



# Optimisation of nutrient budget in agriculture



## D5.2 System requirements



Funded by  
the European Union

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. The European Union cannot be held responsible for them.

# Delivery Report

Project Information	
Acronym	NutriBudget
Title	Optimisation of nutrient budget in agriculture
Project no.	101060455
Type of Action	RIA
Website	<a href="https://www.nutribudget.eu/">https://www.nutribudget.eu/</a>
Deliverable Information	
Title	System requirements
WP number and title	WP5 – Co-create the Nutriplatform
Lead Beneficiary	PwC
Authors	Susan Vrona Béjina (PwC), Florent Béjina (PwC), Vincent Coulette (PwC), Maher Megdamini (PwC), Gerard Ros (WU), Sven Verweij (NMI), Chantal Hendriks (WR)
Reviewers	Gerard Ros (WU), Sven Verweij (NMI)
Description	System specifications for the Nutriplatform prototype
Type	R – Document, report
Dissemination Level	PU - Public
Status	Final
Submission due date	30 <sup>th</sup> April 2025
History of Changes	
Version 0.1	Draft created by PwC 26/4/2025
Version 0.2	Integration of reviewers' feedback updated by PwC 30/4/2025
Version 1.0	Final version submitted 30/4/2025

## List of Figures

Figure 2-1. High-level components and dataflow.....	19
---	----

## List of Tables

Table 2-1. Identified components of the Target Architecture.....	12
Table 2-2. GitLab repository roles.....	14
Table 2-3. Estimated user volume Nutriplatform over time.....	15
Table 2-4. Node configuration description .....	17

## Table of Contents

List of Figures.....	3
List of Tables.....	3
Table of Contents.....	4
Terminology & Acronyms.....	5
Preface.....	9
Executive Summary.....	10
1. Introduction.....	11
2. Objectives and Approach.....	12
2.1. Assumptions.....	12
2.2. Components.....	12
2.3. Security considerations.....	13
2.4. Scalability, Performance, and Redundancy.....	13
2.5. Availability and Service Level Objectives (SLO).....	14
2.6. Minimal CI/CD in a Secure Environment.....	14
2.7. High-Level Deployment Architecture.....	15
2.8. System Requirements.....	15
2.8.1. Load.....	15
2.8.2. Node Architecture.....	16
2.9. Frontend Requirements.....	17
2.10. External APIs Requirements.....	18
2.10.1. General API Requirements.....	18
2.10.2. Endpoints Requirements.....	18
2.11. FaST integration.....	19
2.12. Agricultural reference data.....	20
2.13. Geographical data sources.....	20
3. Conclusions and next steps.....	21
4. Annexes.....	22
4.1 Annex 1 Mermaid mark-up for architecture.....	22

## Terminology & Acronyms

Item	Description
API	An Application Programming Interface is a set of commands, functions, protocols, and objects that programmers can use to create software or interact with an external system. It provides developers with standard commands for performing common operations so they do not have to write the code from scratch.
Backend	Manages the application's logic and interactions with the database.
Cloud Computing	The delivery of computing services over the internet, including storage, processing, and software, typically on a pay-as-you-go basis.
Container	A container or a container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings. Containers are not installed like traditional software programs, which allows them to be isolated from the other software and the operating system itself. Containerized software will always run the same regardless of the environment.
Container orchestration	In modern development, applications are no longer monolithic, but instead are composed of dozens or hundreds of loosely coupled, containerised components that need to work together to allow a given app to function as designed. Container orchestration refers to the process of organising the work of individual components and application layers. A container-orchestration system is a tool for automating deployment, scaling and management of containerized applications across clusters of hosts.
Cookie, Cookies	A piece of text generated by a web browser/internet navigator software that is stored as a file locally on the internet visitors' computer. Initially conceived to store stepwise information regarding page-to-page navigation by a site visitor to the site delivering the site content, its usage has been widely extended, all involving some way of tracking a unique computer-navigator combination visitor. Tracked data can include web pages viewed or the delivery of web site contents including images or advertising links. A number of other digital events such as time spent looking at a page, or clicks performed by internet users via their web navigation activities can also be tracked using cookies.
Database	A database is a data structure that stores organized information.
DevOps	A set of practices that combines software development (Dev) and IT operations to shorten the development lifecycle and deliver high-quality software continuously.
Docker	A platform for developing, shipping, and running applications inside containers.
Docker Swarm	A native clustering and orchestration tool for Docker containers.
Figma	A tool for creating mock-ups and wireframes of software applications.
Frontend	The visible part of the application that users interact with.
GeoJSON	A format for encoding a variety of geographic data structures using JavaScript Object Notation (JSON).
GitLab	A web-based DevOps lifecycle tool that provides a Git repository manager providing wiki, issue-tracking, and CI/CD pipeline features.
GitOps	Automates deployments and manages configurations via Git.
GraphQL	A query language for your API, and a server-side runtime for executing queries by using a type system you define for your data.

