

3.1 High-level Architecture

At deliverable D4.1 [1] our suggested Federated Learning (FL) architecture was described, in which privacy preserving was ensured with the combination of FLOWER [13] and PATE frameworks. However, "*PATE is primarily a transfer learning framework and may not fully address the specific privacy protection requirements inherent in Federated Learning, especially in scenarios involving potentially malicious participants*".

Considering the disadvantages of PATE and the need for privacy protection, another solution is suggested and presented here, which is stronger than FLATE in terms of privacy protection and is currently under development. This solution maintains FLOWER as the FL framework capable of handling the communication among the server, that contains the global model, and the clients which contain the local models. The modification lies in the fact that privacy preserving is assured by applying homomorphic encryption bidirectionally, from client to server and from server to client at each aggregation round by a context server, and also by adding a digital signature between the communication of a client and a server preventing adversary parties from accessing the weights of the model.

3.1.1 Zero-Trust Federated Learning Framework

To address the critical need for secure, privacy-preserving AI services, a Zero Trust Federated Learning (ZTFL) framework is proposed, purpose-built for deployment in sensitive environments such as the NESTLER platform. The framework enforces a zero-trust security model by applying multiple layers of cryptographic protection and robust authentication mechanisms throughout the federated learning lifecycle. It is both data-agnostic and model-agnostic, enabling reusability and adaptation to various use cases.

In this Zero Trust architecture, no entity—whether internal or external—is implicitly trusted. This principle is embedded at every level of our system: from model parameter exchange to key distribution and signature verification. The key components of our FL framework are visually presented in Figure 6 and are also explained below:

- Aggregation Server: Receives encrypted model weights from all participating clients and performs secure aggregation without access to raw or decrypted data. Verifies all inputs via post-quantum signatures before processing. The aggregated result remains encrypted and is distributed back to clients for local decryption and further training.
- Clients: Each client trains locally on private data and communicates only encrypted and signed model updates. Clients receive a signed encryption context from the context server, ensuring consistency and authenticity. All outgoing updates and incoming model weights are verified and decrypted locally, preserving data confidentiality throughout the training process.

- Encryption Context Server: A dedicated server that handles the secure, signed distribution of public and private encryption contexts to clients and the aggregation server. This prevents context tampering and isolates key management.

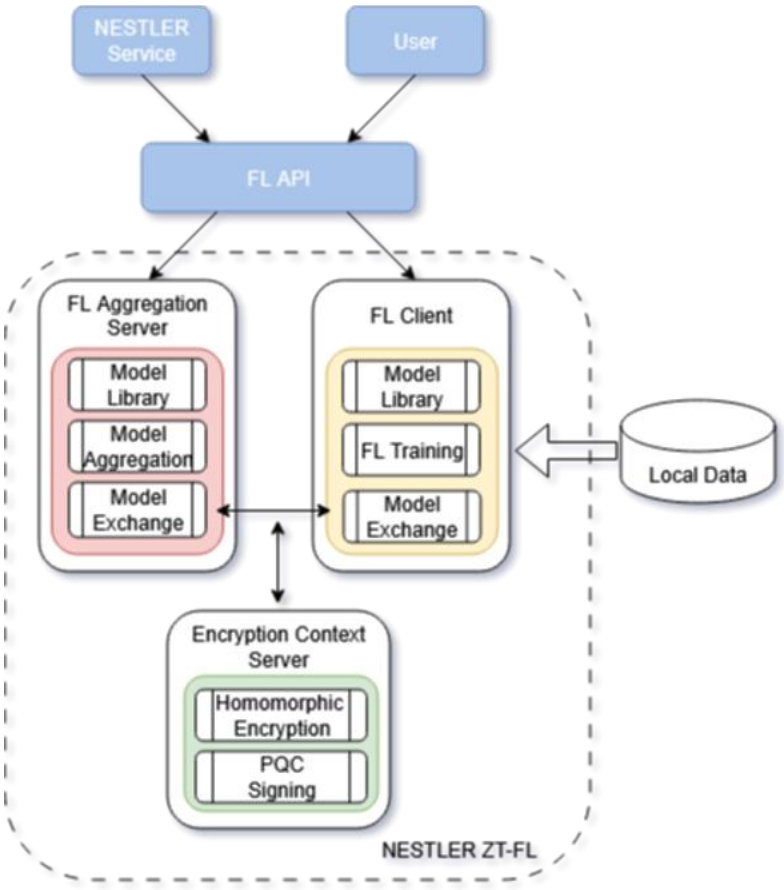

Figure 6: NESTLER AI Framework: ZT-FL stands for Zero-Trust Federated Learning that preserves privacy through homomorphic encryption and digital signing

A detailed and rather interpretable way to explain the way encryption is applied can be shown in *Figure 7*.

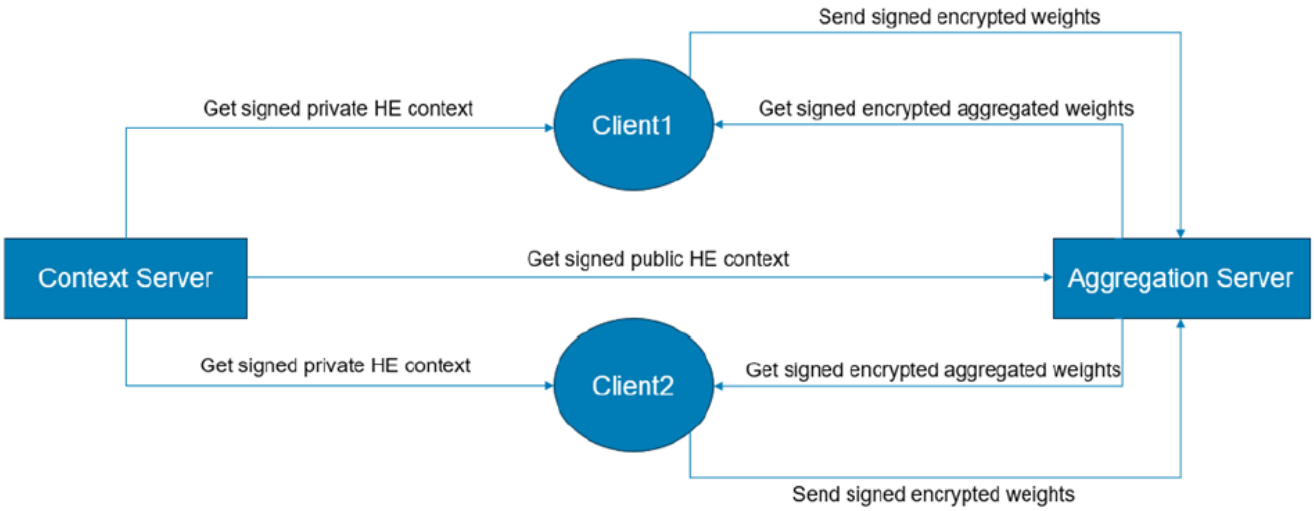

Figure 7: Core components of the proposed ZTFL framework showing the three main entities: Aggregation Server, Clients, and Encryption Context Server, along with their secure communication channels.

The core security measures that the Zero-Trust FL framework includes:

- Federated Learning Framework: Our framework is built on Flower AI. Training is performed locally on clients, with only model weights exchanged between them and the server. TLS is used as the base layer of communication security.
- Homomorphic Encryption (HE): Model weights are encrypted client-side using the CKKS scheme implemented by the TenSEAL library [14]. Encryption context is distributed through the Context Server and refreshed every training round.
- Post-Quantum Cryptographic Signing: Every communication exchange is verified through a signer verifier logic. CRYSTALS-Dilithium digital signature scheme is applied through the liboqs-python library [15].

The two types of encryption that are applied through the context server are briefly explained next.

3.1.1.1 Homomorphic Encryption

Homomorphic Encryption (HE) is employed to safeguard user data privacy during machine learning by enabling secure parameter exchange under encryption. In this setup, neither the raw data nor the model itself is directly shared, and adversaries cannot infer the original information from the encrypted parameters. This significantly reduces the risk of data leakage at the raw-data level.

In the context of Federated Learning (FL), HE is often applied to the model weights or gradients that are exchanged between clients and the central server. Instead of transmitting raw updates, each client encrypts its locally trained parameters before sending them out. The server can then aggregate these encrypted weights—thanks to the homomorphic property—without ever needing to decrypt them. This ensures that even if the communication channel or the server itself is compromised, sensitive information about local datasets cannot be reconstructed from the exchanged updates.

Types of Homomorphic Encryption:

- Partial Homomorphic Encryption: is where the cryptosystem is homomorphic for some operations. A defined operation can be performed infinite times on the ciphertext. These encryption schemes are relatively easy to design.
- Somewhat/Level Homomorphic Encryption: is where the cryptosystem is homomorphic for all operations that can be represented by a logic circuit of bounded depth (leveled: the depth depends on the key sizes). A limited number of addition or multiplication operations are allowed.
- Fully Homomorphic Encryption: is where the cryptosystem is homomorphic for all operations that can be represented by a logic circuit. An infinite number of additions or multiplications for ciphertexts is enabled.
- Bootstrapping: is the operation to reduce noise in the ciphertexts

3.1.1.2 Post-Quantum Cryptographic Signing

Post-Quantum Cryptographic (PQC) Signing is a way to ensure the authenticity and integrity of material exchanged during FL training. Unlike classical digital signatures such as RSA or ECDSA, which are vulnerable to quantum attacks, PQC signing schemes are designed to resist adversaries with access to largescale quantum computers. These schemes are based on computational problems that are believed to be hard even in a quantum setting.

PQC digital signatures do not rely on number-theoretic assumptions like factorization or discrete logarithms. Instead, they use hard problems from lattice theory, multivariate polynomials, hash-based constructions, or error-correcting codes. Their inclusion in FL frameworks ensures that exchanged messages cannot be forged or tampered with by quantum-capable adversaries. Each participant in the system signs its messages using its private key, and signatures are verified using corresponding public keys. This ensures that only authenticated entities can submit model updates or request sensitive cryptographic material.

3.2 Addressing NESTLER Use Cases

The NESTLER project aims to advance the state of privacy-preserving artificial intelligence through the adoption and enhancement of FL technologies across a range of application domains. Within this context, the current study focuses on the critical task of *Tomato Disease Classification*, a representative challenge in agricultural machine learning that has direct implications on sustainable farming practices.

This work explores the comparative performance of FL against typical, centralized machine learning approaches. Through a series of experiments, we investigate the behavior of various model architectures and optimization techniques under both federate and classical training regimes. The objective is to evaluate the trade-offs in terms of convergence, accuracy, and privacy preservation, especially under the constraints of typical agri-data environments such as decentralized data ownership, variable network conditions, and the need for compliance with data protection regulations. The results contribute not only to the scientific understanding of FL in agriculture but also to the practical roadmap for deploying FL in real-world crop monitoring scenarios.

3.2.1 Tomato Disease Recognition

The *Tomato-Village* [16] dataset, that was selected as the training and verification dataset in the Tomato Disease Classification experiments, is a publicly available collection of real-world tomato leaf images captured under natural field conditions in the Jodhpur and Jaipur districts of Rajasthan, India. Unlike other existing datasets such as PlantVillage [17], which are captured in controlled environments, Tomato-Village reflects practical agricultural scenarios, including commonly observed diseases like leaf miner, spotted wilt virus, and nutrient deficiencies. The dataset supports multiclass, multilabel, and object detection tasks, making it a versatile resource for developing and evaluating disease recognition (classification) models in realistic farming contexts. For our case we use the multiclass tomato disease classification variant of the dataset.

The *Tomato-Village* dataset used in our experiments is originally organized into three distinct subsets: training, validation, and testing. Out of a total of 4,526 labeled images, approximately 69.9% (3,163 images) are allocated to the training set, 19.9% (902 images) to the validation set, and 10.2% (461 images) to the test set. This distribution follows a commonly adopted 70-20-10 split strategy, ensuring a balanced setup that supports effective model training while allowing for robust performance evaluation and hyperparameter tuning.

The dataset consists of images from 8 classes, namely:

- Potassium Deficiency,
- Nitrogen Deficiency,
- Late_blight,
- Spotted Wilt Virus,
- Healthy,
- Leaf Miner,
- Magnesium Deficiency,
- Early_blight

Below we depict the class distribution for the whole dataset. It is evident that "Leaf Miner", "Late_Blight", and "Magnesium Deficiency" are the most represented classes while "Potassium Deficiency" and "Healthy" are the least represented. The analytical class distribution for the whole dataset can be seen in *Figure 8*.

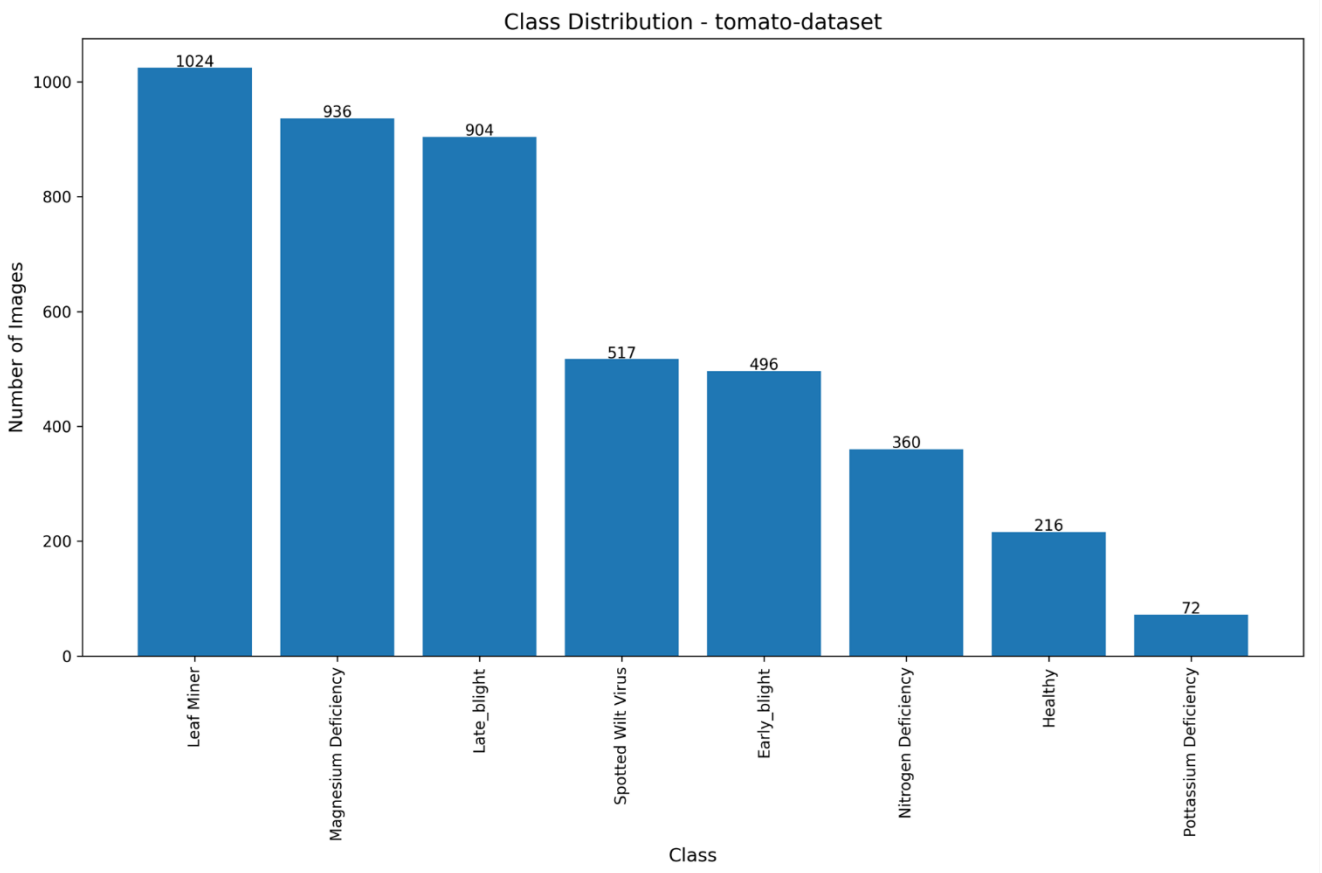

Figure 8: Number of images per class of TomatoVillage dataset.

A Vision Transformer (ViT) [18] model was employed—specifically Google's vit-base-patch16-224 [19] - to evaluate the performance of Transformer-based architectures on our image classification task. ViT represents a shift from traditional Convolutional Neural Networks (CNNs) by treating images as sequences of patches and applying Transformer encoders originally designed for natural language processing. This model divides each image into non-overlapping 16×16 patches, flattens them, and feeds them into a Transformer along with positional embeddings to retain spatial information.

The ViT-base-patch16-224 variant is a powerful yet general-purpose model with 12 transformer layers, 768 hidden dimensions, and 12 attention heads, offering a strong capacity for learning complex visual patterns. Unlike CNNs that rely on local receptive fields, ViT captures global context from the very beginning, making it especially effective in distinguishing fine-grained differences between visually similar classes. Despite its larger parameter count, it generalizes well on image datasets when pre-trained on large corpora and fine-tuned properly. In our case, its ability to leverage global features proved valuable in improving classification accuracy across diverse disease categories in the tomato dataset.

A more intuitive way to understand the architecture of the ViT model is its description in *Figure 9*. The input image is split into fixed-size patches, which are linearly embedded, and position embeddings are added to them to preserve the initial position relation among the patches, and then the resulting

sequence of vectors is fed to a standard Transformer encoder. To perform classification, we use the standard approach of adding an extra learnable classification token to the sequence

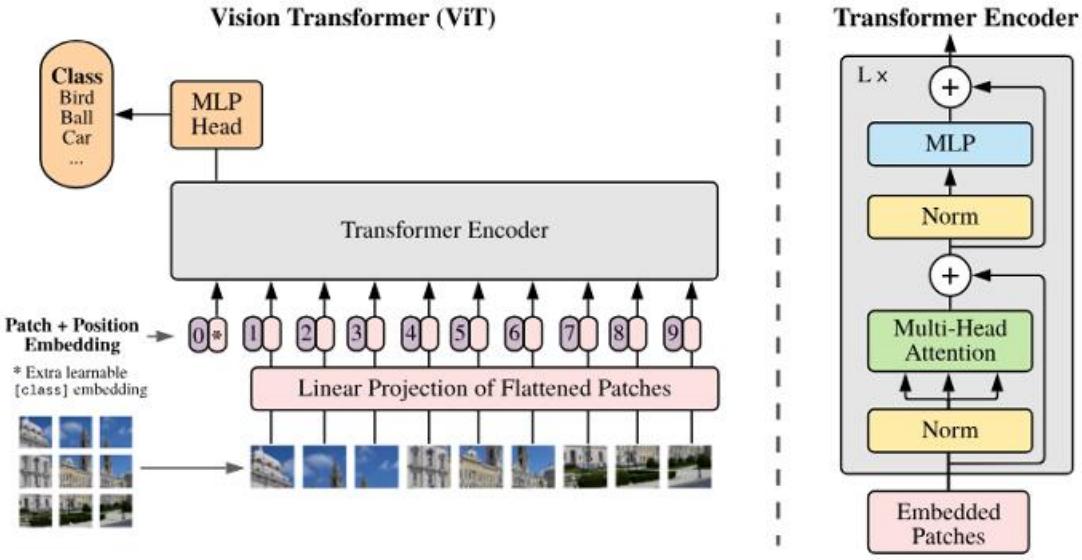

Figure 9: Image Split and classification [Alexey Dosovitskiy 2020].

A series of experiments were performed observing the behavior of each model in a typical centralized setting compared to a FL training setting. For the case of FL training, a portion of 10% of the dataset was selected and used for evaluation of the global model and the rest of the dataset was split in two groups, namely group1 and group2, with each group of data assigned to a local client of the federated system. For the case of classic (centralized) training both groups were used, and the learning rate was set to 10^{-3} with early stopping (patience = 2) and the training duration was 200 epochs, while for federated learning the experiments were set up with the same learning rate for 10 training rounds of 5 epochs each. Weighted CrossEntropy was chosen as the loss function to tackle class imbalance in this multiclass classification task. The formula that describes the Weighted CrossEntropy loss [20] [21] is the following:

$$l_n = -\sum_{c=1}^{C} w_c log \frac{exp\, exp\,(x_n, c)}{\sum_{i=1}^{C} exp\, exp\,(x_n, i)} y_n, c$$

where *x* is the input, *y* is the target, *C* is the number of classes, *w* is the weight of the class *c*.

3.2.2 Tomato Disease Recognition Results

At this subsection the performance of the two types of training, typical centralized training and FL training are reported. The aim is to showcase that even in a FL setting, where each client has less available data, the performance of the final model used for inference is similar compared to the model trained in a centralized manner, therefore it is worth using FL to preserve privacy and leverage multiple sources of data.

As can be noticed from Table 1 and Table 2, the performance of the finetuned ViT model is very good, and the false positives and false negatives are balanced considering precision and recall.

Table 1: Performance of ViT on TomatoVillage test set.

	Precision	Recall	F1	Accuracy
ViT performance	0.8356	0.8388	0.8369	0.8392

Figure 10 displays the training and validation loss. Training duration was set to 200 epochs, but it stopped at the 12th epoch using early stopping mechanism. Validation loss' curve follows the training loss' curve implying a proper way of training.

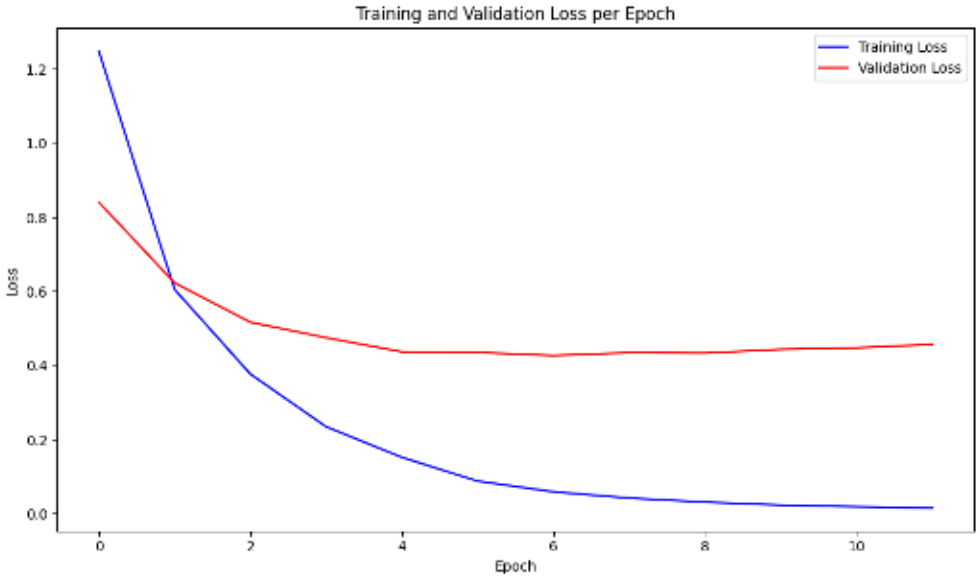

Figure 10: Training and validation loss of ViT (centralized training). Training duration was set for 200 epochs but it stopped at 12th epoch using early stopping.

Table 2 presents the performance metrics of the ViT model on the test set of TomatoVillage dataset per class to better examine which classes have higher potential than others to be recognized.

Table 2: Evaluation results of ViT model on the test set of TomatoVillage dataset. ViT model was trained in a typical manner (centralized training).

	Precision	Recall	F1
Pottassium Deficiency	0.75	0.81	0.78
Nitrogen Deficiency	0.56	0.48	0.51
Late_blight	0.87	0.89	0.88
Spotted Wilt Virus	0.82	0.88	0.85
Healthy	0.97	0.95	0.96
Leaf Miner	0.91	0.89	0.90
Magnesium Deficiency	0.96	0.98	0.97
Early_blight	0.70	0.59	0.64

Table 3 and *Table 4* present the test results of the FL trained global ViT model. Likewise with the aforementioned results of the centralized trained ViT model, the FL model has also a balance between false negatives and false positives, considering precision and recall, and an overall satisfying accuracy.

Table 3: Performance of FL-trained ViT on TomatoVillage test set.

	Precision	Recall	F1	Accuracy
ViT performance	0.8388	0.8382	0.8378	0.8382

The training and validation loss and the validation accuracy of the FL model is depicted in *Figure 11*. Validation loss has a similar curve with the training loss during the FL rounds of training, thus we conclude a good training.

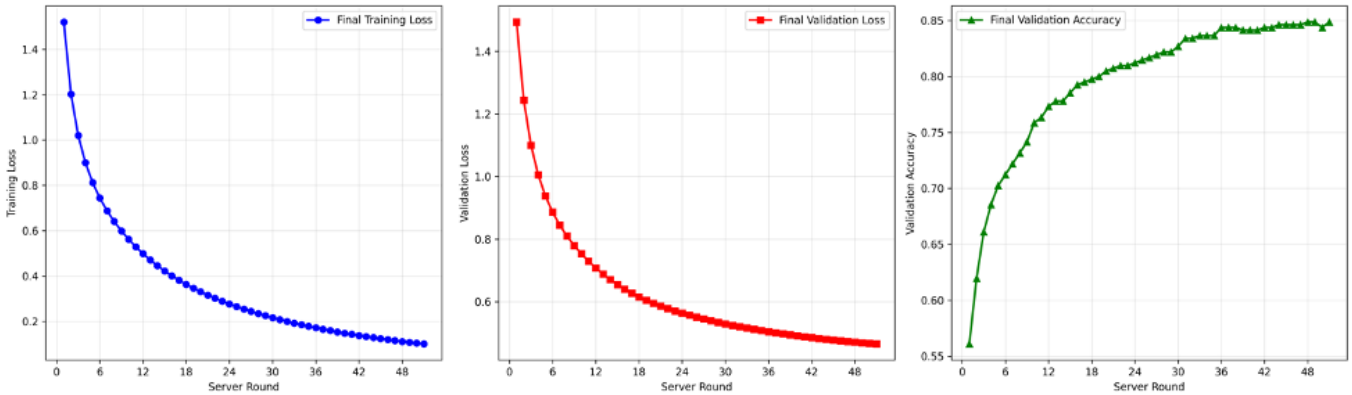

Figure 11: Training & Validation loss (left and central plot) and Validation accuracy (right plot). FL training lasted totally 75 epochs (15 rounds x 5 epochs) but the training stopped at 51st epoch using early stopping.

Table 4: Evaluation results of FL-trained ViT model on the test set of TomatoVillage dataset. ViT model was trained in a federated manner.

	Precision	Recall	F1
Pottassium Deficiency	0.81	0.88	0.84
Nitrogen Deficiency	0.58	0.67	0.62
Late_blight	0.88	0.84	0.86
Spotted Wilt Virus	0.82	0.83	0.83
Healthy	0.93	0.96	0.94
Leaf Miner	0.91	0.86	0.88
Magnesium Deficiency	0.97	0.98	0.98
Early_blight	0.70	0.64	0.67

As a conclusion, we showcased in this section that NESTLER use cases can be addressed through a FL setting of training. Here we performed Tomato Disease Recognition but also other use cases, either based on image data or IoT environmental data, can be implemented in a similar manner.

4 NESTLER Backend Agricultural Services

4.1 Weather Impact Assessment Services

4.1.1 Drought Forecast Service

This section documents the comprehensive methodology, implementation, and evaluation of the drought forecasting models. It represents the culmination of the process initiated in Deliverable D4.1 [1], which covered initial data acquisition, cleaning, and the development of a preliminary algorithmic framework. The work presented here extends through the full pipeline, including data processing, in-depth statistical analysis, model development and validation, and concludes with the successful backend integration of the models into the NESTLER platform.

4.1.1.1 Dataset

This dataset was compiled to develop AI/ML models for drought prediction and analysis. It encompasses historical daily weather data from January 1983 to December 2022 for multiple regions across four African countries:

- Cameroon,
- Nigeria,
- Rwanda, and
- Ethiopia.

The core features of the dataset consist of:

- Daily Precipitation: The amount of rainfall recorded each day.
- Daily Average Soil Skin Temperature: The average temperature of the land surface (soil skin) for each day.

The data was sourced from the NASA Power Data Access Viewer [22], a global repository for historical weather and solar climate data.

To enable supervised learning for drought modelling, the dataset has been labelled. The labels defining drought conditions for the target regions were obtained from the authoritative Global Standardised Precipitation Evapotranspiration Index (SPEI) Database.

4.1.1.2 Training and evaluation of AI/ML models

To build our drought forecast models, we rigorously tested a suite of statistical machine learning models, including K-Nearest Neighbors, Random Forest, Logistic Regression, Decision Trees, Extra Trees, Support Vector Machines, and XGBoost for each region. Statistical models are particularly well-suited for this use case as they are effective at identifying complex patterns and relationships within historical weather data, such as the correlation between precipitation, soil temperature, and subsequent drought conditions.

The models were evaluated against key performance indicators (KPIs), which we prioritized in the following order: Recall, Precision, Accuracy, and Specificity. This order is critical because, for an early warning system like drought prediction, it is more important to correctly identify all potential drought

events (high Recall) even if it means occasionally issuing a false alarm, rather than missing a true drought event, which could have severe consequences. Precision is then prioritized to ensure the alerts generated are as reliable as possible.

Our analysis revealed that the best-performing model varied from region to region, influenced by local climate patterns and data characteristics. Therefore, to build the most efficient and accurate API service, our architecture is designed to integrate the highest-performing model for each specific region.

4.1.1.3 Integration of AI/ML models

The deployment of the drought forecasting service is facilitated through a robust and scalable API built with FastAPI [23] [24], a modern Python web framework. This API serves as the central interface that integrates the machine learning models into an operational environment, enabling reliable drought predictions.

API Architecture and Endpoints:

The drought forecast service exposes several key endpoints designed for interoperability and ease of use including the following web services:

- Retrieve the list of monitoring sites
- Retrieve the drought forecast for a site
- Health endpoint

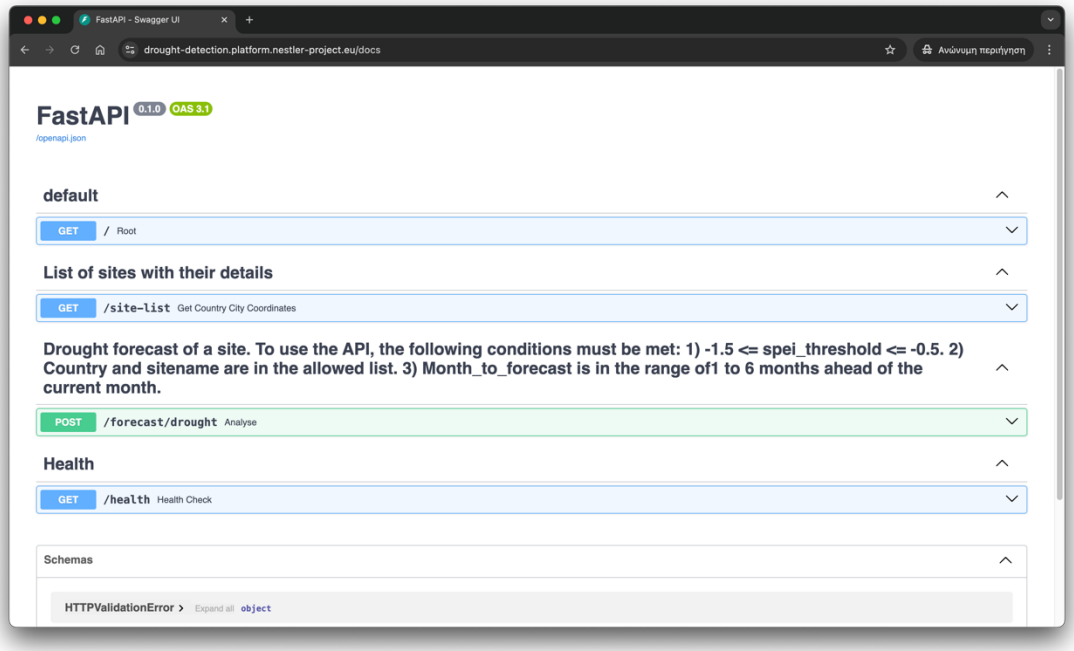

Figure 12: Documentation of Drought Forecast API.

The description of these web services is available in Annex (see section §8.1.1).

Dynamic Model Selection

A key design principle of the integration is the dynamic selection of the optimal model per region. As established during training, the best-performing algorithm (e.g., XGBoost, Random Forest, Extra Trees) varies by location. The API encapsulates this logic; when a forecast request is received, the service automatically labels the dataset, trains, loads, and runs the model that is identified as most effective for that specific city and threshold, ensuring the highest possible accuracy for every prediction.

Data Handling and Preprocessing

The service seamlessly integrates live data into the forecasting pipeline. For each request, it:

- Fetches the most recent monthly soil skin temperature and precipitation data from the NASA POWER API [25] using the site's precise coordinates.
- Executes a sophisticated preprocessing routine that aggregates historical data, handles class imbalance via oversampling, and encodes features (like converting months to binary bits) to construct the exact feature set required by the model before generating the forecast.

Packaging and Deployment in NESTLER infrastructure

The entire service is containerized using Docker [26], ensuring consistent performance across different environments. The container is built from a slim Debian-based Python image and is designed for deployment on cloud infrastructure. The application leverages AWS S3 and S3-like object storages for centralized and secure storage of all datasets and city metadata. Upon startup, the service automatically downloads the latest data from the object storage, making the deployment process agile and decoupled from the data collection lifecycle. This architecture ensures that the integration is not only functional but also secure, scalable, and maintainable within a modern MLOps framework.

