HTL-Donaustadt
Höhere Abteilung für Informatik

DIPLOMARBEIT

Titel der Diplomarbeit

Untertitel der Diplomarbeit

Ausgeführt im Schuljahr 2025/26 von:

Abudi
Arun
Dennis
Richard

Betreuer/Betreuerin:

Mag. Dr. Putzinger

Wien, 30. Dezember 2025

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Max Mustermann

Wien, 30. Dezember 2025

Danksagungen

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat.

Einleitung

Subsubsection 1

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat.

Subsubsection 2

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat.

Abstract

Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Subsubsection 1

Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Subsubsection 2

Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Gendererklärung

Das in dieser Arbeit gewählte generische Maskulinum bezieht sich zugleich auf die männliche, die weibliche und andere Geschlechteridentitäten. Zur besseren Lesbarkeit wird auf die Verwendung männlicher und weiblicher Sprachformen verzichtet. Alle Geschlechteridentitäten werden ausdrücklich mitgemeint, soweit die Aussagen dies erfordern.

Inhaltsverzeichnis

Kapitel 1

Gamification. Gaming design patterns to keep people engaged in apps.

1.1 Introduction

1.1.1 Background and Context

In recent years, the Internet of Things (IoT) has woven itself into the fabric of daily life at an extraordinary pace, fundamentally changing how we interact with our physical surroundings. This digital transformation is perhaps most visible in the agricultural sector, which is currently navigating a shift known as “Agriculture 4.0”. We are seeing a move away from traditional, intuition-based farming toward precise, data-driven decision-making, all enabled by interconnected sensor networks [?]. While industrial agriculture has already reaped significant efficiency gains from this revolution, a parallel and equally important trend is taking root in the consumer world: Smart Urban Gardening [?].

Driven by rapid global urbanization, shrinking living spaces, and a rising awareness of environmental sustainability, urban gardening has grown into a significant movement [?]. This is more than just a hobbyist trend; it signals a shift toward the Social Internet of Things (SIoT) [?]. The SIoT paradigm expands our traditional understanding of IoT by suggesting that objects can establish social relationships, not just with other machines but with people, creating a truly collaborative ecosystem [?]. In this new context, a plant is no longer just a passive biological entity. It becomes an active participant in a digital social network, capable of communicating its needs and status directly to its caretaker.

However, bringing these advanced concepts into the home using consumer-grade hardware creates a unique set of engineering hurdles [?]. Unlike industrial systems, which often rely on stable power grids and dedicated infrastructure, consumer IoT devices for plant care must survive in resource-constrained environments. These devices often need to run for months on small batteries while staying connected to congested and unstable home Wi-Fi networks. This reality demands a rigorous and creative approach to optimizing both hardware and software strategies [?].

1.1.2 Case Study: The "Plant Up!" Project

This thesis uses the "Plant Up!" project as a primary testbed to investigate these architectural frictions. "Plant Up!" is a comprehensive IoT application designed to gamify the experience of plant care, turning routine maintenance into an engaging activity [?]. By providing immediate digital feedback on biological processes, the system bridges the gap between human perception and the actual physiological needs of a plant.

The system is built on a robust three-tier architecture that combines distributed sensor units, a scalable cloud microservices backend, and a user-facing mobile application [?]. At the edge, ESP32-based sensor nodes continuously monitor environmental metrics like soil moisture, temperature, humidity, and light intensity. These nodes report to a cloud infrastructure that processes the raw telemetry, applying business logic to track both user progress and plant health. Finally, the application layer presents this data through an interface that transforms mundane tasks into social achievements and rewards.

1.1.3 Problem Statement

Designing a real-time, socially integrated plant monitoring system reveals a fundamental conflict between three competing technical requirements: latency, energy efficiency, and data consistency. The "Social" aspect of the platform relies heavily on gamification mechanics that demand low latency to keep users immersed [?]. For example, when a user waters their plant, they expect to see that action reflected in the app almost immediately.