Item	Description
Hasura	A GraphQL engine that provides instant, real-time GraphQL APIs over PostgreSQL
Keycloak	An open-source Identity and Access Management (IAM) solution.
Load balancer	A load balancer distributes the processing and traffic evenly across a network, making sure no single device or service is overwhelmed. This is achieved by evenly splitting the traffic load among several different servers.
LPIS	The Land-Parcel Identification System is a system to identify land use for a given country. The LPIS is part of the IACS system.
MapProxy	An open-source proxy for geospatial data that caches, accelerates, and transforms data from existing map services.
Micro service	A micro service is a software development technique that structures an application as a collection of loosely coupled services. In a micro services architecture, services are fine-grained and the protocols are lightweight. The benefit of decomposing an application into different smaller services is that it improves modularity and makes the application easier to understand, develop, test, and more resilient to architecture erosion.
Mock-up	An image or set of images which displays the functional elements of a website or page, typically used for planning a site's structure and functionality and for soliciting feedback from stakeholders.
Module	Deployable, manageable, natively reusable, unit of software that provides a concise interface to consumers.
Node	In the context of server architecture, a Node refers to a distinct unit within a network that performs specific roles and functions.
Node.js	A server-side JavaScript runtime.
OAuth	A secure authentication protocol used to verify user identities and manage access permissions to sensitive information
Open-Source Software	Collaboratively created software code that permits free use, modification and distribution of code under a Open Software license.
OpenAPI	A specification for building APIs that allows both humans and computers to discover and understand the capabilities of a service without access to source code.
Parcel	Takes into account the definitions that vary from one Member State to another, e.g. Farmer's block, Agricultural parcel, Cadastral parcel.
PostgreSQL	An open-source relational database management system emphasizing extensibility and SQL compliance.
PostGIS	A spatial database extender for PostgreSQL, adding support for geographic objects.
Proxy	In computer networks, a proxy server is a server or software program that acts as an intermediary for requests from one resource seeking access to other resources usually over the internet and between servers. The term proxy can also be used in general sense in a computer software system for any component that abstracts access to another component.
OAuth	OAuth is a secure authentication protocol used to verify user identities and manage access permissions to sensitive information. It allows users to grant third-party applications limited access to their resources without sharing their credentials.

Item	Description
Raster tiles	A method for delivering geographic data as pre-rendered images, typically in a grid format. Raster tiles are used for displaying complex imagery like satellite photos and are styled server-side.
Stateless architecture	A stateless software architecture is a design where each request from a client to a server is treated as an independent transaction.
Server	Computer hardware that underpins digital IT systems
Service	A service encapsulates business logic and presents a simple interface that abstracts away the underlying complexity acting as a black box.
Service Level Objective	The term SLO stands for Service Level Objective. It is a key element in service level management and represents a specific measurable characteristic of the service, such as availability, throughput, or response time.
SSL/TLS	SSL stands for Secure Sockets Layer, and TLS stands for Transport Layer Security. Both are cryptographic protocols designed to provide secure communication over a computer network.
Software Updates	Software updates involve the installation of a new version of the software, incorporating either changes in software features or fixes to known software malfunctions.
UX	User Experience is the overall experience of a person using a product such as a website or computer application, especially in terms of how easy or pleasing it is to use.
Vector tiles	A method for delivering geographic data in small chunks, allowing for efficient rendering and interaction with maps. Vector tiles contain vector data (points, lines, polygons) and are styled client-side.
Version	A software version represents a certain state of the software, at a given point in time with an associated set of capabilities. Usually expressed in a number format such as X.Xxx with 'X' representing major features, and 'x' representing minor changes and fixes.
Virtual Machines	In computing, a virtual machine (VM) is an emulation of a computer system. Virtual machines are based on computer architectures and provide functionality of a physical computer.
Virtualization	Virtualization refers to running multiple operating systems on a single physical machine. While most computers only have one operating system installed, virtualization software allows a computer to run several operating systems at the same time.
Vue.js	A progressive JavaScript framework for building user interfaces.
Web application	A web-based application is any program that is accessed over a network connection using HTTP. Web-based applications often run inside a web browser. Web-based applications are also known as web apps.