For the deployment of the drought forecast service in the NESTLER infrastructure, a set of kubernetes manifests were developed:

- *configmap.yaml*: includes non-confidential configuration data in key-value-pairs,
- *secrets.yaml*: includes sensitive base64-encoded data,
- *dataset-pvc.yaml*: defines persistent volume claims (PVC) for storing datasets
- *service.yaml*: exposes a group of pods as a network service
- *deployment.yaml*: defines the desired state for stateless pods
- *ingress.yaml*: manages the network access such as DNS settings, SSL certificates and ingress routes towards the network service

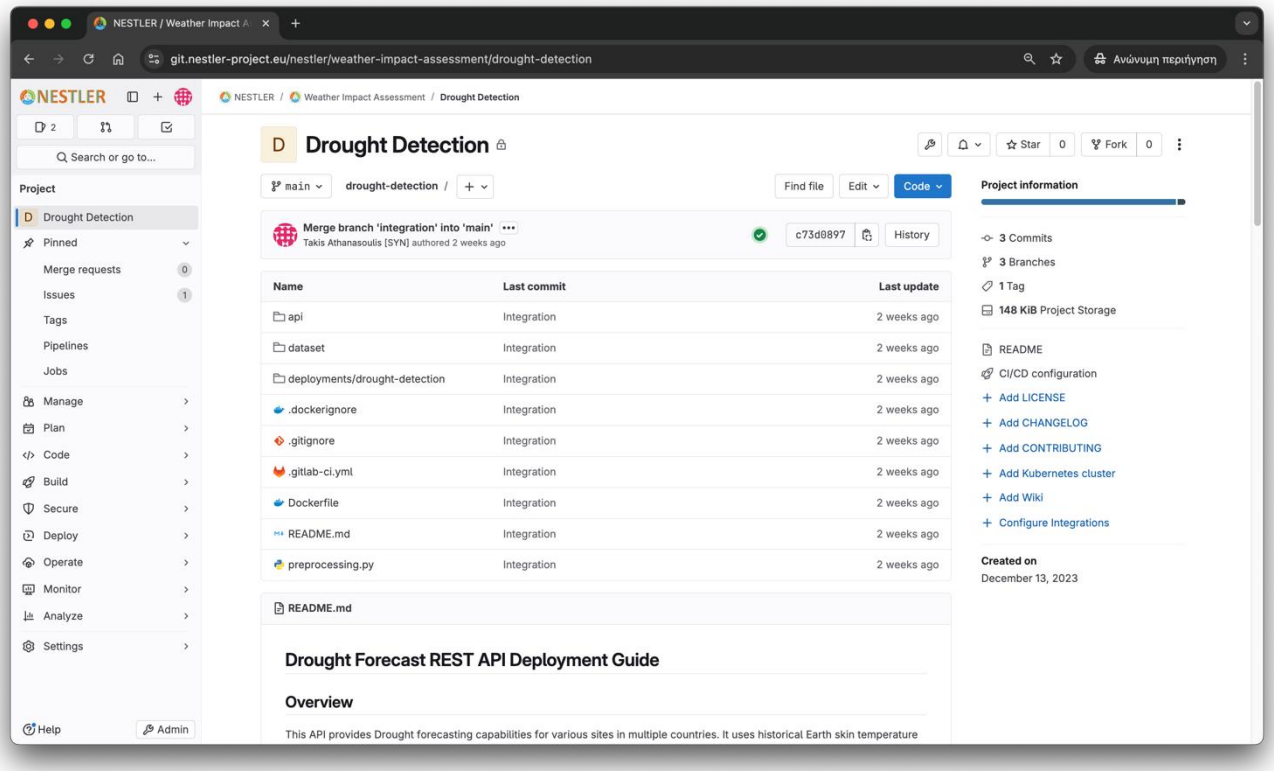

Figure 13: Drought forecast service's source code in NESTLER source code management system.

4.1.2 Flood Forecast Service

This section documents the comprehensive methodology, implementation, and evaluation of the flood forecasting service. It represents the culmination of the process initiated in Deliverable D4.1 [1], which covered initial data acquisition, cleaning, and the development of a preliminary algorithmic framework. The work presented here extends through the full pipeline, including advanced data processing, predictive model development, and validation, and concludes with the successful backend integration of the service into the NESTLER platform.

4.1.2.1 Dataset

The dataset used to develop the Flood AI/ML models was sourced from the Power Data Access Viewer [22], a global historical weather data platform developed by NASA. It comprises daily precipitation measurements spanning from January 1981 to December 2022 for multiple regions across four countries: Cameroon, Ethiopia, Rwanda, and Nigeria.

To establish ground truth for model training, the dataset was labeled based on flood events. This process involved two key steps:

- **Temporal Pattern Identification:** Histograms of cumulated monthly precipitation were analyzed to identify periods with significantly higher rainfall, corresponding to each region's known rainy seasons.
- **Threshold Determination:** Documented news reports of historical flood events in the different regions were leveraged to calibrate and set critical monthly precipitation thresholds. These

validated thresholds were then used to label the entire dataset, marking periods where precipitation levels indicated a high probability of flooding.

This method ensured that the labeling process was not solely based on statistical anomalies but was anchored in real-world occurrences of flooding.

4.1.2.2 Training and evaluation of AI/ML models

To develop our flood forecast models, we trained and evaluated multiple classical machine learning algorithms, including K-Nearest Neighbors, Random Forest, Logistic Regression, Decision Trees, Extra Trees, Support Vector Machines, and XGBoost, across each region. Statistical ML models are particularly well-suited for this use case due to their efficiency, interpretability, and strong performance with structured tabular data, such as historical precipitation records. They provide a robust foundation for identifying patterns indicative of flood events without the complexity and data requirements of deep learning alternatives.

Model performance was assessed using key performance indicators, prioritized in the following order: Recall, Precision, Accuracy, and Specificity. Recall was prioritized to minimize false negatives, ensuring that actual flood events are not missed, which is critical for life-saving early warnings. Precision was followed to limit false alarms, thereby maintaining user trust, while Accuracy and Specificity provided additional validation of overall model reliability.

The top-performing model varied by region, highlighting the importance of localized environmental factors. Consequently, to ensure optimal performance across all areas, we designed our API service to integrate the highest performing model specific to each region.

4.1.2.3 Integration of AI/ML models

The deployment of the flood forecasting service is facilitated through a robust and scalable API built with FastAPI, a modern Python web framework. This API serves as the central interface that integrates the machine learning models into the operational NESTLER platform, enabling real-time flood predictions.

API Architecture and Endpoints

The flood forecast service exposes several key endpoints designed for interoperability and ease of use including the following web services:

- Retrieve the list of monitoring sites
- Retrieve the historical monthly precipitation data for a site
- Retrieve the flood forecast for a site
- Heath endpoint

The description of these web services is available in Annex (see section §0).

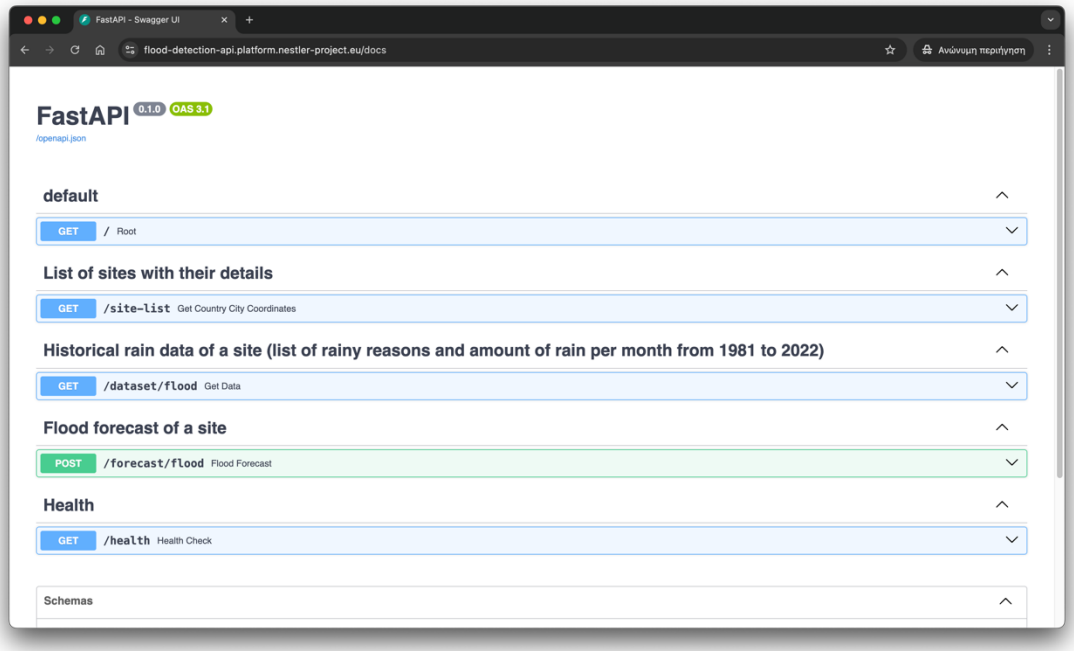

Figure 14: Documentation of Flood Forecast API.

Dynamic Model Selection

A key design principle of the integration is the dynamic selection of the optimal model per region. As established during training, the best-performing algorithm (e.g., Random Forest, XGBoost, Logistic Regression) varies by location. The API encapsulates this logic; when a forecast request is received, the system automatically labels the dataset, trains, loads, and runs the model that is identified as most effective for that specific city, ensuring the highest possible accuracy for every prediction.

Data Handling and Preprocessing

The service seamlessly integrates live data into the forecasting pipeline. For each request, it:

- Fetches the current year's precipitation data up to the present day from the Open-Meteo historical API [27] using the site's coordinates.
- Executes a sophisticated preprocessing routine that imputes any missing monthly values using the site's long-term historical median precipitation data. This ensures robust predictions even with incomplete current-year data.
- Constructs the feature set required by the model (e.g., cumulative precipitation from preceding months) before generating the forecast.

Packaging and Deployment in NESTLER infrastructure

The entire service is containerized using Docker [26], ensuring consistent performance across different environments. The container is built from a slim Debian-based Python image and is designed for deployment on cloud infrastructure. The application leverages AWS S3 and S3-like object storages for centralized and versioned storage of dataset files and metadata. Upon startup, the service automatically downloads the latest dataset from the object storage, making the deployment process agile and decoupled from the model training lifecycle. This architecture ensures that the integration is not only functional but also secure, scalable, and maintainable within a modern MLOps framework.

For the deployment of the flood forecast service in the NESTLER infrastructure, a set of kubernetes manifests were developed:

- *configmap.yaml*: includes non-confidential configuration data in key-value-pairs,
- *secrets.yaml*: includes sensitive base64-encoded data,
- *dataset-pvc.yaml*: defines persistent volume claims (PVC) for storing datasets
- *service.yaml*: exposes a group of pods as a network service
- *deployment.yaml*: defines the desired state for stateless pods
- *ingress.yaml*: manages the network access such as DNS settings, SSL certificates and ingress routes towards the network service

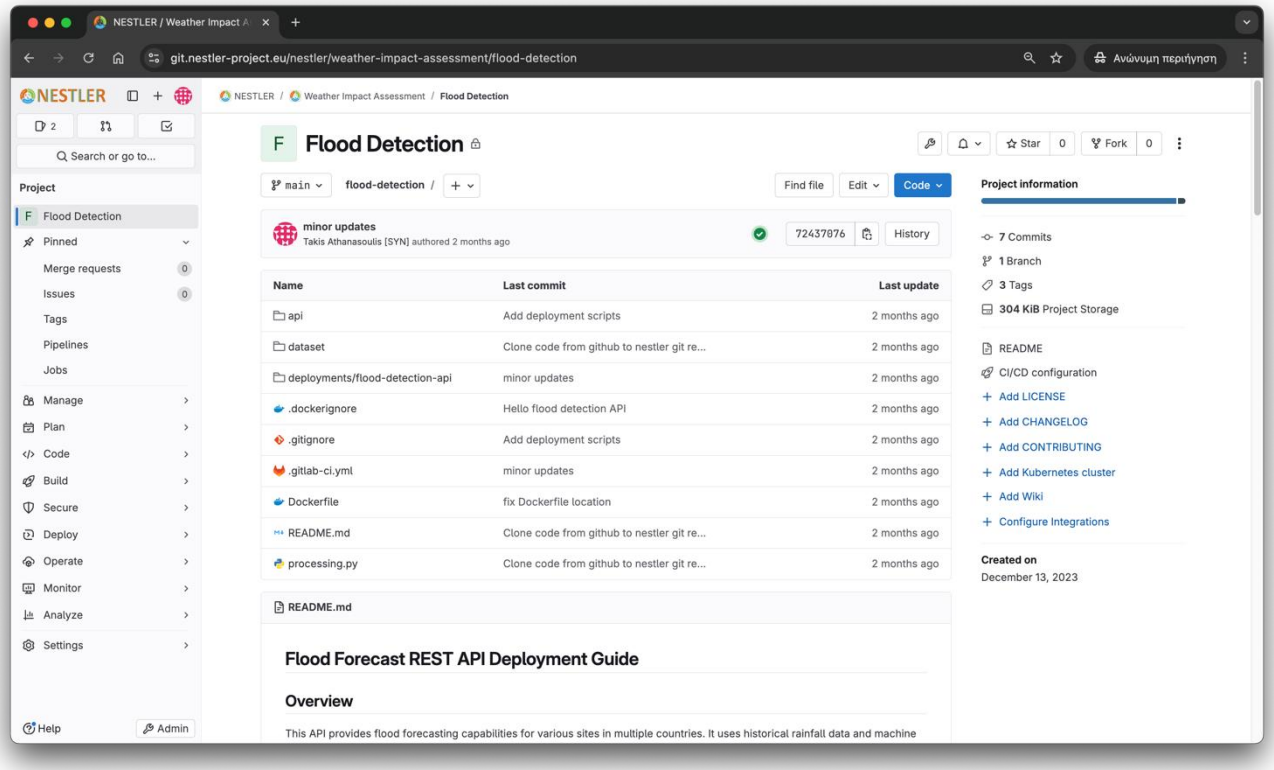

Figure 15: Flood forecast service's source code in NESTLER source code management system.

4.2 Smart Irrigation and Pest Repelling Solutions

As mentioned in Deliverable 4.1 [1], a European Parliament report accounts that farming activities consume almost 70% of global water withdrawals, reaching as much as 95% in some developing countries, while about 13,000 to 16,000 litres of water are needed to produce 1 kg of beef, depending on the production system and the feed. Spain, Italy, Greece and France together account for 88% of the total EU irrigation water [28].

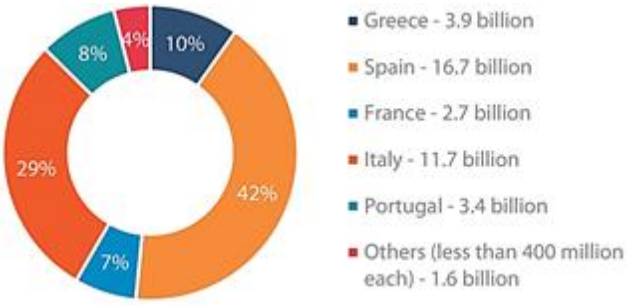

Figure 16: Volume of water for irrigation in EU.

In Africa, of the total amount of water withdrawn, 85% is used in agriculture, 9% for community use and 6% for industry [29]. As it is shown in *Figure 17*, the problem is more important in the North Africa region, however regions such as Cameroon are vulnerable.

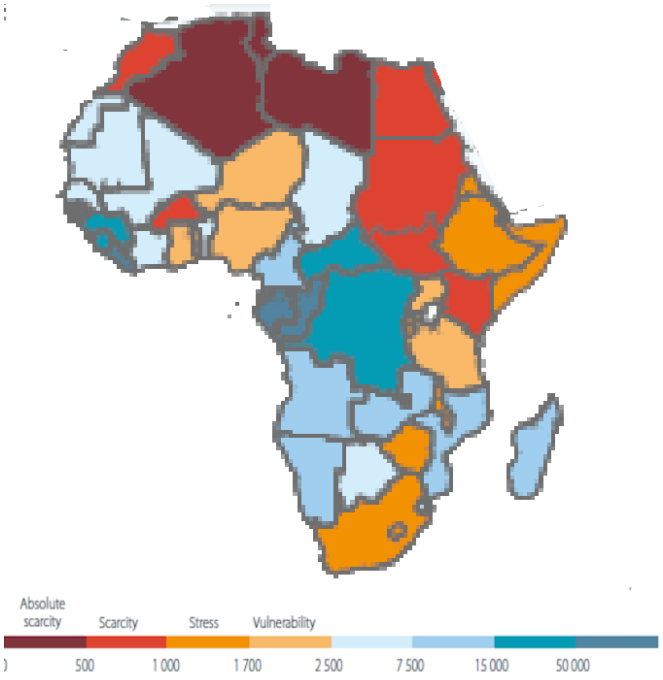

Figure 17: Total renewable water resources per capita.

The NESTLER consortium has identified farm irrigation and pest control as being pressing challenges that farmers face in Africa.

4.2.1 Smart Irrigation Service

To smartly control water usage, we have developed a smart irrigation solution. Environment and weather sensors continuously monitor and return environment and weather conditions to the NESTLER platform through the local node installed in the farm. *Figure 18* presents the high-level view of the Smart Irrigation Service. The Environment Assessor processes and analyzes the sensor readings from the SynField node while the Irrigation Controller drives the valves through the SynField API.

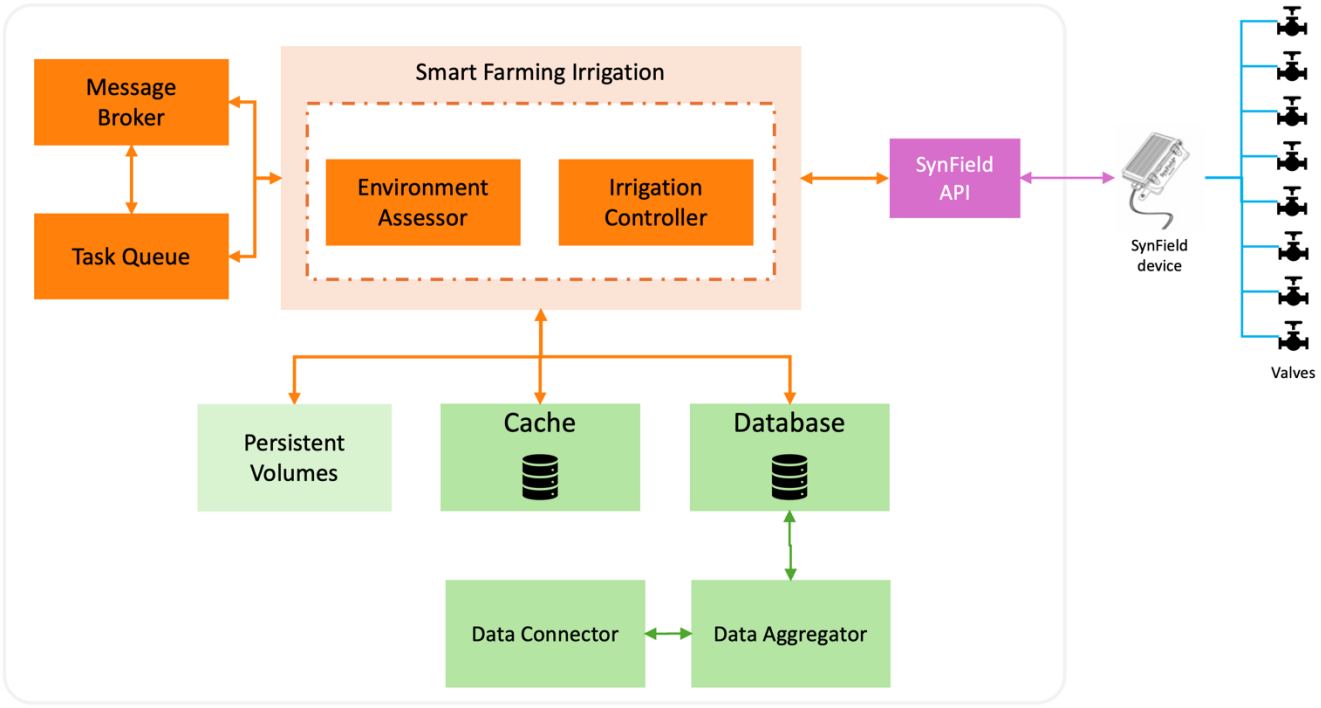

Figure 18: High-level view of the Smart Irrigation Service.

The NESTLER platform then processes the received data and determines if the irrigation system has to be started or stopped by opening or closing the water source supplying the system. The opening and closing of the water source is controlled by a solenoid valve which receives its commands from the NESTLER platform via the local device.

The NESTLER smart irrigation system is composed of:

- A SynField device (X3 model) and peripheral nodes to control the sensors and valves in the farm,
- A Weather station including a rainfall sensor, a temperature and relative humidity sensor, a wind speed/direction sensor and a pyranometer,
- A soil moisture sensor (10HS model),
- Multiple pulse-driven solenoid valves.

An irrigation policy has been applied in Cameroon (tomato plant) that says, “irrigate every day only if soil moisture value is less than SM_1% and less than SM_2% (where $SM_2 > SM_1$)”. It’s worth stressing that the soil moisture thresholds depend on the stage of tomato plant. *Figure 19* depicts the fluctuations of soil moisture at a given site over a three-day time window. Especially, the graph includes three series:

the rainfall is depicted with blue color, the soil moisture is depicted with brown color and the valve's state is depicted with black color. As shown, the irrigation process was automatically suspended on 08/09 due to the rainfall on that day, in contrast to 07/09 and 09/09.

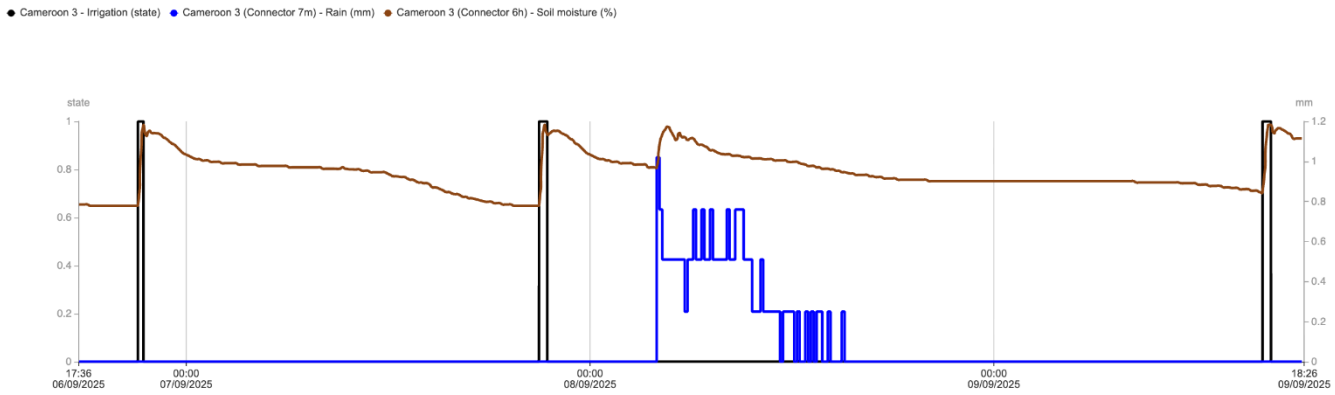

Figure 19: Soil moisture fluctuation using Smart Irrigation Service in Cameroon (tomato plant).

4.2.2 Smart Pest Detection and Repelling Solution

To smartly control pest repellents usage, we have developed an AI based mobile app which is a plant disease and pests diagnosis tool. It allows farmers to identify plant diseases or pests based on an image of a plant taken with the camera of his smartphone. It then provides treatment recommendations: type and quantity of pest repellents to use on the attacked farm.

This mobile app can run offline allowing farmers in areas without network connection to use it. It is a light app that can run on basic android smartphones such as Android 8 and above. The AI models that have been trained to identify pest infestations are integrated into it. This integration allows users to harness the power of advanced machine learning techniques for on-the-spot pest detection and management in agricultural settings.

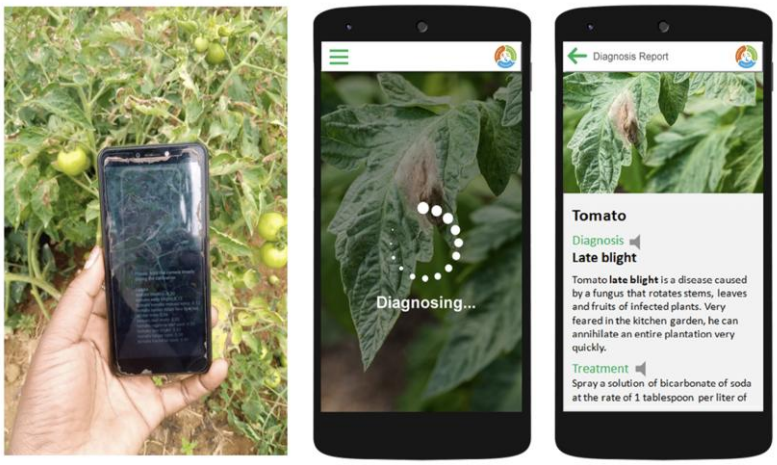

Figure 20: NESTLER mobile application to identify disease on a tomato plant.

4.3 Automated Monitoring Services for Crop Yield Quality, Livestock Wellbeing and Insect Population

4.3.1 Tomato Disease Recognition Service

As outlined in section §3.2, NESTLER works on the tomato disease recognition models. *Figure 21* depicts the high-level architecture of the Tomato Disease Recognition Service based on existing ML model. This service consists of the ML model, the event processor and the disease recognition accessor. Upon receiving a request (e.g. an image), the service schedules the request processing through the Event Processor and generates a specific task using the message broker and task queue components. Then, the Disease Recognition Assessor provides as output indicators with respect to the disease recognition in the tomato image. The Tomato Disease Recognition Service exposes a web service that is available within NESTLER backend and frontend services.

NGINX [30] is used as Ingress, Gunicorn [31] is used as WGSI Server, key-value Redis database [32] is used for caching while Tomato Disease Recognition Service has been implemented in Python programming language using the Django and Django Rest Framework. RabbitMQ [33] is used as message broker and Celery [34] acts as distributed task queue.

Integration of the Tomato Disease Recognition Service with the IAM API is scheduled to be completed by the end of the project.

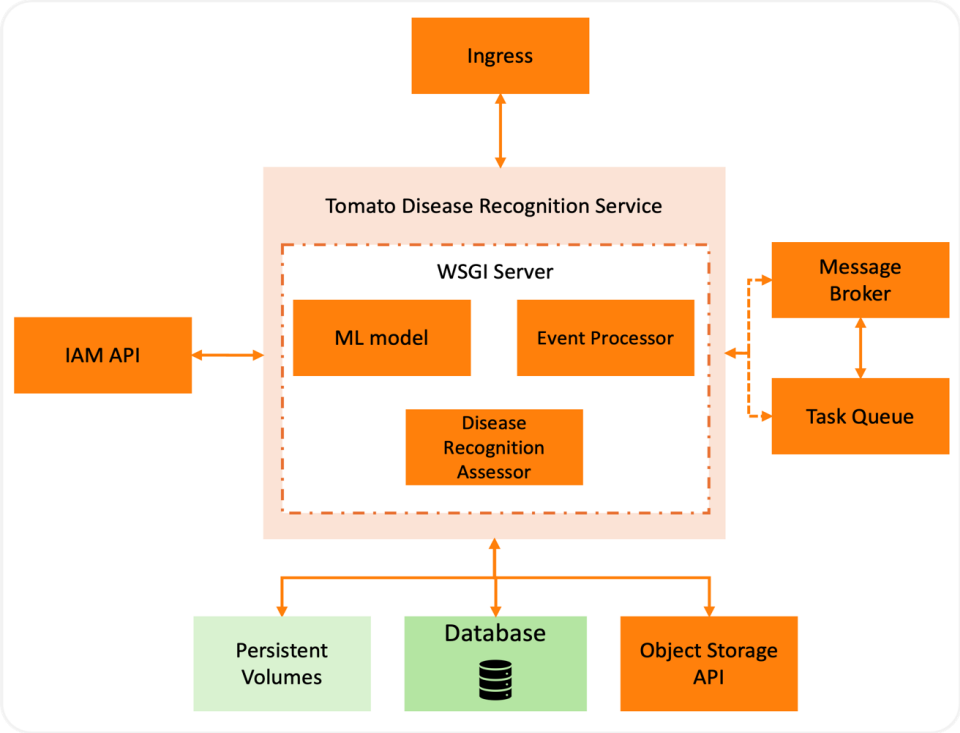

Figure 21: High-level view of the Tomato Disease Recognition Service.

4.3.2 Zoonotic Disease Outbreak Service

One of the fundamental backend services in the NESTLER platform is the Zoonotic Disease Outbreak Service. The regions of each pilot country have been stored in the NESTLER database including attributes such as its ID and name, the country and country code, its geometry and its timezone. Each country's region has been split in smaller rectangles, called tiles, that have been stored as well in the NESTLER relational database. Each tile includes attributes such as its ID, the country code and its geometry.

The Zoonotic Disease Outbreak Service is composed of the Multi-modal Data Processor component, which is responsible for aggregating diverse datasets within the same time window and geographic location. These datasets include flood and drought predictions per tile, as provided by the relevant Weather Impact Assessment services, insect trap captures, their corresponding analyses related to if those insects are disease-vectors, environmental conditions etc. The Zoonotic Disease Outbreak Detector component processes and analyzes the aggregated data within the same time window and geographic location and identifies a possible outbreak of a zoonotic disease. The Risk Assessor translates each identification in a risk indicator (low, moderate, high, critical) per location and time window that is stored in the database.

Finally, the zoonotic disease outbreak risk is available within the NESTLER platform through the GIS API. Each WMS request in GIS API includes the disease of interest, the CRS, the bounding box of the location of interest and the time window. The end-users of the NESTLER platform access the zoonotic disease outbreak risk indicators through an interactive map that depicts the risk using different colors on a daily basis. *Figure 22* depicts the high-level architecture of the Zoonotic Disease Outbreak Service.

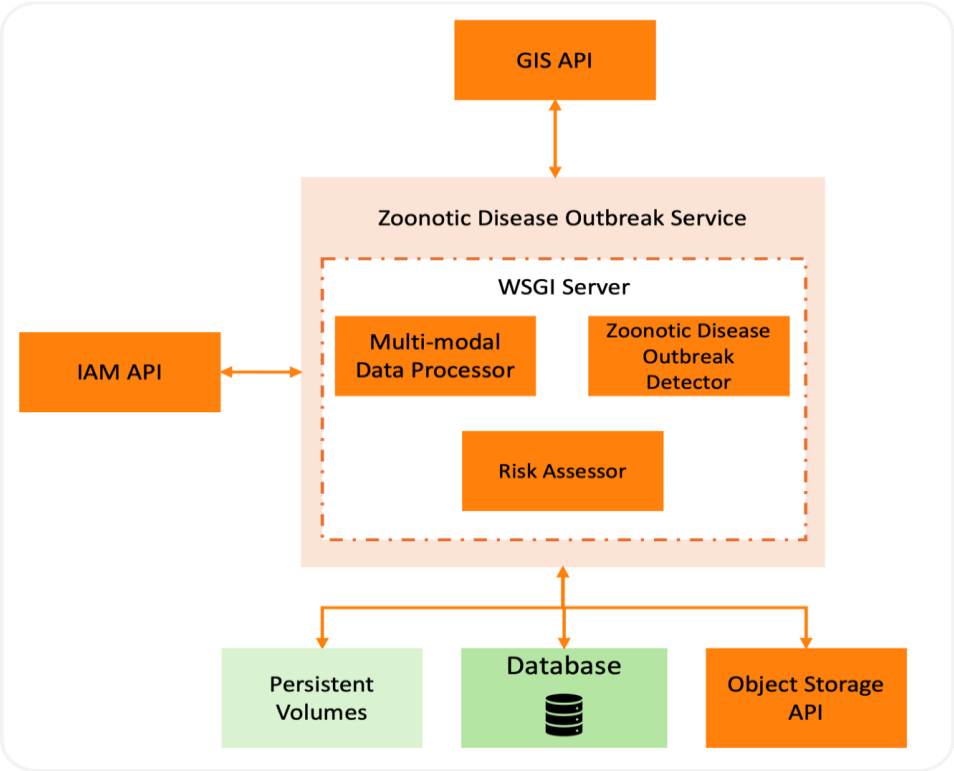

Figure 22: High-level view of the Zoonotic Disease Outbreak Service.

Documentation of the Zoonotic Disease Outbreak Service web services is presented in Annex.

4.4 Economic Risk Assessment Models for Predicting the Yield Quality

This section documents the complete development methodology, implementation and evaluation of the economic risk assessment model. In deliverable D4.1 [1], the data acquisition, cleaning and preliminary algorithmic framework were outlined, while this deliverable includes the entire process, from processing and statistical analysis to model development, validation and backend integration to the NESTLER platform. A high-level overview of the economic risk assessment can be seen in *Figure 23*.