However, to simply remain practical for home use, the hardware cannot drain its battery in a matter of days. It must utilize aggressive power-saving states, such as the ESP32's Deep Sleep mode [?]. While deep sleep dramatically extends battery life by shutting down the CPU and Wi-Fi radio, waking up and reconnecting introduces unavoidable delays that directly clash with the need for real-time responsiveness [?].

Furthermore, the choice of communication protocol dictates the “wake-up tax”, which is the energy cost paid just to establish a connection before a single byte of data is sent [?]. Traditional web protocols like HTTP are robust and universal, but they carry heavy header overhead and rely on verbose text formats. In contrast, lightweight protocols like MQTT are designed specifically to solve these inefficiencies using binary payloads and a publish/subscribe model [?].

Finally, maintaining a consistent view of the system state in such a distributed architecture is a non-trivial challenge. As the CAP theorem (Consistency, Availability, Partition Tolerance) warns us, a distributed system cannot guarantee all three properties simultaneously during a network failure [?]. In a residential IoT setting where network partitions are a common occurrence, the system must make calculated trade-offs between keeping data consistent and keeping the service available.

1.1.4 Research Question

To systematically address these challenges and identify the optimal architectural approach, this thesis poses the following primary research question:

How do MQTT and REST compare in a microservices-based architecture for real-time plant monitoring, specifically regarding latency, throughput, and data consistency, when constrained by the energy limitations of battery-powered IoT devices [?]?

1.2 Theoretical Foundations of Gamification

1.2.1 Definition of Gamification

Gamification, a term that defines how game design elements and mechanics are integrated to non-game contexts, while the main activity remains a non-game. The goal of introducing these elements is to increase user engagement and motivation by making otherwise boring or routine tasks more appealing [?, ?].

1.2.2 Difference between Gamification, Serious Games, and Games for Entertainment

It is important to take note that gamification is not the same as serious games or games for entertainment. Although the terms sound the same, they describe different concepts. While gamification refers to the integration of game elements into non-game contexts in order to increase user engagement and motivation, serious games are complete games that are explicitly designed for a primary purpose other than entertainment, such as education, training, or simulation. A common example is a flight simulator used for pilot training. On the other hand games for entertainment are only produced for enjoyment, without any explicit educational or functional objective [?].

1.2.3 Why Gamification Works in Non-Game Contexts

While many people are skeptical about the effectiveness of gamification, research and practice have shown that it can be an effective strategy for enhancing user engagement and motivation. This raises the questions: 'Why and how?'.

According to the Haufe Akademie, the effectiveness of gamification lies deep in the human psyche, it addresses the basic needs of autonomy, competence, and social relatedness, which are the foundation of human motivation. How gamification achieves this can be explained using three central mechanisms.

- **1:** Gamification integrates the human need for autonomy, competence, and social relatedness not merely through external rewards, but by supporting these needs.
- **2:** It redefines user interaction with a system. Instead of removing all hard and boring tasks to maximize efficiency, gamification introduces voluntary challenges that encourage people to improve themselves. These tasks are transformed into engaging activities in which progress is clearly visualized and displayed with immediate feedback mechanisms such as points, badges and so on.
- **3:** Gamification uses social interaction to build a sense of community. Features such as cooperative challenges and leaderboards satisfy the desire of being noticed by others, thereby people feel more attached to the activity, by getting a competitive feeling. [?].

1.2.4 Motivation Theory

To understand why gamification can be effective in practice, it is necessary to examine the foundations of human motivation. Motivation is the psychological force that initiates, guides, and sustains goal-directed behavior. In the context of gamification, a key distinction is made between engaging in an activity because it is inherently rewarding (intrinsic motivation) and engaging in it to obtain an external outcome (extrinsic motivation) [?].

Motivation should not be thought about as a monolithic construct because it exists on a spectrum. The two primary types, that are relevant to gamification, are intrinsic- and extrinsic motivation.