Acronym	Elaboration
API	Application Programming Interface
CI/CD	Continuous Integration/Continuous Deployment
DST	Decision Support Tool
EU	European Union
FaST	Farm Sustainability Tool
GDPR	General Data Protection Regulation
GIS	Geographic Information System
IAM	Identity and Access Management
K	Potassium
LPIS	Land-parcel identification system
N	Nitrogen
P	Phosphorus
SLO	Service Level Objective
SPA	Single Page Application
SSL/TLS	Secure Sockets Layer / Transport Layer
UI	User Interface
UX	User eXperience
WP	Work Package
WMS	Web Map Service
WMTS	Web Map Tile Service

## Preface

The NutriBudget project aims to develop the prototype of a first-of-its-kind integrated nutrient management platform, called “NutriPlatform”, in various regions across Europe. The NutriPlatform will operate as a decision-support tool for farmers, advisors and regional authorities. Before the end of the project, the NutriPlatform will be tested and available to these users and interested citizens across Europe.

This report describes the outcome of the activities undertaken for the deliverable D5.2 “Collect Nutriplatform system requirements”, undertaken by PwC, in close collaboration with project members from different work packages (WP) whose valuable work is embedded in the tool capabilities and therefore influence the system requirements necessary to build and operate the NutriPlatform: (WP1) Ghent University (Belgium) for agricultural mitigation measures, (WP2) Stichting Wageningen Research (the Netherlands) for nutrient modelling across Europe and (WP3) Wageningen University (the Netherlands) for the expression of gaps between a current and optimal nutrient budget status as NutriKPIs.

Work on this deliverable started in project month 31 (March 2025) resulting in the delivery of this report at the end of project month 32 (April 2025).

We would like to acknowledge the researchers and staff of Wageningen University (the Netherlands), Stichting Wageningen Research (the Netherlands) for their contributions as reviewers of this document.

## Executive Summary

The aim of WP5, is to build a prototype of the Nutriplatform for use as a decision support tool (DST) to assist multiple users (e.g. farmers, farm advisors, local authorities, researchers, and citizens) across various spatial scales (from farm to EU level). The tool will provide users with insights on their current and desired status for nutrient budgets and allow them to discover the most suitable measures to bridge the gap between their current nutrient budget status and an optimal nutrient budget status. The Nutriplatform will enable “scenario evaluation” allowing users to set environmental goals and discover and then select a series of applicable measures affected by market developments / policy implementations, to improve their nutrient budget status and permit the monitoring of changes (in soil fertility, soil C and nutrient budgets) over time.

This report describes the outcome of the activities undertaken for the D5.2 “Collect Nutriplatform system requirements”, undertaken by PwC, starting in M31 and with the delivery of this report at the end of M32.

There is one task associated with this deliverable: Task 5.2 - Collect Nutriplatform system requirements involving the following consortium participants: PwC, UGent, WU and WR. T 5.2 aims to understand the technical constraints and requirements that the Nutriplatform must satisfy in terms of security, scalability, performance and availability for prototype development and deployment. As such, this task consists of analysis of technological requirements to support the interfacing of the Nutriplatform with operational models, algorithms, and business/regulatory rules derived from WP1, WP2 and WP3.

The system requirements also establish expected data volumes, and performance expectations (e.g. number of expected users), and requirements stemming from interconnections with services and models (content, formats, protocols, security arrangements) (aligned with the outputs of WP 1, 2 & 3), and external data sources (e.g. farm data, etc.), as well as the analysis of user stories and functional specifications carried out in T5.1. to facilitate the usage of the tool by “real” users (i.e. end-users).

## 1. Introduction

The aim of work package 5 (WP5), is to build a prototype of the Nutriplatform for use as a decision support tool (DST) to assist multiple users (e.g. farmers, farm advisors, local authorities, researchers, and citizens) across various spatial scales (from farm to EU level). The tool will provide users with insights on their current and desired status for nutrient budgets and allow them to discover the most suitable measures to bridge the gap between their current nutrient budget status and an optimal nutrient budget status. The Nutriplatform will enable “scenario evaluation” allowing users to set environmental goals and discover and then select a series of applicable measures affected by market developments / policy implementations, to improve their nutrient budget status and permit the monitoring of changes (in soil fertility, soil C and nutrient budgets) over time.

This report describes the outcome of the activities undertaken for the deliverable (D) 5.2 “System requirements”, undertaken by PwC, starting in M31, with the delivery of this report at the end of M32.

## 2. Objectives and Approach

There is one task associated with this deliverable: Task 5.2 – Collect Nutriplatform system requirements (PwC, UGent, WU, WR) [M31-M32]. T5.2 aims to understand the technical constraints and expectations that the Nutribudget Decision Support Tool (DST) called Nutriplatform, will need to meet in terms of security, scalability, performance and redundancy. These considerations are necessary for the development of the Nutriplatform prototype, the final deliverable for the WP5 to be delivered in M42 (February 2026) D5.4 Prototype DST, deployed and running on a cloud platform.