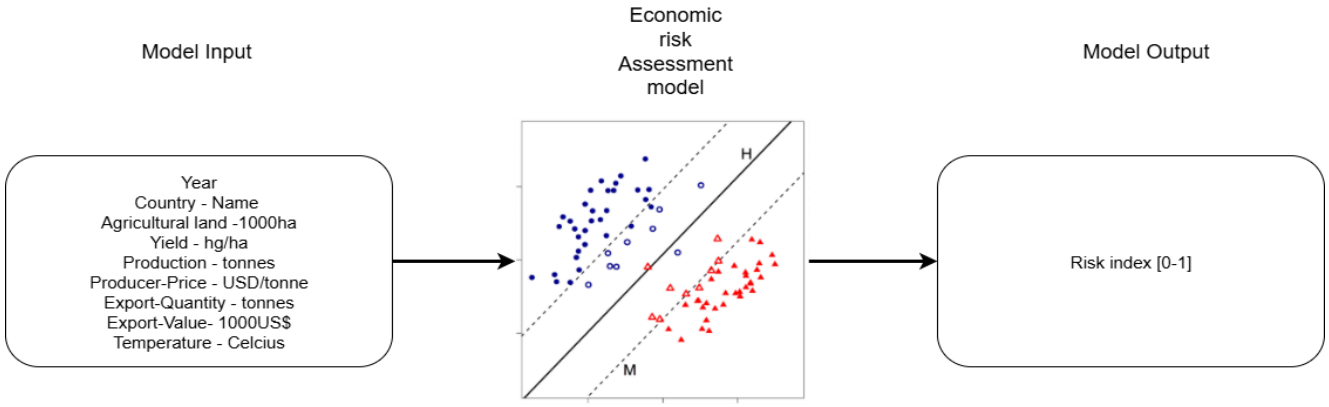

Figure 23: High-level view of the Economic Risk Assessment Model service.

4.4.1 Dataset and Preprocessing

4.4.1.1 Data Acquisition

The dataset for the development of the model was sourced from the Food and Agriculture Organization (FAO) of the United Nations [35], which provides comprehensive agricultural data. The analysis focused on the six African countries of interest: Cameroon, Ethiopia, Kenya, Nigeria, Rwanda, and Uganda and the seven major crops: bananas, cassava, coffee, cowpeas, soybeans, yams, and maize.

The dataset incorporates attributes across multiple domains:

- Agricultural land (country area, land area, agricultural area)
- Crop yields and production (yield per hectare, total production)
- Producer prices (USD per tonne)
- Export quantities and values (tonnes and USD)
- Climate change indicators (temperature variations)

4.4.1.2 Handling Missing Values, Outliers detection and Normalization

Missing values were a major challenge in the datasets so the columns with more than 55% missing values were removed and interpolation methods were applied. Initially, the Least Square Approximation (LSA) method was tested but rejected due to unrealistic values when polynomial degree higher than 2. Then, the Cubic Spline Interpolation (CSI) was chosen as the most reliable, with adaptive rolling windows for clustered missing values, while the Moving Average (MA) method was used as a fallback method, though less accurate than CSI.

Outliers were detected using the Interquartile Range (IQR) method. Instead of discarding them (due to the limited dataset), they were imputed via Linear Regression models, replacing extreme values with predicted values consistent with the overall trend. This ensured integrity without losing valuable data.

All numerical variables were rescaled to a 0-1 range and normalized using the L2 norm, ensuring comparability across attributes and stabilizing the training of machine learning models.

4.4.2 Statistical and Time-Series Analysis

Visual inspection of attributes was first applied to identify data patterns, outliers and inconsistencies, using boxplots, distribution plots and time-series charts to guide preprocessing. To assess relationships between attributes, Euclidean Distance was employed to quantify similarity between time series, while Cross-Correlation measured alignment over time, confirming strong similarities between pairs such as Production and Yield or Export Value and Export Quantity. Dendrogram analysis was also used to cluster attributes with comparable behaviours. Finally, stationarity tests were conducted to evaluate the temporal stability of the series, ensuring proper treatment in regression and predictive modelling.

4.4.3 Economic Risk Model

After preprocessing, each product dataset was transformed into a Probability Dataset, where the likelihood of attribute values was calculated. This was then converted into an Entropy Dataset using Shannon entropy:

$$H(X) = -\sum_{i=1}^{n} P(x_i) log_2(P(x_i))$$

where $P(x_i)$ is the probability of a given value.

Entropy quantifies the uncertainty associated with agricultural attributes, reflecting both unpredictability and risk.

Then, a new Target column was created as the sum of entropies across attributes, representing the **risk index**:

$$Target = AA_{Entropy} + P_{Entropy} + PP_{Entropy} + EV_{Entropy} + EQ_{Entropy} + T_{Entropy}$$

Where:

- *AA* stands for *Agricultural Area*
- *P* stands for *Production*
- *PP* stands for *Producer Price*
- *EV* stands for *Export Value*
- *EQ* stands for *Export Quantity*
- *T* stands for *Temperature Change*

Finally, in order to manage correlated attributes, Principal Component Analysis (PCA) was applied generating composite features that captured maximum variance while reducing dimensionality.

4.4.4 ML Models

The machine learning phase began with Linear Regression, used as a baseline model to capture simple linear relationships, followed by Polynomial Regression, which extended the approach to account for non-linear behaviours in the data. To further enhance predictive capability, Support Vector Regression (SVR) was tested as a more advanced method, leveraging kernel functions to model both linear and non-linear dependencies effectively. Model development followed a standard training and testing split, with 80% of the data (1041 samples) allocated for training and 20% (261 samples) reserved for testing, while preprocessing and normalization ensured consistent and reliable performance across all models.

4.4.5 Results and Evaluation

To assess the predictive capability of the developed economic risk assessment models, a rigorous evaluation was carried out using both statistical and machine learning approaches. For this purpose, two key metrics were selected: R^2 (Coefficient of Determination), which indicates the proportion of variance in the target variable explained by the model and Mean Absolute Error (MAE), which reflects the average absolute difference between predicted and absolute values.

Table 5 provides a concise summary of the performance of three distinct machine learning models. The SVR model emerges as the most effective, boasting a notably high R-squared score of 0.87 and a low MAE of 0.07, underscoring its promising predictive capabilities. Polynomial Regression, while not surpassing SVR, exhibits comparable performance. In contrast, the Linear Regression model lags behind, recording a lower R-squared score of 0.66 and a higher MAE of 0.105, positioning it as the least performing model in this evaluation.

Table 5: Economic risk assessment models' performance.

Model	R^2	Mean Absolute Error
Linear Regression	0.66	0.105
Polynomial Regression	0.82	0.070
Support Vector Regression	0.87	0.070

4.4.6 Integration of Economic Risk Assessment Model into the NESTLER Platform

The work carried out in Task 4.3 has resulted in the development and validation of the economic risk assessment model. The next step is the integration of this model into the NESTLER platform, so that its functionalities and outputs can be made directly accessible to end users through the project's dashboard. To support the integration of the economic risk assessment model within the NESTLER platform, an API (web service) has been developed. API documentation is currently available in the link https://speca.io/eBOS/nestler?key=6e5108b4c642d1fc4cdff6e8e0dc0697 (see *Figure 24*).

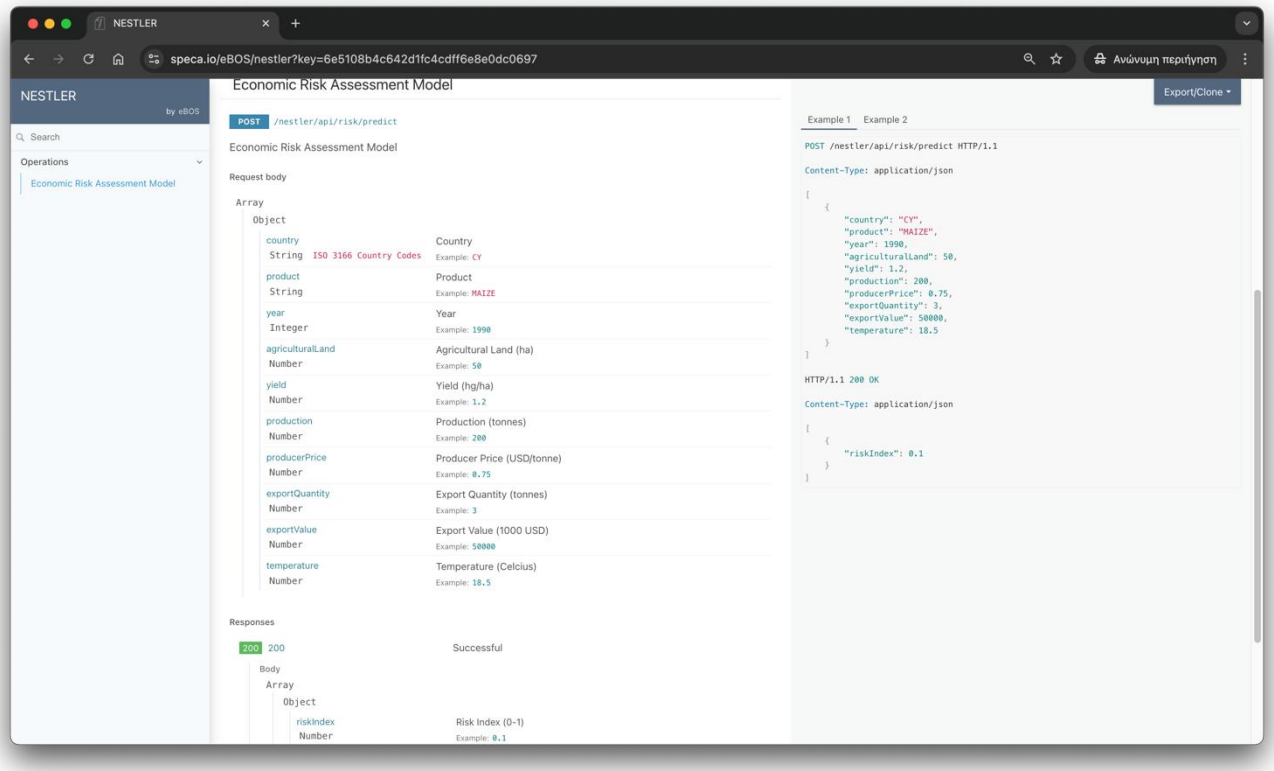

Figure 24: Documentation of the Economic Risk Assessment model API.

A draft mockup has been prepared to illustrate how the model will be visualised in the platform (*Figure 25*). The mockup serves as an initial step towards translating the model into a user-facing service, demonstrating how results such as the risk index can be displayed in a clear and intuitive manner. The full integration of the model will be presented in the final deliverable of WP4, namely D4.4 “NESTLER integrated platform release”.

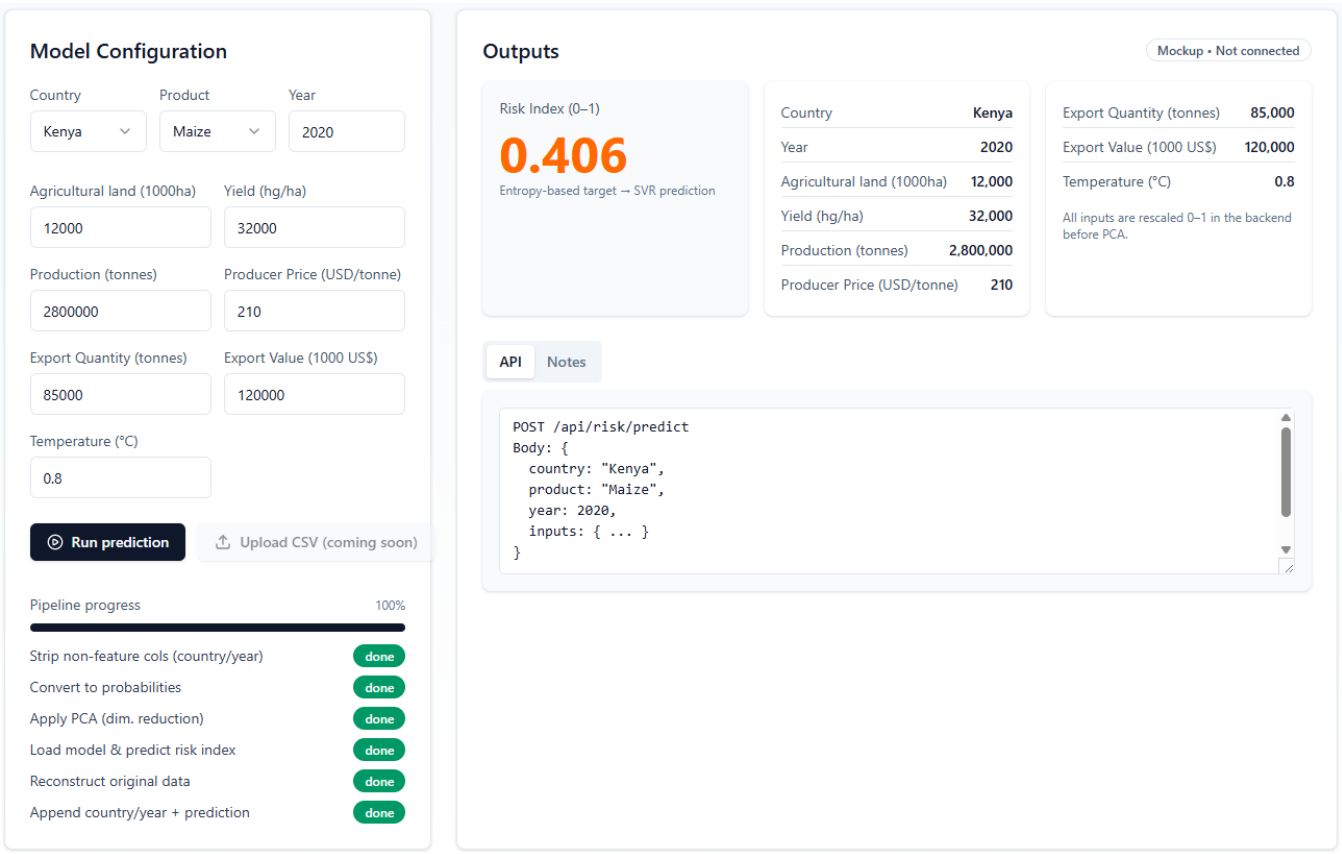

Figure 25: Draft mockup of the Economic Risk Assessment model integration in the NESTLER dashboard.

4.5 NESTLER API

As outlined in section §2.3, the NESTLER backend services provide the NESTLER API, which acts as a communication layer between backend services, frontend services, and the database. *Figure 26* depicts high-level architecture of NESTLER API. Ingress defines rules for routing of external requests to the NESTLER API based on hostnames while Application server implements the Web Server Gateway Interface (WSGI) standard which defines how web applications communicate with the API. Upon receiving a request, the API executes the corresponding operation on the PostgreSQL relational database. For optimized performance in data fetching, the API initially checks the cache; if the data are not present, it retrieves them directly from the database. NGINX [30] is used as Ingress, Gunicorn [31] is used as WGSI Server, key-value Redis database [32] is used for caching while NESTLER API has been implemented in Python programming language using the Django and Django Rest Framework. RabbitMQ [33] is used as message broker and Celery [34] acts as distributed task queue.

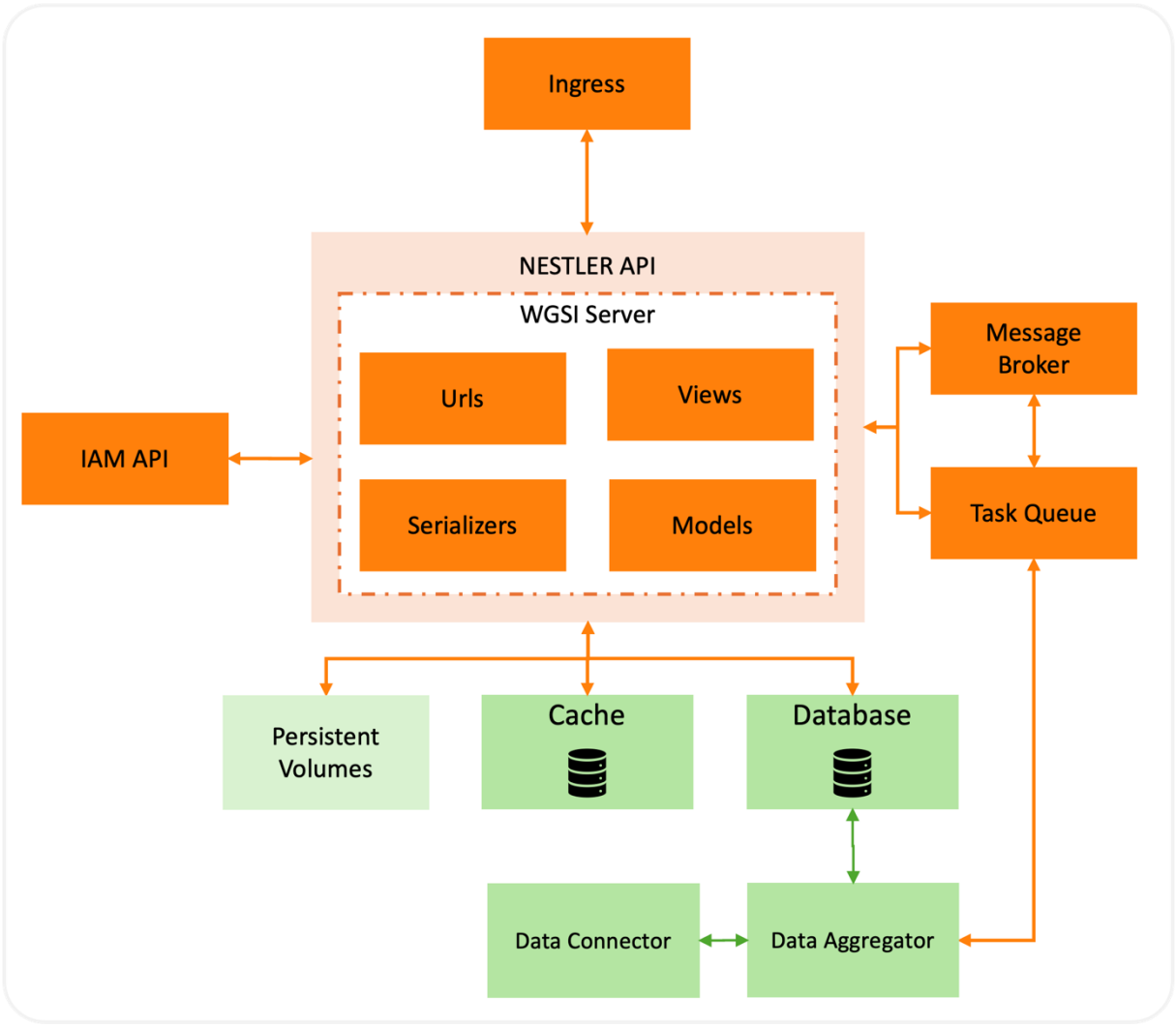

Figure 26: High-level view of NESTLER API

The API is structured into two categories of web services: those that manage systemic data i.e., data that do not change frequently over time (let's say *NESTLER Generic API*) and those that handle service- and user-related data (let's say *NESTLER core API*), which fluctuate over time. *Swagger* framework [36], based on the *OpenAPI* standard [37], is used to provide online documentation for NESTLER API.

Figure 27 depicts an overview of NESTLER Generic API's documentation and the detailed analysis of the list of web services is presented in Annex (see section §8.3).

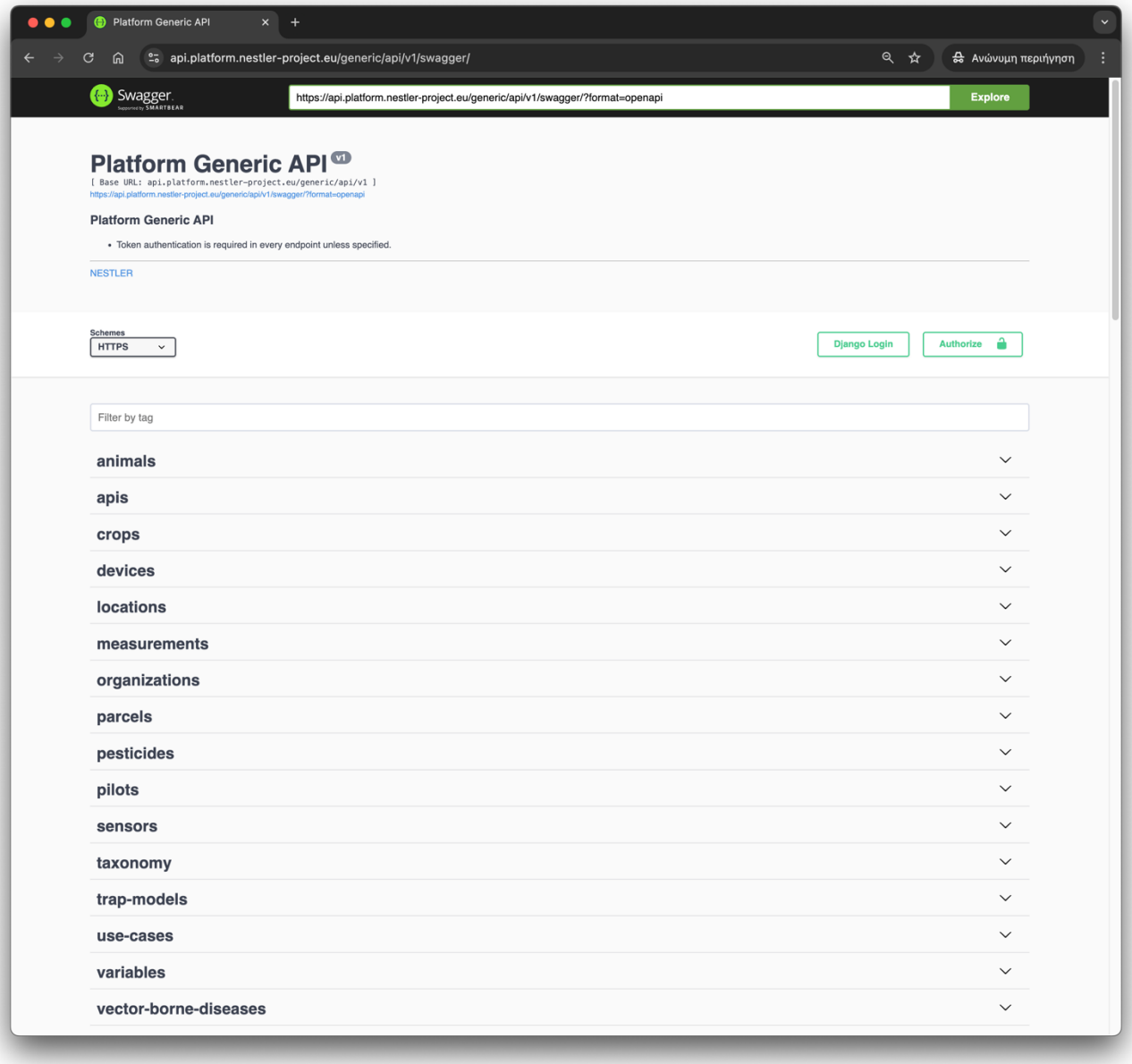

Figure 27: Documentation of the NESTLER Generic API (systemic data)

Figure 28 depicts an overview of NESTLER core API's documentation while the detailed analysis of the list of web services is presented in Annex (see section §8.3).

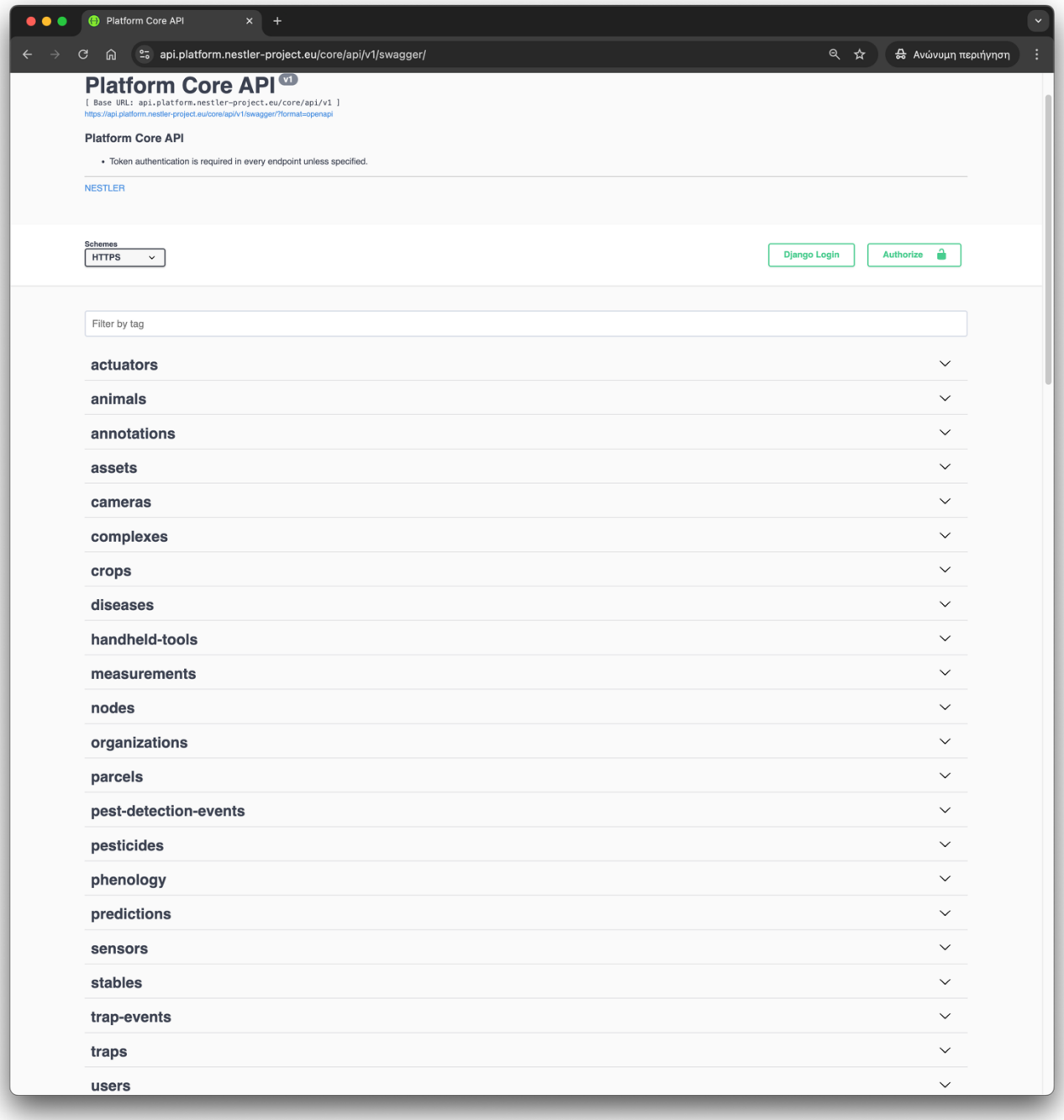

Figure 28: Documentation of the NESTLER Core API (service- and user-related data)

The deployment of the NESTLER API in the infrastructure is applied through helm charts and a set of kubernetes manifests.

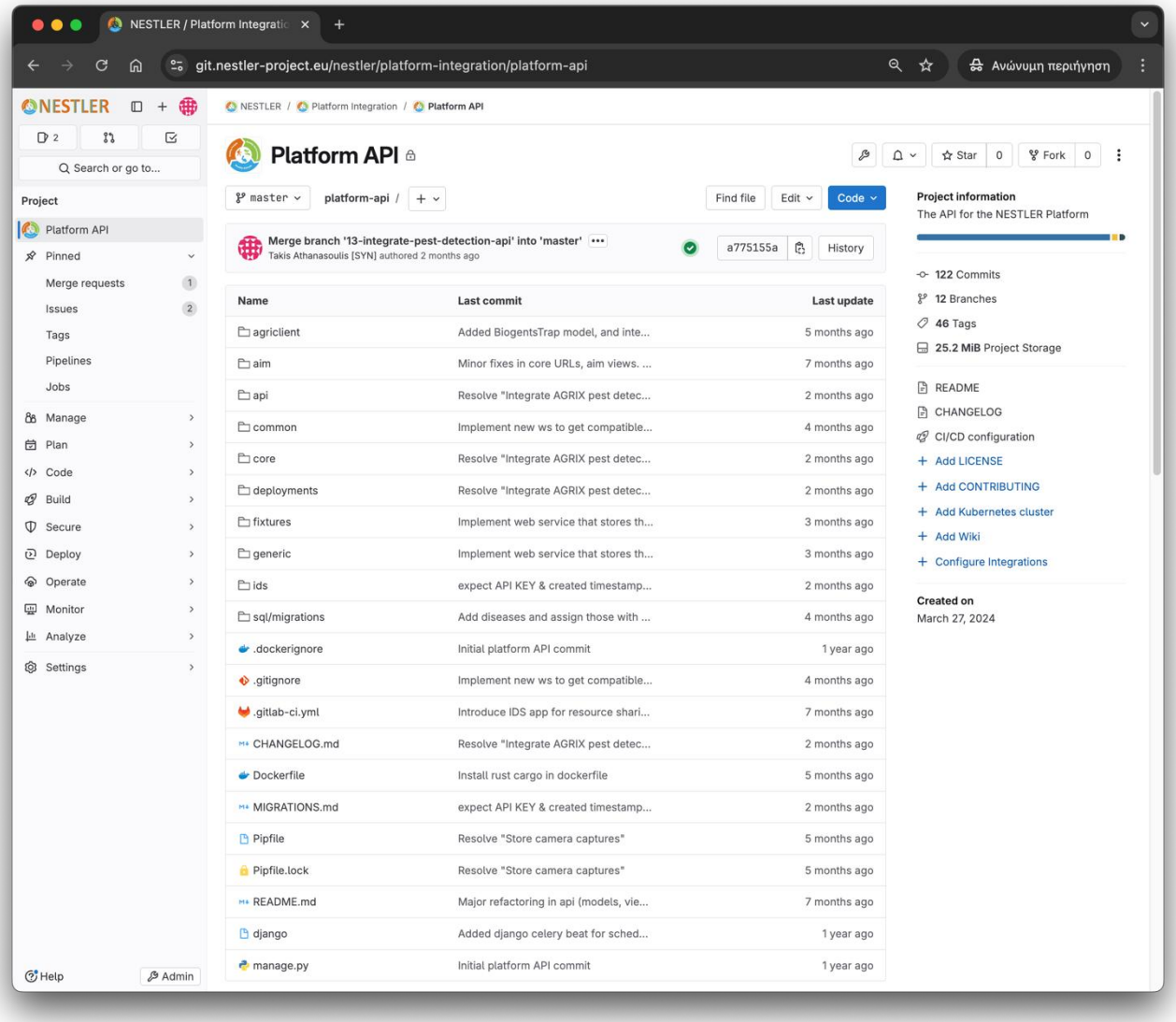

Figure 29: Documentation of NESTLER API source code structure in code repository

4.6 GIS API

GIS (or Geoserver) API provides standardized web services for sharing and accessing geospatial data using protocols defined by the Open Geospatial Consortium (OGC). Three of the most widely used services are WMS (Web Map Service), WFS (Web Feature Service), and WMTS (Web Map Tile Service). A WMS request returns dynamically rendered map images, making it ideal for visualization in GIS clients and web applications. In contrast, WFS provides direct access to the underlying vector data (points, lines, polygons) along with their attributes, enabling querying, filtering, and editing within GIS or data analysis tools. WMTS complements WMS by offering cached, pre-rendered tiles at fixed zoom levels, delivering much faster performance for interactive maps at scale. OGC services allow NESTLER developers to integrate GIS-hosted data into NESTLER dashboard and other applications. WMS is often used to embed

styled maps in dashboards, WFS enables downloading or analyzing spatial datasets programmatically, while WMTS powers fast, tile-based navigation in web mapping interfaces.

Geoserver provides also a REST API [38] through which frontend services can retrieve information about an instance and make configuration changes. Using the REST interface's simple HTTP calls, frontend services are able to configure GeoServer without needing to use the NESTLER GIS Admin Console. Also, the REST API provides the list of workspaces, WMS stores, the WMS/WMTS layers as well as details per WMS layer.

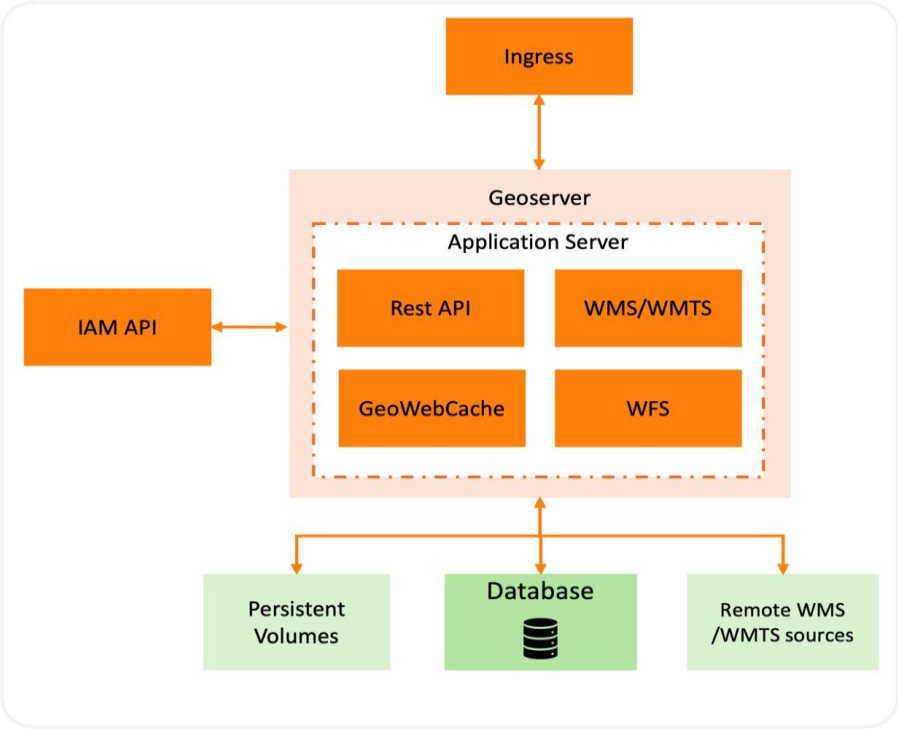

Figure 30: High-level view of GIS API

Geoserver has been implemented in Java programming language using Spring Framework and Tomcat application server. *Figure 30* depicts high-level architecture of NESTLER API. Description of indicative web services is available in section §8.4 while Geoserver web page [38] presents in detail its documentation.