Intrinsic Motivation (The Engine of Engagement)

Intrinsic motivation refers to engaging in an activity because it is satisfying, enjoyable, or meaningful, which is considered essential for long-term engagement [?]. According to Self-Determination Theory (SDT) [?], intrinsic motivation is supported when three basic psychological needs are fulfilled:

- **Autonomy:** The perception of choice and self-direction.
- **Competence:** The experience of mastery and effectiveness.
- **Relatedness:** The feeling of belonging and social connection.

When these needs are satisfied, individuals demonstrate increased curiosity, creativity, and persistence [?].

Extrinsic Motivation (The Spark)

Extrinsic motivation involves performing an activity in order to receive a reward or avoid negative consequences. Common gamification elements such as points, badges, and leaderboards are examples of extrinsic motivators [?].

- **Controlling Extrinsic Motivation:** When rewards are perceived as coercive or punitive, they can undermine intrinsic motivation, resulting in short-term compliance but reduced long-term engagement. A typical example would be a streak-based system, where the user needs to accomplish something regularly to keep his streak going. This can be counterproductive as it puts pressure on the user to perform and can lead to burnout.

– **Autonomy-Supportive Extrinsic Motivation:** On the contrary, when rewards provide informational feedback or signal competence, they can be internalized and reinforce intrinsic motivation. Instead of 'You need to do this!' it is more effective to say 'You did this well!' or 'You are getting better at this!', mostly this is done by just handing out rewards for accomplishments, while not making them feel like a chore.

Effective gamification strategies use extrinsic rewards not as the primary objective, but as a means of supporting intrinsic motivation [?, ?].

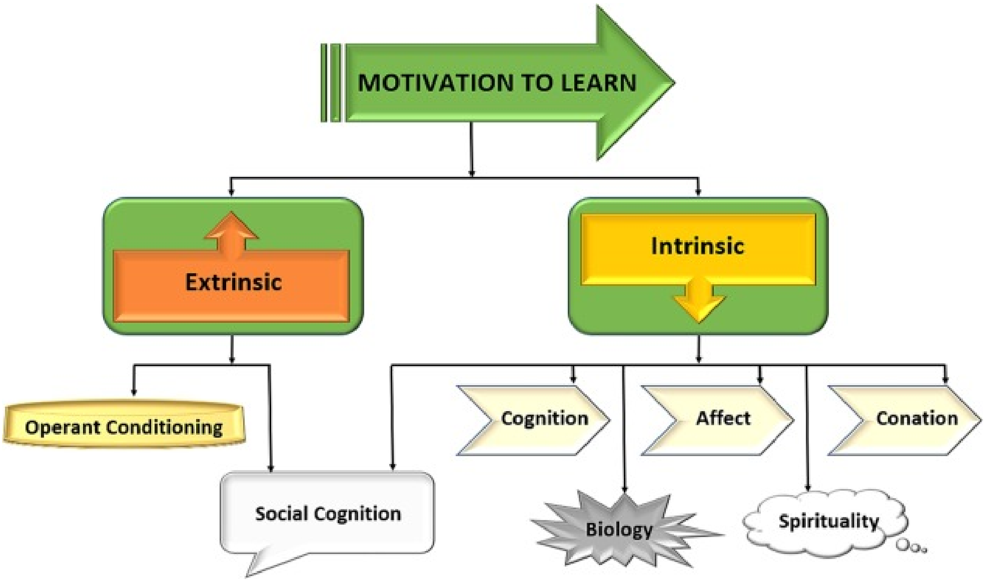

Abbildung 1.1: Motivation to Learn: Intrinsic vs. Extrinsic [?]

Self-Determination Theory (SDT) in Gamification

Self-Determination Theory (SDT) establishes a framework for human motivation based on three fundamental psychological needs: autonomy, competence, and relatedness. Meeting these needs fosters self-determined motivation, which drives enhanced performance, persistence, and creativity. In contrast, failing to support them can significantly diminish motivation and well-being [?, ?].

