PlantMo: Plant Monitoring

Project Proposal & Development Plan

Cartago · Tonet · Toso · Zambonini

Document Status. The following document presents a preliminary and evolving version of the project proposal. The described deliverables, milestones, and architectural decisions may be subject to modification as the project advances. Accordingly, this document should be regarded as a working draft rather than a finalized specification.

Contents

1 Project Summary

This project addresses the challenge of reactive plant care by developing an automated smart monitoring system. The system's scope encompasses the collection, aggregation, and continuous analysis of environmental data from four sensor types: light, temperature, humidity, and soil moisture, to enable data-driven decision-making and real-time visualization. It explicitly excludes direct physical interventions such as automated watering or climate control.

The primary objectives are to design a reliable monitoring infrastructure that supports drift detection and the identification of anomalous environmental trends. System effectiveness will be measured through key performance indicators including Data Availability, Data Latency, System Reliability, Drift Detection Accuracy, and Dashboard Responsiveness. This initiative is relevant because it fundamentally transforms plant cultivation from a manual, snapshot-based approach into a proactive, insight-driven practice. By delivering continuous monitoring and early warning capabilities, the system empowers users to maintain optimal growing conditions and implement timely interventions before environmental issues compromise plant conditions.

2 Deliverables

The project delivers integrated hardware, data pipelines, visualization, and ML components that implement the specified architecture and satisfy all requirements.

#	**Deliverable**	**Description**
1	Fully Configured Data Acquisition Hardware	A physical sensing unit comprising an Arduino microcontroller interfaced with four environmental sensors (light, temperature, humidity, soil moisture). The unit will perform automated sampling and local preprocessing according to defined temporal windows.
2	Operational End-to-End Data Pipeline	A complete software pipeline implementing the specified data flow architecture. This encompasses modules for secure sensor-to-server data transmission, weighted temporal aggregation logic, and automated logging of processed time-series data to the Weights & Biases cloud platform.
3	Secure Data Access & Management System	Implemented authentication mechanisms and secure interfaces (FR09, FR10) incorporating API key validation to control client-side data retrieval and cloud upload operations.
4	Interactive Streamlit Dashboar	A web-based visualization platform (FR11) delivering real-time sensor monitoring and historical trend analysis capabilities (FR12). The dashboard will incorporate alert mechanisms for anomalous patterns and drift detection events (FR14).
5	Machine Learning Monitoring Module	Software components implementing the ML formulation, including statistical aggregation of multi-sensor data streams, drift detection algorithms for identifying distributional shifts and anomalous trends, and visualization tools for rendering data distribution changes (KPI alignment).
6	Comprehensive System Documentation	Technical documentation encompassing system architecture diagrams, deployment instructions, API specifications, and end-user dashboard guides. This includes operational logs and anomaly records to support monitoring and troubleshooting activities (FR15).
7	Deployed & Validated System Instance	A fully integrated, tested, and operational system deployment-spanning hardware, data pipeline, cloud storage, and visualization layers-demonstrating continuous operation, seamless data flow, and compliance with all specified functional and non-functional requirements.

Table 1: Project Deliverables Overview

2.1 Data Pipeline

The Data Pipeline implements the end-to-end data flow defined in the System Specification Document (Section 2.2).

2.2 ML Kernel

The Machine Learning Kernel utilizes the NHITS (Neural Hierarchical Interpolation for Time Series) model, a deep learning architecture designed for scalable and accurate time series forecasting.

NHITS employs a hierarchical, modular structure in which input time series are processed through stacked neural blocks. Each block models a distinct temporal resolution, enabling the capture of both short-term fluctuations and long-term trends within environmental sensor data. This hierarchical decomposition enhances forecasting accuracy while preserving interpretability and computational efficiency.

