Design Document

This Design Document serves as a comprehensive technical specification for the GrowHub MVP. It aims to provide a clear architectural overview and detailed specifications for all components of the system. This document will guide the development process and serve as a reference for all technical stakeholders involved in the project. Glossary of used terms, acronyms, and abbreviations is included at the end of the document.

Version: 001
Created: 11. 7. 2024 by Filip Vallo (Project Manager)
Approved: 14. 7. 2024

1. GENERAL PROJECT INFORMATION

Project name:
GrowHub

Project Manager: Filip Vallo
Email: filip.vallo@gmail.com

Expected start date: 8. 7. 2024
Expected completion date: 1. 9. 2024

2. PROJECT OVERVIEW

2.1 Description

GrowHub is an MVP IoT product designed to help indoor growers monitor and control their growing environments. The system will collect real-time data from various sensors and provide users with a web-based dashboard for monitoring and managing growing conditions. The product will include a set of fully programmable peripheral devices for irrigation scheduling, fertilizer injection, and power control. The dashboard will also allow to set alerts with notifications if any of the measured parameters exceeds set thresholds.

2.2 Purpose and Justification

The purpose of the GrowHub project is to develop an MVP IoT solution that provides hobbyists and professional growers with the tools to monitor and control their indoor growing environments. This solution aims to enhance plant growth by maintaining optimal conditions through automated and precise control of environmental factors.

3.3 Problem Statement

Indoor growers face challenges in maintaining optimal growing conditions for their plants. The lack of precise and automated control over environmental factors can lead to suboptimal plant growth and reduced yields.

3.4 Business Case

Developing the GrowHub MVP will demonstrate the feasibility of an IoT-based solution for indoor growing environments. This project has the potential to attract commercial interest, leading to further development and potential market entry. Successful implementation of the MVP will showcase the value of precise environmental control in improving plant health and yields.

2.5 Vision

Revolution of indoor growing through an innovative IoT solution that empowers both hobbyists and professionals to achieve optimal plant health and yields, by providing precise, automated environmental control and real-time monitoring.

2.6 Goals

- Manufacture a functional prototype of the monitoring IoT system using Raspberry Pi with integrated sensors to measure atmospheric, soil, and water conditions
- Manufacture connected programmable peripheral IoT devices for irrigation control, fertilizer injection, and power control
- Develop a fully operational website with dashboard for live and historical data visualization with notification system and environmental control system for managing growing conditions
- Manufacture embedded PCB that integrates all electronic circuits needed to process digital signals from all connected sensors and to control all custom IoT peripheral devices
- Test the MVP's commercial potential

2.7 Scope Statement

The GrowHub project encompasses the development, testing, and deployment of a fully functional MVP IoT system for indoor growing environment monitoring and management.

2.8 Functional Requirements

2.8.1 IoT device

- Real-time data collection from sensors connected to IoT compute unit: soil (moisture), atmospheric (temperature, humidity, CO2), and water (pH, conductivity)
- Fully programable peripheral IoT devices for irrigation control, fertilizer injection, and power control
- Embedded PCB with custom integrated electronic circuit that processes digital signals from all connected sensors and controls all peripheral devices

2.8.2 Web-based dashboard

- Real-time data collection from IoT device
- Historical data tracking and visualization
- Notification system for alerts when measured parameters exceed thresholds
- Environmental control system for managing growing conditions through peripheral devices

2.9 Non-functional Requirements

- High reliability and accuracy of sensor data
- Secure and responsive web dashboard
- User-friendly interface for both novice and experienced growers
- Scalability to handle multiple users and devices in the future

2.10 Outside of Scope

- Full commercialization and mass production (beyond MVP)
- User registration/authentication and user management system
- Advanced features not essential for the MVP
- Casing and product design

2.11 Risks

- Technical challenges in integrating sensors and developing the dashboard
- Limited budget and resources
- Delays in hardware procurement
- Delays in custom PCB manufacturing from the external vendor
- Quality issues with the manufactured PCB

2.12 Constraints

- Limited budget for hardware procurement and development
- Tight timeline to complete the MVP within the four planned phases
- Dependence on third-party services for cloud hosting and data storage
- Dependence on external vendor timelines for custom PCB manufacturing

2.13 Assumptions

- Availability of necessary hardware components
- Access to development tools and platforms for building the website
- The external vendor will meet the specified quality and timeline requirements for the custom PCB circuit
- Initial funding is sufficient to develop the MVP

12. ARCHITECTURE DECISION RECORD (ADR)

12.1 Hardware

12.1.1 IoT Compute Unit

Date:

Context:
The GrowHub MVP requires a central computing unit for the IoT device that will interface with sensors, control peripheral devices, and communicate with the cloud backend. This unit needs to be capable of running the sensor interfacing software, processing sensor data, and managing real-time communications.

Needs:

- Low cost to align with MVP budget constraints
- Strong community support and available resources for ease of development
- Sufficient processing power to handle sensor data collection and control logic
- GPIO pins to connect sensors and actuators
- Support for Inter-Integrated Circuit (I^2C) serial protocol for interfacing with sensors
- Low power consumption for energy efficiency

Assumptions:

- The development team has basic proficiency in working with single-board computers
- The chosen IoT compute unit has long-term support

Constraints:

- Limited budget for hardware procurement
- Need for rapid prototyping and development of the MVP

Decision:
We decided on ***Raspberry Pi 4 Model B (4GB RAM version)*** as IoT compute unit, knowing its limitations and the implications of this decision.

Decision	**Pros and Cons**
Raspberry Pi 4 Model B (4GB RAM version)	Price: • € 60 per unit Pros: • Powerful quad-core processor and sufficient RAM • Extensive GPIO pins and built-in Wi-Fi • Large community and extensive documentation • Versatile for prototyping and development • Built-in Wi-Fi and Bluetooth Cons: • Relatively higher power consumption • Overkill for simple IoT applications • More expensive than some alternatives

Argument:
The Raspberry Pi 4 Model B offers an excellent balance of performance, cost, and community support for our MVP needs. Its quad-core processor and 4GB of RAM provide sufficient computing power for our IoT application. The built-in Wi-Fi and numerous GPIO pins make it ideal for connecting various sensors and actuators. The vast Raspberry Pi ecosystem and community support align well with our need for rapid development and troubleshooting. While there are cheaper alternatives, the Raspberry Pi's versatility and extensive documentation justify the higher cost for our MVP stage.

Implications:

- We will need to develop a power management strategy to ensure stable operation
- We may need to implement cooling solutions if the device will be running continuously in enclosed spaces
- We may need to consider a more specialized or cost-effective solution for large-scale production in the future

- We will benefit from extensive community support and resources during development
- We will have access to a wide range of compatible sensors and modules, simplifying hardware integration
- The Raspberry Pi's capabilities exceed our immediate MVP needs, providing room for future feature expansion

Viable alternative:

Alternatives	**Pros and Cons**
Raspberry Pi 3 Model B+	Price: • € 40 per unit Pros: • Lower cost than Raspberry Pi 4 Cons: • May struggle with more demanding future applications • Less powerful than Raspberry Pi 4 • Higher power consumption than Arduino Uno Rev3
Arduino Uno Rev3	Price: • € 24 per unit Pros: • Low cost and power consumption • Simple to use for beginners • Good for simple sensor reading and controlling actuation Cons: • Limited processing power and memory • No built-in Wi-Fi (requires separate module) • Less suitable for complex data processing

12.1.2 Atmospheric Sensor Breakout Board

Date:

Context:

Needs:
- Ease of integration with IoT compute unit (Raspberry Pi 4 B)
- Good documentation
-

Assumptions:
- The development team has basic proficiency in XY
- The chosen solution has long-term support

Constraints:
- Limited budget for hardware procurement
- Need for rapid development of the MVP
- Must be compatible with chosen compute unit (Raspberry Pi 4 B)

Decision:
We decided on ***Pimoroni BME680*** breakout board with ***Bosch Sensortec BME680*** atmospheric sensor, knowing its limitations and the implications of this decision.

Decision	Pros and Cons
Pimoroni BME680 (Bosch Sensortec BME680 sensor)	Price: • € 22 Pros: • Cons: •

Argument:

Implications:

-
-

Viable alternatives:

Alternatives	**Pros and Cons**
	Price: • Pros: • Cons: •
	Price: • Pros: • Cons: •

12.1.3 Soil Moisture Sensor Breakout Board with Probe

Date:

Context:

Needs:
- Ease of integration with IoT compute unit (Raspberry Pi 4 B)
- Good documentation
-

Assumptions:
- The development team has basic proficiency in XY
- The chosen solution has long-term support

Constraints:
- Limited budget for hardware procurement
- Need for rapid development of the MVP
- Must be compatible with chosen compute unit (Raspberry Pi 4 B)

Decision:
We decided on ***XY*** soil sensors, knowing their limitations and the implications of this decision.