Supporting Autonomy

Users should experience a sense of control over their actions.

– **Design Patterns:** Optional tasks, multiple paths to success, and customization options. [?]

Supporting Competence

Users need clear feedback that demonstrates progress and skill development.

– **Design Patterns:** Experience points, progress indicators, difficulty scaling, and immediate feedback loops. [?]

Supporting Relatedness

Users are more engaged when they feel socially connected.

– **Design Patterns:** Social feeds, cooperative goals, peer comparison, and mentorship features. [?]

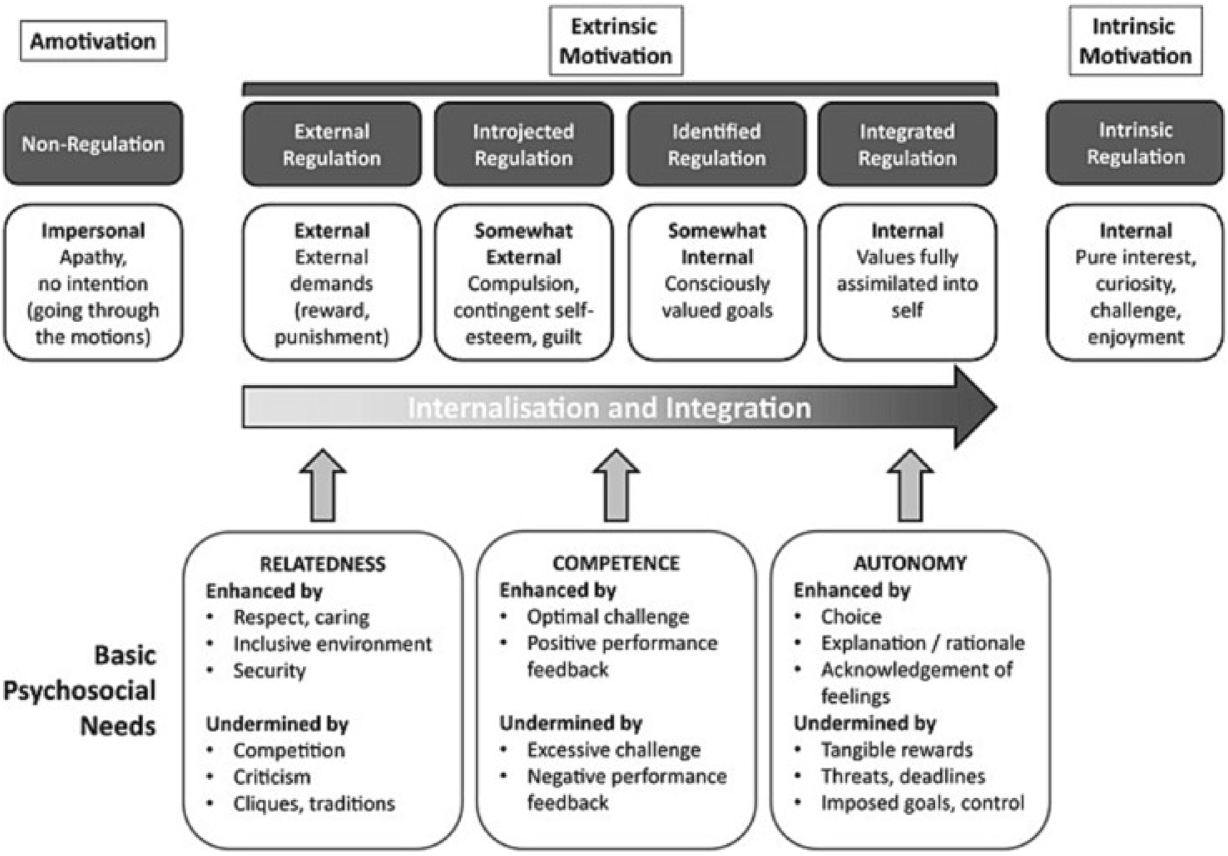

Abbildung 1.2: The Self-Determination Continuum of Motivation [?]