4.7 Identity & Access Manager API

Identity & Access Manager based on the open-source tool Keycloak that provides authentication, authorization, and user federation services for modern applications and services. It supports widely adopted security standards such as OpenID Connect (OIDC), OAuth 2.0, and SAML 2.0, enabling seamless integration with web, mobile, and enterprise systems. Through its single sign-on (SSO) capabilities, social login integration, and fine-grained authorization features, Keycloak simplifies the management of user identities across distributed architectures.

To facilitate automation and management, Keycloak exposes a comprehensive REST API. This API allows NESTLER administrators and developers to perform operations such as creating and managing users, groups, roles, clients, and realms, as well as handling sessions, tokens, and authentication flows. Alongside the Admin REST API, Keycloak provides an Account REST API for end-users to manage their profiles, sessions, and credentials. Complementary to these, backend and frontend services interact with Keycloak through its OIDC/OAuth2 endpoints for token issuance, introspection, and revocation, as well as SAML endpoints for federation. This API-driven approach enables the integration of Keycloak into CI/CD pipelines, microservices, and cloud-native platforms, making it a robust solution for securing modern digital ecosystems.

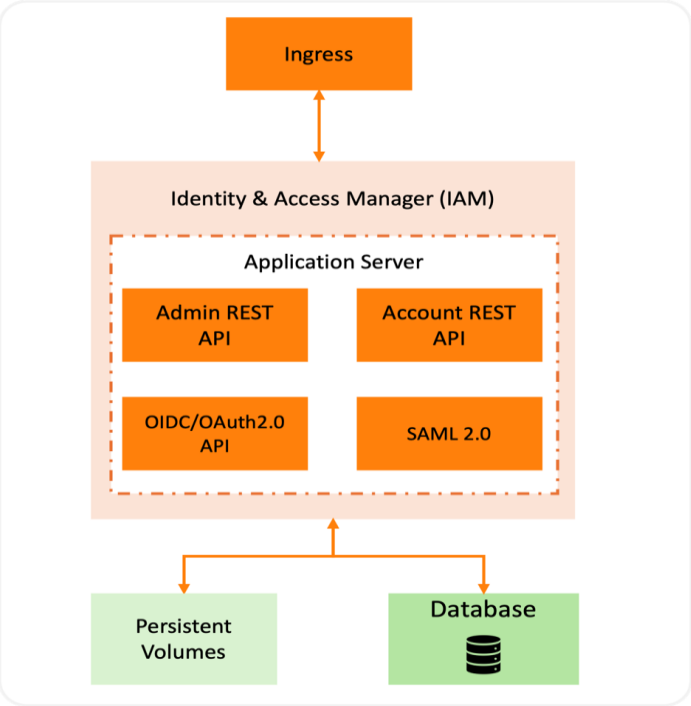

Figure 31: High-level view of Identity & Access Manager APIs

Keycloak has been implemented in Java programming language using Jakarta EE [39], Quarkus [40], JAX-RS and Hibernate [41] (JPA) frameworks and Wildfly [42]/Quarkus are used as application server. PostgreSQL database is used for persistent storage. *Figure 31* depicts high-level architecture of Identity & Access Manager APIs. The API documentation is presented here [43] and here [44].

5 NESTLER Integration Framework

This section serves as an updated version of section 4 of Deliverable D4.1 [1], and focuses on the definition and description of a structured, yet flexible integration framework, able to cater for all NESTLER's subcomponents. Especially, this section provides description of NESTLER's integration framework that serves as its enabler, consisting of various utilities and workflows that facilitate essentials for the realization of every comprising component, i.e.:

- Source Code Management
- DevSecOps Approach
- Continuous Integration and Continuous Deployment (CI/CD)
- Issue Tracking System
- Containerization Tools

In addition, an overview of the deployment platform in use is presented. We plan to approach the integration framework as a constantly adapting and evolving part of NESTLER throughout the entirety of the project's duration, with updates, in-depth descriptions, and documentations to follow as needed, according to the requirements of the NESTLER components to be realized and integrated, thus forming NESTLER platform as-a-whole. The final version will be described in the final deliverable of WP4, namely D4.4 "NESTLER integrated platform release".

5.1 Source Code Management

Source code management (SCM) refers to tracking modifications on the source code, possibly applied in parallel over a common code base. Besides the obvious benefits of tracking, SCM is of vital significance when resolving conflicts in parallel development streams. The easy access to code, snippets or changes on different code versions facilitates the collaboration among developers, typically working remotely. The main and fundamental characteristics (offerings) of SCM are summarized in the following:

- Complete long-term change history of every tracked file
- Branching and merging features
- Traceability

Source code management in the context of NESTLER will be based on git [45], given that its structure is aligned with the project needs for parallel development in multiple branches, and code integration. Indeed, in the context of NESTLER, a self-hosted GitLab instance [46] has been deployed to cater for the essential features of SCM, as well as issue tracking and CI/CD, which will be described in the following. NESTLER's GitLab instance is hosted at URL:

- https://git.nestler-project.eu.

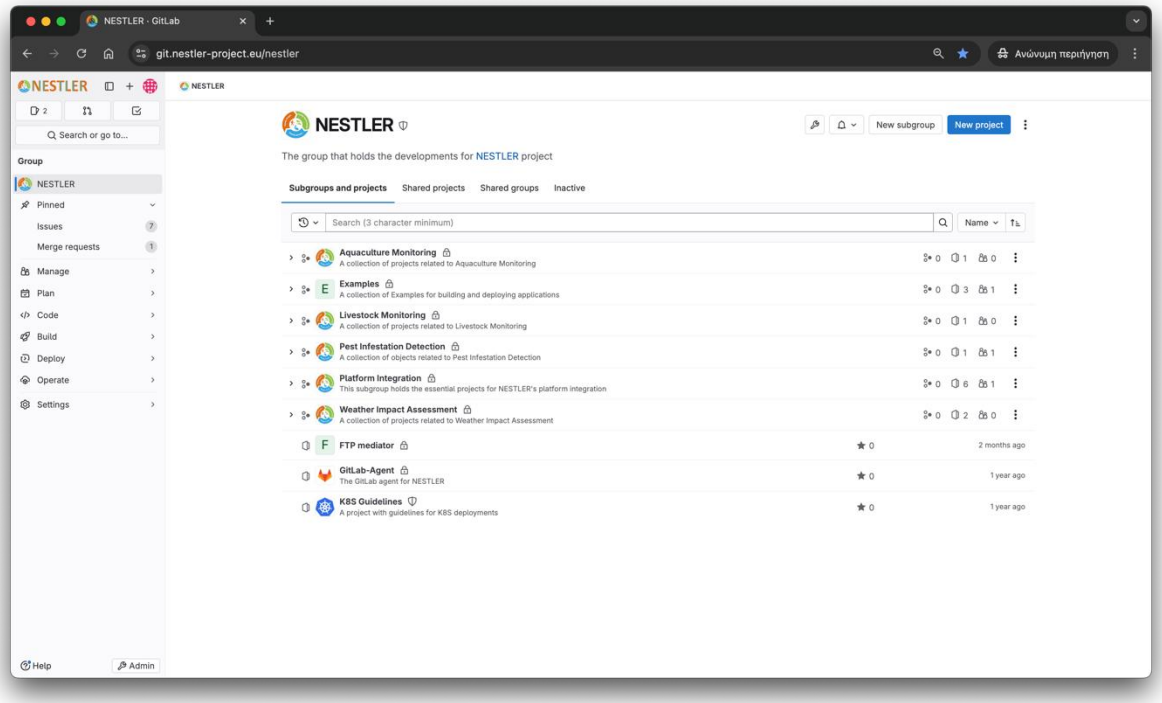

Figure 32: Overview of NESTLER group in NESTLER's self-hosted GitLab instance.

Figure 32 depicts an overview of the available NESTLER's GitLab groups and projects:

- *Aquaculture Monitoring*: includes a collection of projects related to Aquaculture Monitoring,
- *Examples*: includes a collection of Examples for building and deploying applications,
- *Livestock Monitoring*: includes a collection of projects related to Livestock Monitoring,
- *Pest Infestation Detection:* includes a collection of projects related to Pest Infestation Detection,
- *Platform Integration:* includes the core projects for NESTLER's platform integration,
- *Weather impact Assessment:* includes a collection of projects related to Weather Impact Assessment,
- *FTP mediator:* includes the project that is responsible to pick images from ftp service and upload them in the database and the object storage
- *GitLab-Agent*: includes the GitLab agent for NESTLER project
- *K8S Guidelines:* includes the guidelines for source code deployment in kubernetes.

Every project is organized according to a defined structure of files and directories. The most common files and directories in a typical GitLab project repository are:

- Project metadata:
  - *README.md*: describes the software and includes dependencies, usage and deployment instructions
  - *CHANGELOG.md*: lists the changes across versions (tags)
- Project configuration:
  - tsconfig.json / config.js: includes project specific configuration

- .prettierrc / .eslintrc / .pylintrc / .flake8: includes linter/formatter rules and settings

- Git & gitlab specific files:
  - *. gitignore*: includes files and directories that GitLab must ignore
  - *.gitlab-ci.yml*: includes CI/CD definition in the GitLab specifying various stages, e.g. SAST, TEST, BUILD, DEPLOY etc.
- Source code & dependencies:
  - Directories that include the source code (files) depending on the programming language
  - *requirements.txt / Pipfile / package.json / pom.xml / build.gradle*: includes the dependency manifests considering the programming language and used framework
  - *Pipfile.lock / package-lock.json:* includes the snapshot of exact versions of all the dependencies
- Infrastructure & deployment:
  - *Dockerfile:* includes the docker image definition
  - *.dockerignore*: includes files and directories that GitLab must exclude from the build context when building a docker image
  - *deployments/ or k8s/ or manifests/*: includes the Kubernetes manifests such as configmap.yaml, secrets.yaml, service.yaml, deployment.yaml, ingress.yaml, sts.yaml etc.
- Data & resources:
  - *dataset/ or datasets/*: includes sample or reference datasets
  - *docs/*: includes documentation files

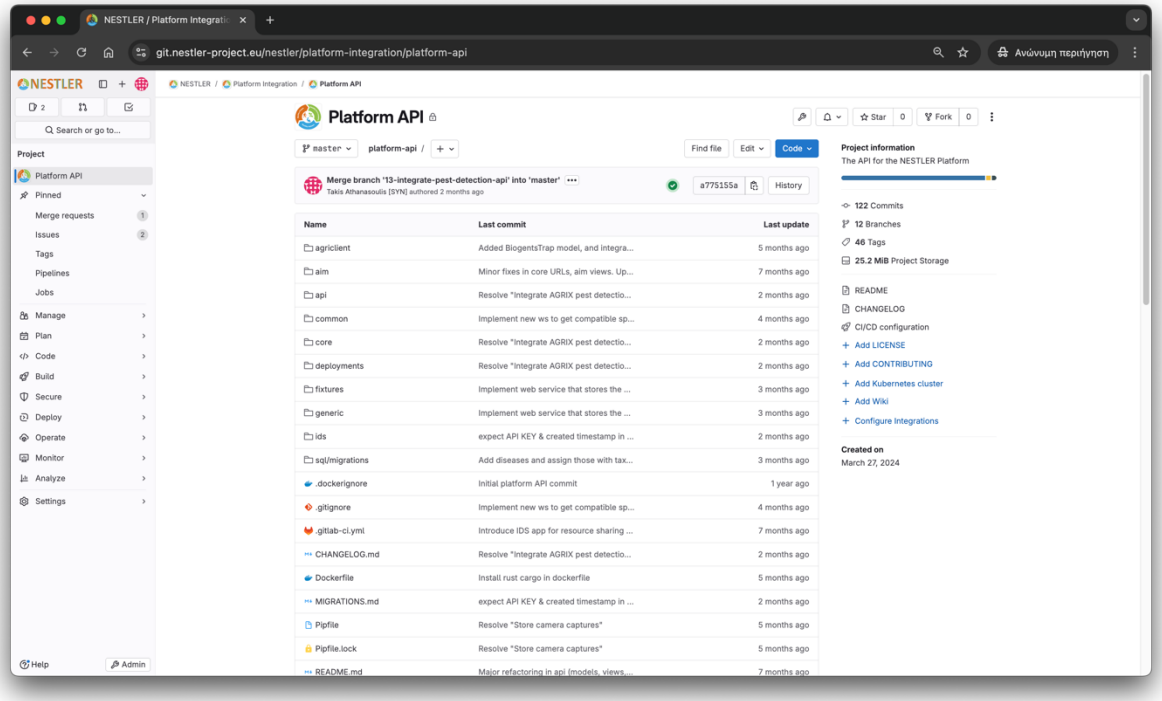

Figure 33: Overview of NESTLER API.

Figure 33 up to *Figure 41* present snapshots of the various files within the context of the NESTLER Platform API and dashboard.

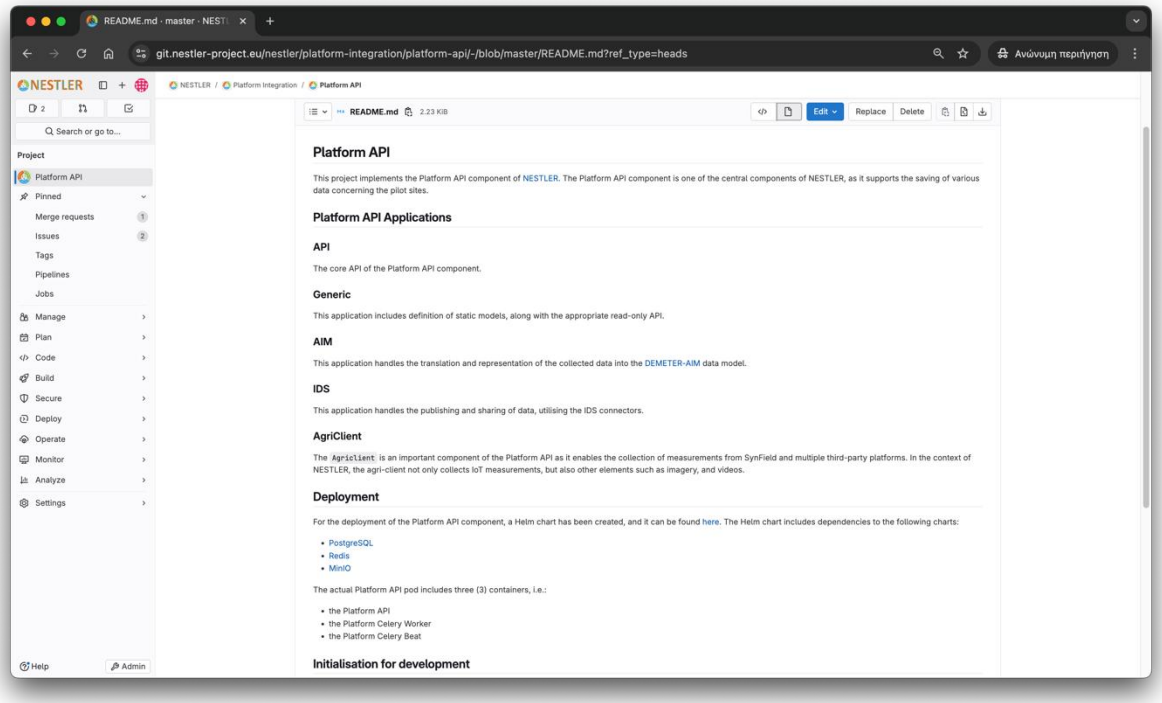

Figure 34: Snapshot of README file in NESTLER API.

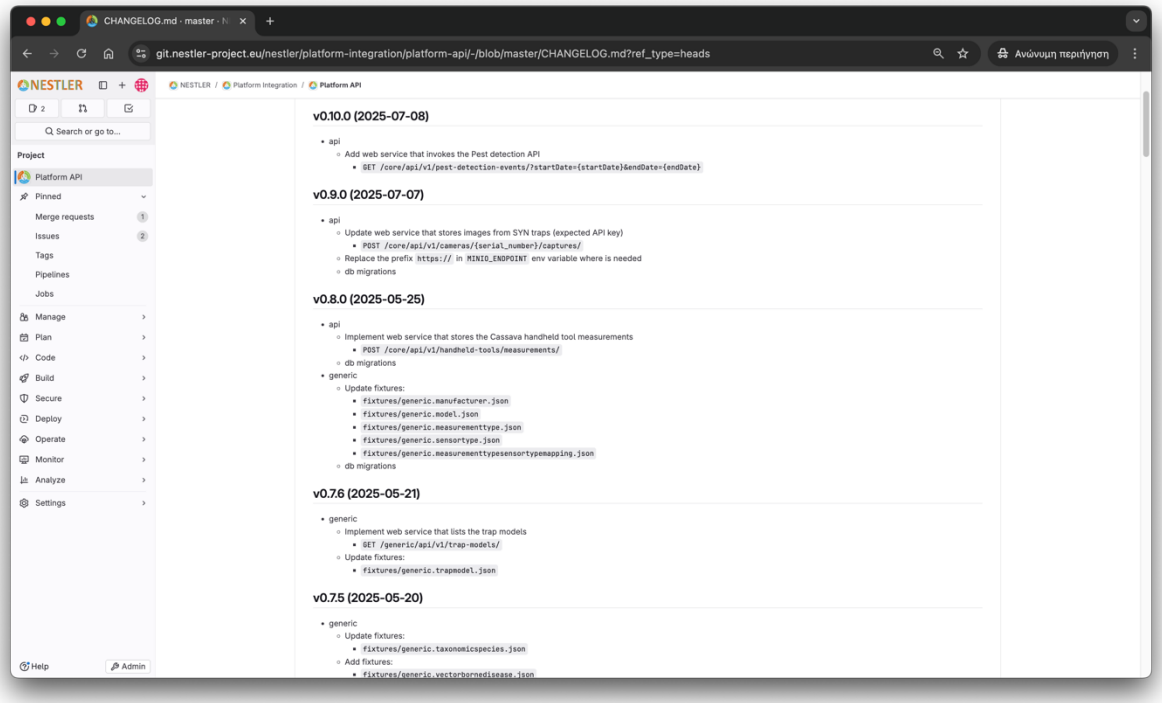

Figure 35: Snapshot of CHANGELOG file in NESTLER API.

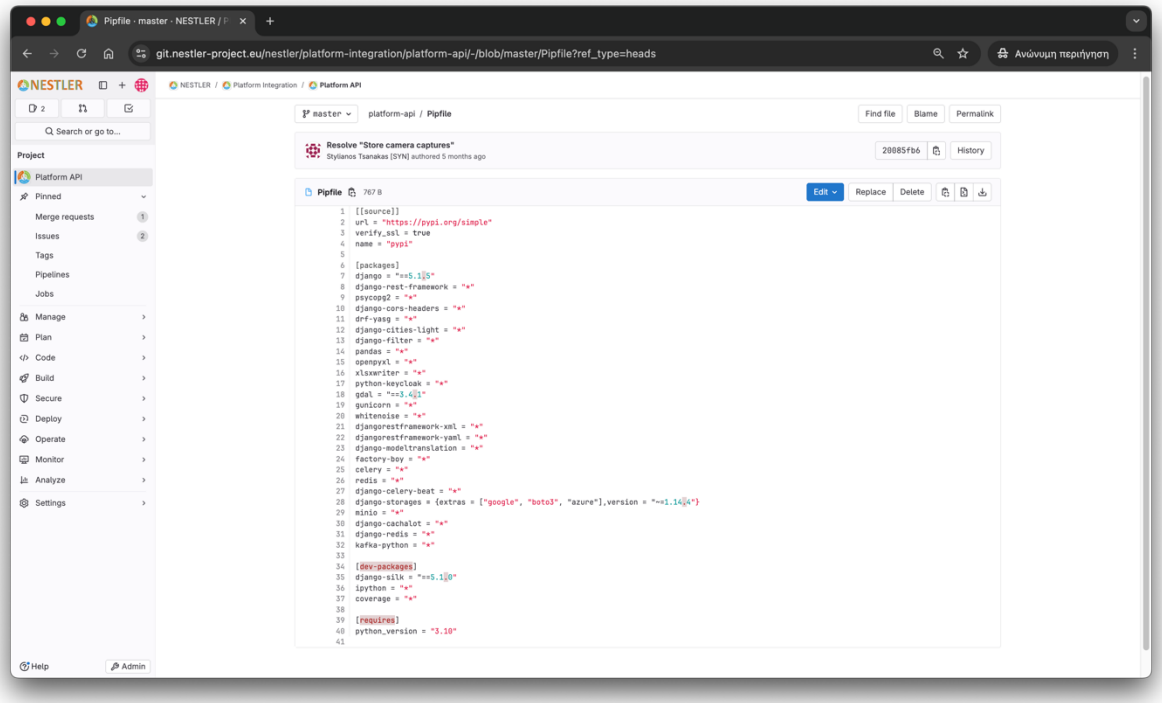

Figure 36: Snapshot of dependencies in NESTLER API.

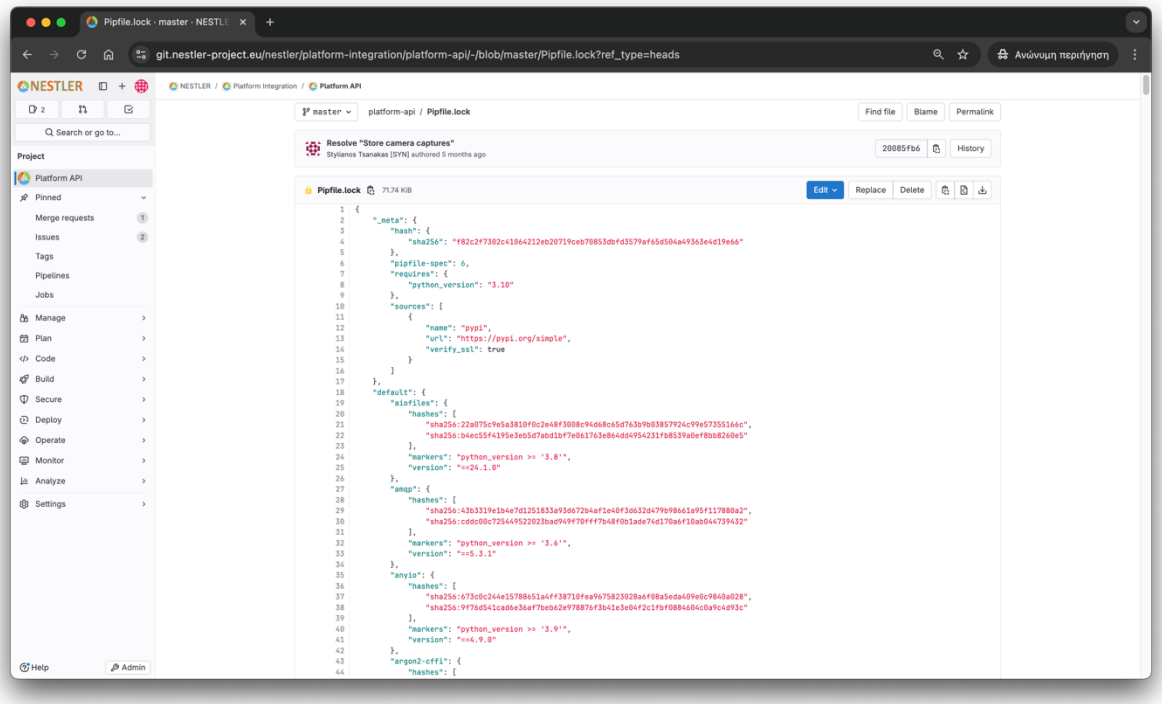

Figure 37: Snapshot of exact dependencies' versions in NESTLER API.

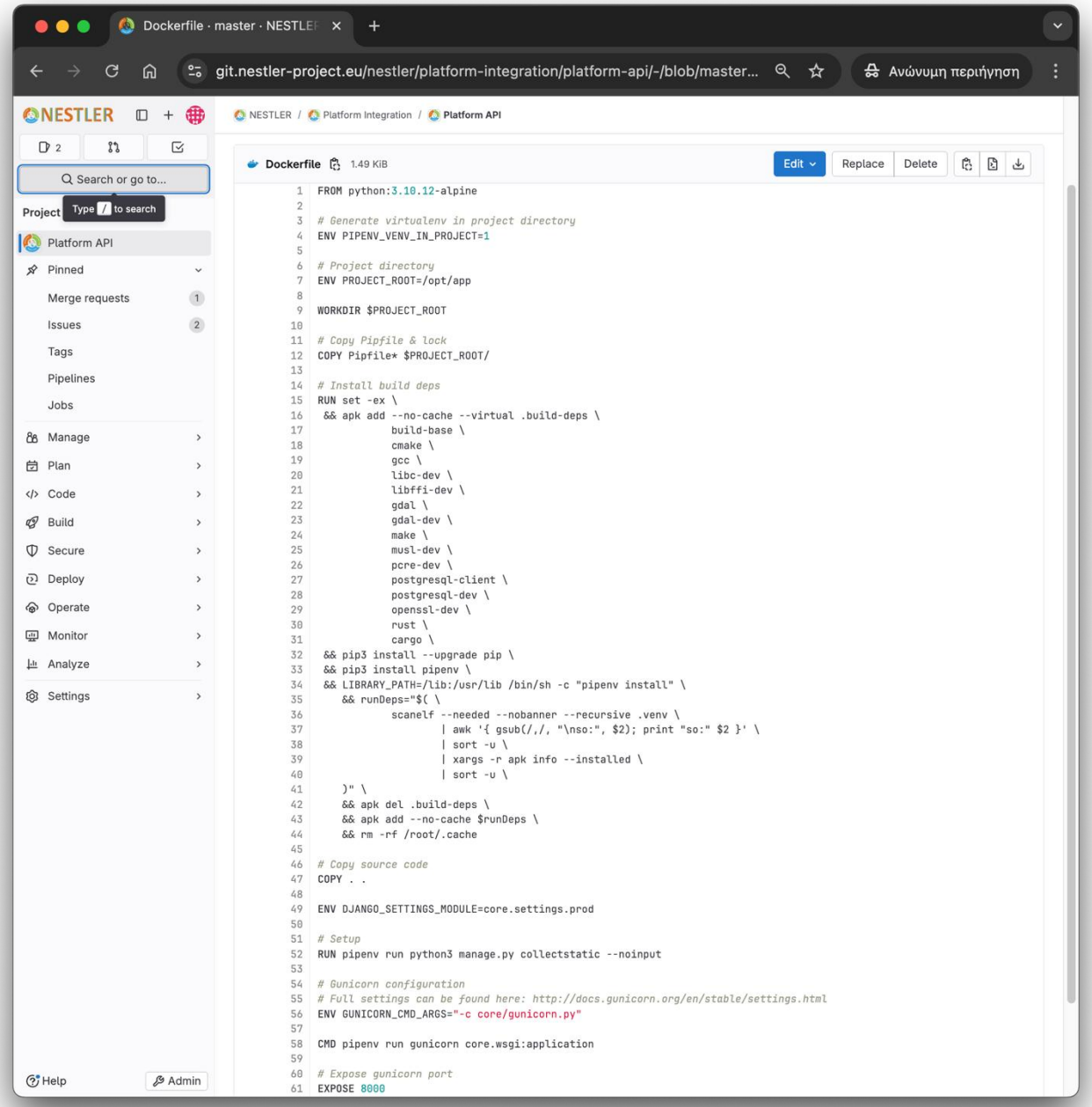

Figure 38: Snapshot of Dockerfile in NESTLER API.

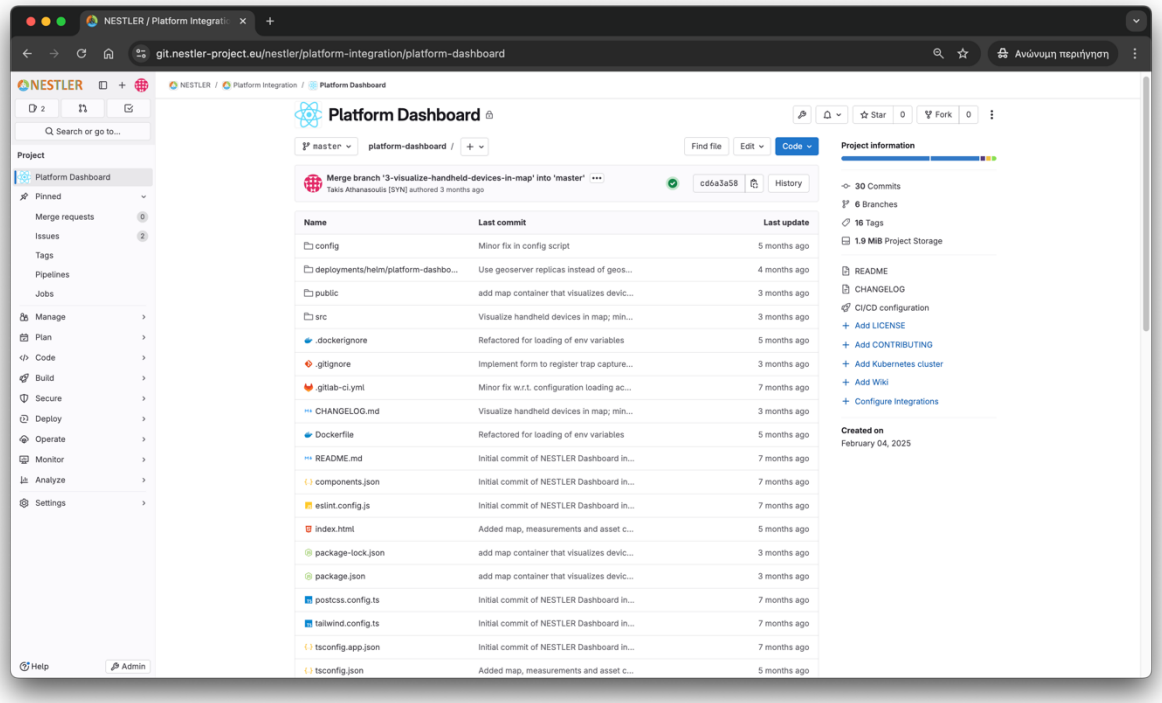

Figure 39: Overview of NESTLER dashboard source code.

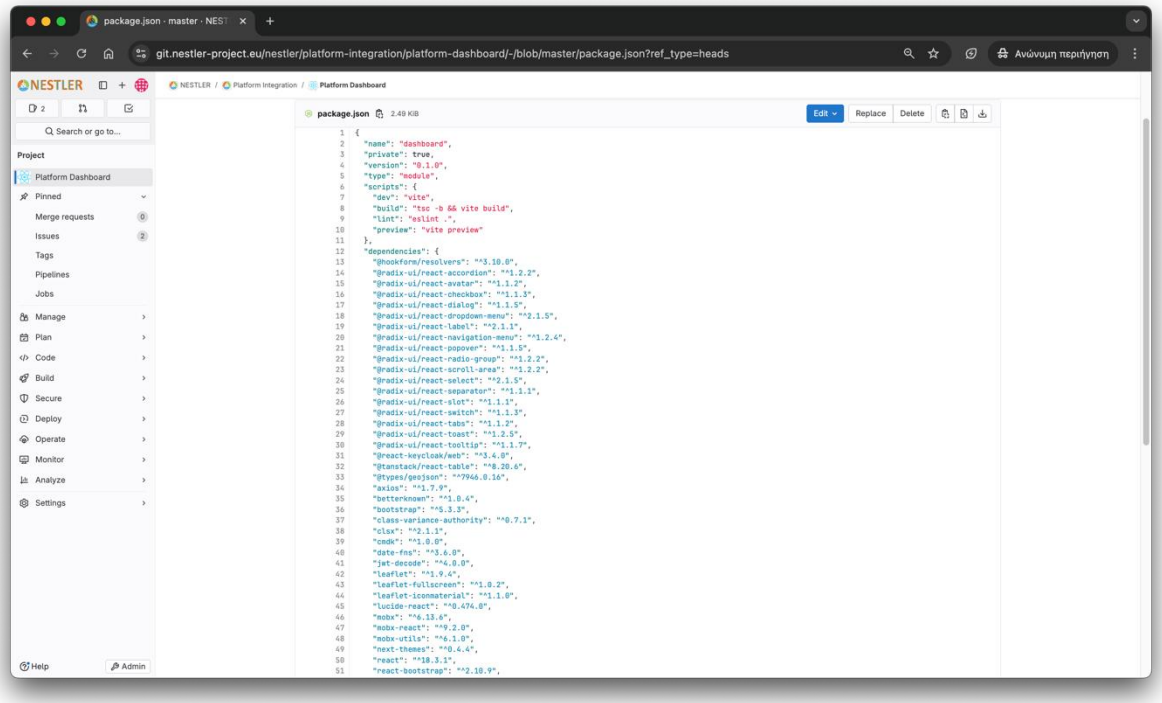

Figure 40: Snapshot of dependencies in NESTLER dashboard.