Decision	Pros and Cons
	Price: • Pros: • Cons: •

Argument:

Implications:

-

-

Viable alternatives:

Alternatives	**Pros and Cons**
	Price: • Pros: • Cons: •
	Price: • Pros: • Cons: •

12.1.4 Water pH Sensor Breakout Board with Probe

Date:

Context:

Needs:
- Ease of integration with IoT compute unit (Raspberry Pi 4 B)
- Good documentation
-

Assumptions:
- The development team has basic proficiency in XY
- The chosen solution has long-term support

Constraints:
- Limited budget for hardware procurement
- Need for rapid development of the MVP
- Must be compatible with chosen compute unit (Raspberry Pi 4 B)

Decision:
We decided on ***XY*** soil sensors, knowing their limitations and the implications of this decision.

Decision	Pros and Cons
Atlas Scientific pH Kit	Price: • Pros: • Cons: •

Argument:

Implications:

-
-

Viable alternatives:

Alternatives	**Pros and Cons**
	Price: • Pros: • Cons: •
	Price: • Pros: • Cons: •

12.1.5 Water Conductivity (EC) Sensor Breakout Board with Probe

Date:

Context:

Needs:
- Ease of integration with IoT compute unit (Raspberry Pi 4 B)
- Good documentation
-

Assumptions:
- The development team has basic proficiency in XY
- The chosen solution has long-term support

Constraints:
- Limited budget for hardware procurement
- Need for rapid development of the MVP
- Must be compatible with chosen compute unit (Raspberry Pi 4 B)

Decision:
We decided on ***XY*** soil sensors, knowing their limitations and the implications of this decision.

Decision	Pros and Cons
Atlas Scientific Conductivity K 0.1 Kit	Price: • $ 220 + shipping Pros: • Cons: •

Argument:

Implications:

-
-

Viable alternatives:

Alternatives	**Pros and Cons**
	Price: • Pros: • Cons: •
	Price: • Pros: • Cons: •

12.2 System Architecture

12.2.1 Software Architectural Style and Software Components

Date:

Context:
The GrowHub MVP requires an architectural style that can efficiently handle IoT device communication, data processing, storage, and user interface interactions. We need to choose an architecture that allows for scalability, maintainability, and rapid development of the MVP.

Needs:
- Low development and operational costs
- Ease of development, deployment, and maintenance
- Deployment of the code and data across multiple physical or virtual servers (Raspberry Pi, cloud-based web application)
- Sufficient performance to handle real-time data processing and user interactions
- Maintainability to ensure the system can be easily updated and managed

Assumptions:
- The development team is familiar with common architectural styles and their benefits

Constraints:
- Strict budget constraints for development and operational costs of the MVP
- Need for rapid development of the MVP
- Must integrate well with chosen deployment toolchain and development practices

Decision:
We decided on **N-tier Distributed Architecture** (architectural style), knowing its limitations and the implications of this decision. Software components (tiers):
- Presentation Tier (web application's frontend)
- Application Tier (web application's backend)
- Data Tier (databases)
- IoT Communication Tier (cloud service)
- Device Tier (IoT Device Software)

Decision	**Pros and Cons**
N-tier Distributed Architecture	Pros: • Clear separation of concerns • Flexible and independently scalable components • Allows optimization of each tier • Supports future growth and technology changes Cons: • Requires careful API design • Potential for increased operational overhead

Argument:
Given the requirement to deploy frontend and backend in the cloud while running Sensor Interfacing Software on Raspberry Pi, a N-Tier Distributed Architecture is the most suitable choice for the GrowHub MVP. This approach balances the need for separation of concerns with the priorities of rapid development and cost-efficiency. It allows for independent development and deployment of the IoT component, backend services, and frontend application, while maintaining a relatively simple overall structure. This architecture provides a good foundation for future scalability without the full complexity of a microservices approach.

Implications:
- We will need to design and implement efficient communication protocols between Raspberry Pi and cloud backend
- We will need to implement robust error handling and data synchronization between layers/modules
- We will need to manage separate frontend and backend environments
- We will need to implement security measures for communication between layers/modules

- Using Layered Architecture ensures a clear separation of concerns, enhancing modularity and reusability of components
- Each layer can be developed, tested, maintained, and scaled independently, simplifying the development process and future updates
- The architecture supports the distributed deployment setup efficiently, ensuring optimal performance and cost-effectiveness
- Initial complexity in system design and deployment is increased compared to a monolithic approach
- There is potential for future modularization of backend services if needed

Viable alternative:

Alternative	**Pros and Cons**
Microservices Architecture with IoT Gateway	Pros: • Highly scalable and flexible • Allows use of different technologies for different services • Enables complex applications with many features Cons: • Highest complexity in system design and deployment • Overkill for initial MVP requirements • Requires sophisticated DevOps practices

12.2.2 IoT Device OS

Date:

Context:
The Raspberry Pi microcomputer (IoT compute unit of GrowHub MVP) requires an operating system (OS) that will support the development and execution of the IoT device software, manage hardware resources, and facilitate communication with sensors, actuators, and the cloud backend.

Needs:
- Free and open-source
- Compatibility with Raspberry Pi 4 Model B
- Community support and extensive documentation
- Regular security updates and long-term support
- Easy integration with various sensors and actuators
- Robust package management system
- Good performance for real-time data processing (efficient resource management)

Assumptions:
- The development team has basic proficiency in Linux-based systems
- The chosen OS will be suitable for both development and deployment

Constraints:
- Limited budget for software procurement
- Need for rapid prototyping and development of the MVP

Decision:
We decided on ***Raspberry Pi OS*** for IoT device OS, knowing its limitations and the implications of this decision.

Decision	**Pros and Cons**
Raspberry Pi OS	Pros: • Official OS for Raspberry Pi with optimized performance • Large community and extensive documentation • Regular updates and long-term support • Pre-installed with Python and many useful tools Cons: • Larger footprint compared to minimal distributions • May include unnecessary components for IoT applications • Default configuration may need optimization for specific use cases

Argument:
Raspberry Pi OS is the official operating system for Raspberry Pi, offering optimized performance and compatibility. It provides a familiar Linux environment with a robust package management system, making it ideal for development and deployment. The OS comes with pre-installed Python support, aligning with our chosen programming language. Its large community, extensive documentation, and regular updates ensure long-term support and security. While there are lighter alternatives, Raspberry Pi OS's balance of features, performance, and ease of use makes it the most suitable choice for our MVP development.

Implications:

- We may need to disable the graphical interface of the device for production to save processing resources
- We should consider creating a custom, stripped-down image for deployment to minimize resource usage and improve boot time
- Using Raspberry Pi OS ensures maximum compatibility with Raspberry Pi hardware and available software libraries
- We will benefit from a stable and well-supported operating system
- We will have access to a wide range of development tools and packages through the built-in package manager

Viable alternatives:

Alternatives	**Pros and Cons**
Ubuntu Core	Pros: • Minimalist, container-based OS designed for IoT devices • Strong focus on security with automated updates • Supports snap packages for easy software management Cons: • Steeper learning curve compared to Raspberry Pi OS • Smaller community compared to Raspberry Pi OS • May require more setup for development environment
DietPi	Pros: • Extremely lightweight, optimized for performance • Minimal installation with option to add required software • Lower power consumption due to optimizations Cons: • Less user-friendly for beginners • Smaller community and less documentation compared to Ubuntu Core • May require more manual configuration for some features

12.2.3 Sensor Interfacing Software

12.3 Deployment Toolchain

12.3.1 Programming Language for Software Development on IoT Device

Date:

Context:
The GrowHub MVP requires a programming language for developing the software that will run on the IoT device (Raspberry Pi) to interface with sensors, perform data collection, communicate with the backend, and control peripheral devices.

Needs:
- Free and open-source
- Ease of development
- Ability to handle real-time data collection and processing
- Compatibility with Raspberry Pi hardware
- Efficient performance on Raspberry Pi hardware
- Good support for interfacing with hardware (GPIO, I2C, SPI)
- Ease of integration with various sensors and actuators
- Strong community and documentation

Assumptions:
- The development team has basic proficiency in IoT development and in chosen programming language for IoT development
- The chosen language has long-term support and is in active development

Constraints:
- Limited budget for software procurement
- Need for rapid development of the MVP
- Limited processing power and memory of Raspberry Pi

Decision:
We decided on ***Python*** for IoT device software development, knowing its limitations and the implications of this decision.

Decision	**Pros and Cons**
Python	Pros: • Excellent library support for Raspberry Pi (e.g., RPi.GPIO, pigpio) • Large community and extensive documentation • Easy to learn and read, rapid development and prototyping Cons: • May face performance issues with complex real-time operations • Higher memory usage compared to compiled languages • Global Interpreter Lock (GIL) can limit concurrency

Argument:
Despite its performance limitations, Python offers the best balance of ease of use, rapid development, and extensive library support for Raspberry Pi and sensor integration, which aligns well with our MVP goals. The cons of slower execution and GIL limitations are outweighed by the pros of faster development time and abundant resources, which are crucial for our MVP timeline. While C++ and Rust offer better performance, their steeper learning curves and longer development times make them less suitable for our current needs.