The Role of Extrinsic Rewards

Gamification is not merely about adding rewards, but about how they are implemented. Extrinsic motivators, which can be internal (rewards) or external (prizes) to the system, are known outcomes that drive play. While they can be strong drivers, they often come at the expense of creative problem-solving and can lead to mechanical gameplay (grinding). This results in retention (negative engagement) rather than enchantment (positive engagement) [?]. However, extrinsic motivators can still be effective by providing short-term motivational boosts that may eventually foster intrinsic interests such as competition or mastery [?, ?].

Rewards that Support Intrinsic Motivation

Intrinsic motivators are linked to a personal desire to master, explore, and enjoy the activity. Unlike extrinsic rewards, they generally lead to enchantment, where users feel a sense of "positive engagement" and are more likely to evangelize the system [?]. Effective rewards in this category function as informational feedback rather than control mechanisms.

- **Badges as Recognition:** Indicators of achievement and mastery.
- **Experience Points as Feedback:** Visualization of personal growth.
- **Virtual Currency for Customization:** Enabling autonomy and self-expression.
- **Titles as Identity Markers:** Reinforcing competence and long-term commitment. [?]

Rewards that Undermine Intrinsic Motivation

Rewards can become counterproductive when they are perceived as controlling or meaningless. In the context of smart gardening applications, these motivational mechanisms are particularly relevant, as user engagement must be sustained over long periods despite repetitive tasks such as monitoring or maintenance.

- **Trivial Action Rewards:** Over-rewarding insignificant actions.
- **Punitive Streak Systems:** Excessive penalties that induce stress.
- **Context-Free Leaderboards:** Rankings without fairness or relevance.
- **Motivational Displacement:** When rewards replace interest in the activity itself [?].

Attribution Theory

Attribution theory examines how individuals explain success and failure. In gamified systems, it is important that users attribute outcomes to their own effort and strategies rather than to luck or fixed ability.

- **Effort-Based Feedback:** Emphasizing controllable factors increases motivation.
- **Failure as Learning Opportunity:** Errors should be framed as temporary and improvable.

Expectancy-Value Theory

Before engaging in an activity, users implicitly evaluate two questions: “Can I succeed?” (expectancy) and “Is this worth my effort?” (value).

1. **Expectancy:** Clear goals and visible progress increase perceived attainability.
2. **Value:** The task must be meaningful, useful, or personally relevant [?].

1.2.5 Summary: Principles of Motivational Gamification

Effective gamification systems do not replace intrinsic motivation with rewards. Instead, they employ extrinsic elements to support autonomy, competence, and relatedness. When rewards are perceived as informational rather than controlling, they foster engagement, learning, and sustainable behavioral change.

1.2.6 Engagement and Habit Formation

- Feedback loops
- Short-term versus long-term engagement
- Daily routines and behavioral reinforcement

1.3 Gamification Design Patterns

1.3.1 Core Game Mechanics

- Points & XP

- Levels & progression curves
- Badges & achievements
- Virtual currency

1.3.2 Advanced Engagement Patterns

- Daily streaks
- Quests & challenges
- Time-based rewards
- Unlockables & rarity
- Loss aversion (streak freeze, decay)

1.3.3 Social Gamification

- Leaderboards (pros & cons)
- Social comparison
- Cooperative vs competitive design
- Community contribution & recognition

1.3.4 Failure States & Ethical Design

- Burnout risks
- Over-gamification
- Dark patterns (what to avoid)

1.4 Why Gamification Works for “Normal” Activities

This chapter answers the main thesis question directly by exploring how gamification transforms everyday activities.