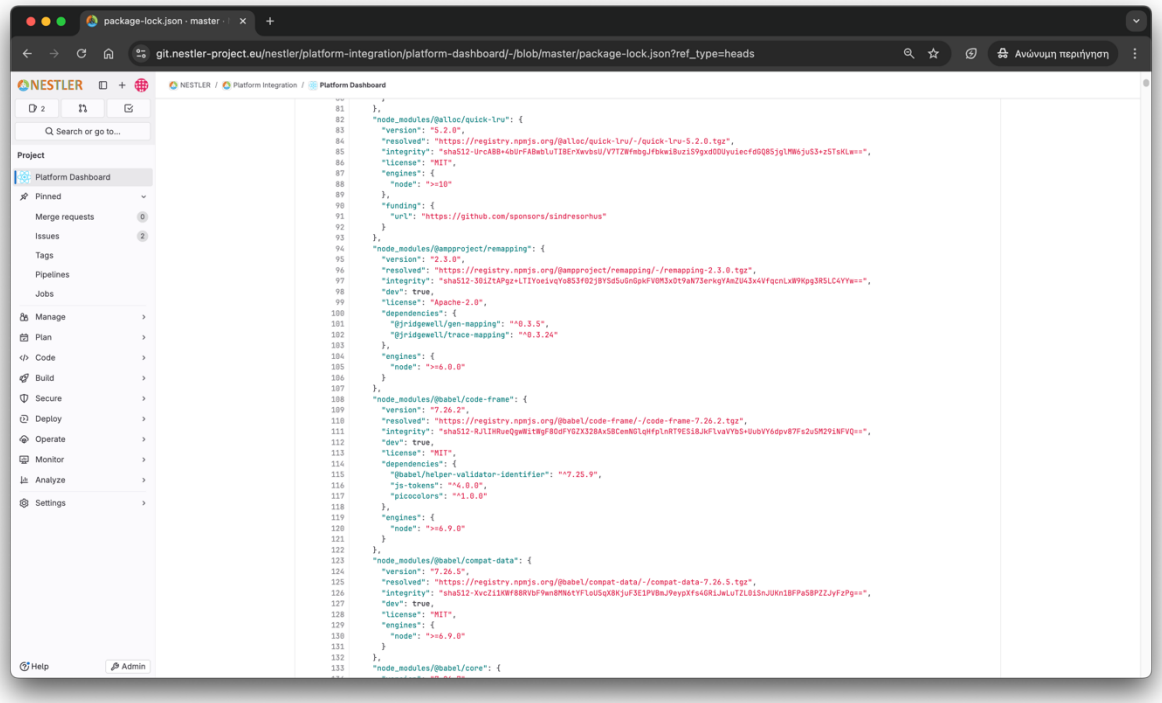

Figure 41: Snapshot of the exact dependencies' versions in NESTLER dashboard.

5.2 Issue Tracking System

The term Issue Tracking System describes a software package that supports the creation and management of issues, and thus, the communication among a collaborating team. An issue could refer to a software bug, a new required or desired feature, functionalities that do not work as expected, etc. Besides the overall issue management and tracking, an Issue Tracking System offers a variety of functionalities that facilitate project planning, such as resource assignment, prioritization of tasks, time constraints, communication among collaborating people and / or teams, notifications, etc.

As mentioned in Deliverable D4.1 [1], GitLab has been selected as Issue Tracking System for the NESTLER project. GitLab offers an advanced and feature-full tool for tracking the evolution of a new idea or the process of problem solving, as part of the GitLab workflow. A snapshot of NESTLER Platform API issues is shown in the below figure. Each issue is linked with the desired milestone: for instance, milestone PM33 stands for project month 33.

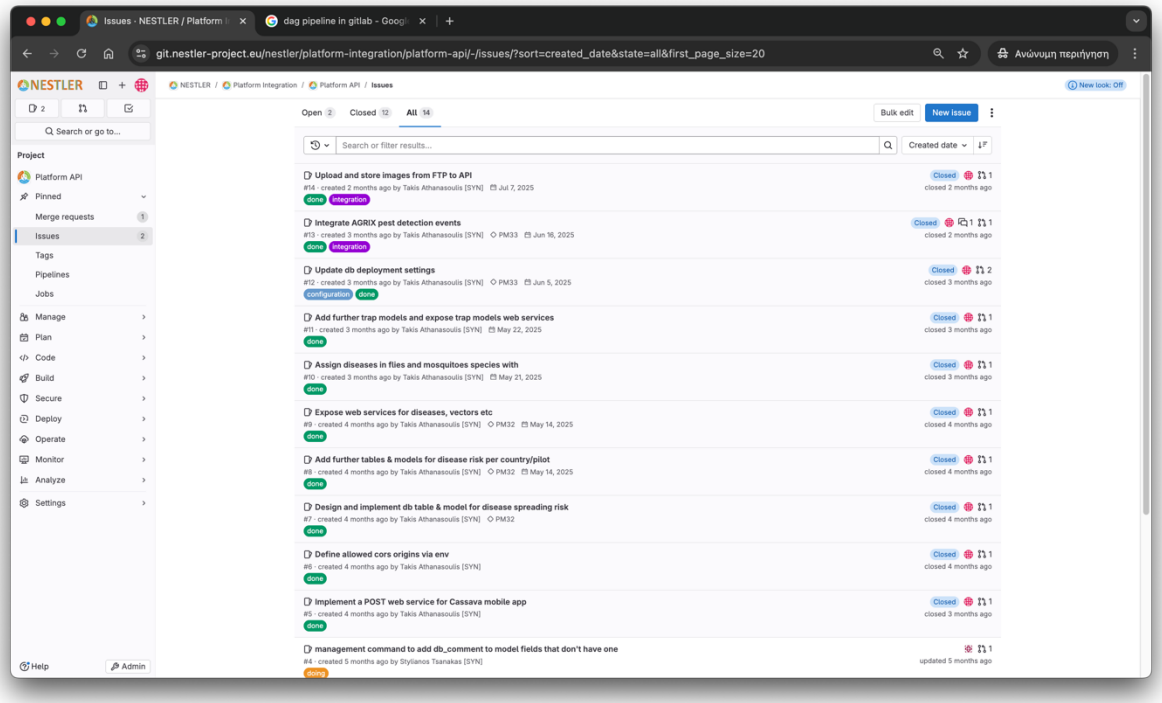

Figure 42: Snapshot of all issues in NESTLER API.

Accessing the Issues tab under the NESTLER group provides a view of the issues across all projects.

5.3 NESTLER DevSecOps Approach

DevSecOps, i.e. development, security, and operations, integrates security at every phase of the software development lifecycle, from the initial design to integration, testing, deployment, and software delivery.

DevSecOps transforms security into an innate part of the Agile and DevOps practices adopted by development organizations. In this manner, security issues are handled as soon as they are detected, instead of relying on Quality Assurance (QA) testing at the end of every development cycle, in the production environment. Therefore, through DevSecOps, software development, as well as release, cycles are accelerated by automating the delivery of secure software without slowing the software development cycle.

According to IBM, some of the best practices in DevSecOps include [47]:

- **Shift left.** This term refers to the movement of security from the right (end) to the left (beginning) of the software delivery process. In a DevSecOps environment, security is an innate part of the development process from the beginning. The cybersecurity architects and engineers become part of the development team, ensuring every component and configuration is configured securely.

- **Security education.** Security is a combination of engineering and compliance. Everyone involved with the delivery process of a software component should be familiar with the fundamental principles of application security, application security testing and other security engineering practices.
- **Culture: communication, people, processes, technology.** The importance of security of processes should be communicated in a concise manner to all participants of a software development team, from product owners to engineers and developers.
- **Traceability, auditability, and visibility.** Traceability allows for tracking of configuration items across the development cycle. Auditability is essential for ensuring compliance with security controls. Finally, visibility is generally considered a good management practice and is extremely important in a DevSecOps environment from the aspect of solid monitoring system to measure the heartbeat of the operations.

Given the best practices, the stages of the NESTLER DevSecOps approach include:

- **Plan and create.** Processes are directly related to the software design and development procedures.
- **Verify, package, and release.** These stages are directly related to the Continuous Integration (CI) and Continuous Delivery (CD), covered by automated CI/CD tools, and specifically GitLab [46].
- **Configure, detect, respond, predict, and adapt.** These stages mostly refer to QA testing on production, as well as the processes related to communicating them to the design and development phases again, in a continuous, circular interaction.

During the DevOps steps, monitoring and analytics tools will allow for proper logging in the whole software lifecycle. Last, but not least, security is introduced within every process of this DevOps cycle, implementing security by design among people, processes and technologies involved in the NESTLER software development and release.

5.4 Continuous Integration and Continuous Deployment (CI/CD)

Continuous Integration (CI) is the practice of code integration into a shared repository and frequent automated building / testing of each alteration. Alternatively, CI implements continuous processes of applying quality control, including but not limited to, a set of software development practices, behaviours, and principles for automation and improvement of integration, and certify software continuously.

CI is usually directly linked to the underlying Source Code Management (SCM) system or distributed version control system, i.e. Git in case of NESTLER. In this context, distribution is a facilitator for collaboration, by allowing developers to work on different branches of code or on the master (or main) branch. Code alterations can be committed without affecting the main repository. Hence, developers are allowed to test a pipeline with their own contributions, and without necessarily incorporating someone else's code. Furthermore, distribution allows for parallel development of features, new functionalities, or bug resolutions, as well as discrete and concurrent release cycles for development and testing. GitLab, i.e. the platform of choice for NESTLER, offers added value services on top of Git.

Continuous Delivery (CD) refers to release management practice in which software is built in a manner that allows for the production releases at any time. Continuous Delivery is accomplished by continuously

integrating late software, building executables, if / when needed, and running automated tests on those executables. These executables are pushed in environments that imitate production ones, to ensure the software will work in a production environment as well. Of course, these production-like environments might imitate the architecture, or deployment platform of the real production environment but not the entirety of its computational resources.

Continuous Deployment is an additional step, providing automated deployment of the application, without human intervention whatsoever. The term Continuous Delivery and Deployment (CD) encapsulates the automated execution of both stages. GitLab offers CD capabilities via the integrated GitLab-CI service [48].

Since both CI and CD practices are highly interconnected, CI/CD refers to integrating the procedures related to CI and CD into a single multi-stage pipeline. In the context of GitLab, the essence of continuous integration, delivery and deployment are the *pipelines* that describe sequential CI and CD operations. Pipelines are composed of *jobs* and *stages* that describe the sequence of *job* execution.

The GitLab jobs are executed by GitLab runners, while jobs of the same stage can be executed concurrently, assuming the existence of multiple runners. If all jobs of a stage are successful, then the pipeline moves to the next stage; otherwise, it is terminated early, if even one job fails. The jobs are executed via GitLab runners, which run the code defined in YAML scripts with a file named ".gitlab-ci.yml" at the root of a repository.

The ".gitlab-ci.yml" file presented below defines the CI/CD pipeline of the NESTLER Platform API. As shown in *Figure 43*, four stages have been defined:

- SAST,
- TEST,
- BUILD,
- DEPLOY.

For each one stage a set of jobs are defined. For instance, the BUILD stage defines the jobs "build-master", "build-tags" that applied in different cases.

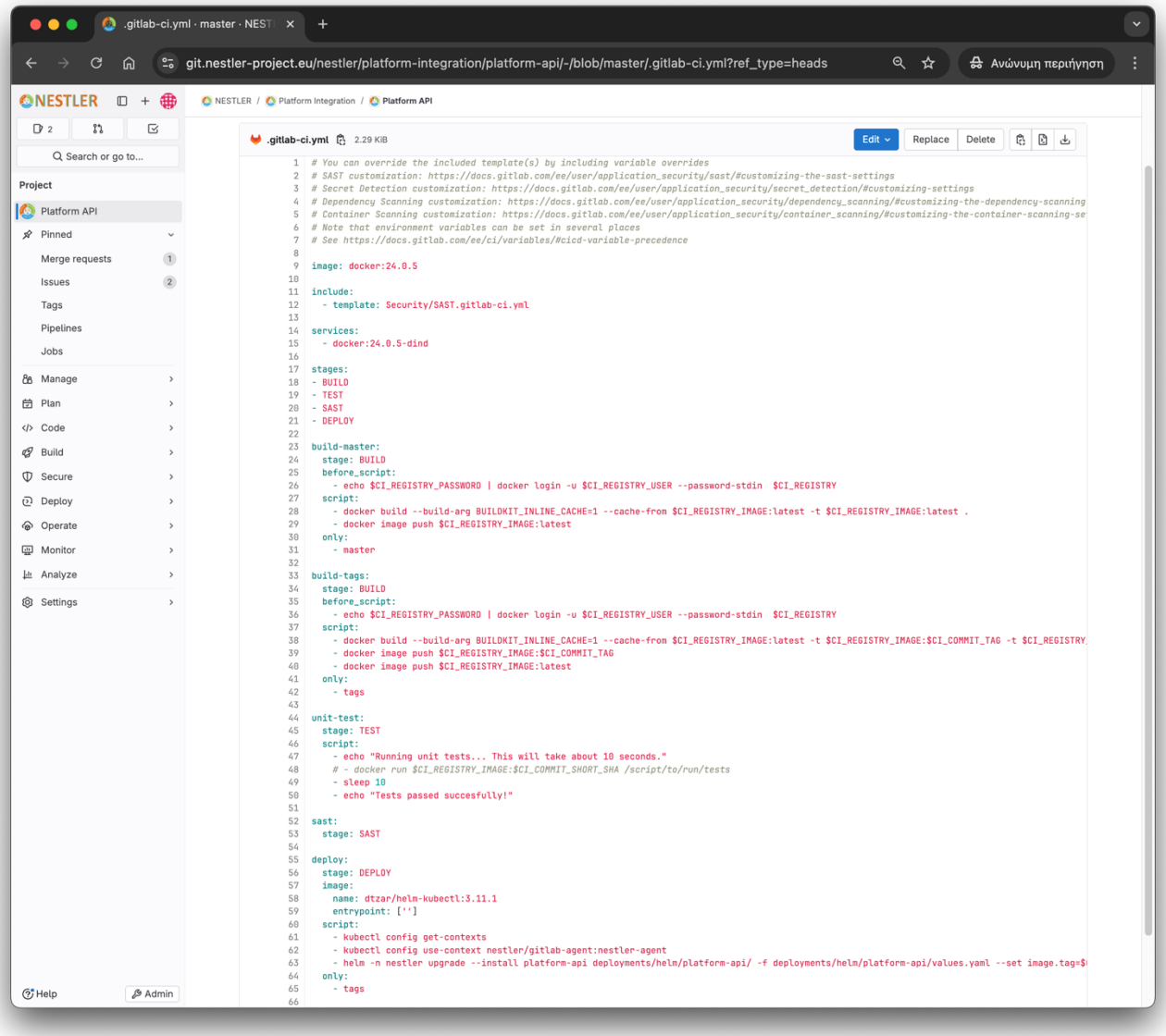

Figure 43: Snapshot of the CI/CD definition for the NESTLER API.

A common development workflow within GitLab is presented in *Figure 44* [49]. The code developments that made in a local branch are committed and pushed in a branch of a remote repository in GitLab, an action which triggers the CI/CD pipeline set for the specific project:

- The developer implements a certain piece of software to be integrated into the core system functionality.
- The developer then introduces one or multiple test cases that unit-test the new piece of code at the level of rationality checking and consistency of outcomes. The unit-testing should be as thorough as possible.
- The developer commits and pushes code developments on their local Git repository to a branch of the remote GitLab repository.

- The developer creates a merge request for their piece of code.
- The CI/CD pipeline set for the specific project is triggered. The CI platform performs the determined unit-testing.
- If the unit-testing is successful, the code is subject to further integrated system testing.
- If the integrated system testing is successful, then the merge request is accepted, and the newly committed source code becomes part of the repository's master (main) branch.
- If the result of the testing is unsuccessful, the merge request is rejected.
- If either the unit or integrated system testing fails, the merge request is rejected. In this case, the developer should consider either a code rewrite to satisfy the unit-testing procedures, or the unit-test rewrite, in case of erroneous testing. The workflow is re-initiated.
- The source code management platform moves to the CD part, building the package and deploying it.
- After successful CD, the new code is running on the deployment infrastructure and is subject to user testing/production.

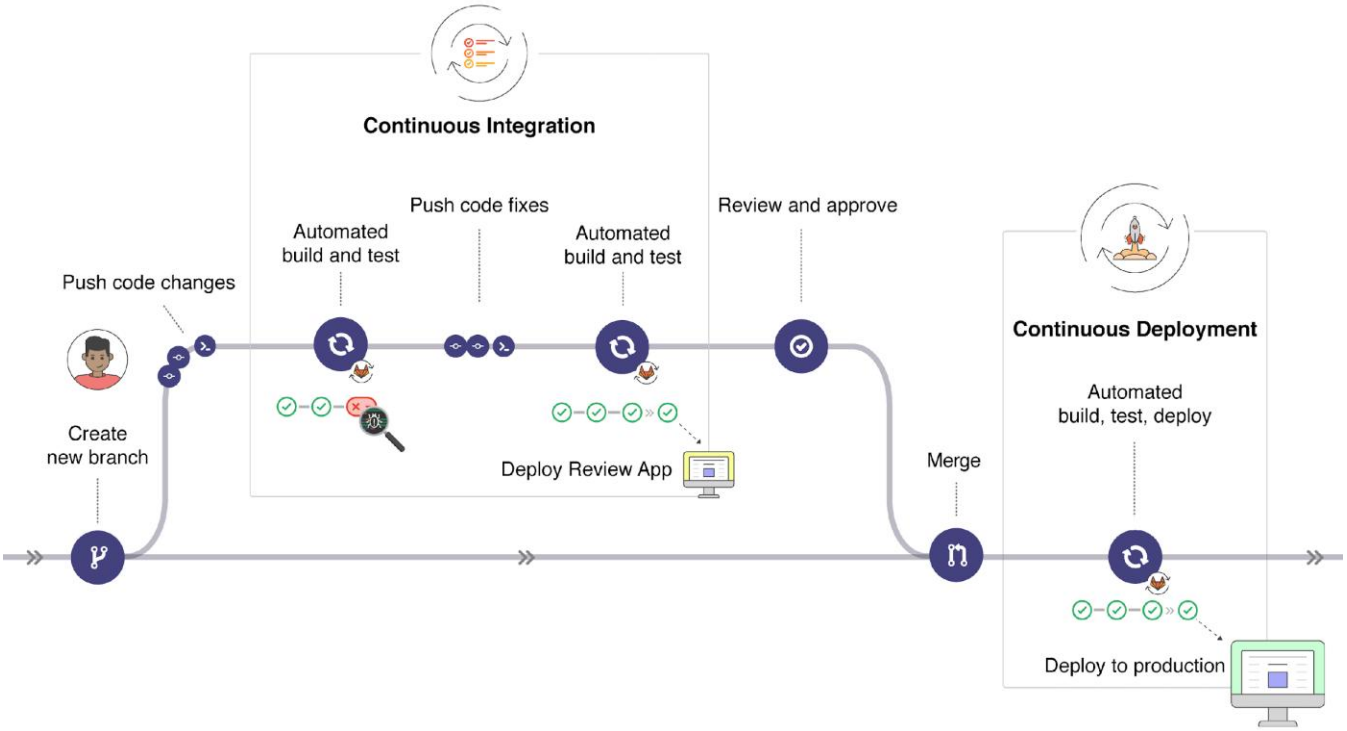

Figure 44: GitLab CI/CD workflow.

Figure 45 lists the pipeline history of the NESTLER platform API while *Figure 46* depicts an implementation and test run of a configured pipeline on top of the NESTLER platform API implemented in Python. *Figure 47* presents the jobs view of a selected pipeline with more details.

The pipelines and jobs are executed by a native GitLab runner, installed alongside GitLab, in a dedicated project server. The overall NESTLER integration deployment is following the cloud-native paradigm, hence, NESTLER components get delivered (deployed) using relevant standard procedures. These procedures include building docker images of the software (based on Dockerfile definition) and

uploading them to a private container registry, testing the uploaded image and, when it comes to tags, deploying them.

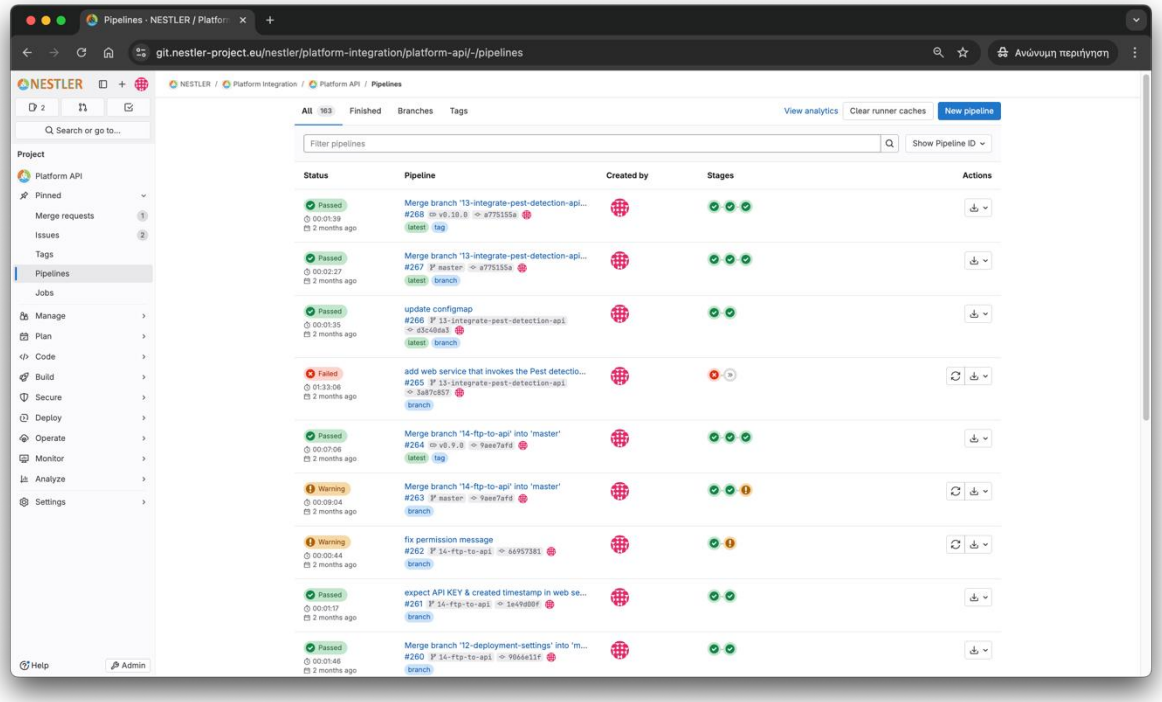

Figure 45: List of pipelines in NESTLER API.

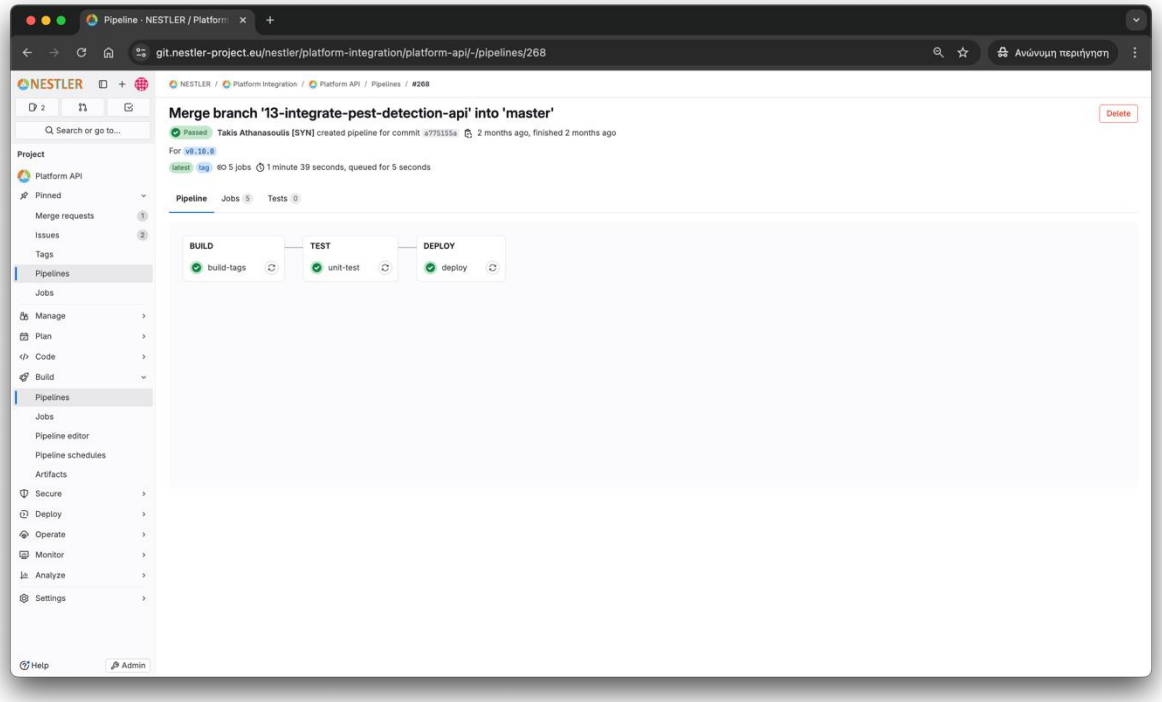

Figure 46: CI / CD Pipeline view of NESTLER API.

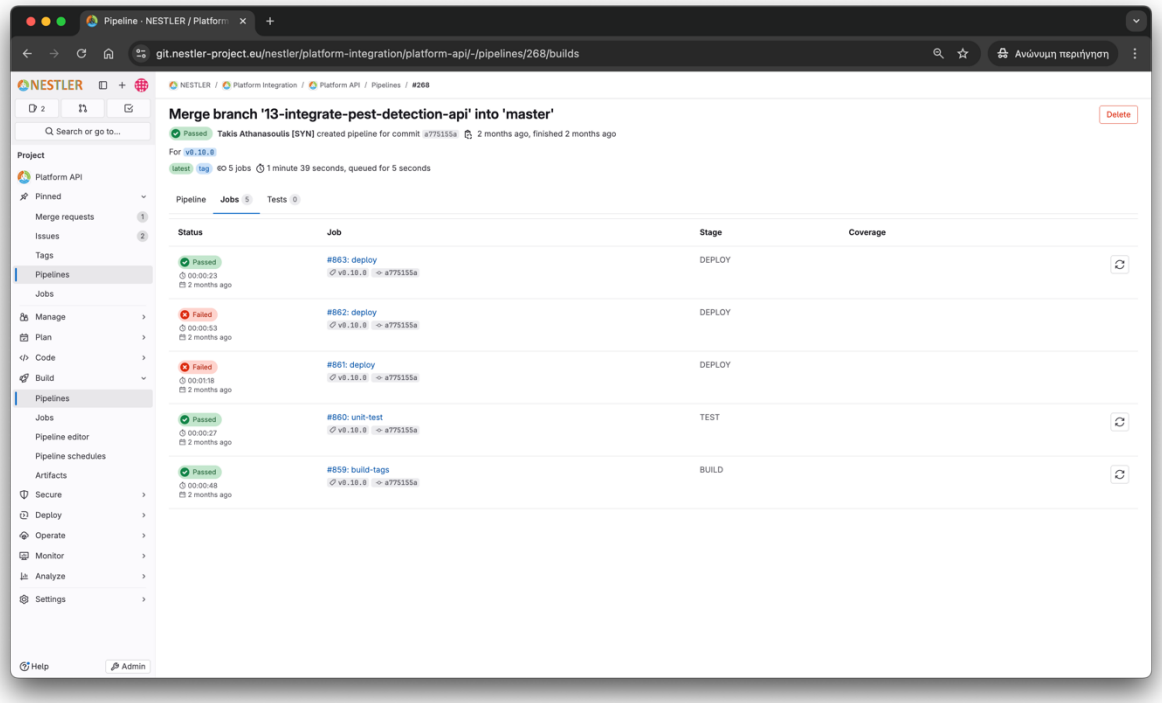

Figure 47: CI / CD pipeline's jobs view of NESTLER API.

5.5 Containerization and Registry Tools

Containerization tools, the major companion and competitor of virtualization, is becoming an increasingly popular, rapidly maturing technology among software development teams, as a facilitator of operations within the DevOps lifecycle. Containerization offers a higher layer of abstraction, over the application layer which packages code and library dependencies. The innate portability of containers among different computing environments has been one of the key features pushing the rapid development and adoption of containerization for a series of highly diverse applications. As it isolates the software from its execution environment, it ensures that the application works uniformly for a variety of systems, both bare-metal and OS-based.

The most common and popular containerization technology is Docker [26]. A Docker image is a lightweight, standalone, executable package of software that encapsulates the entirety of an application's prerequisites, such as code, runtime, system tools and libraries, etc.

Latest GitLab versions readily support the creation of a container registry, where Docker, or other types of container images can be pushed, pulled, and stored. The built-in GitLab container registry feature allows for a complete, self-hosted solution, without the need for deploying an additional, external registry for container images' storage purposes. Indeed, GitLab container registry has been utilised for NESTLER project and is currently hosted at:

- https://registry.git.nestler-project.eu

Figure 48 depicts the available tags of the Platform API in the NESTLER container registry, as captured through NESTLER GitLab instance while *Figure 49* depicts the available tags of the NESTLER dashboard.

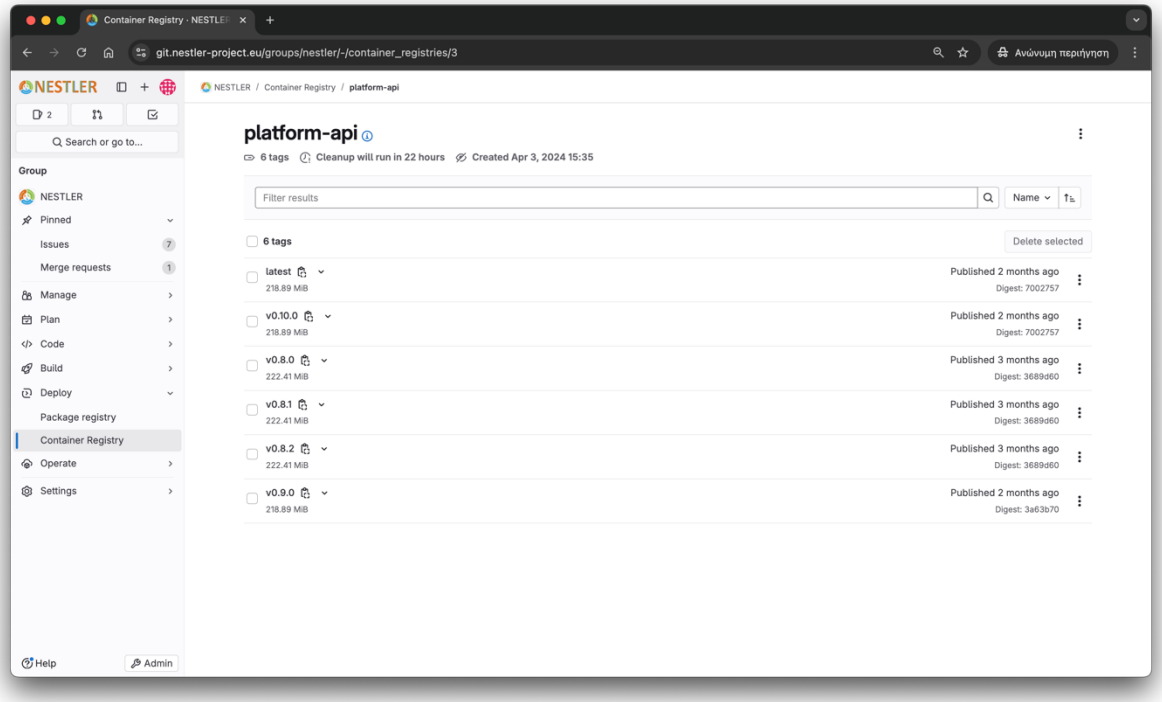

Figure 48: Snapshot of available tags of platform API in NESTLER container registry.