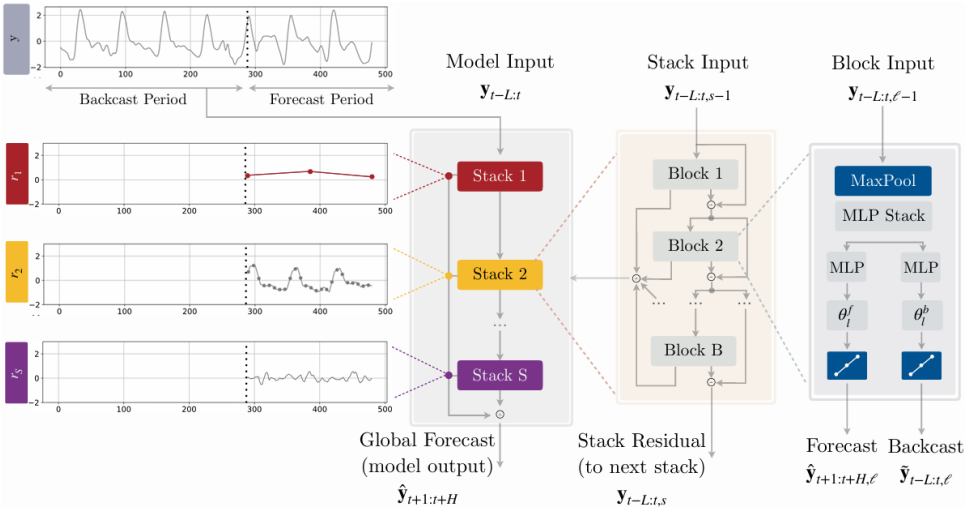

Figure 1: NHITS hierarchical architecture[1]

At each stage, the model decomposes the input signal into two components: a backcast, representing the portion explained by the current block and subtracted from the input before propagation to the next block, and a forecast, which contributes directly to the final prediction (as visualized in Figure 1). This residual learning approach allows NHITS to iteratively refine predictions by directing successive blocks toward increasingly complex or lower-frequency patterns.

A defining feature of NHITS is its interpolation-based forecasting mechanism. Rather than predicting every future time step directly, the model estimates a reduced set of latent forecast points, which are then interpolated to generate the full forecast horizon. This design reduces model complexity, improves stability, and makes NHITS particularly suitable for applications with limited or noisy data, such as real-world environmental sensor readings.

Within the proposed system, the NHITS-based ML Kernel processes multivariate time-series data derived from aggregated environmental sensors, including temperature, humidity, light intensity, and soil moisture. By leveraging historical observations, the

[1]Source: *Challu et al.* Available at: AAAI Conference Proceedings

model forecasts future environmental conditions to support monitoring, trend analysis, and early anomaly detection.

The kernel integrates seamlessly into the broader MLOps pipeline, enabling experiment tracking, model versioning, and performance evaluation via Weights and Biases. Its architectural properties—temporal consistency, scalability, and robustness—align with system requirements, establishing NHITS as a suitable core model for continuous plant environment monitoring.

2.3 CI/CD

The Continuous Integration and Continuous Deployment (CI/CD) strategy is conceived as a future extension of the system, aimed at increasing automation, reproducibility, and reliability of both software and embedded components.

At the current stage of the project, a fully automated CI/CD pipeline has not yet been implemented. This choice is intentional and reflects the exploratory and educational nature of the system, where design validation and functional correctness take priority over deployment automation. Nevertheless, the architecture has been designed to remain compatible with a future CI/CD integration.

The main idea of the proposed CI/CD workflow is to enable automatic updates of the entire software stack upon each commit to the main repository. This includes both embedded and server-side components, as summarized in Table 2.

Component / Task	**Deployment Actions and Configuration**
Arduino Firmware	Automatic update of the Arduino firmware responsible for sensor acquisition and local preprocessing. The pipeline retrieves the latest committed source code, performs basic consistency checks, and deploys the updated firmware to the edge device.
Server-side Scripts	Automatic update of server-side scripts handling data processing, cloud logging, and visualization. Updated Python modules are fetched from the repository and deployed to the local server environment to ensure consistency with the embedded firmware.

Table 2: CI/CD Target Components

In the envisioned setup, a CI/CD tool such as Jenkins is connected to the version-controlled repository and acts as a central automation orchestrator. Upon detecting a new commit, the pipeline executes a predefined sequence of actions, summarized in Table 3.

Pipeline Stage	Execution Description
Source Code Retrieval	The CI/CD service automatically pulls the latest version of the source code from the main branch of the repository after a commit event is detected.
Validation Checks	Basic validation steps are executed, including syntax checks, configuration consistency verification, and dependency resolution for both embedded and server-side code.
Automated Deployment	The validated codebase is deployed to the target components: Arduino firmware is updated on the edge device, while server scripts are deployed to the local processing and visualization environment, without requiring manual intervention.