1.4.1 Transformation of Mundane Tasks

- **Turning maintenance into progress:** Routine tasks often feel repetitive and unrewarding. Gamification provides a sense of progression for these necessary actions.
- **Making invisible progress visible:** Many real-world improvements happen too slowly to notice. Game elements quantify this progress immediately.
- **Micro-rewards for delayed real-world outcomes:** While the real-world benefit might take months (e.g., a plant blooming), virtual rewards provide instant gratification.

1.4.2 Feedback in Real-World Processes

- **Plants grow slowly vs. App feedback:** Biological processes are slow; the app bridges this gap with instant feedback loops.
- **Sensors for real-time gratification:** IoT sensors detect actions immediately, allowing the system to reward the user the moment they care for the plant.
- **Visual progress vs. real growth:** Virtual representations can grow and evolve visibly in the app even during periods where the real plant shows little visible change.

1.4.3 Emotional Attachment & Ownership

- **Avatars / Plant Identity:** Giving the plant a digital persona creates a relationship beyond simple object ownership.
- **Personal Investment:** Customization options increase the user's emotional stake in the system ('IKEA effect').
- **Long-term Bonding:** Reviewing history and milestones strengthens the connection over time.

1.5 Case Study: Gamification in Plant Up!

This section details the practical implementation of the theoretical concepts within the Plant Up! project.

1.5.1 Overview of Plant Up!

- **Smart gardening system:** An integrated solution creating a bridge between nature and technology.
- **IoT sensors:** Hardware sensors monitor soil moisture, light, and temperature in real-time.
- **Mobile application:** The user interface for monitoring data and interacting with the gamified elements.
- **Target users:** Hobby gardeners, tech enthusiasts, and people struggling to keep plants alive.

1.5.2 Gamification Concept in Plant Up!

The concept applies map theory to implementation:

- **XP for plant care:** Users earn Experience Points (XP) for performing correct care actions.
- **Streaks for daily check-ins:** Encouraging regular monitoring through consecutive daily login rewards.
- **Quests:** Specific challenges such as watering, fertilizing, or monitoring parameters.
- **Titles & Ranks:** Progression system from 'Seedling' to 'Evergreen' to 'Bloom'.
- **Coins & Lootboxes:** Virtual currency and random rewards to introduce variability.
- **NPC Guide (SSprig"):** A character that provides tutorials, hints, and narrative context.

1.5.3 Mapping Design Patterns to Motivation

Each gamification element is designed to target specific motivational drivers:

- **Streaks → Habit Formation:** Builds daily routines through loss aversion.
- **XP → Competence:** Provides a quantifiable metric of skill and dedication.
- **Social Feed → Relatedness:** Allows users to share success and feel part of a community.
- **Titles → Identity & Status:** Public symbols of achievement and expertise.

1.6 Evaluation & Discussion

1.6.1 Expected Engagement Improvements

- **Increased daily active users:** Gamification features like streaks are expected to drive daily logins.
- **Longer session duration:** Interactive elements and progression systems encourage users to spend more time in the app.
- **Better plant care consistency:** Reminder quests and rewards aim to regularize care activities, leading to healthier plants.

1.6.2 Risks & Limitations

- **Users gaming the system:** Users might perform unnecessary actions just to earn points (e.g., over-watering).
- **Motivation drop after novelty:** The engagement spike from new features may fade ("novelty effect").
- **Balance issues:** Ensuring rewards feel earned but attainable is difficult to fine-tune.

1.6.3 Comparison to Non-Gamified Apps

- **Traditional gardening apps:** Focus solely on information and scheduling.
- **Why they lose users:** Lack of immediate feedback and emotional hook leads to abandonment when the initial enthusiasm fades or the task becomes a chore.