More specifically, the T5.2 consists of the analysis of technological requirements, notably analysing endpoints, inputs and outputs of operational models, algorithms, and business/regulatory rules derived from WP1, WP2 and WP3, but also the analysis of user stories and functional specifications carried out in T5.1.

T5.2 will lead to the identification of the list of interconnections with services and models (content, formats, protocols, security arrangements) (aligned with outputs of WP 1, 2 & 3), performance expectations, expected data volumes and required processing, expected external data sources (e.g. farm data, etc.) to facilitate the usage of the tool by “real” users (i.e. end-users) and the list of services to be built, as well as constraints on their deployment (e.g. redundancy, high availability, etc)

### 2.1. Assumptions

- Aside from GDPR compliance, no regulatory constraints apply to the Nutriplatform.
- Consider a simple architecture and standard technologies for a seamless handover at the end of the project.
- Consider a Docker infrastructure for multi-cloud compatibility considerations (OpenStack, VMWare, public cloud, etc.).
- Take into account the technologies mastered by the development team.

### 2.2. Components

Components are the technological building blocks of the Nutriplatform. The proposed components are based on open source software. Open source software (OSS)<sup>1</sup> is software that is shared under a license that permits free use, modification, and distribution of the human-readable source code. Open source software development permits flexibility and customisation of components to meet the specific needs of the Nutriplatform solution. It has the advantage of being free for non-commercial use, and benefit from stability, reliability and increased implementation speed afforded to open software components that have been developed and vetted collaboratively by of a large community of developers.

The listing of potential components appears in Table 2-1 below. This listing will be refined and finalised in the following deliverable D5.3 Architecture and wireframes (M36).

*Table 2-1. Identified components of the Target Architecture*

Component	Description	Technologies & Tools
<b>Backend</b>	Manages the application's logic and interactions with the database	Hasura for API, supplemented by Node.js modules for additional functionalities.
<b>Frontend</b>	The visible part of the application that users interact with	Vue SPA
<b>Database</b>	Stores the application's data	PostgreSQL/PostGIS

<sup>1</sup> <https://opensource.com/resources/what-open-source>

<b>Proxy</b>	Handles HTTP requests and redirects them to the appropriate services	Nginx, Traefik
<b>Monitoring</b>	Monitors the performance and health of the application	Prometheus for metrics collection, Grafana for data visualization
<b>IAM</b>	Manages user identities and access	Keycloak
<b>GitOps</b>	Automates deployments and manages configurations via Git.	Portainer for container management, SwarmCD for Docker Swarm, Composed for Docker Compose automation
<b>IaC</b>	Infrastructure as Code	Terraform, Ansible
<b>Testing</b>	Testing strategies	Selenium or Cypress for end-to-end testing. JMeter for performance and load testing. TestCafe for cross-browser testing. Postman for API testing.

### 2.3. Security considerations

In developing the open source based Nutriplatform, we are committed to enforcing robust security measures to safeguard data and ensure the reliability of our system. Our approach includes compliance with the General Data Protection Regulation (GDPR) and other relevant legal frameworks to protect user privacy and data integrity. We implement secure authentication protocols, such as OAuth, to verify user identities and manage access permissions to sensitive information strictly.

For data encryption, all data in transit will be encrypted using SSL/TLS protocols, and sensitive data at rest will be protected with advanced encryption techniques. Regular security audits and manual code reviews are conducted to identify and mitigate potential vulnerabilities, leveraging automated tools for static code analysis to enhance the security of the codebase.

We will monitor and update all software dependencies, ensuring that best practices in secure coding are followed, including thorough code reviews. Detailed logging mechanisms and real-time monitoring systems are established to detect and respond promptly to any suspicious activities.

Furthermore, as an open-source project, we encourage participation and feedback from the community to enhance security measures and maintain transparency in our processes. By integrating these security strategies, we aim to create a resilient and trustworthy platform that meets the high standards expected by the European Commission and our users.

### 2.4. Scalability, Performance, and Redundancy

Our platform is designed with a stateless architecture, fully containerized using Docker, and hosted on virtual machines in a cloud environment. A stateless software architecture is a design approach where each request within the platform is treated as an independent transaction. This model enhances scalability and simplifies infrastructure management, as there is no need to maintain information regarding the state of the application (for example during an operation to fulfil a platform request). It ensures that the platform is able to handle increased load in the form of requests (scalable) and can recover more easily in the case of failures.

This design ensures effortless scalability, as containers can be easily duplicated and distributed across available resources to efficiently handle variable loads. By grouping services with strong interdependencies on the same virtual machine, we optimize performance and reduce latency, enhancing the overall responsiveness of the application.

The choice of Docker over Kubernetes stems from our goal to simplify the handover process at the end of the project. This decision facilitates the transition for any team wishing to adopt the platform for further

development or hosting. Docker's straightforward architecture allows teams focused on operational management to use and develop the platform without requiring in-depth Kubernetes expertise.