Table 3: Proposed CI/CD Pipeline Stages

This approach would reduce the risk of configuration drift, ensure consistency between deployed components, and support rapid iteration during development. However, due to the limited scope of the current project and the absence of strict production constraints, manual deployment remains acceptable and more transparent at this stage.

Future work may focus on progressively enabling CI/CD automation once the system stabilizes and operational requirements become more stringent.

2.4 Monitoring

The following phase ensures the continuous, reliable, and transparent operation of the system after deployment. Its primary objective is to observe system behavior in real time, verify compliance with the defined Key Performance Indicators (KPIs), and promptly detect anomalies or deviations that may affect data quality, system stability, or machine learning reliability. Monitoring activities are strictly post-deployment and focus on operational supervision rather than development or training.

The monitoring process provides visibility across the system, spanning sensor data acquisition, data pipeline execution, machine learning behavior, and user-facing visualization. Sensor data are continuously logged to a cloud-based monitoring platform, enabling centralized storage, historical analysis, and traceability. The operational status of the system, including online and offline connectivity, is explicitly tracked to ensure transparency regarding data synchronization and availability.

A dedicated visualization dashboard supports real-time and historical monitoring of environmental variables such as light intensity, temperature, humidity, and soil moisture. Users can dynamically inspect recent measurements and longer-term trends, enabling timely identification of abnormal patterns or unexpected environmental changes. For selected sensors, current values are continuously compared against predefined threshold ranges; violations are interpreted as potential anomalies and are highlighted within the monitoring interface.

Beyond raw signal observation, monitoring includes the detection of statistical changes in data distributions. Data drift monitoring evaluates whether incoming sensor data deviate from the distributions observed during model training, providing early signals of

potential degradation in model validity. Detected drift events are logged and visualized to support informed decisions regarding model retraining or system recalibration, in accordance with the project's governance strategy.

To ensure stable operation during continuous execution, the monitoring application maintains a bounded local buffer containing the most recent aggregated samples. This mechanism preserves short-term contextual information while preventing uncontrolled memory growth. All monitoring events, including anomalies, drift detections, and system status changes, are systematically logged to support debugging, auditing, and long-term performance evaluation.

Monitoring Area	**Observed Aspects**
Data Acquisition	Availability, continuity, and validity of sensor measurements collected from the environmental sensing units.
Data Pipeline	Correctness of temporal aggregation, data latency, and reliability of cloud-based logging mechanisms.
Machine Learning	Detection of data drift and identification of anomalous changes in sensor data distributions over time.
Visualization	Responsiveness, correctness, and real-time update behavior of the monitoring dashboard.
System Operation	Overall operational stability, including online/offline status and continuous execution reliability.

Table 4: Monitoring areas and observed operational aspects

3 Milestones

To ensure effective monitoring of project progress, a defined set of Key Milestones has been established.

Each milestone represents a verifiable project outcome, signaling the formal completion of a major development phase or deliverable. This milestone-based framework provides clear, objective checkpoints for tracking schedule adherence, managing resources, and facilitating informed governance decisions throughout the project lifecycle.

M1	The project framework, including objectives, technical scope, development methodology, and collaboration workflow, is formally defined and approved.
M2	The hardware infrastructure, consisting of the Arduino platform and required sensors, is procured, assembled, and verified for data acquisition readiness.
M3	The software development environment and monitoring infrastructure, including the experiment tracking platform (W&B), are successfully deployed and validated.
M4	A synthetic data generation pipeline is implemented and validated to support early-stage testing and system prototyping.
M5	An integrated system prototype is implemented, combining data acquisition, machine learning components, and interactive monitoring dashboards.
M6	Data acquisition, storage, and visualization pipelines are fully operational, enabling continuous and reliable data flow from hardware to dashboards.
M7	Monitoring, data drift detection, and automated alerting mechanisms are implemented and validated under varying operational conditions.
M8	Project results are consolidated through final evaluation and analysis, and all required documentation is completed, marking formal project closure.

4 WBS: Work Breakdown Structure

To organize the project effectively, we established a Work Breakdown Structure (WBS). The WBS breaks down the project into clear, manageable phases, tasks, and subtasks, serving as a foundation for planning, scheduling, and assigning responsibilities throughout the development process.