Implications:
- We will need to manage Python package dependencies (e.g., using virtual environments)
- We may need to optimize critical sections of code for performance
- We may need to reassess language choice in future iterations if performance becomes a critical issue
- We should consider using asynchronous programming techniques to mitigate GIL limitations

- Using Python will streamline the development process due to its simplicity and the availability of libraries
- Python ensures compatibility with existing Raspberry Pi setups and broad community support for troubleshooting and enhancements
- Performance of Python might be slightly lower compared to C/C++, but this is offset by the faster development and ease of use

Viable alternatives:

Alternatives	**Pros and Cons**
C++	Pros: • High performance and efficient resource utilization • Extensive control over system resources • Well-suited for real-time applications Cons: • Steeper learning curve, longer development time • More prone to memory-related errors (e.g., memory leaks) • Less forgiving for beginners, potentially leading to more bugs
Rust	Pros: • Memory safety without garbage collection • High performance and strong concurrency support • Growing ecosystem for embedded development Cons: • Steeper learning curve, longer development time • Smaller community compared to Python or C++ • Fewer libraries and resources specifically for Raspberry Pi development

12.3.2 Programming Language for Web Application Backend Development

Date:

Context:
In the context of developing the GrowHub MVP, we need to choose an appropriate programming language for the backend services that will handle data storage, data processing, and API interactions. The backend will support real-time data updates, historical data tracking, and provide an interface for the frontend application.

Needs:
- Free and open-source
- Ease of development and integration with IoT devices
- Efficient performance for real-time data processing and API handling
- Support for both RESTful APIs and WebSockets for real-time updates
- Robust library and framework support
- Good support for database integration
- Security and reliability of data handling and storage
- Strong community and documentation
- Scalability to handle multiple users and devices in the future

Assumptions:
- The development team has basic proficiency in backend development and in chosen programming language for backend services
- The chosen language has long-term support and is in active development

Constraints:
- Limited budget for software procurement
- Need for rapid development of the MVP
- Must integrate well with chosen IoT device technologies

Decision:
We decided on ***JavaScript*** with ***Node.js*** framework for web application's backend development, knowing its limitations and the implications of this decision.

Decision	**Pros and Cons**
JavaScript with Node.js framework	Pros: • Non-blocking I/O model suitable for real-time applications • Large ecosystem (npm) with numerous libraries • JavaScript on both frontend and backend (full-stack JS) • Excellent for handling concurrent connections Cons: • Single-threaded, may struggle with CPU-intensive tasks • Callback hell if not managed properly • Potential for introducing security vulnerabilities if not configured correctly

Argument:
JavaScript (Node.js) offers an excellent balance of performance, ease of use, and a vast ecosystem of libraries for backend development. Its non-blocking, event-driven architecture is well-suited for handling real-time data from IoT devices. While Python could also be a good choice, Node.js's performance in I/O-intensive operations makes it more suitable for our IoT backend needs.

Implications:

- We will need to implement proper error handling and asynchronous programming patterns
- We will need to ensure proper security measures are in place, as Node.js can be vulnerable if not configured correctly
- We should be cautious of callback hell and use modern JavaScript features (async/await) to manage asynchronous operations

- Using Node.js will allow for efficient handling of concurrent connections from multiple IoT devices
- Using Node.js ensures a consistent development environment for both frontend and backend, simplifying the development process
- Node.js's asynchronous nature will help in handling real-time data efficiently
- The extensive npm ecosystem will provide libraries to accelerate development
- Performance for CPU-intensive tasks might be lower compared to compiled languages, but this is offset by its efficiency in I/O operations

Viable alternatives:

Alternatives	**Pros and Cons**
Python	Pros: • Extensive libraries for data processing and analysis • Easy to learn and use, rapid development and prototyping • Large community and extensive documentation Cons: • May face performance issues with complex real-time operations • Higher memory usage compared to compiled languages • Global Interpreter Lock (GIL) can limit concurrency
Java with Spring Boot framework	Pros: • Robust and scalable, suitable for enterprise applications • Strong typing and OOP principles for maintainable code • Excellent performance and concurrency handling Cons: • Steeper learning curve, longer development time • Higher memory usage, especially for small-scale applications • More verbose code

12.3.3 Programming Languages for Web Application Frontend Development

Date:

Context:
The GrowHub MVP requires a programming language and framework for developing the frontend web application that will serve as the user interface for monitoring and controlling the IoT devices. We need to choose a language and framework that can efficiently handle real-time data visualization and user interactions while allowing for rapid development of the MVP.

Needs:
- Free or open-source
- Ease of development and integration with backend services
- Efficient performance for rendering and updating real-time data
- Strong ecosystem and library support for web development
- Good support for creating interactive and responsive user interfaces
- Scalability to accommodate future feature additions

Assumptions:
- The development team has basic proficiency in web development and in chosen programming language for web application development
- The chosen language has long-term support and is in active development

Constraints:
- Limited budget for software procurement
- Need for rapid development of the MVP
- Must integrate well with chosen backend technologies

Decision:
We decided on ***JavaScript (Vue.js)*** for web application's frontend development, knowing its limitations and the implications of this decision.

Decision	**Pros and Cons**
JavaScript with Vue.js framework	Pros: • Simple and intuitive API, gentle learning curve • Excellent performance and small bundle size • Flexible and incrementally adoptable • Growing ecosystem and community support Cons: • Smaller ecosystem compared to React or Angular • Potential for decision paralysis due to flexible nature

Argument:
JavaScript (Vue.js) offers an excellent balance of performance, ease of use, and a growing ecosystem. Vue.js's component-based architecture and reactive data model are well-suited for building interactive dashboards with real-time updates. Its gentle learning curve and excellent documentation will facilitate rapid development of the MVP. While React and Angular are also strong contenders, Vue.js's simplicity and performance make it more suitable for our MVP timeline and requirements.

Implications:

- We will need to implement proper state management for handling real-time data updates
- We will need to ensure proper optimization techniques are used for handling large datasets in real-time visualizations
- We may need to use additional libraries for complex data visualization (e.g., D3.js or Chart.js)
- We should leverage Vue.js's single-file component structure for maintainable code

- Using Vue.js will allow for efficient creation of reactive user interfaces
- Using Vue.js will ensure a modular and maintainable codebase, facilitating future updates and feature additions
- Vue.js ecosystem (Vuex for state management, Vue Router for routing) will provide necessary tools for building a robust application

Viable alternatives:

Alternatives	Pros and Cons
JavaScript with React framework	Pros: • Large ecosystem and community support • Virtual DOM for efficient updates Cons: • Steeper learning curve, especially with additional libraries (Redux, React Router) • Requires additional libraries for full functionality • JSX syntax may be unfamiliar to some developers
JavaScript with Angular framework	Pros: • Full-featured framework with everything included • Strong typing with TypeScript for robust code • Excellent for large-scale applications • Enforces consistent coding practices Cons: • Steeper learning curve and more complex setup • Heavier and potentially slower for small applications • Opinionated structure may be overkill for MVP • Slower development speed due to boilerplate

12.3.4 Integrated Development Environment (IDE) for Software Development on IoT Device (Python)

Date:

Context:
The GrowHub MVP requires an Integrated Development Environment (IDE) for Python development, primarily for the IoT device software. We need to choose an IDE that facilitates efficient coding, debugging, and integration with the Raspberry Pi.

Needs:
- Free or low cost
- Ease of development and debugging
- Robust support for Python development
- Efficient code editing, auto-completion, and refactoring tools
- Integrated debugging capabilities
- Support for version control systems
- Ability to work on Raspberry Pi or remote development

Assumptions:
- The development team has basic proficiency in using IDEs
- The chosen IDE has long-term support and is in active development

Constraints:
- Limited budget for software procurement
- Need for rapid development of the MVP
- Limited processing power and memory of Raspberry Pi
- Must work well with the Raspberry Pi ecosystem
- Must be compatible with chosen programming language for software development on IoT device (Python)

Decision:
We decided on ***PyCharm** (**Community Edition**)* for Python programming IDE, knowing its limitations and the implications of this decision.

Decision	**Pros and Cons**
PyCharm (Community Edition)	Price: • Free and open-source Pros: • Purpose-built for Python development • Robust out-of-the-box features for Python • Excellent code analysis and refactoring tools • Integrated virtual environment management Cons: • Heavier resource usage compared to VS Code • Steeper learning curve for beginners • Limited customization in Community Edition

Argument:
PyCharm Community Edition offers a comprehensive, purpose-built environment for Python development. Its robust out-of-the-box features, including advanced code analysis, refactoring tools, and integrated virtual environment management, make it an excellent choice for our IoT project. While it may have a steeper learning curve and higher resource usage compared to lighter alternatives, the productivity gains from its Python-specific features outweigh these drawbacks. PyCharm's strong support for scientific libraries and data science tools aligns well with our project's potential future needs in data analysis and visualization.

Implications:
- We may need to allocate more powerful development machines to handle PyCharm's higher resource requirements
- Using PyCharm will provide a comprehensive development environment tailored for Python, enhancing productivity and efficiency
- The decision to use PyCharm ensures robust support for Python development, including debugging, testing, and database management
- We will benefit from PyCharm's advanced code analysis and refactoring tools, potentially improving code quality and development speed
- The integrated virtual environment management will streamline our Python package handling

Viable alternatives:

Alternatives	**Pros and Cons**
Visual Studio Code (VS Code)	Price: • Free and open-source Pros: • Lightweight and fast performance • Extensive marketplace for extensions • Built-in terminal and debugger • Remote development capabilities Cons: • Requires some initial configuration for optimal development • May lack some advanced Python-specific features without extensions
Atom	Price: • Free and open-source Pros: • Highly customizable and extensible • Built-in package manager for easy extension installation • Active community with numerous packages Cons: • Can be slower to start up compared to lighter editors • Requires additional packages for full Python IDE functionality • May have performance issues with very large files

12.3.5 Integrated Development Environment (IDE) for Software Development of Web Application (JavaScript, HTML, CSS)

Date:

Context:
The GrowHub MVP requires an Integrated Development Environment (IDE) for developing the web application using JavaScript, specifically for Vue.js frontend and Node.js backend development. We need to choose an IDE that facilitates efficient coding, debugging, and integration with our chosen tech stack and cloud services.