Additionally, cloud-based deployment offers inherent redundancy and reliability. The virtual machines are configured to allow automatic failover and resource balancing, ensuring minimal disruption and high availability of services. Through this configuration, we aim to deliver an efficient, adaptable, and easily maintainable system that meets future scalability and performance demands while ensuring a smooth transition for subsequent teams.

## 2.5. Availability and Service Level Objectives (SLO)

By adopting Docker Swarm technology to orchestrate and deploy our containerized project, we aim to provide a platform with enhanced availability and resilience. Docker Swarm facilitates high availability through cluster orchestration, ensuring service continuity despite isolated failures. However, taking into account the necessary human interventions for certain operational processes, we acknowledge the possibility of short interruptions.

Our goal is to minimize service disruptions with potential downtimes limited to a few hours during critical scenarios, such as complete architecture rebuilds and database backup applications. We are committed to ensuring adequate response times through integrated load balancing, optimizing request distribution during normal operation periods. The system's continuity and resilience are further reinforced by backup and data restoration practices that aim to minimize downtime in case of major issues. Furthermore, we strive to provide regular updates and deployments with minimal service reductions, using progressive update processes.

By combining Docker Swarm orchestration with a stateless architecture and appropriate recovery strategies, we are committed to maintaining a high quality of service while being transparent about the limitations inherent to manual management of operational issues.

## 2.6. Minimal CI/CD in a Secure Environment

In our setup, software development is conducted on ultra-secure machines provided by PwC, and the code is subsequently pushed to a public GitLab server. Given that our development and staging environments are hosted in a UGent Research private cloud, seamless communication between systems is limited. With these constraints, the most feasible CI/CD process primarily involves a combination of manual builds and secure transfers. Developers can push their commits to GitLab, where automated build and test pipelines can run. However, subsequent steps require manual interventions, including the use of secure file transfers to move builds to the staging environments.

The public GitLab repository will be accessible using the standard roles as described in Table 2-2 below.

*Table 2-2. GitLab repository roles*

Role	Description
Guest	This role applies to private and internal projects only.
Planner	
Reporter	
Developer	
Maintainer	
Owner	
Minimal Access	Available for the top-level group only

Deployments in staging will need to be controlled using simple deployment scripts, without the potential for direct integration or orchestration between systems. This process, while basic, remains functional and upholds strict security requirements.

## 2.7. High-Level Deployment Architecture

The deployment architecture of the platform is designed to provide farmers, farm advisors, policy makers and researchers with effective tools for nutrient management. The system includes several key components: a front office (what the user interacts with), a back office (software components composed of services), a server (hardware, or hardware abstraction in virtual machines), and integrated APIs from other Work Packages. It also requires the capability to display maps, satellite images, and users' farm plots.

To meet these needs, the following technologies will be utilized: Hasura and Node.js for the backend, an Identity and Access Management (IAM) system with Keycloak, and a front-end built on Vue.js. OAuth authentication will be implemented with Keycloak as the provider. A tiling proxy server and a GIS tiles server will use a PostgreSQL database equipped with the PostGIS module. Additionally, the system will include monitoring tools with Prometheus and Grafana, and container orchestration via Docker Swarm, complemented by GitOps practices.

The infrastructure will consist of three environments: staging, pre-production, and production. Although there are no specific performance or scalability requirements as stated by the Nutribudget proposal, the architecture will leverage the chosen technologies to achieve a satisfactory Service Level Objective (SLO). Security is ensured by SSL encryption for all incoming connections from the Internet, as well as secure authentication with OAuth and Keycloak.

The project will integrate various external APIs, both internal APIs from other WPs and external services such as OpenStreetMap. The application will be deployed in select countries within the European Union<sup>2</sup>.

This architecture aims to provide a robust, secure, and scalable foundation to address the diverse needs of the project, considering the specifics of the deployment environments and regions.

## 2.8. System Requirements

### 2.8.1. Load

Expected load for the Nutriplatform over time is depicted in Table 2-3 below.

*Table 2-3. Estimated user volume Nutriplatform over time*

Time frame	Minimum	Maximum
<b>2025</b>	50	3 000
<b>2026 - 2029</b>	100	3 000
<b>2030 and beyond</b>	3 000	40 000

<sup>2</sup> The application will support at a minimum, the languages and user groups associated with the 5 study pilot regions (Belgium - Atlantic Region, Finland - Boreal Region, Switzerland - Continental Region, Italy - Continental Region, Spain - Mediterranean Region) and will be expanded to additional countries when possible during the development of the prototype.

## 2.8.2. Node Architecture

Our server architecture is divided into several nodes, each with specific roles and performance requirements tailored to its functions.