To provide a clear visual guide, Figure 2 shows a high-level overview of the WBS. This diagram highlights the main project phases and illustrates how they are logically divided, offering an intuitive understanding of the project's structure and how different work packages relate to one another.

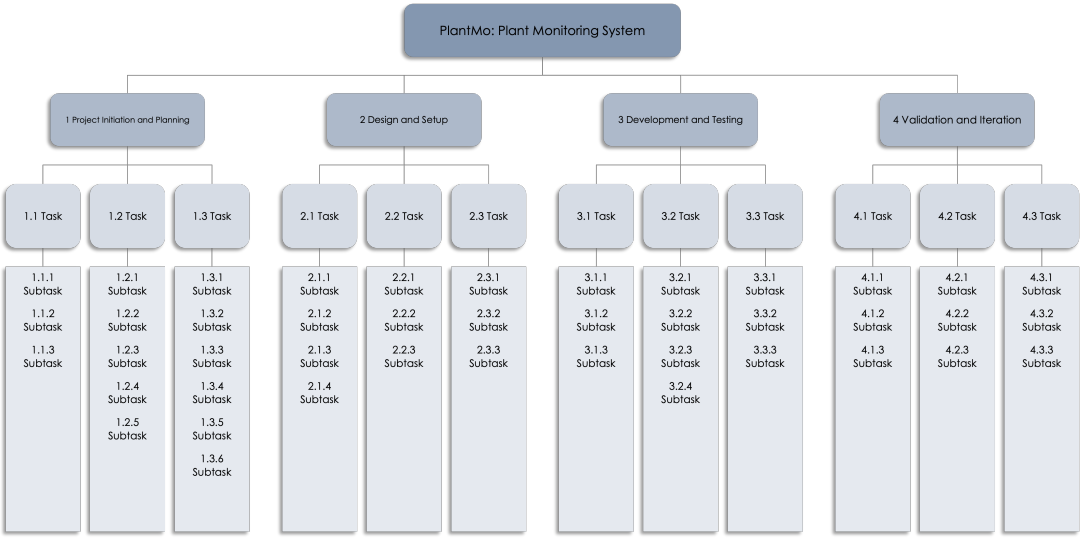

Figure 2: WBS Architecture

For a detailed view, Table 5 lists the full WBS in a structured table format. Each activity includes a task identifier and a brief description, helping to define the scope for every project phase. This table acts as a practical reference for ongoing planning, progress tracking, and project documentation.

Table 5: Work Breakdown Structure (WBS) of the Project

Task ID	Task Description
1.0 Project Initiation & Planning	
1.1	Preliminary research and ideation activities.
1.1.1	Project idea definition.
1.1.2	Brainstorming sessions.
1.1.3	Program and objective analysis.
1.2	Technical analysis and requirements identification.
1.2.1	Review of Arduino documentation.
1.2.2	Study of Arduino hardware configuration.
1.2.3	Sensor selection.

Continued on next page

Task ID	Task Description
1.2.4	Server selection.
1.2.5	Programming language selection.
1.3	Project organization and tool setup.
1.3.1	Role assignment.
1.3.2	Agile sprint planning.
1.3.3	Meeting schedule definition.
1.3.4	GitHub repository setup.
1.3.5	Telegram communication channel setup.
1.3.6	Jira project setup.
2.0 Design & Setup	
2.1	Documentation and system specifications.
2.1.1	System Specification Document preparation.
2.1.2	Functional requirements definition.
2.1.3	Non-functional requirements definition.
2.1.4	UML diagram creation.
2.2	Environment and package configuration.
2.2.1	Weights & Biases (W&B) platform setup.
2.2.2	Python package selection.
2.2.3	Development environment setup.
2.3	Hardware assembly and synthetic data preparation.
2.3.1	Sensor procurement.
2.3.2	Arduino board assembly.
2.3.3	Synthetic dataset creation.
3.0 Development & Testing	
3.1	Core system development.
3.1.1	Machine learning model implementation.
3.1.2	Streamlit dashboard development.
3.1.3	Preliminary data collection.
3.2	Testing and intermediate validation.
3.2.1	Preliminary data visualization.
3.2.2	Data testing through Streamlit interface.
3.2.3	Demo testing.
3.2.4	Drift detection testing.
3.3	Phase documentation and review.