Needs:
- Free or low cost
- Ease of development, debugging, and deployment
- Support for JavaScript (Node.js, Vue.sj) and related web technologies (HTML, CSS)
- Efficient code editing, auto-completion, and refactoring tools
- Integrated debugging capabilities
- Support for version control systems
- Ability to integrate with AWS services for deployment

Assumptions:
- The development team has basic proficiency in using IDEs
- The chosen IDE has long-term support and is in active development

Constraints:
- Limited budget for software procurement
- Need for rapid development of the MVP
- Must be compatible with chosen programming language
  for software development of web application (JavaScript, HTML, CSS)

Decision:
We decided on ***PyCharm*** (***Professional Edition***) for JavaScript, HTML, and SCC programming IDE, knowing its limitations and the implications of this decision.

Decision	Pros and Cons
PyCharm (Professional Edition) (includes WebStorm and DataGrip)	Price: • € 12 per month (first month free) Pros: • Comprehensive IDE for data science and web development • Robust out-of-the-box features for both Python and JavaScript • Advanced code analysis and refactoring tools • Advanced debugging and testing tools • Remote development capabilities • Integrates well with various frameworks and tools Cons: • Higher system requirements compared to VS Code • Paid software with recurring cost

Argument:
PyCharm Professional Edition offers a comprehensive, feature-rich environment for both Python and JavaScript development, which aligns well with our full-stack development needs. It provides robust support for Vue.js and Node.js, along with excellent debugging capabilities and intelligent code assistance. While it comes with a cost, its all-in-one solution for both frontend and backend development can potentially increase productivity and streamline our development process. The professional features, including advanced frameworks support and remote development capabilities, make it a powerful tool for our MVP development and beyond.

Implications:

- We will need to allocate budget for PyCharm Professional Edition licenses
- We should leverage PyCharm's advanced debugging and testing tools to improve code quality
- Using PyCharm (Professional Edition) will provide a robust and versatile development environment for JavaScript, enhancing our productivity
- We will benefit from integrated support for Python (IoT device), JavaScript (web application) and SQL (databases)
- PyCharm's out-of-the-box support for Vue.js and Node.js will reduce setup time for these frameworks
- The remote development features may be beneficial when working with AWS services

Viable alternatives:

Alternatives	**Pros and Cons**
WebStorm	Price: • € 8.30 per month (first month free) Pros: • Comprehensive IDE for web development • Robust out-of-the-box features for JavaScript • Advanced code analysis and refactoring tools Cons: • Higher system requirements compared to VS Code • Paid software with recurring cost
Visual Studio Code (VS Code)	Price: • Free and open-source Pros: • Lightweight and fast performance • Extensive marketplace for extensions • Strong community support and regular updates Cons: • Requires some initial configuration for optimal use • May lack some advanced features without extensions

12.3.6 Source Code Version Control Management

Date:

Context:
The GrowHub MVP requires a system for managing versions of the source code. This system needs to track changes, manage different versions of the codebase, and support parallel development workflows for potential future expansion of development team.

Needs:
- Free or low cost
- Ease of use and good integration with chosen IDE (PyCharm)
- Good documentation and community support
- Support for distributed version control with ability to manage branches and merging
- Support for code review processes and CI/CD pipelines

Assumptions:
- The development team has basic proficiency in version control systems
- The chosen solution has long-term support and is in active development
- The chosen system will be used for managing all project-related code

Constraints:
- Limited budget for software procurement
- Need for rapid development of the MVP
- Must be compatible with chosen programming languages (Python, JavaScript, HTML, CSS), IDEs (PyCharm) and cloud services

Decision:
We decided on ***GitHub*** (***Free Version***) for source code hosting with ***Git*** for version control management, knowing their limitations and the implications of this decision.

Decision	Pros and Cons
GitHub (Free Version) with Git	Free-Tier (only for public repositories): • Unlimited storage for codebase (files less than 50 MB) • Up to 3 collaborators Pros: • Industry standard, widely used and understood • Excellent integration with most development tools • Robust features for code review and collaboration • Large community and extensive documentation • Good integration with CI/CD tools Cons: • Potential security concerns with public repositories if not properly managed

Argument:
GitHub, coupled with Git, offers a powerful and flexible solution for version control that aligns well with our project needs. Git's distributed nature allows for efficient offline work and branching, which is crucial for our development workflow. GitHub provides an excellent platform for code hosting, issue tracking, and collaboration. Its integration with our chosen IDE (PyCharm) and potential CI/CD tools make it a natural fit for our development ecosystem. The free tier offered by GitHub is sufficient for our MVP stage, making it a cost-effective choice.

Implications:
- We will need to establish Git workflows and branching strategies
- We will need to ensure proper security practices for managing repository access
- We should implement code review processes using GitHub's pull request feature
- We should establish guidelines for commit messages and documentation to maintain code quality
- We should leverage GitHub Actions for potential CI/CD integration

- Using GitHub with Git will provide a robust and versatile platform for managing source code and version control, enhancing our productivity
- We will benefit from GitHub's issue tracking for project management
- We will benefit from extensive community support and a wide range of integrations, facilitating rapid development and collaboration

Viable alternatives:

Alternatives	**Pros and Cons**
Bitbucket (Free Version)	Free-Tier: • Unlimited storage for codebase • 1 GB free large file storage • Up to 5 collaborators Pros: • Good integration with Atlassian tools • Built-in CI/CD with Bitbucket Pipelines • Private repositories in free tier Cons: • Smaller community compared to GitHub • Less third-party tool integration
GitLab (Free Version)	Free-Tier: • Unlimited storage for codebase • Up to 5 collaborators Pros: • Integrated CI/CD pipelines • Built-in project management tools • Self-hosting option available Cons: • Less widely used than GitHub • Less generous free-tier offerings compared to GitHub

12.3.7 Artifact Management

Date:

Context:
The GrowHub MVP requires a system for managing artifacts (packages) produced during the software development lifecycle. These artifacts include compiled code, libraries, and other binary files that are not typically stored in version control systems like Git. Artifact management is essential for handling deployable components (build outputs). The chosen solution must facilitate the storage, versioning, and retrieval of these artifacts, ensuring that builds are reproducible and that the correct versions are deployed to different environments.

Needs:
- Free or low cost
- Ease of integration with chosen IDEs (PyCharm) and various cloud service providers
- Good documentation and community support
- Version control of build outputs and dependencies
- Support for different types of artifacts (e.g., Python packages, Node.js modules, Docker images)

Assumptions:
- The development team has basic proficiency in artifact management
- The chosen solution has long-term support and is in active development
- The project will primarily produce Python and JavaScript artifacts

Constraints:
- Limited budget for software procurement
- Need for rapid development of the MVP
- Must be compatible with chosen IDEs (PyCharm)

Decision:
We decided on ***GitHub** (**Free Version**)* with GitHub Packages feature for artifact management, knowing its limitations and the implications of this decision.

Decision	**Pros and Cons**
GitHub (Free Version)	Free-Tier (only for public repositories): • 500 MB for storing artifacts with GitHub Packages feature Pros: • No additional tool required • Supports various package types (npm, PyPI, Docker, etc.) • Built-in access control aligned with repository permissions Cons: • Storage and bandwidth limits on free plans • Less flexible than some dedicated artifact management solutions • Limited to GitHub ecosystem

Argument:
Using GitHub for artifact management offers a seamless integration with our existing GitHub workflow, which aligns well with our MVP development process. It supports the package types we need (Python and JavaScript), and its built-in access control simplifies management. While it may have some limitations compared to dedicated artifact management tools, its integration with our existing toolchain and the simplicity it offers make it an excellent choice for our MVP stage.

Implications:

- We will need to set up GitHub Packages for our repositories and configure our build processes to publish artifacts
- We will need to integrate GitHub Packages into our CI/CD pipeline for automated artifact publishing
- We may need to monitor our usage to ensure we stay within the free tier limits
- We should establish naming conventions and versioning strategies for our packages
- We will benefit from the integrated security features of GitHub, such as dependency scanning

Viable alternatives:

Alternative	**Pros and Cons**
PyCharm (Professional Edition) with Amazon S3	Free-Tier: • We already own subscription for PyCharm (Professional Edition) • 5 GB of S3 storage per month (first 12 months) • 20,000 GET, 2,000 PUT requests per month (first 12 months) Pros: • Our primary IDE • S3 provides reliable, scalable storage for larger artifacts • Good integration with other AWS services Cons: • Lacks some advanced features of dedicated artifact management tools • Requires manual setup for S3 integration

12.3.8 Continuous Integration

Date:

Context:
The GrowHub MVP requires a system for automatically building and testing code changes as they are pushed to the repository. This ensures that new changes integrate well with existing code and don't introduce bugs.