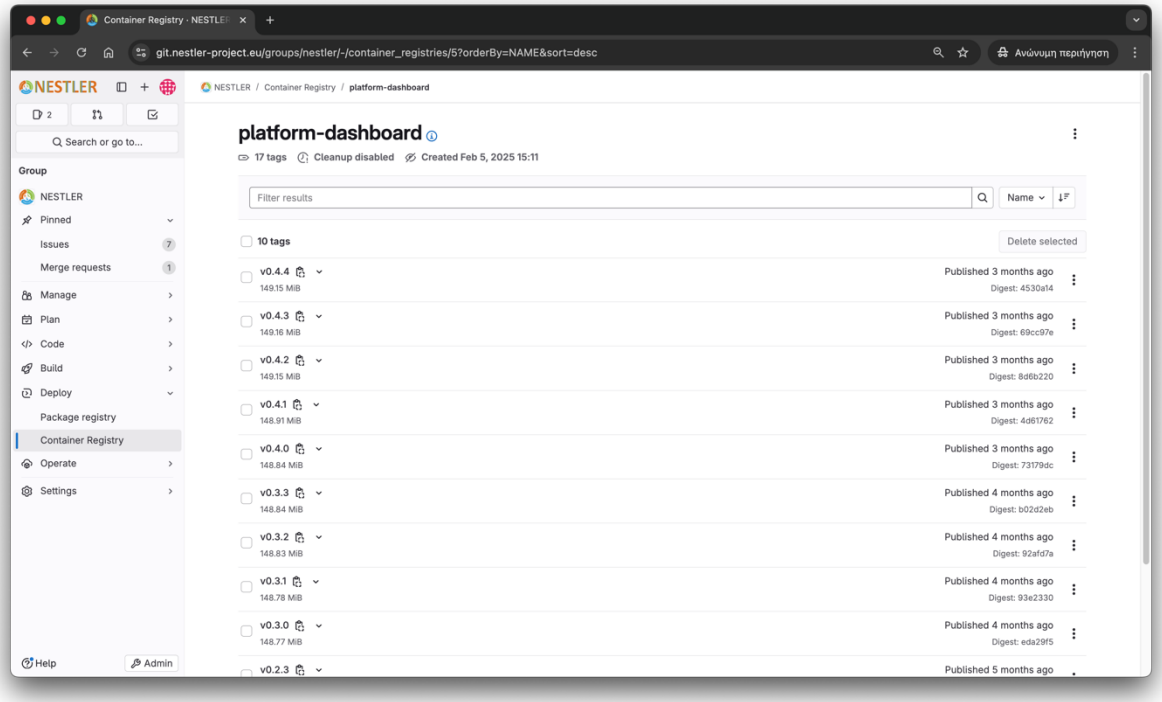

Figure 49: Snapshot of available tags of platform dashboard in NESTLER container registry.

With the built-in container registry, a GitLab user does not need an account for an external registry (e.g. Dockerhub [50]). Instead of this, the user can build, push, and pull container images to GitLab's container registry, given that their role within a project gives them sufficient privileges for such an operation. Typically, every user with a *Developer* or higher role can push and pull images to and from the container registry. The images of every project lie under the respective group and / or subgroup of the project. The URL schemes defined for each project are:

- https://*registry.git.nestler-project.eu/nestler/{subgroup_name}/{project_name}:<tag>, or*
- https://*registry.git.nestler-project.eu/nestler/{project_name}:<tag>*

For instance, the Platform API project lies under the Platform Integration subgroup of the NESTLER group. The Platform API docker image with tag v0.10.0 is then accessible via the URL:

- https://*registry.git.nestler-project.eu/nestler/platform-integration/platform-api:v0.10.0*

5.6 Deployment Platform

NESTLER components are going to be deployed in a Kubernetes cluster hosted by SYN. *Figure 50* lists the available nodes of the Kubernetes cluster.

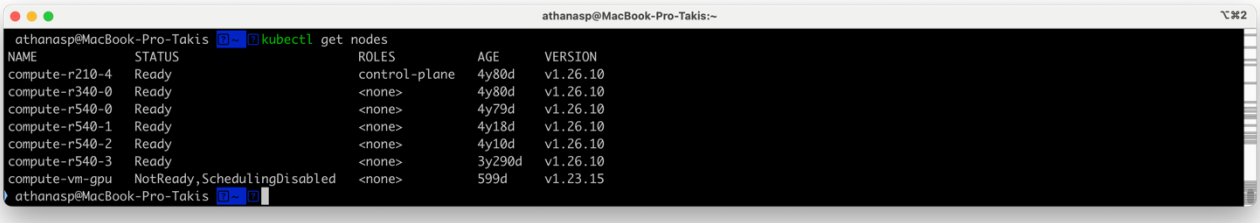

Figure 50: Nodes of the Kubernetes cluster.

In this context, a dedicated NESTLER namespace has been created within this cluster (see *Figure 51*).

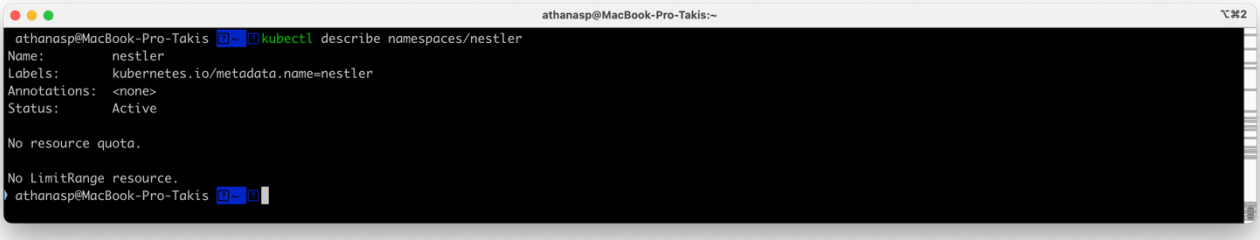

Figure 51: NESTLER namespace in the Kubernetes cluster.

Considering that data is of vital significance in the context of NESTLER, and the project is following the cloud-native paradigm, the existence of cloud-native storage is mandatory. On this ground, storage has been configured utilizing the open-source source rook framework [51], combined with Ceph [52] as a software storage cluster. With this combination, effective and configurable storage duplication can be

achieved transparently and effortlessly for the end-user application, without the need to adhere to specified component design practices to support it. *Figure 52* depicts the list the available storage classes in the Kubernetes cluster.

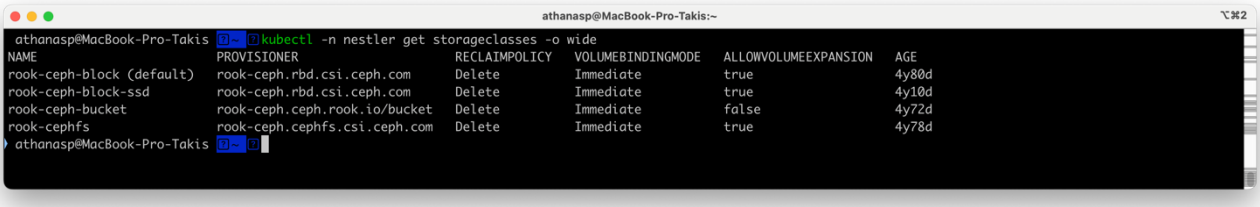

Figure 52: Storage classes of the Kubernetes cluster.

In Kubernetes, a Persistent Volume Claim (PVC) is a way for NESTLER components/services to request and use persistent storage. Unlike regular pods, which are ephemeral and lose their data when restarted or deleted, PVCs provide durable storage that outlives the pod's lifecycle. They act as an abstraction layer between the application and the underlying storage. By defining a PVC in a manifest, each component/service can request a specific amount of storage with defined access modes, and Kubernetes will bind it to a suitable Persistent Volume (PV). This ensures data persistence, portability, and consistency across deployments in the cluster. Access mode RWO stands for "read-write-once" and the volume can be mounted by a single pod while RWX stands for "read-write-many" and the volume can be mounted by multiple pods at the same time. *Figure 53* lists the available PVCs in the NESTLER namespace.

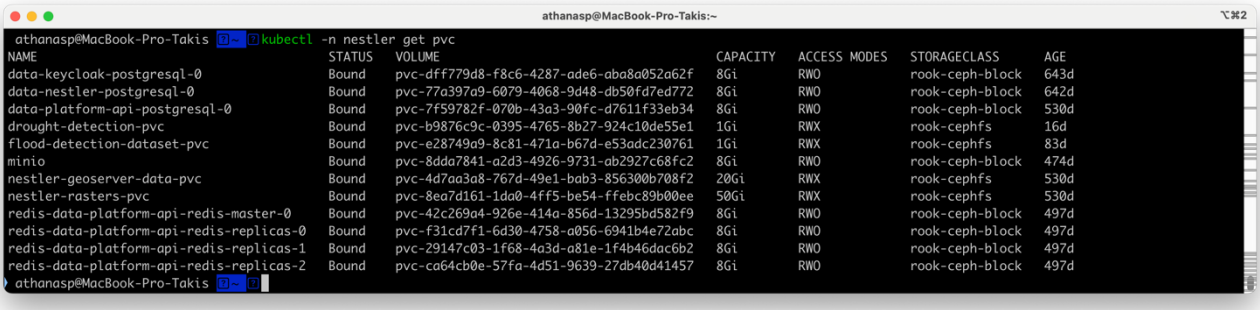

Figure 53: Permanent Volume Claims (PVC) of the NESTLER namespace.

Figure 54 depicts the available configuration manifests (ConfigMap) in the NESTLER namespace. The ConfigMap is an API object used to store non-confidential configuration data in key–value pairs, keeping configuration separate from the component/service source code. This allows services to remain portable and flexible, since the same container image can be deployed across environments with different configuration values. ConfigMaps can be consumed by pods in several ways: as environment variables, as command-line arguments, or by mounting them as configuration files inside a container. Typical use cases include defining application settings, database connection URLs, feature flags, or other runtime parameters that should not be hard coded into the application image. By externalizing configuration

through ConfigMaps, Kubernetes enables easier updates and environment-specific customization without rebuilding or redeploying application containers.

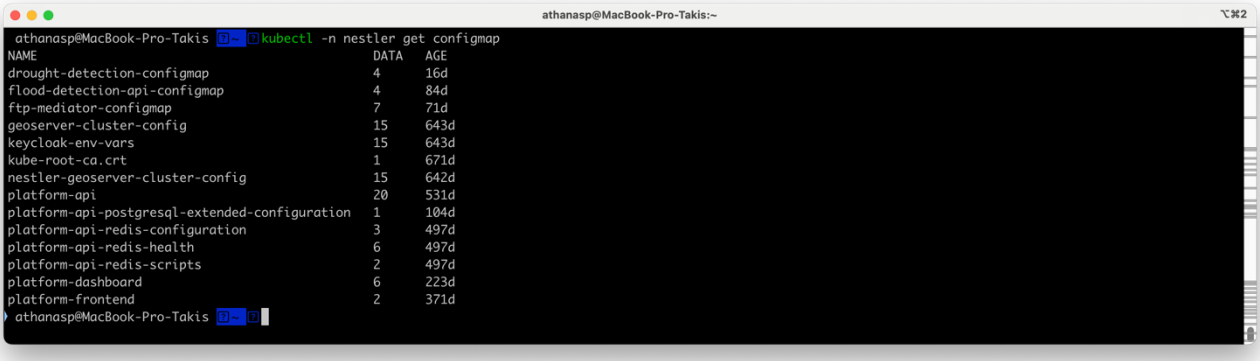

Figure 54: Configuration manifests of the NESTLER namespace.

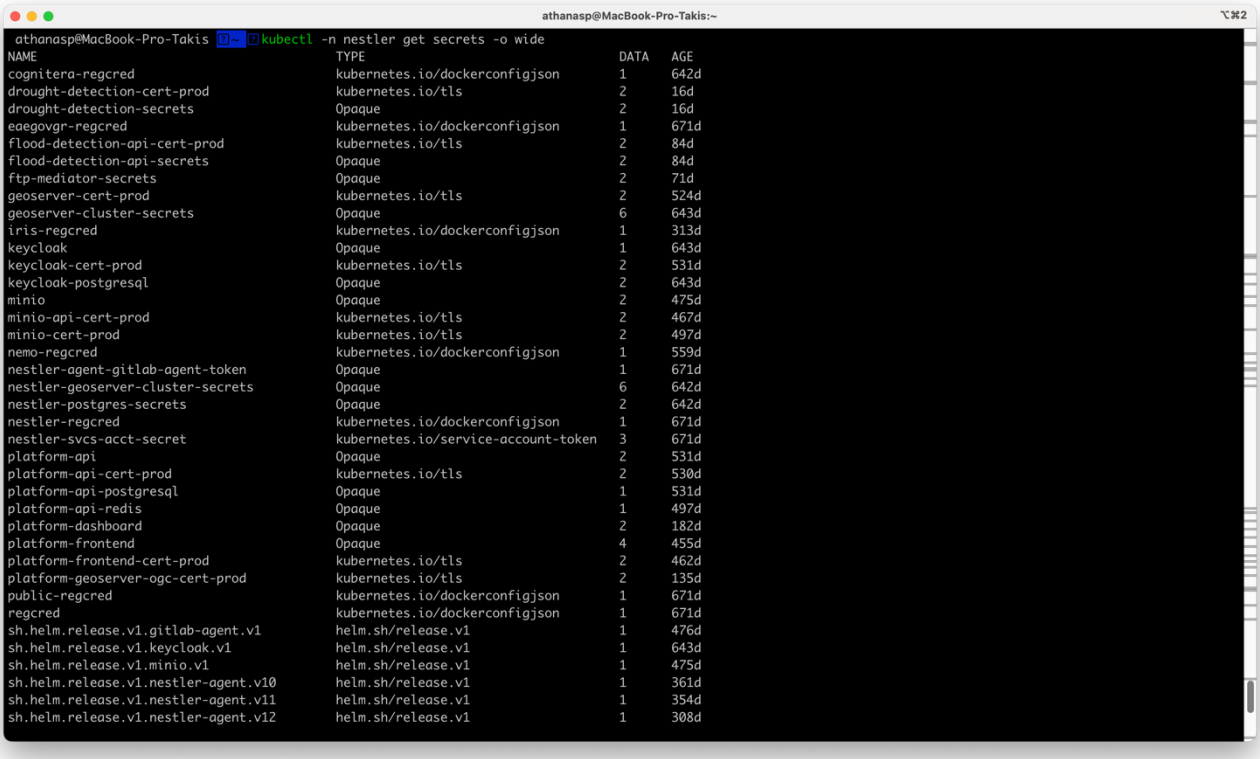

Figure 55: Secret manifests of the NESTLER namespace.

Figure 55 depicts a snapshot of the available secret manifests in the NESTLER namespace. Each secret manifest is used to store and manage sensitive information, such as passwords, API keys, tokens in a secure way. Secrets are like ConfigMaps but specifically designed for confidential data, as their contents are base64-encoded and can be mounted into pods as files or injected as environment variables. By externalizing sensitive values into secret objects, applications avoid hardcoding credentials inside

container images or configuration files, improving both security and maintainability. Secrets can be referenced in deployments, pods, or other resources, ensuring that only authorized components can access them when needed.

In Kubernetes, each deployment is used to manage stateless services by defining the desired state of pods and ensuring that the specified number of replicas are always running. It supports rolling updates, rollbacks, and scaling, making it ideal for workloads like web servers, APIs, or frontend applications where pods are interchangeable. *Figure 56* shows the current deployments in the NESTLER namespace. On the other hand, a StatefulSet (STS) is designed for stateful applications that require stable network identities, persistent storage, and ordered deployment or scaling. Each pod in an STS has a unique, stable identifier and can be linked to its own PV, which makes it suitable for databases, distributed systems, and applications that maintain data consistency. *Figure 57* depicts the available StatefulSets in the NESTLER platform. In both cases the IMAGES column reflects the exact version of docker image that is used.

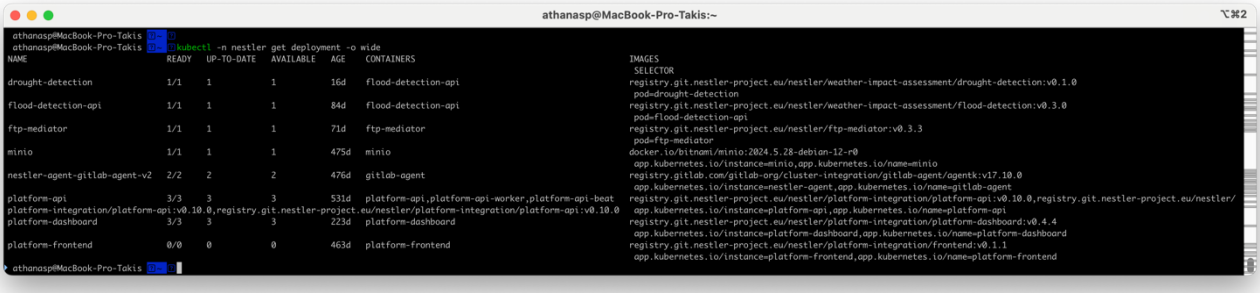

Figure 56: List of kubernetes deployments in the NESTLER namespace.

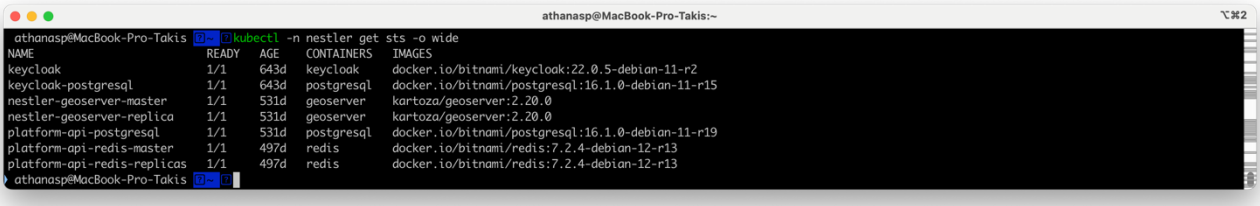

Figure 57: List of kubernetes statefulsets in the NESTLER namespace.

Each service provides a stable way to access a set of pods. Since Pods can be recreated and their IPs change dynamically, Services act as an abstraction layer, assigning a permanent virtual IP (ClusterIP) or exposing Pods outside the cluster (NodePort or LoadBalancer). Services also provide built-in load balancing by distributing traffic across all healthy pod instances that match a selector. *Figure 58* shows the current Kubernetes services in the NESTLER namespace. Several services exist to automate the management of DNS records in external DNS providers based on the state of the Kubernetes cluster. ExternalDNS service bridges kubernetes with the external DNS provider, ensuring that DNS names (like https://dashboard.platform.nestler-project.eu/) automatically point to the correct Services or Ingresses

in your cluster. As an example, the “platform-api-external-dns” is provided, and its metadata is shown in *Figure 59*.

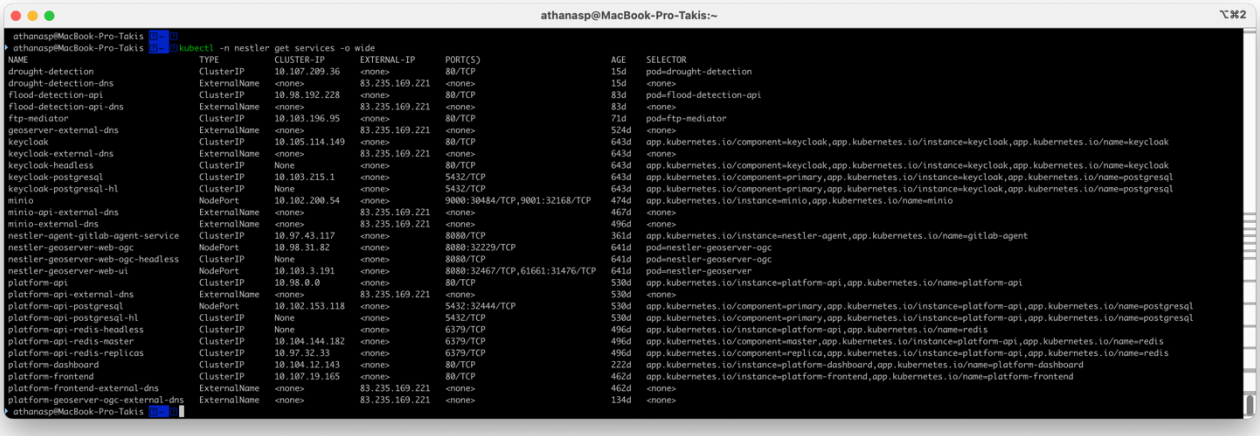

Figure 58: List of kubernetes services in the NESTLER namespace.

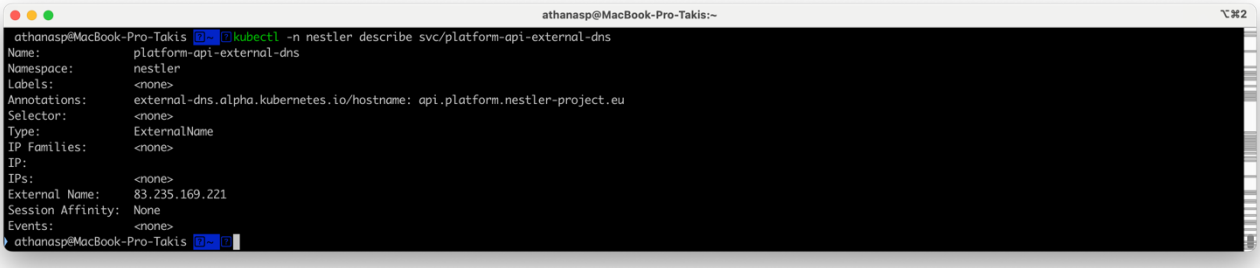

Figure 59: Metadata inspection of an external DNS service in the NESTLER namespace.

On the other hand, the Pod is the smallest deployable unit, representing one or more tightly coupled containers that share the same network namespace and storage volumes. Pods are ephemeral by nature — if one fails or is deleted, it is usually recreated by a controller (like a Deployment or StatefulSet). They are the fundamental building blocks where application workloads actually run. *Figure 60* shows the running Kubernetes pods in the NESTLER namespace.

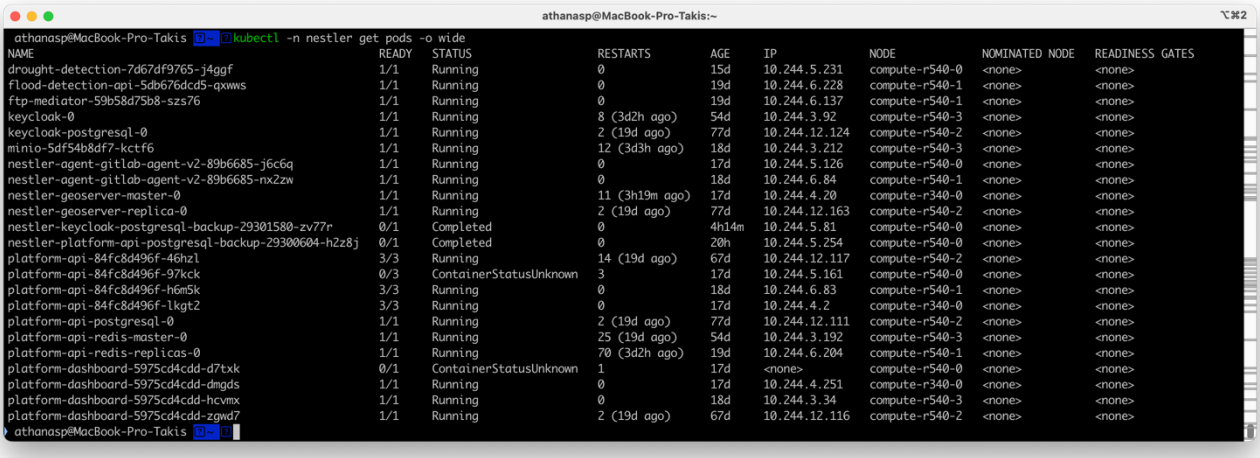

Figure 60: List of running pods in the NESTLER namespace.

In Kubernetes, a certificate manifest is used to define and manage TLS/SSL certificates that enable secure communication between clients and services. Certificates are often stored as Secrets and can be automatically issued and renewed using tools like cert-manager, ensuring encrypted traffic and compliance with security policies. *Figure 61* shows the current certificates in the NESTLER namespace.

Figure 61: List of current certificates *in the NESTLER namespace.*

An IngressRoute is a custom resource (commonly used with the Traefik [53] ingress controller) that extends the standard Ingress functionality by offering more advanced routing capabilities, such as path- and host-based rules, middleware for authentication or rate limiting, and traffic splitting for canary releases. Together, the certificate manifest and IngressRoute provide both secure and flexible routing of external traffic into the cluster: the certificate ensures encrypted HTTPS connections, while the IngressRoute controls how those requests are directed to the appropriate services. *Figure 62* shows the current certificates in the NESTLER namespace.

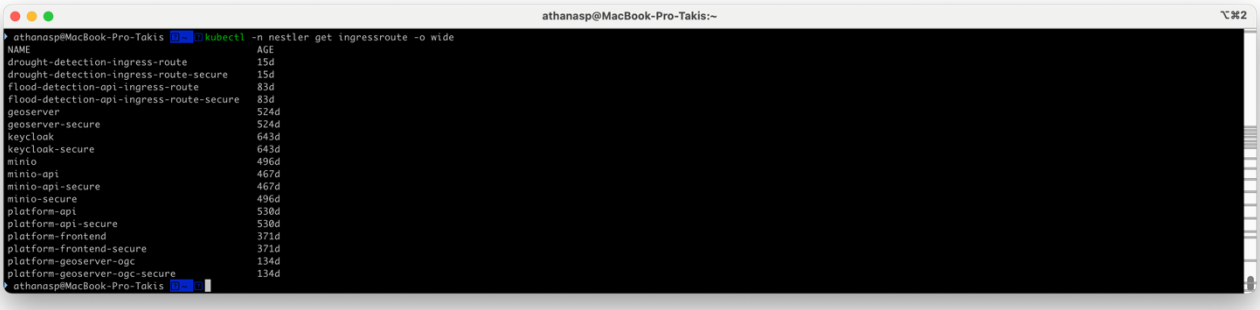

Figure 62: List of current ingress routes in the NESTLER namespace.

To ensure regular and automated backups of the NESTLER platform databases, kubernetes CronJobs have been configured. A CronJob allows scheduling of Jobs to run periodically at fixed times, dates, or intervals, like the traditional cron service in Linux. By using this mechanism, backup tasks can be executed reliably without manual intervention, ensuring that database snapshots are created and stored according to the defined schedule. This approach not only enhances data protection and recovery capabilities but also integrates seamlessly into the Kubernetes environment, providing consistency and resilience in managing critical data. *Figure 63* shows the current cronjobs in the NESTLER namespace.

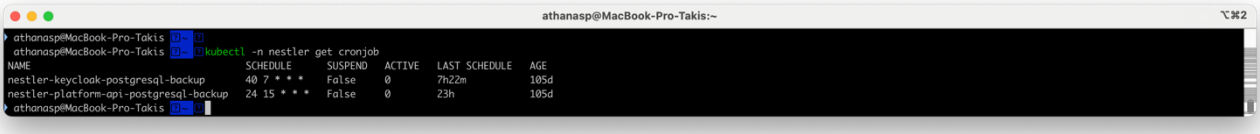

Figure 63: List of current cronjobs in the NESTLER namespace.

As a final notice, since Kubernetes might not be available to all developers during the coding and debugging procedures, docker-compose versions of the deployments will be available to the component developers, for the local development to be transformed into an easier procedure, closer to the real environment, rendered possible by the utilization of containers for development as well as local testing.

6 Conclusions

This deliverable constitutes an update of deliverable D4.1. It presents the updated NESTLER reference architecture shaped by user needs and both functional and non-functional requirements are demonstrated, with particular focus on the integration of data sources as well as backend and frontend services.

Also, provides an update on NESTLER AI Framework (NAIF), that is designed to facilitate the deployment of various Federated Learning frameworks, ensuring a versatile and efficient environment for training models using Federated Learning.

Another aspect of this document was the current NESTLER backend implementation of AI algorithms and agricultural services developed as part of the NESTLER platform to enable intelligent data processing, decision support, and the provision of advanced digital farming functionalities.

Moreover, this document provides an update on the NESTLER Integration framework and tools, which is designed to be dynamic and scalable, incorporating a DevSecOps approach that includes Source Code Management, CI/CD practices, Issue Tracking, Containerization and Deployment insights.

This document will be updated as the project progresses and its final version will be reported by the project consortium in Deliverable D4.4, entitled “NESTLER Integrated Platform Release” by the end of the project (in M42).

7 References

[1] NESTLER Consortium, D4.1 Initial NESTLER backend implementation of AI algorithms (v2.0), 2024.

[2] NESTLER Consortium, D3.1 Remote Sensing Technologies and Multimodal Data Aggregation Protocols.

[3] "SynField Smart Farming and Irrigation," [Online]. Available: https://app.synfield.gr/en/.

[4] "Keycloak: Open Source Identity and Access Management," [Online]. Available: https://www.keycloak.org/.

[5] "OpenID Connect," [Online]. Available: https://openid.net/.

[6] IETF, "The OAuth 2.0 Authorization Framework," [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6749.

[7] IETF, "Security Assertion Markup Language (SAML)," [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7522.

[8] "Geoserver: open source server for sharing geospatial data," [Online]. Available: https://geoserver.org/.

[9] "MinIO: Exascale object store for AI data, agentic computing, and analytics," [Online]. Available: https://www.min.io/.

[10] NESTLER Consortium, D3.2: NESTLER implementation of data aggregation protocols and AI algorithms.

[11] "The DEMETER Agricultural Information Model," [Online]. Available: https://ec.europa.eu/enrd/evaluation/knowledge-bank/demeter-agricultural-information-model_en.html.

[12] NESTLER Consortium, D4.2 NESTLER frontend implementation of GIS services.

[13] "Flower: A Friendly Federated AI Framework," [Online]. Available: https://flower.ai/.

[14] "OpenMined/TenSEAL," [Online]. Available: https://github.com/OpenMined/TenSEAL. [Accessed 24 July 2025].

[15] "open-quantum-safe/liboqs-python," [Online]. Available: https://github.com/open-quantum-safe/liboqspython.. [Accessed 25 July 2025].

[16] R. K. S. a. G. C. G. M. Gehlot, "Tomato-Village: a dataset for end-to-end tomato disease detection in a real-world environment," *Multimedia Systems,* vol. 29, no. 6, pp. 3305-3328, 1 12 2023.

[17] D. P. a. M. S. Hughes, "An open access repository of images on plant health to enable the development of mobile disease diagnostics through machine learning and crowdsourcing.," in *ArXiv abs/1511.08060*, 2015.

[18] L. B. A. K. D. W. X. Z. T. U. M. D. M. M. G. H. S. G. J. U. N. H. Alexey Dosovitskiy, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *arXiv,* vol. abs/2010.11929, 2020.

[19] "google/vit-base-patch16-224," [Online]. Available: https://huggingface.co/google/vit-base-patch16-224. [Accessed 3 6 2025].

[20] Yaoshiang Ho, Samuel Wookey, "The Real-World-Weight Cross-Entropy Loss Function: Modeling the Costs of Mislabeling," *IEEE Access,* vol. 8, pp. 4806 - 4813, 2019.

[21] "Weighted cross entropy loss formula," [Online]. Available: https://docs.pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html.

[22] "NASA POWER: Data access viewer," [Online]. Available: https://power.larc.nasa.gov/data-access-viewer/.

[23] "FastAPI: a high performance framework," [Online]. Available: https://fastapi.tiangolo.com/.

[24] "FastAPI: source code," [Online]. Available: https://github.com/fastapi.

[25] "NASA POWER: API documentation," [Online]. Available: https://power.larc.nasa.gov/docs/services/api/.

[26] "Docker: Accelerate how you build, share, and run applications," [Online]. Available: https://www.docker.com/.

[27] "Open Meteo API," [Online]. Available: https://open-meteo.com/en/docs.

[28] "Irrigation in EU agriculture," [Online]. Available: https://www.europarl.europa.eu/RegData/etudes/BRIE/2019/644216/EPRS_BRI(2019)644216_EN.pdf.

[29] U. Nations, "Sustainable Land and Water Management (SLWM) including Integrated Watershed Management Stratégies to ensure Food Security in Africa," [Online].

[30] "Nginx: an HTTP web server," [Online]. Available: https://nginx.org/.

[31] "Gunicorn: a Python WSGI HTTP Server," [Online]. Available: https://gunicorn.org/.

[32] "Redis: a key-value database," [Online]. Available: https://redis.io/.

[33] "RabbitMQ: a powerful, enterprise grade open source messaging and streaming broker," [Online]. Available: https://www.rabbitmq.com/.

[34] "Celery: Distributed Task Queue," [Online]. Available: https://docs.celeryq.dev/en/stable/.

[35] "Food and Agriculture Organization: comprehensive agricultural data," [Online]. Available: https://www.fao.org/faostat/en/#data/QCL.

[36] "Swagger: API Development," [Online]. Available: https://swagger.io/.

[37] "OpenAPI: API description standard," [Online]. Available: https://www.openapis.org/.

[38] "Geoserver: REST API," [Online]. Available: https://docs.geoserver.org/2.20.x/en/user/rest/index.html.

[39] "Jakarta EE:," [Online]. Available: https://jakarta.ee/.

[40] "Quarkus: A Kubernetes Native Java stack tailored for OpenJDK HotSpot and GraalVM, crafted from the best of breed Java libraries and standards," [Online]. Available: https://quarkus.io/.

[41] "Hibernate: object-relational mapping framework for Java," [Online]. Available: https://hibernate.org/.

[42] "WildFly: A powerful, modular, & lightweight application serve," [Online]. Available: https://www.wildfly.org/.

[43] "Keycloak: REST API," [Online]. Available: https://www.keycloak.org/docs-api/latest/rest-api/index.html.

[44] "Securing applications and services with OpenID Connect," [Online]. Available: https://www.keycloak.org/securing-apps/oidc-layers.

[45] "Git: a free and open source distributed version control system," [Online]. Available: https://git-scm.com/.

[46] Gitlab, "Source code management," [Online]. Available: https://about.gitlab.com/.

[47] IBM Cloud Education, "DevSecOps," 2020. [Online]. Available: https://www.ibm.com/cloud/learn/devsecops.

[48] "GitLab Continuous Integration & Delivery," [Online]. Available: https://about.gitlab.com/product/continuous-integration/.

[49] "Introduction to CI/CD with GitLab," [Online]. Available: https://docs.gitlab.com/ee/ci/introduction/index.html.