Continued on next page

Task ID	Task Description
3.3.1	Project Proposal and Development Plan documentation.
3.3.2	Operational governance and versioning documentation.
3.3.3	Demo review meeting.
4.0 Validation & Iteration	
4.1	Deployment and monitoring activities.
4.1.1	Plant status monitoring via Telegram API with automatic alerts.
4.1.2	Real-environment testing.
4.1.3	Arduino optimization evaluation.
4.2	Expansion and scalability evaluation.
4.2.1	Integration of additional plants.
4.2.2	CI/CD trial of implementation.
4.2.3	Evaluation of additional sensors.
4.3	Final data collection and project closure.
4.3.1	Additional data gathering and modification of plant environmental conditions.
4.3.2	Final results review.
4.3.3	Project closure documentation.

5 Sprint Plan

The project follows an iterative model based on weekly, time-boxed sprints. This agile structure ensures incremental delivery, regular assessment, and adaptability to feedback. The plan below maps the Work Breakdown Structure to specific sprints, respecting a logical dependency flow where foundational work enables subsequent development and testing.

Sprint ID	**Date**	**Week**	**WBS Tasks**
Sprint 0	08/12/2025	Week 1	1.1
Sprint 1	15/12/2025	Week 2	1.2
Sprint 2	22/12/2025	Week 3	1.3
Sprint 3	29/12/2025	Week 4	2.1 , 2.2.3
Sprint 4	05/01/2026	Week 5	2.2.1, 2.2.2, 2.3.1-2.3.3
Sprint 5	12/01/2026	Week 6	3.1.1, 3.1.3
Sprint 6	19/01/2026	Week 7	3.2
Sprint 7	26/01/2026	Week 8	3.3
Sprint 8	02/02/2026	Week 9	4.1
Sprint 9	09/02/2026	Week 10	4.2
Sprint 10	16/02/2026	Week 11	4.3

The execution of this plan was governed by a structured coordination framework. Weekly sprints were anchored by two key ceremonies: a planning session on Mondays to set objectives and assign tasks, and a review with retrospective on Fridays to assess progress and identify improvements. Daily coordination occurred via a dedicated Telegram group for real-time discussion and documentation, while Jira provided formal task tracking, visibility, and accountability through the entire sprint lifecycle.

The sprint sequence implements a clear project phasing:

- **Sprints 0-2 (Foundation):** Project initiation and planning.
- **Sprints 3-4 (Design and Setup):** System specification and environment preparation.
- **Sprints 5-7 (Development and Review):** Core implementation, testing, and internal documentation.
- **Sprints 8-10 (Validation and Closure):** Real-world deployment, scalability assessment, and project conclusion.

This phased approach ensures core deliverables are stabilized before exploring extended functionality in the final validation stage.

6 Definition of Done and Definition of Ready

Clear and objective completion and readiness criteria are essential to ensure consistent development quality, predictable progress, and effective coordination across project activities. This project adopts formal **Definition of Ready (DoR)** and **Definition of Done (DoD)** frameworks to establish measurable conditions for initiating and closing development activities across all project phases.

6.1 Definition of Done (DoD)

The Definition of Done establishes the conditions under which any work item is considered fully completed. This provides shared understanding across all team members and serves as a quality gate for sprint reviews and milestone validation.

Criterion	Requirements
Implementation Complete	Development completed according to System Specification Document and approved architectural design.
Requirements Satisfied	All applicable functional requirements (FRs) and non-functional requirements (NFRs) fully met.
Testing and Validation	Comprehensive testing completed with no critical defects. Validation against acceptance criteria documented.
Data Flow Operation	All data flows, logging mechanisms, and monitoring features operate correctly under expected conditions.
KPI Compliance	System behavior aligns with defined Key Performance Indicators for availability, latency, reliability, and accuracy.
Documentation Updated	All relevant technical documentation completed or updated, including configuration and operational guides.
Formal Acceptance	Deliverable reviewed and accepted during sprint review or milestone evaluation process.

Table 6: Definition of Done (DoD) Criteria

The Definition of Done ensures completed work is not only functionally correct but also validated, documented, and ready for integration or deployment. This approach promotes transparency, accountability, and overall system quality throughout the project lifecycle.

6.2 Definition of Ready (DoR)

The Definition of Ready specifies the conditions that must be satisfied before any work item can enter the development phase. This ensures implementation begins only when sufficient information, resources, and dependencies are confirmed, thereby improving efficiency and reducing uncertainty.