Needs:
- Free or low cost
- Ease of use and good integration with chosen IDE (PyCharm) and version control system (GitHub)
- Good documentation and community support
- Automatic build and test execution on code pushes
- Support for Python (IoT device software) and JavaScript (web application)
- Reporting of build and test results
- Good documentation and community support

Assumptions:
- The development team has basic proficiency in CI practices
- The chosen solution has long-term support and is in active development
- The CI system will be used for all system components (IoT device software, web application backend and frontend)

Constraints:
- Limited budget for software procurement
- Need for rapid development of the MVP
- Must be compatible with chosen IDEs (PyCharm) and version control system (GitHub)

Decision:
We decided on ***GitHub*** *(**Free Version**)* with GitHub Actions feature for continuous integration (CI), knowing its limitations and the implications of this decision.

Decision	**Pros and Cons**
GitHub (Free Version)	Free-Tier (only for public repositories): • Unlimited CI/CD build minutes with GitHub Actions feature • Up to 3 collaborators Pros: • No additional tool required • Configurable workflows for different types of tests and builds • Built-in reporting of results Cons: • Learning curve for configuring workflows • Limited to GitHub ecosystem

Argument:
GitHub Actions feature provides a robust and integrated solution for continuous integration (CI) that aligns perfectly with our existing version control choice. It offers the ability to automate our build and testing process without the need for additional external services. The tight integration with our GitHub repositories means we can set up workflows that automatically run our tests and builds on every push or pull request. This solution is cost-effective (free for our current needs) and eliminates the need to manage a separate CI tool, simplifying our overall toolchain.

Implications:

- We will need to create and maintain GitHub Actions workflow files (.github/workflows) in our repositories
- We will need to write comprehensive tests for both our IoT device software (Python) and web application (JavaScript)
- We may need to mock certain AWS services for testing purposes
- We should set up different workflows for unit tests, integration tests, and builds
- We should design our test suites to run efficiently within the free tier limits
- We should integrate build and test result reporting into our development workflow
- Using GitHub Actions will provide a robust and integrated solution for continuous integration, enhancing the efficiency and reliability of the development processes
- We will benefit from immediate feedback on the impact of our code changes

Viable alternatives:

Alternatives	**Pros and Cons**
CircleCI	Free-Tier (only for public repositories): • 6,000 build minutes per month Pros: • Fast build times • Supports Docker out of the box Cons: • More complex configuration for advanced setups • Separate tool from version control system • More limited free tier compared to GitHub Actions
Travis CI	Free-Tier (only for public repositories): • 1,000 build minutes per month Pros: • Well-established CI tool with good community support • Easy to set up and configure Cons: • Separate tool from version control system • More limited free tier compared to GitHub Actions

12.3.9 Configuration Management of IoT compute unit

Date:

Context:
The GrowHub MVP requires a configuration management solution for the IoT compute unit (Raspberry Pi 4 B) to ensure consistent and reproducible setups across development and production environments. This tool will manage software installations and system configurations.

Needs:
- Free and open-source
- Ease of use and good integration with chosen IoT compute unit (Raspberry Pi 4 B)
- Good documentation and community support
- Agentless operation to minimize resource usage on the Raspberry Pi
- Support for version control of configurations
- Ability to manage multiple Raspberry Pi devices (for future scalability)

Assumptions:
- The development team has basic proficiency in configuration management
- The chosen solution has long-term support and is in active development
- The solution will be used primarily for the Raspberry Pi, but may extend to other parts of the infrastructure in the future

Constraints:
- Limited budget for software procurement
- Need for rapid development of the MVP
- Must be compatible with chosen IoT compute unit (Raspberry Pi 4 B) and version control system (GitHub)

Decision:
We decided on ***Ansible*** for configuration management of IoT compute unit (Raspberry Pi 4 B), knowing its limitations and the implications of this decision.

Decision	Pros and Cons
Ansible	Pros: • Agentless, minimizing resource usage on Raspberry Pi • Python-based, aligning with existing project technology • Easy to learn with human-readable YAML syntax • Extensive module library for various tasks • Good integration with version control systems (Git) • Strong community support and documentation Cons: • May require additional setup for Windows development environments

Argument:
Ansible provides a lightweight, agentless configuration management solution that aligns well with our project's needs and constraints. Its Python-based architecture fits seamlessly with our existing technology stack, and its simplicity allows for rapid adoption and implementation. The YAML-based playbooks are easy to read and maintain, which is crucial for our MVP development timeline. While there might be a slight learning curve, the extensive documentation and community support mitigate this concern. Ansible's ability to scale from managing a single Raspberry Pi to multiple devices also supports our potential future needs.

Implications:
- We will need to set up an Ansible control node, likely on the development machines
- We will need to integrate Ansible playbooks into our version control workflow
- We may need to set up a staging environment to test Ansible playbooks before applying to production
- We should create a library of reusable Ansible roles for common configurations

- Using Ansible will provide a consistent and reproducible method for configuring our Raspberry Pi devices
- We will benefit from improved documentation of our system configuration through Ansible playbooks

Viable alternatives:

Alternatives	Pros and Cons
Puppet (Open Source Version)	Pros: • Mature tool with extensive ecosystem • Good for managing large infrastructures • Strong reporting capabilities Cons: • Steeper learning curve compared to Ansible • Requires an agent on managed nodes, which could impact Raspberry Pi performance • Uses its own domain-specific language, which may take time to learn
Chef Infra	Pros: • Powerful and flexible configuration management • Good integration with cloud services • Strong support for compliance and security automation Cons: • Complex for small-scale projects like our MVP • Ruby-based, which doesn't align with our Python-centric stack • Requires more setup and maintenance compared to Ansible

12.3.10 Testing Framework for Sensor Interfacing Software on IoT Device (Python)

Date:

Context:
Testing the sensor interfacing software on the IoT device is critical to ensure reliable data collection and communication. Selecting a suitable testing framework for Python will help automate the testing process, ensuring that the software interacts correctly with the sensors and handles data appropriately.

Needs:
- Free and open-source
- Ease of use and good integration with chosen IDE (PyCharm)
- Good documentation and community support
- Support for unit testing and integration testing
- Ability to mock hardware interactions for isolated testing
- Good performance on Raspberry Pi hardware
- Clear and informative test result reporting

Assumptions:
- The development team has basic proficiency in Python testing
- The chosen framework has long-term support and is in active development

Constraints:
- Limited budget for software procurement
- Need for rapid development of the MVP
- Limited processing power and memory of Raspberry Pi
- Must be compatible with chosen IDEs (PyCharm)
  and continuous integration system (GitHub)

Decision:
We decided on ***pytest*** for Python testing framework, knowing its limitations and the implications of this decision.

Decision	Pros and Cons
pytest	Pros: • Simple and easy to use • Powerful feature set including parameterized testing and fixtures • Large ecosystem of plugins • Good integration with PyCharm • Supports both unit and integration testing • Active community and extensive documentation Cons: • Limited features compared to third-party frameworks • Slight learning curve for advanced features • May require additional plugins for some specific testing needs

Argument:
pytest offers a great balance of simplicity and power, making it well-suited for our IoT device testing needs. Its simple syntax allows for rapid test development, crucial for our MVP timeline. The extensive plugin ecosystem provides flexibility to address various testing scenarios, including hardware mocking if needed. pytest's good integration with PyCharm (our chosen IDE) and GitHub Actions (our CI system) ensures a smooth development and testing workflow. While there's a slight learning curve for advanced features, the benefits in terms of readability, maintainability, and extensibility outweigh this drawback for our project.

Implications:
- We will need to install pytest and potentially some plugins on our development machines and the Raspberry Pi
- We will need to integrate pytest into our GitHub Actions CI workflow
- We may need to create custom fixtures for mocking hardware interactions
- We should establish coding standards for writing tests using pytest
- We should leverage pytest's parameterized testing features for thorough sensor data testing
- We should consider using pytest-cov for code coverage reporting

- Using pytest as the primary testing framework will provide a robust and flexible testing setup for the sensor interfacing software
- We will benefit from strong community support and extensive documentation, facilitating rapid development and reliable testing

Viable alternatives:

Alternative	**Pros and Cons**
Unittest	Pros: • Included with Python, no additional installation required • Includes mock library for mocking hardware interfaces and sensor data Cons: • Limited features compared to third-party frameworks • More verbose test writing compared to pytest • Less expressive assertion messages

12.3.11 Testing Framework for Web Application Backend (Node.js)

Date:

Context:
Testing the web application backend developed in Node.js is crucial for ensuring that the API endpoints, business logic, and data handling processes work correctly. Selecting a suitable testing framework will help automate the testing process, ensuring the reliability and quality of the backend services.