The following contains a description of each of the nodes designed to support the various components of the Nutriplatform to ensure efficient orchestration, smooth operations, and secure management.

### 1. **Node: Manager**

The primary role of this node is orchestration and cluster management. Each manager node is equipped with 2 vCPU, 4 GB RAM, and 40 GB SSD to host the Swarm control plane, along with a secret store and necessary configurations. Three manager nodes are deployed to ensure redundancy and high availability management.

### 2. **Node: Worker**

Worker nodes are dedicated to running application services. With a configuration of 8 vCPU, 16 GB RAM, and between 100 and 200 GB SSD, these nodes support components such as the Node.js backend, Hasura, and proxy services like MapProxy and pg\_tileserv. Three worker nodes are used to ensure scalability and processing capacity.

### 3. **Node: Database**

This node is responsible for persistent storage and spatial queries. Each node is configured with 8 vCPU, 16 GB RAM, and between 200 and 500 GB SSD, optimized for PostgreSQL with PostGIS. Two database nodes are deployed for increased redundancy and performance.

### 4. **Node: Keycloak**

Dedicated to identity and access management, this node utilizes Keycloak to provide authentication, SSO, and OIDC services. It is configured with 2 vCPU, 4 GB RAM, and 40 GB SSD, and is set up as a single instance.

### 5. **Node: Observability**

This node is tasked with metrics and log collection, integrating Prometheus, Grafana, and Loki. With a configuration of 2 vCPU, 8 GB RAM, and 100 GB SSD, a single observability node is deployed.

### 6. **Node: Backup/Ops**

Designed for the maintenance of backups and scheduled jobs, this node is equipped with 2 vCPU, 4 GB RAM, and 40 GB SSD. It manages database backups, cron tasks, and system housekeeping.

These systems are structured to provide a robust foundation that supports operational efficiency, security, and allows for scalability to meet the current and future needs of the platform. Node configuration appears in Table 2-4 below.

Table 2-4. Node configuration description

Node	Role	Performance requirements	# Nodes	Main Components
<b>Manager</b>	Orchestration cluster management	2 vCPU 4 GB RAM 40 GB SSD	3	Swarm control plane secret store configs
<b>Worker</b>	App services runtime	8 vCPU, 16 GB RAM, 100–200 GB SSD	3	Node.js backend Hasura pg_tileserv MapProxy Nginx + Traefik Vite.js
<b>Database</b>	Persistent storage, spatial queries	8 vCPU 16 GB RAM 200–500 GB SSD	2	PostgreSQL + PostGIS (app + reference) Keycloak DB
<b>Keycloak</b>	Identity & access management	2 vCPU 4 GB RAM 40 GB SSD	1	Auth, SSO, OIDC
<b>Observability</b>	Metrics & logs collection	2 vCPU 8 GB RAM 100 GB SSD	1	Prometheus Grafana Loki
<b>Backup / Ops</b>	Scheduled jobs maintenance	2 vCPU 4 GB RAM 40 GB SSD	1	DB backups cron tasks housekeeping

## 2.9. Frontend Requirements

The frontend application is designed as a Single Page Application (SPA) utilizing robust and mature open-source technologies to ensure efficiency, scalability, and a seamless user experience.

The application adheres to best practices for security, ensuring robust protection against vulnerabilities. OAuth is implemented for secure authentication, providing a reliable method for user verification.

The application employs responsive design principles to ensure optimal performance across various devices. It incorporates state-of-the-art user experience (UX) methodologies, with Figma screens designed by professionals, using the NutriBudget graphical charter, to guarantee a user-friendly and accessible interface. The application is compatible with desktops, tablets, and smartphones via the up-to-date standard browsers for a seamless user experience. The application will be fully internationalized to operate in the user's preferred working language.

The following technologies are considered:

- Language: Typescript
- Framework: Vue.js with Vite.js as a build and packaging tool
- UI Components: Quasar Framework
- Client-Side Routing and State Management: Vue router and Pinia
- Server-Side State Management: ApolloGraphQL or TanStack Query
- Maps: MapLibre
- Charts: Apache ECharts
- Internationalization: GNU gettext
- Testing: Vitest

## 2.10. External APIs Requirements

The Nutriplatform is designed to be flexible and adaptable to integrate with a wide variety of external APIs. Whether dealing with diverse authentication methods, endpoint structures, data formats, or rate limits, we are equipped to accommodate and efficiently respond to various integration requirements, ensuring seamless connectivity and interoperability. We will use Hasura as an API Gateway and the primary entry point for the backend. This enables us to deliver a unified GraphQL schema to the frontend.