Criterion	Requirements
Objective & Scope	Clear definition aligned with project goals and milestones. Acceptance boundaries explicitly stated.
WBS Alignment	Explicitly linked to Work Breakdown Structure element. Assigned to specific sprint with defined timeline.
Requirements Clarity	All functional and non-functional requirements identified. No unresolved ambiguities or contradictions.
Resources Available	Required hardware, datasets, and software tools (Arduino platform, sensors, cloud services) configured and accessible.
Dependencies Resolved	All preceding tasks, milestones, or external component dependencies resolved or formally accepted.
Acceptance Criteria	Measurable and verifiable criteria defined for objective evaluation of completion.
Technical Feasibility	Assessment completed with reasonable effort and duration estimates. No blocking technical constraints.

Table 7: Definition of Ready (DoR) Criteria

By enforcing these readiness conditions, the project ensures development activities begin in a controlled context, supporting predictable sprint execution and effective resource utilization.

7 Resources & Infrastructure

The successful implementation of the proposed system relies on a combination of hardware components, software tools, cloud services, and organizational resources. This section describes the resources and infrastructure required to support system development, deployment, monitoring, and validation across the project lifecycle. The selected resources are aligned with the defined functional and non-functional requirements and support scalability, reliability, and reproducibility.

7.1 Hardware Resources

Resource	Description
Arduino Microcontroller	Primary data acquisition and local processing unit responsible for interfacing with environmental sensors and performing preliminary aggregation operations.
Environmental Sensors	Sensors for monitoring light intensity, temperature, humidity, and soil moisture, providing continuous measurements of plant environmental conditions.
Local Server	Personal computer used for data aggregation, system integration and execution of monitoring services.
Personal Computer	Used exclusively for software development and source code implementation purposes.

Table 8: Hardware resources supporting data acquisition and system execution

7.2 Software and Development Tools

Tool	Purpose
Python	Primary programming language used for data processing, machine learning components, and dashboard development.
Arduino CLI	Command Line Interface for microcontroller programming, sensor integration, and firmware deployment.
Streamlit	Framework used to develop interactive dashboards enabling real-time monitoring and historical data exploration.
GitHub	Version control and collaboration platform supporting source code management, issue tracking, and team collaboration.

Table 9: Software tools and development resources

7.3 Cloud and Infrastructure Services

Service	Description
Weights & Biases (W&B)	Cloud-based platform used for logging time-series sensor data, experiment tracking, and monitoring system performance and behavior.
API Access Control	Secure interfaces using API keys to regulate data upload and retrieval operations, ensuring controlled access to cloud-stored resources.
Network Connectivity	Internet connectivity enabling continuous data transmission from the local system to cloud infrastructure and remote monitoring services.

Table 10: Cloud services and infrastructure components

7.4 Development and Execution Environment

Component	Description
Python Virtual Environments	Isolated environments used to manage dependencies and ensure consistent execution across different development machines.
Configuration Files	Environment configuration and dependency specification files enabling reproducible setup and deployment.
Cross-Platform Support	Design choices ensuring portability and correct execution across different operating systems without functional degradation.

Table 11: Development and execution environment resources

7.5 Plant and Growing Components

Element	Description
Syngonium podophyllum	Used as a houseplant due to its adaptability to indoor environments and low maintenance requirements.
Soil	Nutrient-rich growing medium that provides physical support to the plant roots, ensuring proper anchorage, aeration, and availability of essential minerals.
Water	Essential resource for plant survival.
Pot	Container to hold the soil and plant, allowing root development and facilitating drainage, stability, and placement in indoor or outdoor environments.

Table 12: Plant components and growing elements

7.6 Human and Organizational Resources

Resource	Description
Project Team	The project team consisted of four key roles, each contributing specific expertise to the successful development of the system. The team was composed of a Project Manager (Ernesto Zambonini), a Data Analyst (Emanuele Toso), and two Software Developers (Marco Cartago and Lorenzo Tonet), collaboratively responsible for system design, implementation, testing, and documentation activities.
Agile Methodology	Sprint-based development process including planning sessions, reviews, and retrospectives to ensure incremental delivery and adaptability.
Coordination Tools	Task tracking and communication platforms supporting transparency, coordination, and accountability throughout the project lifecycle.

Table 13: Human and organizational resources