Needs:
- Free and open-source
- Ease of use and good integration with chosen IDE (PyCharm)
- Good documentation and community support
- Support for unit testing and integration testing
- Ability to mock external services and databases
- Good performance and fast test execution
- Compatibility with continuous integration system (GitHub Actions)
- Clear and informative test result reporting
- Support for asynchronous testing

Assumptions:
- The development team has basic proficiency in JavaScript (Node.js) testing
- The chosen framework has long-term support and is in active development

Constraints:
- Limited budget for software procurement
- Need for rapid development of the MVP
- Must be compatible with chosen IDEs (PyCharm)
  and continuous integration system (GitHub)

Decision:
We decided on ***Jest*** for JavaScript (Node.js) testing framework, knowing its limitations and the implications of this decision.

Decision	**Pros and Cons**
Jest	Pros: • All-in-one solution (test runner, assertion library, mocking) • Minimal configuration required • Built-in code coverage reporting • Excellent mocking capabilities • Supports asynchronous testing • Good integration with most IDEs including PyCharm • Active community and extensive documentation Cons: • Can be slower for large test suites

Argument:
Jest offers a comprehensive testing solution that aligns well with our needs for the Node.js backend. Its all-in-one approach, including built-in assertion library and mocking capabilities, reduces the need for additional testing libraries and simplifies our testing setup. The minimal configuration required allows for rapid test development, crucial for our MVP timeline. Jest's support for asynchronous testing is particularly valuable for our Node.js backend, where many operations are asynchronous. While it may be slower for very large test suites, this is unlikely to be a significant issue for our MVP stage.

Implications:
- We will need to install Jest and potentially some additional plugins in our project
- We will need to integrate Jest into our GitHub Actions CI workflow
- We may need to optimize test execution if test suites become very large in the future
- We should establish coding standards for writing tests using Jest
- We should leverage Jest's snapshot testing for API responses where appropriate
- We should use Jest's built-in code coverage reporting to monitor test coverage

- Using Jest as JavaScript (Node.js) testing framework will provide a comprehensive and flexible testing setup for the web application backend
- We will benefit from strong community support and extensive documentation, facilitating rapid development and reliable testing

Viable alternatives:

Alternatives	**Pros and Cons**
Mocha with Chai	Pros: • Large ecosystem of plugins • Supports both browser and Node.js testing Cons: • Requires more setup and configuration • Need to choose and integrate separate assertion and mocking libraries
Jasmine	Pros: • Built-in assertion and mocking capabilities • Simple syntax Cons: • Less actively developed compared to Jest • Smaller ecosystem and community

12.3.12 Testing Framework for Web Application Frontend (Vue.js)

Date:

Context:

Needs:
- Free and open-source
- Ease of use and good integration with chosen IDE (PyCharm)
- Good documentation and community support
- Support for unit testing and integration testing

Assumptions:
- The development team has basic proficiency in JavaScript (Vue.js) testing
- The chosen framework has long-term support and is in active development

Constraints:
- Limited budget for software procurement
- Need for rapid development of the MVP
- Must be compatible with chosen IDEs (PyCharm)
  and continuous integration system (GitHub)

Decision:
We decided on ***Jest*** for JavaScript (Vue.js) testing framework, knowing its limitations and the implications of this decision.

Decision	**Pros and Cons**
	Pros: • Cons: •

Argument:

Implications:
-
-

Viable alternatives:

Alternatives	**Pros and Cons**
	Pros: • Cons: •
	Pros: • Cons: •

12.3.13 Cloud Service Provider

Date:

Context:
The GrowHub MVP requires a cloud service provider to host our web application and databases, as well as to facilitate communication with IoT devices. We need a provider that offers a comprehensive suite of services to support our distributed architecture and IoT integration.

Needs:
- Offer of free-tier options for needed cloud services
- Ease of development, deployment, and monitoring
- Comprehensive set of services including compute, storage, database, and IoT-specific offerings
- Reliable performance and high availability
- Strong security features and compliance certifications
- Good documentation and developer support
- Good integration with our IoT compute unit (Raspberry Pi 4 Model B)
- Scalable infrastructure to accommodate future growth
- Global data center presence for potential future expansion

Assumptions:
- The development team has basic proficiency in cloud service management
- The chosen provider will offer long-term support and service continuity
- We can use the free-tier offerings of cloud provider to minimize our expenses for cloud services

Constraints:
- Limited budget for cloud services
- Need for rapid development of the MVP
- Must work well with the Raspberry Pi ecosystem

Decision:
We decided on ***Amazon Web Services** (**AWS**)* for provider of cloud services, knowing its limitations and the implications of this decision.

Decision	**Pros and Cons**
Amazon Web Services (AWS)	Price: • € XY monthly subscription per account Pros: • Comprehensive service offerings, including robust IoT services • Extensive free tier ideal for MVP development • Large community and extensive documentation Cons: • Can be complex for beginners • Potential for unexpected costs if not managed carefully

Argument:
Amazon Web Services (AWS) offers a comprehensive suite of services that align well with our needs. Their extensive free tier offerings allow us to develop and initially deploy our MVP with minimal costs. AWS's global infrastructure, robust security features, and extensive documentation support our current needs and future scalability. While other providers like Google Cloud and Microsoft Azure offer similar services, AWS's more extensive free tier and specific IoT services (like IoT Core) make it more suitable for our MVP development and IoT integration.

Implications:
- We will need to carefully manage our usage to stay within free tier limits where possible
- We will need to implement proper security measures and follow AWS best practices for IoT device management
- We will need to invest time in learning AWS-specific tools and best practices
- We may face challenges if we decide to migrate to a different cloud provider in the future
- We should design our architecture to take advantage of AWS-specific services while avoiding over-dependence
- We should monitor our cloud spending closely to prevent unexpected costs

- Using AWS will ensure access to a reliable and scalable cloud infrastructure
- AWS's comprehensive service offerings will allow seamless integration of various components (frontend, backend, IoT communication, and database)
- We will benefit from robust documentation and community support, facilitating rapid development and deployment

Viable alternatives:

Alternatives	**Pros and Cons**
Microsoft Azure	Price: • € XY monthly subscription per account Pros: • Strong integration with Microsoft technologies • Comprehensive set of services, including IoT offerings • Good hybrid cloud capabilities Cons: • Less extensive free tier compared to AWS • Can be more expensive for some services • Learning curve may be steeper for non-Microsoft developers
Google Cloud Platform (GCP)	Price: • € XY monthly subscription per account Pros: • Strong data analytics and machine learning capabilities • User-friendly interface and good documentation Cons: • Less extensive free tier compared to AWS • Fewer IoT-specific services • Smaller community compared to AWS

12.3.14 Cloud Service for Communication with IoT Devices

Date:

Context:
The GrowHub MVP requires a cloud service to facilitate secure and efficient communication between our IoT compute unit (Raspberry Pi) and the cloud backend. This service needs to handle device connectivity, data ingestion, and message routing for our IoT system.

Needs:
- Offer of free-tier option
- Ease of management and debugging
- Secure and reliable communication protocol for IoT devices
- Support for real-time data ingestion and processing
- Scalability to handle multiple devices and high message throughput
- Device management capabilities
- Integration with other AWS services
- Support for various messaging patterns (publish/subscribe, request/response)
- Compatibility with our chosen IoT device (Raspberry Pi) and its software stack (Python)

Assumptions:
- The development team has basic proficiency in IoT communication protocols and AWS services
- The chosen cloud service will integrate seamlessly with the Raspberry Pi 4 and the overall system architecture

Constraints:
- Limited budget for cloud services
- Need for rapid development of the MVP
- We will be using AWS as our primary and only cloud service provider

Decision:
We decided on ***IoT Core*** for cloud service for communication with IoT devices, knowing its limitations and the implications of this decision.

Decision	**Pros and Cons**
AWS IoT Core	Free-Tier: • 500,000 minutes of connection per month (first 12 months) • 250,000 job operations per month (first 12 months) • 225,000 messages per month (first 12 months) Pros: • Easy integration with other AWS services • Supports MQTT protocol Cons: • Potential for costs to increase with scale

Argument:
AWS IoT Core provides a managed cloud platform that lets connected devices easily and securely interact with cloud applications and other devices. It offers robust security features, scalability, and seamless integration with other AWS services. The service supports MQTT protocol, which is ideal for our IoT communication needs. AWS IoT Core's ability to trigger AWS Lambda functions or send data directly to services like DynamoDB aligns well with our architecture. While there are other IoT communication services available, AWS IoT Core's deep integration with the AWS ecosystem we've chosen makes it the most suitable option for our MVP.

Implications:

- We will need to implement AWS IoT SDK in our Raspberry Pi software
- We may need to carefully manage the number of connected devices and messages to stay within free tier limits
- We may need to set up IoT rules to route device data to the appropriate AWS services
- We should design our device communication protocol to efficiently use MQTT topics
- We should implement proper error handling and reconnection logic in our device software

- Using AWS IoT Core will ensure secure and reliable communication between the IoT devices and the cloud backend
- We will have the ability to easily integrate with other AWS services for data processing and storage
- The development team will benefit from robust documentation and community support, facilitating rapid development and deployment
- We will benefit from built-in security features like X.509 certificate-based authentication

Viable alternative:

Alternative	Pros and Cons
AWS IoT Greengrass	Free-Tier: • No free-tier options Pros: • Supports local processing and ML inference • Works offline and syncs when connected Cons: • More complex to set up and manage • May be overkill for simple IoT projects • Requires more powerful edge devices

12.3.15 Cloud Deployment Service for Time-series Databases

Date:

Context:
The GrowHub MVP requires a cloud service to host and manage our time-series databases. It needs to efficiently ingest, store, and query large volumes of time-stamped data from various sensors. We need a database that can scale with our needs.