To connect external APIs, we will develop Node.js services with a server such as Express or Fastify. These services will enhance the main GraphQL schema with additional REST actions or GraphQL subschemas, while the API Gateway manages the authentication and authorization layers. While this is technically feasible, at this time, we do not anticipate extending the API Gateway with externally deployed remote schemas.

For maps, we provide flexibility by using MapProxy to connect to external map tile providers. If necessary, we can also import, store, and serve vector tiles, such as agricultural parcel geometries, using PostGIS and pg\_tileserv. We may also be able to add external WMS layers to the maps (to be confirmed). By incorporating Keycloak, we are also ready to integrate other authentication services if needed in the future.

### 2.10.1. General API Requirements

- **Documentation:** APIs must be provided with comprehensive OpenAPI or Swagger documentation.
- **Environment:** A test/validation environment must be available for integration and testing purposes.
- **Availability:** APIs must ensure a high level of availability to minimize downtime and disruptions.
- **Scalability:** APIs must support usage by hundreds of simultaneous connections to handle high traffic efficiently.
- **Security:** APIs must use HTTPS to ensure secure communications.

### 2.10.2. Endpoints Requirements

This overview is based on the current information available and will be further refined and completed once we receive the specifications from the Nutrimodel web services through future planned workshops with WP2. The specific levels of application will be clarified later.

#### *General level*

- Mitigation measure catalogue
- Reference types (see 2.12 on Agricultural reference data)

#### *Farm and field level*

Applicable at farm and field level (to be clarified).

- Default values for inputs
- Nutrient budgets (actual and desired)
- KPIs
- Recommended mitigation measures
- Mitigation measure optimization

### Region level

Applicable at EU-27, country, NUTS-2 and NUTS-3 level.

- Default values for inputs
- Nutrient budgets (actual and desired)
- KPIs
- Recommended mitigation measures

A schematic representation of the Nutriplatform component architecture and data flows appears in Figure 2-1 below.

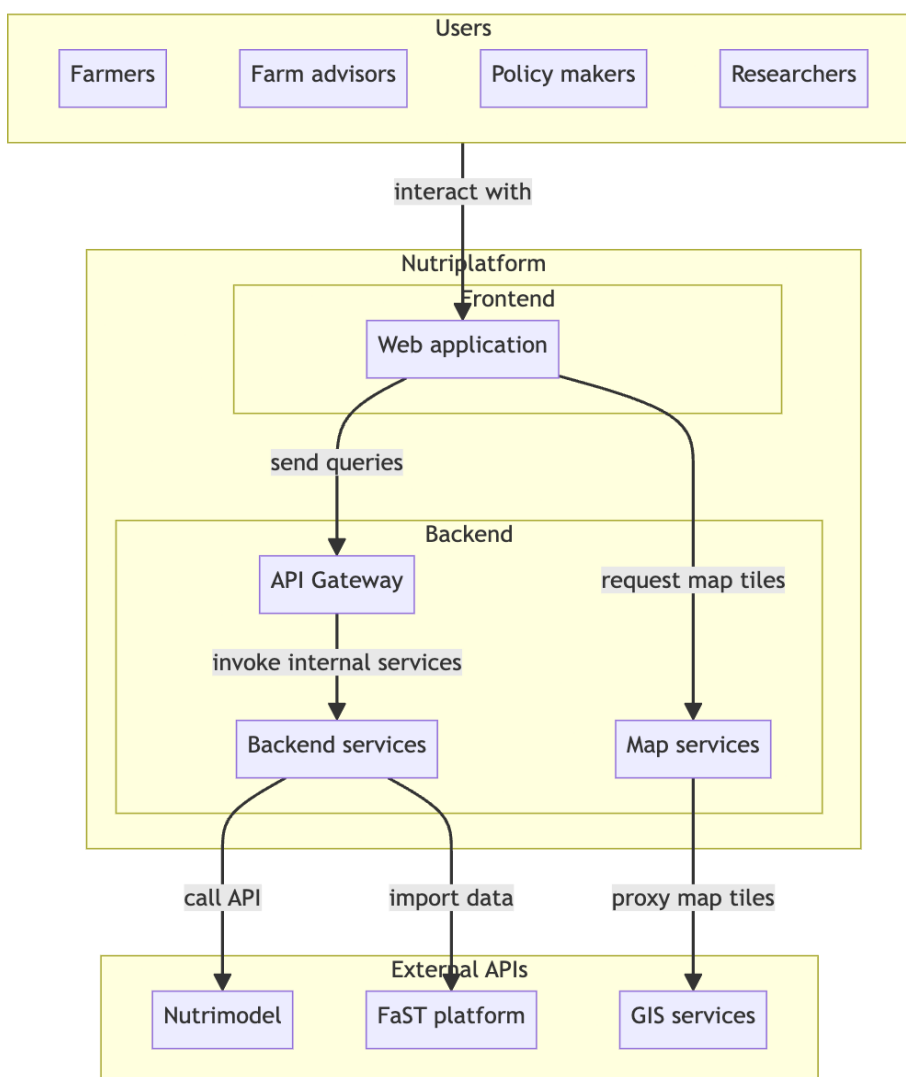


Figure 2-1. High-level components and dataflow

## 2.11. FaST integration

For farms from a region covered by the FaST platform, the application will include functionalities to import their data from the existing FaST platform. To do so, the following conditions must be met:

- The FaST platform is currently running in the farmers's region
- The authority managing the platform has agreed to share data with the NutriBudget DST

- The FaST platform administrators has performed one of these two actions for NutriBudget using the FaST administrative portal:
  - o Created an API key for the NutriBudget DST with the sufficient permissions to read the farmer's data
  - o Registered NutriBudget as an add-on in the platform allowing each individual farmer to allow the NutriBudget add-on to access their data using the FaST farmer application

## 2.12. Agricultural reference data

The application will require the following reference data to be provided either by the API itself or in structured formats such as JSON files (and kept up to date in case of changes):

- Crop types
- Animal types
- Fertilizer types (mineral and organic)
- Soil types

These types must include all relevant sub-data or metadata used in the API inputs such as categories, usages, applications, etc...

This data will be used in the multi-language UI so the reference data items must use standard IDs (or a reference language like English) so they can be referenced in a translation file.

It will be established if the reference data is likely to change over time and with what frequency so that an appropriate strategy will be put in place to manage this possibility.

## 2.13. Geographical data sources

The application will require external map tile layers to show farm locations, farm parcels and aggregated data on web maps. For general and farm-level maps, we need a base map layer such as OpenStreetMap and a Satellite map layer such as Google Maps satellite: we can use either raster or vector tile sources compatible with the MapLibre style specification. For region-level maps, we need geographical definitions for the EU-27, countries, NUTS-2 and NUTS-3 regions in a structured format like GeoJSON to display them on top of base layers. Farm parcels will be preferably imported from the FaST platform for the regions where FaST is deployed. If not available, farmers will be able to select or create their parcels using map editing tools. To display all available agricultural parcels for a given region we will leverage the publicly available data in two ways:

- Data files in geographical formats (such as shapefiles or gml) are processed then stored as geometries in the PostGIS database to be served as vector tiles using the pgtileserv server.
- Alternatively, parcels are displayed as a WMS/WMTS layer if a Web Map Service is available for the given region.

We might need a geocoding API to allow users to search for farms or geographic locations on maps. In this case, a freely usable API for non-commercial projects will suffice.

### 3. Conclusions and next steps

This document detailed the technical constraints and expectations for the Nutriplatform prototype to meet requirements in terms of security, scalability, performance and redundancy (availability) for the prototype.

Additional workshops are scheduled with WP2 and WP3, and a continued review and analysis of user stories and functional specifications carried out in T5.1.

The approach described in this document at this stage, is considered a “work in progress” as it will necessarily evolve through the development of a mock-up of wireframes for the next deliverable D5.3 Architecture blueprints and interface wireframes during M36.

## 4. Annexes

### 4.1 Annex 1 Mermaid mark-up for architecture

```

graph TD
  subgraph Users
    Farmer[Farmers]
    FarmAdvisor[Farm advisors]
    PolicyMaker[Policy makers]
    Researcher[Researchers]
  end

  subgraph Nutriplatform
    subgraph Frontend
      Application[Web application]
    end
    subgraph Backend
      API_Gateway[API Gateway]
      Backend_Services[Backend services]
      Map_Services[Map services]
    end
  end

  subgraph ExternalServices[External APIs]
    Nutrimodel_APIs[Nutrimodel]
    FaST_Platform[FaST platform]
    GIS_Services[GIS services]
  end

  Users --> |interact with| Application
  Application --> |send queries| API_Gateway
  Application --> |request map tiles| Map_Services
  API_Gateway --> |invoke internal services| Backend_Services
  Backend_Services --> |import data| FaST_Platform
  Backend_Services --> |call API| Nutrimodel_APIs
  Map_Services --> |proxy map tiles| GIS_Services
  
```

URL Mermaid : <https://www.mermaidflow.app/editor>



## Optimisation of nutrient budget in agriculture

### Project Coordinators:

Prof. Erik Meers, [Erik.Meers@UGent.be](mailto:Erik.Meers@UGent.be)

Dr. Ivona Sigurnjak, [Ivona.Sigurnjak@UGent.be](mailto:Ivona.Sigurnjak@UGent.be)

Ghent University, Sint Pietersnieuwstraat 25, Ghent 9000, Belgium.

### The Consortium:



Funded by  
the European Union