[50] "Docker Hub Container Image Library: App Containerization," [Online]. Available: https://hub.docker.com/.

[51] "Rook :Open-Source, Cloud-Native Storage for Kubernetes," [Online]. Available: https://rook.io.

[52] "Ceph: Home page," [Online]. Available: https://ceph.io.

[53] "Traefik: the cloud native application proxy," [Online]. Available: https://traefik.io/traefik.

[54] "The International Union for the Protection of New Varieties of Plants (UPOV)," [Online]. Available: https://www.upov.int/portal/index.html.en.

8 Annex – API documentation

8.1 Weather Impact Assessment Services

8.1.1 Drought Forecast Service API

The following subsections present the drought forecast API, which provide the respective services.

Table 6: Monitoring sites of the drought forecast service.

Field	Description
Title	Monitoring sites of the drought forecast service (DFS)
ID	DFS-001
Description	Returns a comprehensive list of all supported monitoring sites, detailing the available countries and their associated cities
HTTP Method(s)	GET
Endpoint	https://drought-detection.platform.nestler-project.eu/site-list
Authentication	N/A
Header(s)	N/A
Query parameter(s)	N/A
Request payload	N/A
Successful response	
	200 OK, JSON object
Error response	
	400 Bad request
	500 Internal error
Notes	N/A

Table 7: Provision of the drought forecast for a site.

Field	Description
Title	Drought forecast for a given site
ID	DFS-002
Description	The core forecasting endpoint. It accepts a location (country and city), a SPEI threshold for defining drought, and a future month for the forecast. The API then dynamically retrieves the latest soil temperature and precipitation data, preprocesses it, and executes the region-specific best-performing model to return a drought forecast (e.g., "drought" or "no drought"), the model's name, and its full suite of performance metrics (Recall, Precision, Accuracy, Specificity).
HTTP Method(s)	POST
Endpoint	https://drought-detection.platform.nestler-project.eu/forecast/drought
Authentication	N/A
Header(s)	Accept: application/json
Query parameter(s)	N/A
Request payload	N/A
Successful response	
	200 OK, JSON object
Error response	
	400 Bad request
	500 Internal error
Notes	N/A

Table 8: Heath endpoint of the drought forecast service.

Field	Description
Title	Health of the drought forecast service
ID	DFS-003
Description	Provides a health check of the service, verifying its connectivity to cloud storage and the availability of its dataset
HTTP Method(s)	GET
Endpoint	https://drought-detection.platform.nestler-project.eu/health
Authentication	N/A
Header(s)	Accept: application/json
Query parameter(s)	N/A
Request payload	N/A
Successful response	
	200 OK, JSON object
Error response	
	400 Bad request
	500 Internal error
Notes	N/A

8.1.2 Flood Forecast Service API

The following subsections present the flood forecast API, which provide the respective services.

Table 9: Monitoring sites of the flood forecast service.

Field	Description
Title	Monitoring sites of the flood forecast service (FFS)
ID	FFS-001
Description	Returns a comprehensive list of all supported regions (cities and countries) along with their specific rainy seasons, providing necessary metadata for client applications.
HTTP Method(s)	GET
Endpoint	https://flood-detection-api.platform.nestler-project.eu/site-list
Authentication	N/A
Header(s)	Accept: application/json
Query parameter(s)	N/A
Request payload	N/A
Successful response	
	200 OK, JSON object
Error response	
	400 Bad request
	500 Internal error
Notes	N/A

Table 10: Provision of the historical monthly precipitation data for a site of the flood forecast service.

Field	Description
Title	Historical monthly precipitation data for a given site (FFS)
ID	FFS-002
Description	Provides access to historical monthly precipitation data for any given site from 1981 to 2022, crucial for analysis and contextualizing forecasts.
HTTP Method(s)	GET
Endpoint	https://flood-detection-api.platform.nestler-project.eu/dataset/flood
Authentication	N/A
Header(s)	Accept: application/json
Query parameter(s)	N/A
Request payload	N/A
Successful response	
	200 OK, JSON object
Error response	
	400 Bad request
	500 Internal error
Notes	N/A

Table 11: Provision of the flood forecast for a site.

Field	Description
Title	Flood forecast for a given site (FFS)
ID	FFS-003
Description	The core forecasting endpoint. It accepts a location (country and city) and a target rainy season period. The API then dynamically retrieves the current year's precipitation data, preprocesses it, and executes the region-specific best-performing model to return a flood forecast (binary outcome), the model's name, and its full suite of performance metrics (Recall, Precision, Accuracy, Specificity).
HTTP Method(s)	POST
Endpoint	https://flood-detection-api.platform.nestler-project.eu/forecast/flood
Authentication	N/A
Header(s)	Accept: application/json
Query parameter(s)	N/A

Request payload	{
	"country": "string",
	"sitename": "string",
	"period": "string",
	"thres": 0.5
	}

Successful response	
	200 OK, JSON Object
Error response	
	400 Bad request
	500 Internal error
Notes	N/A

Table 12: Heath endpoint of the flood forecast service.

Field	Description
Title	Health of the flood forecast service
ID	FFS-004
Description	Provides a health check of the service, verifying its connectivity to cloud storage and the availability of its dataset
HTTP Method(s)	GET
Endpoint	https://flood-detection-api.platform.nestler-project.eu/health
Authentication	N/A
Header(s)	N/A
Query parameter(s)	N/A
Request payload	N/A
Successful response	
	200 OK, JSON Object
Error response	
	400 Bad request
	500 Internal error
Notes	N/A

8.2 Zoonotic Disease Outbreak Service API

Table 13 presents the web service that the zoonotic outbreak risk by given disease, date and geographic area (bounding box - bbox). Although the default response format is PNG8, further formats are also supported.

Table 13: Disease-specific zoonotic outbreak risk in given date and geographic area.

Field	Description
Title	Disease-specific zoonotic outbreak risk.
ID	ZDS-001
Description	Provides the zoonotic outbreak risk in given disease, date and geographic area.
HTTP Method(s)	GET
Endpoint	https://geoserver-api.platform.nestler-project.eu/geoserver/NESTLER/wms
Authentication	IAM
Header(s)	Accept: image/* Authorization: Bearer {token}
Query parameter(s)	• service=WMS • request=GetMap • layers={DISEASE_LAYER_NAME} • styles={DISEASE_LAYER_STYLE} • format=image:png8 • transparent=true • version=1.3.0 • time={YYYY-MM-DD} • width=256 • height=256 • crs=EPSG:3857 • bbox={minX,minY,maxX,maxY}
Request payload	N/A
Successful response	
	200 OK, PNG8
Error response	
	401 Unauthorized
	400 Bad request
	500 Internal error
Notes	N/A

8.3 NESTLER API

8.3.1 NESTLER Generic API

Table 14: Retrieve the list of Animal Category.

Field	Description
Title	Animal Category read-only view set.
ID	API-SYS-001
Description	Animal Category read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/animals/categories/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (AnimalCategoryRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 15: Retrieve the list of animal health conditions.

Field	Description
Title	Animal Health Condition read-only view set.
ID	API-SYS-002
Description	Animal Health Condition read-only view set.

Field	Description
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/animals/health-conditions/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (AnimalHealthConditionRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 16: Retrieve the list of the animal reproductive conditions.

Field	Description
Title	Animal Reproductive Condition read-only view set.
ID	API-SYS-003
Description	Animal Reproductive Condition read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/animals/reproductive-conditions/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A

Field	Description
Successful response	
	200, Object list (AnimalReproductiveConditionRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 17: Retrieve the list of the animal sex choices.

Field	Description
Title	Animal Sex read-only view set.
ID	API-SYS-004
Description	Animal Sex read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/animals/sex/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (AnimalSexRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error

Field	Description
Notes	N/A

Table 18: Retrieve the list of animal species.

Field	Description
Title	Animal Species read-only view set.
ID	API-SYS-005
Description	Animal Species read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/animals/species/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (AnimalSpeciesRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 19: Retrieve the list of the available APIs.

Field	Description
Title	API read-only view set.
ID	API-SYS-006
Description	API read-only view set.

Field	Description
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/apis/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (APIRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 20: Retrieve the list of the API providers.

Field	Description
Title	API Provider read-only view set.
ID	API-SYS-007
Description	API Provider read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/apis/providers/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	

Field	Description
	200, Object list (APIProviderRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 21: Retrieve the list of APIs per provider.

Field	Description
Title	API by provider read-only view set.
ID	API-SYS-008
Description	API by provider read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/apis/providers/{provider_id}/apis/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (APIRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 22: Retrieve the list of the crop phenologies.

Field	Description
Title	CropPhenology read-only view set.
ID	API-SYS-010
Description	CropPhenology read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/crops/phenologies/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page.
Request payload	N/A
Successful response	
	200, Paginated list of objects (CropPhenologyRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 23: Retrieve the list of UPOV crops [54].

Field	Description
Title	UPOVCrop read-only view set.
ID	API-SYS-011
Description	UPOVCrop read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/crops/species/
Authentication	Yes

Field	Description
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (UPOVCropRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 24: Retrieve the list of taxonomic species by UPOV crop.

Field	Description
Title	Taxonomic Species by UPOV Crop read-only view set.
ID	API-SYS-012
Description	Taxonomic Species by UPOV Crop read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/crops/species/{species_id}/pests/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (TaxonomicSpeciesRetrieve)
Error response	
	400 Bad request

Field	Description
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 25: Retrieve the list of crop phenologies by UPOV crop.

Field	Description
Title	CropPhenology by UPOVCrop read-only view set.
ID	API-SYS-013
Description	CropPhenology by UPOVCrop read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/crops/species/{species_id}/phenologies/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page.
Request payload	N/A
Successful response	
	200, Paginated list of objects (CropPhenologyRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 26: Retrieve the list of crop varieties by UPOV crop.

Field	Description
Title	CropVariety by UPOVCrop read-only view set.
ID	API-SYS-014
Description	CropVariety by UPOVCrop read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/crops/species/{species_id}/varieties/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (CropVarietyRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 27: Retrieve the list of crop varieties.

Field	Description
Title	CropVariety read-only view set.
ID	API-SYS-015
Description	CropVariety read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/crops/varieties/
Authentication	Yes

Field	Description
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (CropVarietyRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 28: Retrieve the list of crop phenologies by crop variety.

Field	Description
Title	CropPhenology by CropVariety read-only view set.
ID	API-SYS-016
Description	CropPhenology by CropVariety read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/crops/varieties/{variety_id}/phenologies/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page.
Request payload	N/A
Successful response	
	200, Paginated list of objects (CropPhenologyRetrieve)
Error response	

Field	Description
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 29: Retrieve the list of devices.

Field	Description
Title	Device read-only view set.
ID	API-SYS-017
Description	Device read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/devices/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (DeviceRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 30: Retrieve the list of the actuating service types.

Field	Description
Title	ActuatingServiceType view set.
ID	API-SYS-018
Description	ActuatingServiceType view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/devices/actuators/services/types/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (ActuatingServiceTypeRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 31: Retrieve the list of the actuator types.

Field	Description
Title	ActuatorType view set.
ID	API-SYS-019
Description	ActuatorType view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/devices/actuators/types/
Authentication	Yes

Field	Description
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (ActuatorTypeRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 32: Retrieve the list of the devices' manufactures.

Field	Description
Title	Manufacturer read-only view set.
ID	API-SYS-021
Description	Manufacturer read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/devices/manufacturers/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (ManufacturerRetrieve)
Error response	
	400 Bad request

Field	Description
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 33: Retrieve the list of the device models.

Field	Description
Title	Model read-only view set.
ID	API-SYS-022
Description	Model read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/devices/models/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (ModelRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 34: Retrieve the list of the sensor types.

Field	Description
Title	SensorType read-only view set.
ID	API-SYS-023
Description	SensorType read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/devices/sensors/types/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (SensorTypeRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 35: Retrieve the list of the cities.

Field	Description
Title	City read-only view set.
ID	API-SYS-024
Description	City read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/locations/cities/
Authentication	Yes

Field	Description
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page.
Request payload	N/A
Successful response	
	200, Paginated list of objects (CityRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 36: Retrieve the list of the countries.

Field	Description
Title	Country read-only view set.
ID	API-SYS-025
Description	Country read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/locations/countries/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page.
Request payload	N/A
Successful response	
	200, Paginated list of objects (CountryRetrieve)
Error response	

Field	Description
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 37: Retrieve the list of the geographic regions.

Field	Description
Title	Region read-only view set.
ID	API-SYS-026
Description	Region read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/locations/regions/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page.
Request payload	N/A
Successful response	
	200, Paginated list of objects (RegionRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 38: Retrieve the list of the geographic sub-regions.

Field	Description
Title	SubRegion read-only view set.
ID	API-SYS-027
Description	SubRegion read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/locations/subregions/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page.
Request payload	N/A
Successful response	
	200, Paginated list of objects (SubRegionRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 39: Retrieve the list of the aggregation time period choices.

Field	Description
Title	AggregationTimePeriod read-only view set.
ID	API-SYS-028
Description	AggregationTimePeriod read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/measurements/aggregations/periods/
Authentication	Yes

Field	Description
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (AggregationTimePeriodRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 40: Retrieve the list of the aggregation types.

Field	Description
Title	AggregationType read-only view set.
ID	API-SYS-029
Description	AggregationType read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/measurements/aggregations/types/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (AggregationTypeRetrieve)
Error response	
	400 Bad request

Field	Description
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 41: Retrieve the list of the measurement origins.

Field	Description
Title	MeasurementOrigin read-only view set.
ID	API-SYS-030
Description	MeasurementOrigin read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/measurements/origins/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (MeasurementOriginRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 42: Retrieve the list of the measurement types.

Field	Description
Title	MeasurementType read-only view set.
ID	API-SYS-032
Description	MeasurementType read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/measurements/types/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (MeasurementTypeRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 43: Retrieve the list of the organizations (project consortium).

Field	Description
Title	Organization read-only view set.
ID	API-SYS-033
Description	Organization read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/organizations/
Authentication	Yes

Field	Description
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (OrganizationRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 44: Retrieve the list of the pilots by organization.

Field	Description
Title	Pilot by organization read-only view set.
ID	API-SYS-034
Description	Pilot by organization read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/organizations/{organization_id}/pilots/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (PilotRetrieve)
Error response	
	400 Bad request

Field	Description
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 45: Retrieve the list of the crop varieties.

Field	Description
Title	CropVariety read-only view set.
ID	API-SYS-035
Description	CropVariety read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/parcels/crop-varieties/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (CropVarietyRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 46: Retrieve the list of the irrigation system types.

Field	Description
Title	IrrigationSystemType read-only view set.

Field	Description
ID	API-SYS-036
Description	IrrigationSystemType read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/parcels/irrigation-systems/types/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (IrrigationSystemTypeRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 47: Retrieve the list of the land utilization types.

Field	Description
Title	LandUtilisation read-only view set.
ID	API-SYS-037
Description	LandUtilisation read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/parcels/land-utilisations/types/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}

Field	Description
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (LandUtilisationRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 48: Retrieve the list of the soil texture types.

Field	Description
Title	SoilTextureType read-only view set.
ID	API-SYS-038
Description	SoilTextureType read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/parcels/soil-textures/types/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (SoilTextureTypeRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden

Field	Description
	500 Internal error
Notes	N/A

Table 49: Retrieve the list of the UPOV crops.

Field	Description
Title	UPOVCrop read-only view set.
ID	API-SYS-039
Description	UPOVCrop read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/parcels/upov-crops/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (UPOVCropRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 50: Retrieve the list of the crop varieties per UPOV crop.

Field	Description
Title	CropVariety by UPOVCrop read-only view set.
ID	API-SYS-040
Description	CropVariety by UPOVCrop read-only view set.

Field	Description
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/parcels/upov-crops/{upov_crop_id}/varieties/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (CropVarietyRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 51: Retrieve the list of the pilots.

Field	Description
Title	Pilot read-only view set.
ID	API-SYS-045
Description	Pilot read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/pilots/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	

Field	Description
	200, Object list (PilotRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 52: Retrieve the list of the use cases per pilot.

Field	Description
Title	Use case by pilot read-only view set.
ID	API-SYS-046
Description	Use case by pilot read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/pilots/{pilot_id}/use-cases/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (UseCaseRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 53: Retrieve the list of the sensor types.

Field	Description
Title	SensorType read-only view set.
ID	API-SYS-047
Description	SensorType read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/sensors/types/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (SensorTypeRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 54: Retrieve the list of the taxonomic categories.

Field	Description
Title	Taxonomic Category read-only view set.
ID	API-SYS-048
Description	Taxonomic Category read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/taxonomy/categories/
Authentication	Yes

Field	Description
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (TaxonomicCategoryRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 55: Retrieve the list of the taxonomic classes.

Field	Description
Title	Taxonomic Class read-only view set.
ID	API-SYS-049
Description	Taxonomic Class read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/taxonomy/classes/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (TaxonomicClassRetrieve)
Error response	
	400 Bad request

Field	Description
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 56: Retrieve the list of the taxonomic family.

Field	Description
Title	Taxonomic Family read-only view set.
ID	API-SYS-050
Description	Taxonomic Family read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/taxonomy/families/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (TaxonomicFamilyRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 57: Retrieve the list of the taxonomic genus.

Field	Description
Title	Taxonomic Genus read-only view set.
ID	API-SYS-051
Description	Taxonomic Genus read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/taxonomy/genes/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (TaxonomicGenusRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 58: Retrieve the list of the taxonomic kingdoms.

Field	Description
Title	Taxonomic Kingdom read-only view set.
ID	API-SYS-052
Description	Taxonomic Kingdom read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/taxonomy/kingdoms/
Authentication	Yes

Field	Description
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (TaxonomicKingdomRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 59: Retrieve the list of the taxonomic orders.

Field	Description
Title	Taxonomic Order read-only view set.
ID	API-SYS-055
Description	Taxonomic Order read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/taxonomy/orders/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (TaxonomicOrderRetrieve)
Error response	
	400 Bad request

Field	Description
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 60: Retrieve the list of the taxonomic plylum.

Field	Description
Title	Taxonomic Phylum read-only view set.
ID	API-SYS-056
Description	Taxonomic Phylum read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/taxonomy/phyla/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (TaxonomicPhylumRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 61: Retrieve the list of the taxonomic species.

Field	Description
Title	Taxonomic Species read-only view set.
ID	API-SYS-057
Description	Taxonomic Species read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/taxonomy/species/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page.
Request payload	N/A
Successful response	
	200, Paginated list of objects (TaxonomicSpeciesRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 62: Retrieve the list of the UPOV crops by taxonomic species.

Field	Description
Title	UPOV Crop by Taxonomy Species read-only view set.
ID	API-SYS-058
Description	UPOV Crop by Taxonomy Species read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/taxonomy/species/{species_id}/hosts/
Authentication	Yes

Field	Description
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (UPOVCropRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 63: Retrieve the list of the vector-borne diseases by taxonomic species.

Field	Description
Title	Vector borne diseases list that by given taxonomic species
ID	API-SYS-060
Description	Vector borne diseases list that by given taxonomic species
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/taxonomy/species/{species_id}/vector-borne-diseases/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page.
Request payload	N/A
Successful response	
	200, Paginated list of objects (VectorBorneDiseaseRetrieve)
Error response	

Field	Description
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 64: Retrieve the list of the trap models.

Field	Description
Title	Trap model list
ID	API-SYS-061
Description	Trap model list
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/trap-models/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page.
Request payload	N/A
Successful response	
	200, Paginated list of objects (TrapModelRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 65: Retrieve the list of the taxonomic species.

Field	Description
Title	Taxonomic Species list
ID	API-SYS-062
Description	Taxonomic Species list
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/trap-models/{trap_model_id}/taxonomy/species/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page.
Request payload	N/A
Successful response	
	200, Paginated list of objects (TaxonomicSpeciesRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 66: Retrieve the list of the use cases.

Field	Description
Title	Use case read-only view set.
ID	API-SYS-063
Description	Use case read-only view set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/use-cases/
Authentication	Yes

Field	Description
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (UseCaseRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 67: Retrieve the list of the vector-borne diseases.

Field	Description
Title	Fetch the collection of vector-borne diseases
ID	API-SYS-070
Description	Fetch the collection of vector-borne diseases
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/generic/api/v1/vector-borne-diseases/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page.
Request payload	N/A
Successful response	
	200, Paginated list of objects (VectorBorneDiseaseRetrieve)
Error response	
	400 Bad request

Field	Description
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

8.3.2 NESTLER Core API

Table 68: Retrieve the list of the installed actuators.

Field	Description
Title	List Actuators
ID	API-CORE-001
Description	List Actuators
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/actuators/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (ActuatorRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 69: Retrieve details of an installed actuator.

Field	Description
Title	Retrieve an Actuator

Field	Description
ID	API-CORE-002
Description	Retrieve an Actuator
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/actuators/{actuator_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, ()
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 70: Update the installed actuator.

Field	Description
Title	Update an Actuator
ID	API-CORE-002
Description	Update an Actuator
HTTP Method	PUT
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/actuators/{actuator_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	{ "id": 0,

Field	Description
	"identifier": "string", "node": 0, "parcel": 0, "stable": 0, "actuator_type": 0, "description": "string", "geom": "string", "point": "string", "manual": true, "updated": true, "latest_update": "2025-09-30T06:43:07.278Z", "value": 0 }
Successful response	
	200, Object (ActuatorRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 71: Delete the installed actuator.

Field	Description
Title	Delete an Actuator
ID	API-CORE-002
Description	Delete an Actuator
HTTP Method	DELETE
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/actuators/{actuator_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}

Field	Description
Query parameter(s)	
Request payload	N/A
Successful response	
	204
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 72: Retrieve the list of stables' animals.

Field	Description
Title	List Stable Animals
ID	API-CORE-003
Description	List Stable Animals
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/animals/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (StableAnimalRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error

Field	Description
Notes	N/A

Table 73: Retrieve the list of animal groups.

Field	Description
Title	List Stable Animal Groups
ID	API-CORE-004
Description	List Stable Animal Groups
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/animals/groups/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (StableAnimalRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 74: Retrieve details of the animal group.

Field	Description
Title	Retrieve a Stable Animal Group
ID	API-CORE-005
Description	Retrieve a Stable Animal Group
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/animals/groups/{group_id}/

Field	Description
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, ()
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 75: Update the animal group.

Field	Description
Title	Update a Stable Animal Group
ID	API-CORE-005
Description	Update a Stable Animal Group
HTTP Method	PUT
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/animals/groups/{group_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	{ "id": 0, "group": 0, "identifier": "string", "name": "string", "birth_date": "2025-10-03", "death_date": "2025-10-03",

Field	Description
	"reproductive_condition": 0, "health_condition": 0, "sex": 0, "species": 0, "lactation_stage": 2147483647, "days_in_milk": 2147483647 }
Successful response	
	200, Object (StableAnimalRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 76: Delete the animal group.

Field	Description
Title	Delete a Stable Animal Group
ID	API-CORE-005
Description	Delete a Stable Animal Group
HTTP Method	DELETE
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/animals/groups/{group_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	204
Error response	

Field	Description
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 77: Retrieve the list of animals by group.

Field	Description
Title	List Animals by Group
ID	API-CORE-006
Description	List Animals by Group
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/animals/groups/{group_id}/animals/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (StableAnimalRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 78: Register a new animal group.

Field	Description
Title	Create a Stable Animal Group

Field	Description
ID	API-CORE-006
Description	Create a Stable Animal Group
HTTP Method	POST
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/animals/groups/{group_id}/animals/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	

Request payload	{
	"id": 0,
	"group": 0,
	"identifier": "string",
	"name": "string",
	"birth_date": "2025-10-03",
	"death_date": "2025-10-03",
	"reproductive_condition": 0,
	"health_condition": 0,
	"sex": 0,
	"species": 0,
	"lactation_stage": 2147483647,
	"days_in_milk": 2147483647
	}

Successful response	
	201, Object (StableAnimalRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 79: Retrieve details of an animal.

Field	Description
Title	Retrieve a Stable Animal
ID	API-CORE-007
Description	Retrieve a Stable Animal
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/animals/{animal_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, ()
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 80: Update details of an animal.

Field	Description
Title	Update a Stable Animal
ID	API-CORE-007
Description	Update a Stable Animal
HTTP Method	PUT
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/animals/{animal_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	

Field	Description

Request payload	{
	"id": 0,
	"group": 0,
	"identifier": "string",
	"name": "string",
	"birth_date": "2025-10-03",
	"death_date": "2025-10-03",
	"reproductive_condition": 0,
	"health_condition": 0,
	"sex": 0,
	"species": 0,
	"lactation_stage": 2147483647,
	"days_in_milk": 2147483647
	}

Successful response	
	200, Object (StableAnimalRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 81: Delete an animal.

Field	Description
Title	Delete a Stable Animal
ID	API-CORE-007
Description	Delete a Stable Animal
HTTP Method	DELETE
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/animals/{animal_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}

Field	Description
Query parameter(s)	
Request payload	N/A
Successful response	
	204, ()
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 82: Upload and store captures from traps (via camera).

Field	Description
Title	Upload Camera Capture
ID	API-CORE-012
Description	Upload Camera Capture
HTTP Method	POST
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/cameras/{serial_number}/captures/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	• File • created
Successful response	
	201, Object (CameraCaptureCreate)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error

Field	Description
Notes	N/A

Table 83: Retrieve the list of the complexes.

Field	Description
Title	List Activity Complexes
ID	API-CORE-013
Description	List Activity Complexes
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/complexes/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (ActivityComplexRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 84: Register a new complex.

Field	Description
Title	Create an Activity Complex
ID	API-CORE-013
Description	Create an Activity Complex
HTTP Method	POST
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/complexes/

Field	Description
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	

Request payload	{
	"id": 0,
	"pilot": 0,
	"name": "string",
	"description": "string",
	"location": "string",
	"geom": "string",
	"point": "string",
	"allow_organization_management": true,
	"organization_wide_publishing": true
	}

Successful response	
	201, Object (ActivityComplexRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 85: Retrieve details of a complex.

Field	Description
Title	Retrieve an Activity Complex
ID	API-CORE-014
Description	Retrieve an Activity Complex
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/complexes/{complex_id}/
Authentication	Yes

Field	Description
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, ()
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 86: Update a complex.

Field	Description
Title	Update an Activity Complex
ID	API-CORE-014
Description	Update an Activity Complex
HTTP Method	PUT
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/complexes/{complex_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	{ "id": 0, "pilot": 0, "name": "string", "description": "string", "location": "string", "geom": "string", "point": "string",

Field	Description
	"allow_organization_management": true, "organization_wide_publishing": true }
Successful response	
	200, Object (ActivityComplexRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 87: Delete a complex.

Field	Description
Title	Delete an Activity Complex
ID	API-CORE-014
Description	Delete an Activity Complex
HTTP Method	DELETE
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/complexes/{complex_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	204
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error

Field	Description
Notes	N/A

Table 88: Retrieve the list of parcels by complex.

Field	Description
Title	List Parcels by Complex
ID	API-CORE-016
Description	List Parcels by Complex
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/complexes/{complex_id}/parcels/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	complex: Search parcel by complex category: Search parcel by category soil_texture_type: Search parcel by soil texture type irrigation_system_type: Search parcel by irrigation system type
Request payload	N/A
Successful response	
	200, Object list (ParcelRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 89: Register a new parcel.

Field	Description
Title	Create a Parcel
ID	API-CORE-016
Description	Create a Parcel
HTTP Method	POST

Field	Description
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/complexes/{complex_id}/parcels/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	complex: Search parcel by complex category: Search parcel by category soil_texture_type: Search parcel by soil texture type irrigation_system_type: Search parcel by irrigation system type
Request payload	{ "id": 0, "complex": 0, "parent_parcel": 0, "name": "string", "description": "string", "area": "string", "geom": "string", "point": "string", "location": "string", "category": 0, "soil_texture_type": 0, "irrigation_system_type": 0, "allow_organization_management": true, "organization_wide_publishing": true, "last_planted_at": "2025-10-03T06:54:30.234Z", "crops": [{ "id": 0, "parcel": 0, "crop_species": 0, "crop_variety": 0, "geom": "string", "point": "string", "valid_from": "2025-10-03T06:54:30.234Z", "valid_to": "2025-10-03T06:54:30.234Z", "last_planted_at": "2025-10-03T06:54:30.234Z", "production_amount": "string", "soil_texture_type": 0,

Field	Description
	"name": "string", "planting_row_distance": 0, "planting_distance_in_row": 0, "planting_density": 0, "year": 32767 }] }
Successful response	
	201, Object (ParcelRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 90: Retrieve the list of stable by complex.

Field	Description
Title	List Stables by Complex
ID	API-CORE-017
Description	List Stables by Complex
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/complexes/{complex_id}/stables/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (StableRetrieve)
Error response	

Field	Description
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 91: Register a new stable.

Field	Description
Title	Create a Stable
ID	API-CORE-017
Description	Create a Stable
HTTP Method	POST
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/complexes/{complex_id}/stables/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	{ "id": 0, "complex": 0, "name": "string", "description": "string", "geom": "string", "point": "string", "location": "string", "allow_organization_management": true, "organization_wide_publishing": true, "groups": [{ "id": 0, "stable": 0, "animal_category": 0, "name": "string",

Field	Description

	"animals": [
	{
	"id": 0,
	"group": 0,
	"identifier": "string",
	"name": "string",
	"birth_date": "2025-10-03",
	"death_date": "2025-10-03",
	"reproductive_condition": 0,
	"health_condition": 0,
	"sex": 0,
	"species": 0,
	"lactation_stage": 2147483647,
	"days_in_milk": 2147483647
	}
	]
	}
	]
	}

Successful response	
	201, Object (StableRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 92: Retrieve the list of crops.

Field	Description
Title	List Parcel Crops
ID	API-CORE-018
Description	List Parcel Crops

Field	Description
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/crops/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (ParcelCropRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 93: Retrieve details of a parcel's crop.

Field	Description
Title	Retrieve a Parcel Crop
ID	API-CORE-019
Description	Retrieve a Parcel Crop
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/crops/{crop_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, ()
Error response	

Field	Description
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 94: Update a parcel's crop.

Field	Description
Title	Update a Parcel Crop
ID	API-CORE-019
Description	Update a Parcel Crop
HTTP Method	PUT
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/crops/{crop_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	{ "id": 0, "parcel": 0, "crop_species": 0, "crop_variety": 0, "geom": "string", "point": "string", "valid_from": "2025-09-03T06:56:32.855Z", "valid_to": "2025-09-03T06:56:32.855Z", "last_planted_at": "2025-09-03T06:56:32.855Z", "production_amount": "string", "soil_texture_type": 0, "name": "string", "planting_row_distance": 0, "planting_distance_in_row": 0, "planting_density": 0,

Field	Description
	"year": 2025 }
Successful response	
	200, Object (ParcelCropRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 95: Delete a parcel's crop.

Field	Description
Title	Delete a Parcel Crop
ID	API-CORE-019
Description	Delete a Parcel Crop
HTTP Method	DELETE
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/crops/{crop_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	204
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error

Field	Description
Notes	N/A

Table 96: Retrieve the list of measurements of a parcel's crop.

Field	Description
Title	List Measurements by Parcel Crop
ID	API-CORE-021
Description	List Measurements by Parcel Crop
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/crops/{crop_id}/measurements/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page. sensor: Search measurements by sensor ID measurement_type: Search measurements by measurement type ID complex: Search measurements by complex parcel: Search measurements by parcel stable: Search measurements by stable from_date: The date to fetch measurements from. Accepted formats: YYYY-mm-dd (e.g. 2024-05-17) YYYY-mm-ddTHH:MM:ss (e.g. 2024-05-17T15:30:32) to_date: The date to fetch measurements to. Accepted formats: YYYY-mm-dd (e.g. 2024-05-17) YYYY-mm-ddTHH:MM:ss (e.g. 2024-05-17T15:30:32)
Request payload	N/A
Successful response	
	200, Object list (MeasurementRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 97: Register a new measurement of a parcel's crop.

Field	Description
Title	Create a Measurement by Parcel Crop
ID	API-CORE-021
Description	Create a Measurement by Parcel Crop
HTTP Method	POST
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/crops/{crop_id}/measurements/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page. sensor: Search measurements by sensor ID measurement_type: Search measurements by measurement type ID complex: Search measurements by complex parcel: Search measurements by parcel stable: Search measurements by stable from_date: The date to fetch measurements from. Accepted formats: YYYY-mm-dd (e.g. 2024-05-17) YYYY-mm-ddTHH:MM:ss (e.g. 2024-05-17T15:30:32) to_date: The date to fetch measurements to. Accepted formats: YYYY-mm-dd (e.g. 2024-05-17) YYYY-mm-ddTHH:MM:ss (e.g. 2024-05-17T15:30:32)

Request payload	{
	"id": 0,
	"parcel": 0,
	"crop": 0,
	"stable": 0,
	"sensor": 0,
	"group": 0,
	"measurement_type": 0,
	"value": "string",
	"unit": "string",
	"depth": 100,
	"depth_range": "string",
	"date_observed": "2025-09-03T06:59:15.835Z",
	"origin": 0,
	"target": 0,
	"aggregation": 0,
	"aggregation_period": 0
	}

Field	Description
Successful response	
	201, Object (MeasurementRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 98: Retrieve the latest measurements of a parcel's crop.

Field	Description
Title	Measurement by Crop List / Create View.
ID	API-CORE-022
Description	Measurement by Crop List / Create View.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/crops/{crop_id}/measurements/latest/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page.
Request payload	N/A
Successful response	
	200, Paginated list of objects (MeasurementRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 99: Retrieve phenologies of a parcel's crop.