Needs:
- Offer of free-tier option
- Ease of deployment, management and maintenance
- High-speed data ingestion to handle real-time sensor data
- Efficient storage and retrieval of time-series data
- Support for time-based queries and aggregations
- Ability to handle large volumes of data over time
- Scalability to accommodate increasing numbers of sensors and data points
- Easy Integration with AWS IoT Core

Assumptions:
- The development team has basic proficiency in database management
- The chosen database will integrate well with our backend and other AWS services
- The sensor data will primarily consist of numerical values with timestamps
- Historical data will need to be retained for analysis and reporting

Constraints:
- Limited budget for cloud services
- Need for rapid development of the MVP
- Must be compatible with chosen backend technology stack (Node.js runtime)
- We will be using AWS as our primary and only cloud service provider

Decision:
We decided on ***Amazon DynamoDB*** for cloud service for hosting time-series databases, knowing its limitations and the implications of this decision.

Decision	**Pros and Cons**
Amazon DynamoDB	Free-Tier: • 25 GB of storage (always free) • 200,000,000 requests per month (always free) Pros: • Fully managed, serverless NoSQL database • Seamless scalability and high performance • Built-in support for time-series data Cons: • Potential for higher costs with increased usage • Requires careful data modeling to avoid performance issues

Argument:
While Amazon DynamoDB is not specifically designed for time-series data like Timestream, it offers a flexible and scalable solution that can be adapted to handle sensor data effectively. The decision to use DynamoDB is primarily driven by its more generous free tier offering, which is crucial for our MVP stage. DynamoDB's seamless integration with other AWS services, including AWS IoT Core, aligns well with our AWS-centric architecture. Its ability to handle high write throughput and low-latency reads makes it suitable for real-time sensor data ingestion and querying. Although it may require more careful data modeling, the maturity of the service and extensive community support will aid in implementing best practices for time-series data in DynamoDB.

Implications:
- We will need to design a data model optimized for time-series data in DynamoDB, potentially using a composite primary key with device ID as the partition key and timestamp as the sort key
- We will need to carefully monitor our usage to ensure we stay within free tier limits
- We may need to implement additional caching mechanisms (e.g., DAX) if we require faster read performance for frequently accessed data
- We may need to use DynamoDB Streams in combination with AWS Lambda for real-time data processing and aggregations
- We should plan for potential future migration to a specialized time-series database if our needs outgrow DynamoDB's capabilities
- We should implement appropriate TTL (Time to Live) settings to manage data retention and storage costs
- We should consider using DynamoDB's on-demand capacity mode as we scale beyond the free tier to handle variable sensor data ingestion rates

- Using Amazon DynamoDB will provide a scalable, reliable, and cost-effective database solution for managing time-series data from IoT devices
- The decision to use separate services for time-series and relational data will optimize performance and scalability

Viable alternatives:

Alternative	**Pros and Cons**
Amazon Timestream	Free-Tier: • 100 GB of Magnetic tier storage (first 30 days) • 750 GB-hour of Memory tier storage (first 30 days) • 50 GB for data ingestion of data storage (first 30 days) Pros: • Automatically scales to handle variable data ingestion rates • Native integration with AWS IoT Core and other AWS services • Automatic data tiering and SQL-compatible query language Cons: • Relatively new service with evolving best practices • Potential for higher costs as data volume grows beyond free tier • Limited third-party tool support

12.3.16 Cloud Deployment Service for Relational Databases

Date:

Context:
The GrowHub MVP requires a cloud service to host and manage our relational SQL databases to handle user management and application state data. This database needs to efficiently store and retrieve structured data, support complex queries, and maintain data integrity.

Needs:

- Offer of free-tier option
- Ease of deployment, management and maintenance
- Reliable storage for user data and application state
- Support for complex queries and transactions
- High availability and data durability
- Backup and recovery capabilities
- Support for ACID properties
- Scalability to accommodate potential future growth

Assumptions:

- The development team has experience with relational database management
- The data structure for users and application state is well-defined and relatively stable
- The chosen database will integrate well with our backend and other AWS services

Constraints:

- Limited budget for cloud services
- Need for rapid development of the MVP
- Must be compatible with chosen backend technology stack (Node.js runtime)
- We will be using AWS as our primary and only cloud service provider

Decision:
We decided on ***Amazon RDS*** *with* ***MySQL*** *engine* for cloud service for hosting relational SQL databases, knowing its limitations and the implications of this decision.

Decision	**Pros and Cons**
Amazon RDS with MySQL engine	Free-Tier: • 20 GB of storage + 20 GB of backup storage (first 12 months) • 750 hours of t2.micro EC2 instance per month (first 12 months) Pros: • Managed service, reducing operational overhead • Familiar MySQL engine with wide community support • Good performance for relational data and complex queries • Automatic backups and easy scaling Cons: • Limited vertical scaling options in free tier • Less flexibility compared to self-managed databases

Argument:
Amazon RDS with MySQL engine provides a robust, managed relational database service that aligns well with our needs for user management and application state data. The familiar MySQL engine ensures compatibility with existing tools and libraries, and the wide community support can aid in problem-solving. RDS's managed nature reduces operational overhead, allowing us to focus on application development rather than database management. The free tier offering is sufficient for our MVP needs, providing enough compute and storage resources to get started. As a mature service, RDS offers reliable performance, automated backups, and scaling options, which are crucial for maintaining data integrity and availability.

Implications:

- We will need to design our database schema to efficiently handle user data and application state
- We will need to monitor our usage to ensure we stay within free tier limits and plan for potential cost increases
- We will need to implement connection pooling in our application to efficiently manage database connections
- We may need to optimize queries and implement indexing strategies as the data volume grows
- We should implement appropriate access controls and security measures to protect user data
- We should set up automated backups and test restore procedures to ensure data recoverability
- We should consider implementing a caching layer (e.g., Amazon ElastiCache) if read performance becomes a bottleneck

- Using Amazon RDS will ensure robust and reliable management of relational data for user and application state
- We will benefit from seamless integration with other AWS services, facilitating rapid development and reliable data management

Viable alternatives:

Alternative	**Pros and Cons**
Amazon DynamoDB	Free-Tier: • 25 GB of storage (always free) • 200,000,000 requests per month (always free) Pros: • Fully managed, serverless NoSQL database • Seamless scalability and high performance Cons: • Less suitable for complex relational data and joins • May require significant changes to data access patterns

12.3.17 Cloud Deployment Service for Web Application Backend

Date:

Context:
The GrowHub MVP requires a cloud service to host our web application's backend, which will handle data processing, API requests, and business logic. This service needs to be scalable and cost-effective, and compatible with our chosen backend technology stack (Node.js).

Needs:
- Offer of free-tier option
- Ease of deployment and management
- Ease of Integration with other AWS services (e.g., AWS IoT Core, databases)
- Scalability to handle varying loads
- Support for RESTful APIs and WebSocket connections
- Reasonable performance for real-time data processing

Assumptions:
- The development team has basic proficiency in backend development and cloud services
- The chosen cloud service will integrate seamlessly with other AWS services and the overall system architecture

Constraints:
- Limited budget for cloud services
- Need for rapid development of the MVP
- Must be compatible with chosen backend technology stack (Node.js runtime) and distributed version control system (Git)
- We will be using AWS as our primary and only cloud service provider

Decision:
We decided on ***AWS Elastic Beanstalk*** with ***Amazon S3*** as cloud services for hosting web application backend, knowing their limitations and the implications of this decision.

Decision	**Pros and Cons**
AWS Elastic Beanstalk with Amazon S3	Free-Tier: • 750 hours of t2.micro EC2 instance per month (first 12 months) • 5 GB of S3 storage per month (first 12 months) • 20,000 GET and 2,000 PUT requests per month (first 12 months) Pros: • Managed platform, reducing operational overhead • Easy deployment and scaling • Provides monitoring and health checks Cons: • Less fine-grained control compared to EC2 • May have limitations for very specific configurations

Argument:
AWS Elastic Beanstalk provides a managed platform that simplifies the deployment and scaling of web applications. It supports Node.js and integrates seamlessly with other AWS services. Elastic Beanstalk automates the details of capacity provisioning, load balancing, and application health monitoring, reducing operational complexity. While there are other options like EC2 or Lambda, Elastic Beanstalk offers a good balance of control and convenience, making it suitable for our MVP stage. Its free tier offering also aligns with our cost constraints for initial development and testing.

Implications:

- We will need to structure our Node.js application to work with Elastic Beanstalk's deployment model
- We will need to set up proper IAM roles for Elastic Beanstalk to interact with other AWS services
- We may need to carefully monitor our usage to stay within free tier limits
- We may face some limitations in fine-grained control compared to managing our own EC2 instances
- We should utilize Elastic Beanstalk's environment variables for configuration management
- We should implement proper logging and use AWS CloudWatch for monitoring

- Using AWS Elastic Beanstalk will simplify the deployment and management of the backend services, allowing the development team to focus on application logic
- We will benefit from automated scaling and load balancing capabilities
- We will benefit from robust documentation and community support, facilitating rapid development and deployment
- Elastic Beanstalk's integration with other AWS services will ensure a seamless and efficient backend architecture

Viable alternatives:

Alternative	**Pros and Cons**
Amazon EC2 with Amazon S3	Free-Tier: • 750 hours of t2.micro EC2 instance per month (first 12 months) • 5 GB of S3 storage per month (first 12 months) • 20,000 GET and 2,000 PUT requests per month (first 12 months) Pros: • Full control over the server environment • Flexible and customizable • Can be more cost-effective for optimized workloads Cons: • Requires more management and operational overhead • Responsibility for scaling and high availability

12.3.18 Cloud Deployment Service for Web Application Frontend

Date:

Context:
The GrowHub MVP requires a cloud service to host our web application's frontend, which will provide the user interface for monitoring and controlling IoT devices. This service needs to be fast, reliable, and cost-effective for serving static content.

Needs:
- Offer of free-tier option
- Ease of deployment and management
- Ease of Integration with other AWS services (e.g., Elastic Beanstalk, Amazon S3)
- Efficient hosting and delivery of static files (HTML, CSS, JavaScript)
- Support for single-page applications (SPA)
- Secure communication (HTTPS)
- Scalability to handle varying loads
- Global content delivery for low-latency access

Assumptions:
- The development team has basic proficiency in frontend development and cloud services
- The chosen cloud service will integrate seamlessly with other AWS services and the overall system architecture

Constraints:
- Limited budget for cloud services
- Need for rapid development of the MVP
- Must be compatible with chosen frontend technology stack (Vue.js runtime) and distributed version control system (Git)
- We will be using AWS as our primary and only cloud service provider

Decision:
We decided on ***AWS Amplify*** as cloud service for hosting web application frontend, knowing its limitations and the implications of this decision.

Decision	**Pros and Cons**
AWS Amplify	Free-Tier: • 5 GB stored per month (always free) • 15 GB served per month (always free) • 1000 build minutes per month (always free) Pros: • Supports real-time data updates • Integrated hosting and CI/CD pipeline • Built-in support for single-page applications Cons: • Limited control over underlying infrastructure compared to EC2

Argument:
AWS Amplify provides a comprehensive solution for hosting and managing frontend applications, offering features beyond simple static hosting. It supports modern web frameworks, provides an integrated CI/CD pipeline, and offers easy integration with other AWS services. Its built-in support for single-page applications aligns well with our frontend architecture. While it may offer more features than we initially need for static hosting, it provides a scalable path for future development and simplifies the deployment process. The free tier offering is generous enough for our MVP stage, making it a cost-effective choice.

Implications:
- We will need to learn and implement Amplify-specific configurations for our frontend application
- We may need to adapt our development workflow to take full advantage of Amplify's features
- We should leverage Amplify's built-in support for single-page applications in our Vue.js frontend
- We should be mindful of the build minutes usage to stay within the free tier limits
- We should explore Amplify's authentication and API features for potential future use
- We will benefit from robust documentation and community support, facilitating rapid development and deployment
- We will benefit from an integrated hosting and CI/CD pipeline, simplifying our deployment process and management of the frontend
- We will have easier integration with other AWS services, which could be beneficial as our application grows

Viable alternatives:

Alternatives	**Pros and Cons**
Amazon CloudFront with Amazon S3	Free-Tier: • 5 GB of S3 storage per month (first 12 months) • 20,000 GET and 2,000 PUT requests per month (first 12 months) • 2 million HTTP/HTTPS requests per month (first 12 months) • 50 GB data transfer out to the internet (first 12 months) Pros: • Fine-grained control over caching and distribution Cons: • Limited support for server-side operations • Requires separate solution for backend API • Requires more configuration and management than Amplify
Amazon EC2	Free-Tier: • 750 hours of t2.micro EC2 instance per month (first 12 months) • 30 GB of Elastic Block Storage (EBS) per month (first 12 months) Pros: • Full control over the server environment Cons: • Requires more management and operational overhead

12.3.19 Cloud Configuration Management

Date:

Context:

Needs:

- Free or low cost
- Ease of use and good integration with chosen IDE (PyCharm)
- Good documentation and community support
-

Assumptions:

- The development team has basic proficiency in XY
- The chosen solution has long-term support and is in active development

Constraints:

- Limited budget for software procurement
- Need for rapid development of the MVP

Decision:
We decided on ***XY*** for configuration management, knowing its limitations and the implications
of this decision.

Decision	**Pros and Cons**
	Price: • Pros: • Cons: •

Argument:

Implications:

-
-

Viable alternatives:

Alternatives	Pros and Cons
	Price: • Pros: • Cons: •
	Price: • Pros: • Cons: •

12.3.20 Cloud Monitoring

Date:

Context:

Needs:

- Free or low cost
- Good documentation and community support
-

Assumptions:

- The development team has basic proficiency in XY
- The chosen solution has long-term support and is in active development

Constraints:

- Limited budget for software procurement
- Need for rapid development of the MVP

Decision:
We decided on ***XY*** for cloud monitoring, knowing its limitations and the implications of this decision.

Decision	**Pros and Cons**
	Price: • Pros: • Cons: •

Argument:

Implications:

-
-

Viable alternatives:

Alternatives	**Pros and Cons**
	Price: • Pros: • Cons: •
	Price: • Pros: • Cons: •

12.4 External Partners

12.4.1 External PCB Manufacturer

Date:

Context:

Needs:

- Low cost
- Good references and easy to use website
-

Assumptions:

-

Constraints:

- Limited budget for hardware procurement
- Need for rapid development of the MVP

Decision:

We decided on ***XY*** for external PCB manufacturer, knowing its limitations and the implications
of this decision.

Decision	Pros and Cons
	Price: • Pros: • Cons: •

Argument:

Implications:

-
-

Viable alternatives:

Alternatives	Pros and Cons
	Price: • Pros: • Cons: •
	Price: • Pros: • Cons: •

13. APPENDICES

13.1 Technology Stack

13.2 Third-party Libraries

13.3 References

- Python documentation
- Vue.js documentation
- Node.js documentation
- MongoDB documentation

13.4 Glossary of Terms

- **Application Programming Interface** (**API**): A set of protocols, routines, and tools for building software applications. An API specifies how software components should interact and allows different software systems to communicate with each other.
- **Configuration Item** (**CI**): A uniquely identifiable element within a configuration management system, which can be managed and controlled through the configuration management process. CIs can be components of hardware, software, documentation, or any other essential part of a project that requires management to ensure consistency and quality throughout its lifecycle.
- **Daily Scrum** (Daily Stand-up): Formal short daily event of Scrum framework (typically 15 minutes) held every day of the Sprint. It is a key event in the Scrum framework where the Development Team synchronizes their work and plans for the next 24 hours.
- **Internet of Things** (**IoT**): The network of physical objects embedded with sensors, software, and other technologies that connect and exchange data with other devices and systems over the internet. These objects, or "things," can range from everyday household items to sophisticated industrial tools.
- **Minimum Viable Product** (**MVP**): The most basic version of a product that is still functional and usable by early adopters. It includes only the essential features needed to meet the initial user needs and provide feedback for future development.
- **Printed Circuit Board** (**PCB**) - Flat board made of insulating material, typically fiberglass, with conductive pathways or "traces" etched or printed onto it. These traces connect various electronic components such as resistors, capacitors, and integrated circuits, which are soldered onto the board.
- **Raspberry Pi**: A series of small single-board computers developed in the UK by the Raspberry Pi Foundation. It's widely used in IoT projects due to its low cost, modularity, and open architecture.
- **Sensor**: A device that detects and responds to some type of input from the physical environment. The output is generally a signal that is converted to human-readable display at the sensor location or transmitted electronically over a network for reading or further processing.
- **Scrum**: An agile framework for managing complex projects, most commonly used in software development. It is designed to help teams work together, encouraging collaboration, accountability, and iterative progress toward

a well-defined goal. Scrum enables teams to deliver high-value products by dividing the work into short, manageable cycles called Sprints.

- **Sprint**: Fixed time-boxed period (usually 1-4 weeks) of Scrum framework during which a Scrum team works to complete a specific set of work items from the Product Backlog. Each Sprint is a mini-project aimed at creating a usable and potentially releasable increment of the product.
- **Sprint Planning**: Formal event of Scrum framework that kicks off the Sprint. During this meeting, the team defines the work to be completed during the upcoming Sprint.
- **Sprint Retrospective**: Formal event of Scrum framework held after the Sprint Review and before the next Sprint Planning, where the Scrum team reflects on the past Sprint to identify improvements for future Sprints.
- **Sprint Review**: Formal event of Scrum framework held at the end of each Sprint, where the Scrum team and stakeholders inspect the increment of the product developed during the Sprint and discuss feedback and future work.
- **User Interface** (**UI**): The point of interaction between a user and a computer system, typically through a set of visual elements like buttons, menus, and icons. It encompasses both the visual design and the interactive elements of a software application.
- **Version Control**: System that records changes to a file or set of files over time so that specific versions can be recalled later. It is essential for managing source code in software development projects.