Field	Description
Title	List Phenologies by Parcel Crop
ID	API-CORE-024
Description	List Phenologies by Parcel Crop
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/crops/{crop_id}/phenology/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (ParcelCropPhenologyRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 100: Register the current phenology of a parcel's crop.

Field	Description
Title	Create a Phenology by Parcel Crop
ID	API-CORE-024
Description	Create a Phenology by Parcel Crop
HTTP Method	POST
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/crops/{crop_id}/phenology/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	

Field	Description

Request payload	{
	"id": 0,
	"parcel": 0,
	"crop": 0,
	"phenology": 0,
	"date_observed": "2025-09-03T07:01:02.515Z",
	"remarks": "string"
	}

Successful response	
	201, Object (ParcelCropPhenologyRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 101: Retrieve the list of diseases.

Field	Description
Title	List Diseases
ID	API-CORE-025
Description	List Diseases
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/diseases/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (DiseaseRetrieve)
Error response	

Field	Description
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 102: Retrieve the list of diseases' stages.

Field	Description
Title	List Disease Stages
ID	API-CORE-028
Description	List Disease Stages
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/diseases/stages/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (DiseaseStageRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 103: Retrieve details of a disease's stage.

Field	Description
Title	Retrieve a Disease Stage
ID	API-CORE-029
Description	Retrieve a Disease Stage
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/diseases/stages/{stage_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, ()
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 104: Update an existing disease's stage.

Field	Description
Title	Update a Disease Stage
ID	API-CORE-029
Description	Update a Disease Stage
HTTP Method	PUT
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/diseases/stages/{stage_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	

Field	Description

Request payload	{
	"id": 0,
	"disease": 0,
	"stage": 0,
	"date_observed": "2025-09-03T07:03:25.474Z"
	}

Successful response	
	200, Object (DiseaseStageRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 105: Delete an existing disease's stage.

Field	Description
Title	Delete a Disease Stage
ID	API-CORE-029
Description	Delete a Disease Stage
HTTP Method	DELETE
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/diseases/stages/{stage_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	204
Error response	
	400 Bad request

Field	Description
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 106: Retrieve details of an existing disease.

Field	Description
Title	Retrieve a Disease
ID	API-CORE-030
Description	Retrieve a Disease
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/diseases/{disease_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, ()
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 107: Update an existing disease.

Field	Description
Title	Update a Disease
ID	API-CORE-030
Description	Update a Disease
HTTP Method	PUT
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/diseases/{disease_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	

Request payload	{
	"id": 0,
	"parcel": 0,
	"stable": 0,
	"crop": 0,
	"taxonomic_species": 0,
	"non_taxonomic_entity": 0,
	"geom": "string",
	"point": "string",
	"date_observed": "2025-09-03T07:04:25.660Z"
	}

Successful response	
	200, Object (DiseaseRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 108: Delete an existing disease.

Field	Description
Title	Delete a Disease
ID	API-CORE-030
Description	Delete a Disease
HTTP Method	DELETE
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/diseases/{disease_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	204
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 109: Register a new disease stage.

Field	Description
Title	Create a Disease Stage
ID	API-CORE-032
Description	Create a Disease Stage
HTTP Method	POST
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/diseases/{disease_id}/stages/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	

Field	Description
Request payload	N/A
Successful response	
	201, Object (DiseaseStageRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 110: Register new measurements coming from crop yield quality device.

Field	Description
Title	Register the measurements of the handheld tool
ID	API-CORE-033
Description	Register the measurements of the handheld tool
HTTP Method	POST
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/handheld-tools/measurements/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	{ "DeviceName": "string", "Firmware": "string", "DeviceID": "string", "FriendlyName": "string", "Battery": 0, "Temperature": 0, "Reflectometer": 0, "DeviceRssi": 0, "Location": "string", "Latitude": 0,

Field	Description
	"Longitude": 0, "LastReadout": "string" }
Successful response	
	201, Object (HandheldToolCreate)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 111: Retrieve the list of measurements.

Field	Description
Title	List Measurements
ID	API-CORE-034
Description	List Measurements
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/measurements/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page. sensor: Search measurements by sensor ID measurement_type: Search measurements by measurement type ID complex: Search measurements by complex parcel: Search measurements by parcel stable: Search measurements by stable from_date: The date to fetch measurements from. Accepted formats: YYYY-mm-dd (e.g. 2024-05-17) YYYY-mm-ddTHH:MM:ss (e.g. 2024-05-17T15:30:32) to_date: The date to fetch measurements to. Accepted formats: YYYY-mm-dd (e.g. 2024-05-17) YYYY-mm-ddTHH:MM:ss (e.g. 2024-05-17T15:30:32)
Request payload	N/A
Successful response	

Field	Description
	200, Object list (MeasurementRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 112: Retrieve the list of measurements' groups.

Field	Description
Title	List Measurement Groups
ID	API-CORE-035
Description	List Measurement Groups
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/measurements/groups/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page. complex: Search measurements by complex parcel: Search measurements by parcel stable: Search measurements by stable from_date: The date to fetch measurements from. Accepted formats: YYYY-mm-dd (e.g. 2024-05-17) YYYY-mm-ddTHH:MM:ss (e.g. 2024-05-17T15:30:32) to_date: The date to fetch measurements to. Accepted formats: YYYY-mm-dd (e.g. 2024-05-17) YYYY-mm-ddTHH:MM:ss (e.g. 2024-05-17T15:30:32)
Request payload	N/A
Successful response	
	200, Object list (MeasurementGroupRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden

Field	Description
	500 Internal error
Notes	N/A

Table 113: Retrieve the list of a measurement group.

Field	Description
Title	Retrieve a Measurement Group
ID	API-CORE-036
Description	Retrieve a Measurement Group
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/measurements/groups/{group_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, ()
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 114: Delete a measurement group.

Field	Description
Title	Delete a Measurement Group
ID	API-CORE-036
Description	Delete a Measurement Group
HTTP Method	DELETE

Field	Description
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/measurements/groups/{group_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	204, ()
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 115: Retrieve the list of the latest measurements.

Field	Description
Title	Measurement View Set.
ID	API-CORE-039
Description	Measurement View Set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/measurements/latest/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page.
Request payload	N/A
Successful response	
	200, Paginated list of objects (MeasurementRetrieve)
Error response	

Field	Description
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 116: Retrieve details of a measurement.

Field	Description
Title	Retrieve a Measurement
ID	API-CORE-040
Description	Retrieve a Measurement
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/measurements/{measurement_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, ()
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 117: Update an existing measurement.

Field	Description
Title	Update a Measurement
ID	API-CORE-040
Description	Update a Measurement
HTTP Method	PUT
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/measurements/{measurement_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	

Request payload	{
	"id": 0,
	"parcel": 0,
	"crop": 0,
	"stable": 0,
	"sensor": 0,
	"group": 0,
	"measurement_type": 0,
	"value": "string",
	"unit": "string",
	"depth": 2147483647,
	"depth_range": "string",
	"date_observed": "2025-09-03T07:10:43.739Z",
	"origin": 0,
	"target": 0,
	"aggregation": 0,
	"aggregation_period": 0
	}

Successful response	
	200, Object (MeasurementRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated

Field	Description
	403 Forbidden
	500 Internal error
Notes	N/A

Table 118: Delete an existing measurement.

Field	Description
Title	Delete a Measurement
ID	API-CORE-040
Description	Delete a Measurement
HTTP Method	DELETE
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/measurements/{measurement_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	204
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 119: Retrieve the list of installed nodes (IoT devices).

Field	Description
Title	List Nodes
ID	API-CORE-041
Description	List Nodes

Field	Description
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/nodes/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (NodeRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 120: Retrieve details of an installed node (IoT device).

Field	Description
Title	Retrieve a Node
ID	API-CORE-042
Description	Retrieve a Node
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/nodes/{node_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, ()
Error response	

Field	Description
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 121: Update details of an installed node (IoT device).

Field	Description
Title	Update a Node
ID	API-CORE-042
Description	Update a Node
HTTP Method	PUT
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/nodes/{node_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object (NodeRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 122: Delete an installed node (IoT device).

Field	Description
Title	Delete a Node
ID	API-CORE-042
Description	Delete a Node
HTTP Method	DELETE
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/nodes/{node_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	204
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 123: Retrieve the list of installed actuators.

Field	Description
Title	List Actuators by Node
ID	API-CORE-043
Description	List Actuators by Node
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/nodes/{node_id}/actuators/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	

Field	Description
Request payload	N/A
Successful response	
	200, Object list (ActuatorRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 124: Register a new actuator.

Field	Description
Title	Create an Actuator
ID	API-CORE-043
Description	Create an Actuator
HTTP Method	POST
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/nodes/{node_id}/actuators/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	{ "id": 0, "identifier": "string", "node": 0, "parcel": 0, "stable": 0, "actuator_type": 0, "description": "string", "geom": "string", "point": "string", "manual": true,

Field	Description
	"updated": true, "latest_update": "2025-09-03T07:14:34.483Z", "value": 0 }
Successful response	
	201, Object (ActuatorRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 125: Retrieve the list of sensors in an installed IoT device.

Field	Description
Title	List Sensors by Node
ID	API-CORE-044
Description	List Sensors by Node
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/nodes/{node_id}/sensors/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	parcel: Search sensor by parcel node: Search sensor by node sensor_type: Search sensor by sensor type
Request payload	N/A
Successful response	
	200, Object list (SensorRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated

Field	Description
	403 Forbidden
	500 Internal error
Notes	N/A

Table 126: Register a new sensor in an installed IoT device.

Field	Description
Title	Create a Sensor
ID	API-CORE-044
Description	Create a Sensor
HTTP Method	POST
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/nodes/{node_id}/sensors/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	parcel: Search sensor by parcel node: Search sensor by node sensor_type: Search sensor by sensor type
Request payload	{ "id": 0, "identifier": "string", "node": 0, "parcel": 0, "stable": 0, "sensor_type": 0, "depth": 2147483647, "description": "string", "measurements": [{ "id": 0, "parcel": 0, "crop": 0, "stable": 0, "sensor": 0, "group": 0,

Field	Description
	"measurement_type": 0, "value": "string", "unit": "string", "depth": 2147483647, "depth_range": "string", "date_observed": "2025-09-03T07:14:10.703Z", "origin": 0, "target": 0, "aggregation": 0, "aggregation_period": 0 }] }
Successful response	
	201, Object (SensorRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 127: Retrieve the list of complexes by organization.

Field	Description
Title	List Activity Complexes by Organization
ID	API-CORE-045
Description	List Activity Complexes by Organization
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/organizations/{organization_id}/complexes/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}

Field	Description
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (ActivityComplexRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 128: Register a new complex.

Field	Description
Title	Create an Activity Complex
ID	API-CORE-045
Description	Create an Activity Complex
HTTP Method	POST
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/organizations/{organization_id}/complexes/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N { "id": 0, "pilot": 0, "name": "string", "description": "string", "location": "string", "geom": "string", "point": "string", "allow_organization_management": true, "organization_wide_publishing": true

Field	Description
	}
Successful response	
	201, Object (ActivityComplexRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 129: Retrieve the list of parcels.

Field	Description
Title	List Parcels
ID	API-CORE-046
Description	List Parcels
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/parcels/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	complex: Search parcel by complex category: Search parcel by category soil_texture_type: Search parcel by soil texture type irrigation_system_type: Search parcel by irrigation system type
Request payload	N/A
Successful response	
	200, Object list (ParcelRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error

Field	Description
Notes	N/A

Table 130: Retrieve details of a parcel.

Field	Description
Title	Retrieve a Parcel
ID	API-CORE-047
Description	Retrieve a Parcel
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/parcels/{parcel_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, ()
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 131: Update details of a parcel.

Field	Description
Title	Update a Parcel
ID	API-CORE-047
Description	Update a Parcel
HTTP Method	PUT
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/parcels/{parcel_id}/

Field	Description
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	{ "id": 0, "complex": 0, "parent_parcel": 0, "name": "string", "description": "string", "area": "string", "geom": "string", "point": "string", "location": "string", "category": 0, "soil_texture_type": 0, "irrigation_system_type": 0, "allow_organization_management": true, "organization_wide_publishing": true, "last_planted_at": "2025-10-03T07:19:33.761Z", "crops": [{ "id": 0, "parcel": 0, "crop_species": 0, "crop_variety": 0, "geom": "string", "point": "string", "valid_from": "2025-09-03T07:19:33.761Z", "valid_to": "2025-09-03T07:19:33.761Z", "last_planted_at": "2025-09-03T07:19:33.761Z", "production_amount": "string", "soil_texture_type": 0, "name": "string", "planting_row_distance": 0, "planting_distance_in_row": 0,

Field	Description
	"planting_density": 0, "year": 32767 }] }
Successful response	
	200, Object (ParcelRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 132: Delete a parcel.

Field	Description
Title	Delete a Parcel
ID	API-CORE-047
Description	Delete a Parcel
HTTP Method	DELETE
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/parcels/{parcel_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	204
Error response	
	400 Bad request
	401 Unauthenticated

Field	Description
	403 Forbidden
	500 Internal error
Notes	N/A

Table 133: Retrieve the list of crops in a parcel.

Field	Description
Title	List Crops by Parcel
ID	API-CORE-048
Description	List Crops by Parcel
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/parcels/{parcel_id}/crops/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (ParcelCropRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 134: Register a new crop in a parcel.

Field	Description
Title	Create a Parcel Crop
ID	API-CORE-048
Description	Create a Parcel Crop

Field	Description
HTTP Method	POST
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/parcels/{parcel_id}/crops/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	

Request payload	{
	"id": 0,
	"parcel": 0,
	"crop_species": 0,
	"crop_variety": 0,
	"geom": "string",
	"point": "string",
	"valid_from": "2025-09-03T07:21:01.496Z",
	"valid_to": "2025-09-03T07:21:01.496Z",
	"last_planted_at": "2025-09-03T07:21:01.496Z",
	"production_amount": "string",
	"soil_texture_type": 0,
	"name": "string",
	"planting_row_distance": 0,
	"planting_distance_in_row": 0,
	"planting_density": 0,
	"year": 32767
	}

Successful response	
	201, Object (ParcelCropRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 135: Retrieve the list of measurements of a parcel.

Field	Description
Title	List Measurements by Parcel
ID	API-CORE-049
Description	List Measurements by Parcel
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/parcels/{parcel_id}/measurements/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page. sensor: Search measurements by sensor ID measurement_type: Search measurements by measurement type ID complex: Search measurements by complex parcel: Search measurements by parcel stable: Search measurements by stable from_date: The date to fetch measurements from. Accepted formats: YYYY-mm-dd (e.g. 2024-05-17) YYYY-mm-ddTHH:MM:ss (e.g. 2024-05-17T15:30:32) to_date: The date to fetch measurements to. Accepted formats: YYYY-mm-dd (e.g. 2024-05-17) YYYY-mm-ddTHH:MM:ss (e.g. 2024-05-17T15:30:32)
Request payload	N/A
Successful response	
	200, Object list (MeasurementRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 136: Register a measurement in parcel level.

Field	Description
Title	Create a Measurement by Parcel
ID	API-CORE-049
Description	Create a Measurement by Parcel

Field	Description
HTTP Method	POST
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/parcels/{parcel_id}/measurements/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page. sensor: Search measurements by sensor ID measurement_type: Search measurements by measurement type ID complex: Search measurements by complex parcel: Search measurements by parcel stable: Search measurements by stable from_date: The date to fetch measurements from. Accepted formats: YYYY-mm-dd (e.g. 2024-05-17) YYYY-mm-ddTHH:MM:ss (e.g. 2024-05-17T15:30:32) to_date: The date to fetch measurements to. Accepted formats: YYYY-mm-dd (e.g. 2024-05-17) YYYY-mm-ddTHH:MM:ss (e.g. 2024-05-17T15:30:32)

Request payload	{
	"id": 0,
	"parcel": 0,
	"crop": 0,
	"stable": 0,
	"sensor": 0,
	"group": 0,
	"measurement_type": 0,
	"value": "string",
	"unit": "string",
	"depth": 2147483647,
	"depth_range": "string",
	"date_observed": "2025-10-03T07:22:30.196Z",
	"origin": 0,
	"target": 0,
	"aggregation": 0,
	"aggregation_period": 0
	}

Successful response	
	201, Object (MeasurementRetrieve)
Error response	
	400 Bad request

Field	Description
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 137: Retrieve the list of latest measurements of a parcel.

Field	Description
Title	List Measurements by Parcel
ID	API-CORE-051
Description	List Measurements by Parcel
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/parcels/{parcel_id}/measurements/latest/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page. sensor: Search measurements by sensor ID measurement_type: Search measurements by measurement type ID complex: Search measurements by complex parcel: Search measurements by parcel stable: Search measurements by stable from_date: The date to fetch measurements from. Accepted formats: YYYY-mm-dd (e.g. 2024-05-17) YYYY-mm-ddTHH:MM:ss (e.g. 2024-05-17T15:30:32) to_date: The date to fetch measurements to. Accepted formats: YYYY-mm-dd (e.g. 2024-05-17) YYYY-mm-ddTHH:MM:ss (e.g. 2024-05-17T15:30:32)
Request payload	N/A
Successful response	
	200, Object list (MeasurementRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 138: Retrieve the list of installed nodes (IoT devices) of a parcel.

Field	Description
Title	List Nodes by Parcel
ID	API-CORE-053
Description	List Nodes by Parcel
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/parcels/{parcel_id}/nodes/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (NodeRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 139: Register a new node (IoT device) in a parcel.

Field	Description
Title	Create a Node for a Parcel
ID	API-CORE-053
Description	Create a Node for a Parcel
HTTP Method	POST
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/parcels/{parcel_id}/nodes/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	

Field	Description

Request payload	{
	"id": 0,
	"complex": 0,
	"parcel": 0,
	"stable": 0,
	"name": "string",
	"description": "string",
	"sampling_period": 3600,
	"device": 0
	}

Successful response	
	201, Object (NodeRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 140: Retrieve the list of pest detection events in given time range.

Field	Description
Title	Fetch the pest detection events in specific date range
ID	API-CORE-054
Description	Fetch the pest detection events in specific date range
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/pest-detection-events/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	startDate: Filter by the starting date endDate: Filter by the ending date page: Filter by page number for pagination (starting from 1) limit: Number of results per page (maximum 50)

Field	Description
Request payload	N/A
Successful response	
	200, ()
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 141: Retrieve the list of the phenologies.

Field	Description
Title	List Phenologies
ID	API-CORE-057
Description	List Phenologies
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/phenology/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (ParcelCropPhenologyRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 142: Retrieve details of a phenology.

Field	Description
Title	Retrieve a Phenology
ID	API-CORE-058
Description	Retrieve a Phenology
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/phenology/{phenology_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 143: Update details of a phenology.

Field	Description
Title	Update a Phenology
ID	API-CORE-058
Description	Update a Phenology
HTTP Method	PUT
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/phenology/{phenology_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	

Field	Description
Request payload	N/A
Successful response	
	200, Object (ParcelCropPhenologyRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 144: Delete a phenology.

Field	Description
Title	Delete a Phenology
ID	API-CORE-058
Description	Delete a Phenology
HTTP Method	DELETE
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/phenology/{phenology_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	204, ()
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 145: Retrieve the list of sensors.

Field	Description
Title	List Sensors
ID	API-CORE-061
Description	List Sensors
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/sensors/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	parcel: Search sensor by parcel node: Search sensor by node sensor_type: Search sensor by sensor type
Request payload	N/A
Successful response	
	200, Object list (SensorRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 146: Retrieve details of a sensor.

Field	Description
Title	Retrieve a Sensor
ID	API-CORE-062
Description	Retrieve a Sensor
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/sensors/{sensor_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}

Field	Description
Query parameter(s)	
Request payload	N/A
Successful response	
	200, ()
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 147: Update details of a sensor.

Field	Description
Title	Update a Sensor
ID	API-CORE-062
Description	Update a Sensor
HTTP Method	PUT
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/sensors/{sensor_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	{ "id": 0, "identifier": "string", "node": 0, "parcel": 0, "stable": 0, "sensor_type": 0, "depth": 2147483647, "description": "string", "measurements": [

Field	Description

	{
	"id": 0,
	"parcel": 0,
	"crop": 0,
	"stable": 0,
	"sensor": 0,
	"group": 0,
	"measurement_type": 0,
	"value": "string",
	"unit": "string",
	"depth": 2147483647,
	"depth_range": "string",
	"date_observed": "2025-09-03T07:30:14.439Z",
	"origin": 0,
	"target": 0,
	"aggregation": 0,
	"aggregation_period": 0
	}
	]
	}

Successful response	
	200, Object (SensorRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 148: Delete a sensor.

Field	Description
Title	Delete a Sensor
ID	API-CORE-062

Field	Description
Description	Delete a Sensor
HTTP Method	DELETE
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/sensors/{sensor_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	204, ()
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 149: Retrieve the list of measurements of a sensor.

Field	Description
Title	List Measurements by Sensor
ID	API-CORE-063
Description	List Measurements by Sensor
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/sensors/{sensor_id}/measurements/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page. sensor: Search measurements by sensor ID measurement_type: Search measurements by measurement type ID complex: Search measurements by complex parcel: Search measurements by parcel stable: Search measurements by stable

Field	Description
	from_date: The date to fetch measurements from. Accepted formats: YYYY-mm-dd (e.g. 2024-05-17) YYYY-mm-ddTHH:MM:ss (e.g. 2024-05-17T15:30:32) to_date: The date to fetch measurements to. Accepted formats: YYYY-mm-dd (e.g. 2024-05-17) YYYY-mm-ddTHH:MM:ss (e.g. 2024-05-17T15:30:32)
Request payload	N/A
Successful response	
	200, Object list (MeasurementRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 150: Register a new measurement of a sensor.

Field	Description
Title	Create a Measurement by Sensor
ID	API-CORE-063
Description	Create a Measurement by Sensor
HTTP Method	POST
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/sensors/{sensor_id}/measurements/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page. sensor: Search measurements by sensor ID measurement_type: Search measurements by measurement type ID complex: Search measurements by complex parcel: Search measurements by parcel stable: Search measurements by stable from_date: The date to fetch measurements from. Accepted formats: YYYY-mm-dd (e.g. 2024-05-17) YYYY-mm-ddTHH:MM:ss (e.g. 2024-05-17T15:30:32) to_date: The date to fetch measurements to. Accepted formats: YYYY-mm-dd (e.g. 2024-05-17) YYYY-mm-ddTHH:MM:ss (e.g. 2024-05-17T15:30:32)
Request payload	N/A

Field	Description
Successful response	
	201, Object (MeasurementRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 151: Retrieve the latest measurements of a sensor.

Field	Description
Title	Measurement List / Create View.
ID	API-CORE-064
Description	Measurement List / Create View.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/sensors/{sensor_id}/measurements/latest/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page.
Request payload	N/A
Successful response	
	200, Paginated list of objects (MeasurementRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 152: Retrieve the list of stables.

Field	Description
Title	List Stables
ID	API-CORE-065
Description	List Stables
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/stables/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (StableRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 153: Retrieve details of a stable.

Field	Description
Title	Retrieve a Stable
ID	API-CORE-066
Description	Retrieve a Stable
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/stables/{stable_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	

Field	Description
Request payload	N/A
Successful response	
	200, ()
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 154: Update details of a stable.

Field	Description
Title	Update a Stable
ID	API-CORE-066
Description	Update a Stable
HTTP Method	PUT
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/stables/{stable_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	{ "id": 0, "complex": 0, "name": "string", "description": "string", "geom": "string", "point": "string", "location": "string", "allow_organization_management": true, "organization_wide_publishing": true, "groups": [

Field	Description

	{
	"id": 0,
	"stable": 0,
	"animal_category": 0,
	"name": "string",
	"animals": [
	{
	"id": 0,
	"group": 0,
	"identifier": "string",
	"name": "string",
	"birth_date": "2025-09-03",
	"death_date": "2025-09-03",
	"reproductive_condition": 0,
	"health_condition": 0,
	"sex": 0,
	"species": 0,
	"lactation_stage": 2147483647,
	"days_in_milk": 2147483647
	}
	]
	}
	]
	}

Successful response	
	200, Object (StableRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 155: Delete a stable.

Field	Description
Title	Delete a Stable
ID	API-CORE-066
Description	Delete a Stable
HTTP Method	DELETE
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/stables/{stable_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	204, ()
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 156: Retrieve the list of measurements of a stable.

Field	Description
Title	List Measurements by Stable
ID	API-CORE-068
Description	List Measurements by Stable
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/stables/{stable_id}/measurements/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}

Field	Description
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page. sensor: Search measurements by sensor ID measurement_type: Search measurements by measurement type ID complex: Search measurements by complex parcel: Search measurements by parcel stable: Search measurements by stable from_date: The date to fetch measurements from. Accepted formats: YYYY-mm-dd (e.g. 2024-05-17) YYYY-mm-ddTHH:MM:ss (e.g. 2024-05-17T15:30:32) to_date: The date to fetch measurements to. Accepted formats: YYYY-mm-dd (e.g. 2024-05-17) YYYY-mm-ddTHH:MM:ss (e.g. 2024-05-17T15:30:32)
Request payload	N/A
Successful response	
	200, Object list (MeasurementRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 157: Register a new measurement in a stable.

Field	Description
Title	Create a Measurement by Stable
ID	API-CORE-068
Description	Create a Measurement by Stable
HTTP Method	POST
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/stables/{stable_id}/measurements/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page. sensor: Search measurements by sensor ID measurement_type: Search measurements by measurement type ID complex: Search measurements by complex parcel: Search measurements by parcel stable: Search measurements by stable from_date: The date to fetch measurements from. Accepted formats: YYYY-mm-dd (e.g. 2024-05-17) YYYY-mm-

Field	Description
	ddTHH:MM:ss (e.g. 2024-05-17T15:30:32) to_date: The date to fetch measurements to. Accepted formats: YYYY-mm-dd (e.g. 2024-05-17) YYYY-mm-ddTHH:MM:ss (e.g. 2024-05-17T15:30:32)

Request payload	{
	"id": 0,
	"parcel": 0,
	"crop": 0,
	"stable": 0,
	"sensor": 0,
	"group": 0,
	"measurement_type": 0,
	"value": "string",
	"unit": "string",
	"depth": 2147483647,
	"depth_range": "string",
	"date_observed": "2025-09-03T07:37:18.131Z",
	"origin": 0,
	"target": 0,
	"aggregation": 0,
	"aggregation_period": 0
	}

Successful response	
	201, Object (MeasurementRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 158: Retrieve the list of latest measurements of a stable.

Field	Description
Title	Measurement by Stable List / Create View.
ID	API-CORE-070
Description	Measurement by Stable List / Create View.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/stables/{stable_id}/measurements/latest/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page.
Request payload	N/A
Successful response	
	200, Paginated list of objects (MeasurementRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 159: Retrieve the list of installed nodes (IoT devices) of a stable.

Field	Description
Title	List Nodes by Stable
ID	API-CORE-072
Description	List Nodes by Stable
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/stables/{stable_id}/nodes/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}

Field	Description
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (NodeRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 160: Register a new node (IoT device) in a stable.

Field	Description
Title	Create a Node
ID	API-CORE-072
Description	Create a Node
HTTP Method	POST
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/stables/{stable_id}/nodes/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	201, Object (NodeRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error

Field	Description
Notes	N/A

Table 161: Retrieve the list of traps' events (insect captures).

Field	Description
Title	TrapEvent View Set.
ID	API-CORE-073
Description	TrapEvent View Set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/trap-events/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: A page number within the paginated result set. page_size: Number of results to return per page.
Request payload	N/A
Successful response	
	200, Paginated list of objects (TrapEventRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 162: Retrieve details of a trap's event.

Field	Description
Title	TrapEvent View Set.
ID	API-CORE-074
Description	TrapEvent View Set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/trap-events/{trap_event_id}/

Field	Description
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object (TrapEventRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 163: Update details of a trap's event.

Field	Description
Title	TrapEvent View Set.
ID	API-CORE-074
Description	TrapEvent View Set.
HTTP Method	PUT
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/trap-events/{trap_event_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	{ "trap": 0, "captured_number": 0, "collection_timestamp": "2025-09-03T07:43:10.224Z", "measurements": [{ "taxonomic_species": 0,

Field	Description
	"disease": 0, "infected_number_total": 0, "infected_number_male": 0, "infected_number_female": 0 }] }
Successful response	
	200, Object (TrapEventCreateUpdate)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 164: Delete a trap's event.

Field	Description
Title	TrapEvent View Set.
ID	API-CORE-074
Description	TrapEvent View Set.
HTTP Method	DELETE
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/trap-events/{trap_event_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	204
Error response	

Field	Description
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 165: Retrieve the list of installed insects' traps.

Field	Description
Title	Get the list of traps
ID	API-CORE-075
Description	Get the list of traps
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/traps/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: Number of pages page_size: Results number per page. The default page size is 20000 while the max page size is 20000. id: Search based on trap ID code: Search based on trap's code serial_number: Search based on trap's serial_number model: Search based on trap's model ID privacy_level: Search based on trap's privacy level
Request payload	N/A
Successful response	
	200, Paginated list of objects (TrapRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 166: Register a new insect's traps.

Field	Description
Title	Register a new trap
ID	API-CORE-075
Description	Register a new trap
HTTP Method	POST
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/traps/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: Number of pages page_size: Results number per page. The default page size is 20000 while the max page size is 20000. id: Search based on trap ID code: Search based on trap's code serial_number: Search based on trap's serial_number model: Search based on trap's model ID privacy_level: Search based on trap's privacy level
Request payload	N/A
Successful response	
	201, Object (TrapCreateUpdate)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 167: Retrieve the overview list of the insects' traps.

Field	Description
Title	Get the list of traps overview
ID	API-CORE-076
Description	Get the list of traps overview
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/traps/overview/

Field	Description
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	page: Number of pages page_size: Results number per page. The default page size is 20000 while the max page size is 20000. id: Search based on trap ID code: Search based on trap's code serial_number: Search based on trap's serial_number model: Search based on trap's model ID privacy_level: Search based on trap's privacy level
Request payload	N/A
Successful response	
	200, Paginated list of objects (TrapOverviewRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 168: Retrieve details of an insect's trap.

Field	Description
Title	Handle the trap entity
ID	API-CORE-077
Description	Handle the trap entity
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/traps/{trap_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object (TrapRetrieve)

Field	Description
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 169: Retrieve the list of users.

Field	Description
Title	User View Set.
ID	API-CORE-078
Description	User View Set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/users/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (UserRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 170: Retrieve the list of roles.

Field	Description
Title	Role View Set.
ID	API-CORE-079
Description	Role View Set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/users/roles/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object list (RoleRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 171: Retrieve details of a user.

Field	Description
Title	User View Set.
ID	API-CORE-080
Description	User View Set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/users/{user_id}/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	

Field	Description
Request payload	N/A
Successful response	
	200, Object list (UserRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 172: Retrieve the privileges of a user.

Field	Description
Title	User Privilege View Set.
ID	API-CORE-081
Description	User Privilege View Set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/users/{user_id}/privilege/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object (UserPrivilegeRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

Table 173: Retrieve the role(s) of a user.

Field	Description
Title	User Role View Set.
ID	API-CORE-082
Description	User Role View Set.
HTTP Method	GET
Endpoint	https://api.platform.nestler-project.eu/core/api/v1/users/{user_id}/role/
Authentication	Yes
Header(s)	Accept: application/json Authorization: Bearer {token}
Query parameter(s)	
Request payload	N/A
Successful response	
	200, Object (UserRoleRetrieve)
Error response	
	400 Bad request
	401 Unauthenticated
	403 Forbidden
	500 Internal error
Notes	N/A

8.4 GIS API

This section presents indicative web services of the GIS API. For instance, *Table 174* describes the web service that lists the WMS stores while *Table 175* provides the list of WMS layers.

Table 174: GIS web service that provides the list of WMS stores.

Field	Description
Title	WMS stores
ID	GISREST-001
Description	Provides the list of WMS stores
HTTP Method(s)	GET
Endpoint	https://geoserver-api.platform.nestler-project.eu/geoserver/rest/workspaces/NESTLER/wmsstores/
Authentication	(Plan to use IAM)
Header(s)	Accept: application/json, Authorization: Bearer {token}
Query parameter(s)	N/A
Request payload	None
Successful response	
	200 OK
Error response	
	400 Bad request
	500 Internal error
Notes	N/A

Table 175: GIS web service that provides the list of WMS layers.

Field	Description
Title	WMS layers by given store
ID	GISREST-002
Description	Provides the list of WMS layers
HTTP Method(s)	GET

Endpoint	https://geoserver-api.platform.nestler-project.eu/geoserver/rest/workspaces/NESTLER/wmsstores/{wmsstore}/wmslayers/
Authentication	(Plan to use IAM)
Header(s)	Accept: application/json, Authorization: Bearer {token}
Query parameter(s)	N/A
Request payload	None
Successful response	
	200 OK
Error response	
	400 Bad request
	500 Internal error
Notes	N/A

Table 176: GIS web service that provides information of given WMS layer.

Field	Description
Title	WMS layer information
ID	GISREST-003
Description	Provides information of the requested WMS layer
HTTP Method(s)	GET
Endpoint	https://geoserver-api.platform.nestler-project.eu/geoserver/rest/workspaces/NESTLER/wmsstores/{wmsstore} /wmslayers/{wmslayer}.json
Authentication	(Plan to use IAM)
Header(s)	Accept: application/json, Authorization: Bearer {token}
Query parameter(s)	N/A
Request payload	None
Successful response	
	200 OK, JSON object

Error response	
	400 Bad request
	500 Internal error
Notes	N/A