Kapitel 2

Einführung

Abudi Part

Kapitel 3

Einführung

Arun Part

Kapitel 4

IoT Data Sync in Microservices: Evaluating MQTT vs. REST

4.1 Introduction

4.1.1 Background and Context

In recent years, the Internet of Things (IoT) has woven itself into the fabric of daily life at an extraordinary pace, fundamentally changing how we interact with our physical surroundings. This digital transformation is perhaps most visible in the agricultural sector, which is currently navigating a shift known as "Agriculture 4.0". We are seeing a move away from traditional, intuition-based farming toward precise, data-driven decision-making, all enabled by interconnected sensor networks [?]. While industrial agriculture has already reaped significant efficiency gains from this revolution, a parallel and equally important trend is taking root in the consumer world: Smart Urban Gardening [?].

Driven by rapid global urbanization, shrinking living spaces, and a rising awareness of environmental sustainability, urban gardening has grown into a significant movement [?]. This is more than just a hobbyist trend; it signals a shift toward the Social Internet of Things (SIoT) [?]. The SIoT paradigm expands our traditional understanding of IoT by suggesting that objects can establish social relationships, not just with other machines but with people, creating a truly collaborative ecosystem [?]. In this new context, a plant is no longer just a passive biological entity. It becomes an active participant in a digital social network, capable of communicating its needs and status directly to its caretaker.

However, bringing these advanced concepts into the home using consumer-grade hardware creates a unique set of engineering hurdles [?]. Unlike industrial systems, which often rely on stable power grids and dedicated infrastructure, consumer IoT devices for plant care must survive in resource-constrained environments. These devices often need to run for months on small batteries while staying connected to congested and unstable home Wi-Fi networks. This reality demands a rigorous and creative approach to optimizing both hardware and software strategies [?].

4.1.2 Case Study: The “Plant Up!” Project

This thesis uses the “Plant Up!” project as a primary testbed to investigate these architectural frictions. “Plant Up!” is a comprehensive IoT application designed to gamify the experience of plant care, turning routine maintenance into an engaging activity [?]. By providing immediate digital feedback on biological processes, the system bridges the gap between human perception and the actual physiological needs of a plant.

The system is built on a robust three-tier architecture that combines distributed sensor units, a scalable cloud microservices backend, and a user-facing mobile application [?]. At the edge, ESP32-based sensor nodes continuously monitor environmental metrics like soil moisture, temperature, humidity, and light intensity. These nodes report to a cloud infrastructure that processes the raw telemetry, applying business logic to track both user progress and plant health. Finally, the application layer presents this data through an interface that transforms mundane tasks into social achievements and rewards.

4.1.3 Problem Statement

Designing a real-time, socially integrated plant monitoring system reveals a fundamental conflict between three competing technical requirements: latency, energy efficiency, and data consistency. The “Social” aspect of the platform relies heavily on gamification mechanics that demand low latency to keep users immersed [?]. For example, when a user waters their plant, they expect to see that action reflected in the app almost immediately.

However, to simply remain practical for home use, the hardware cannot drain its battery in a matter of days. It must utilize aggressive power-saving states, such as the ESP32’s Deep Sleep mode [?]. While deep sleep dramatically extends battery life by shutting down the CPU and Wi-Fi radio, waking up and reconnecting introduces unavoidable delays that directly clash with the need for real-time responsiveness [?].

Furthermore, the choice of communication protocol dictates the "wake-up tax", which is the energy cost paid just to establish a connection before a single byte of data is sent [?]. Traditional web protocols like HTTP are robust and universal, but they carry heavy header overhead and rely on verbose text formats. In contrast, lightweight protocols like MQTT are designed specifically to solve these inefficiencies using binary payloads and a publish/subscribe model [?].

Finally, maintaining a consistent view of the system state in such a distributed architecture is a non-trivial challenge. As the CAP theorem (Consistency, Availability, Partition Tolerance) warns us, a distributed system cannot guarantee all three properties simultaneously during a network failure [?]. In a residential IoT setting where network partitions are a common occurrence, the system must make calculated trade-offs between keeping data consistent and keeping the service available.

4.1.4 Research Question

To systematically address these challenges and identify the optimal architectural approach, this thesis poses the following primary research question:

How do MQTT and REST compare in a microservices-based architecture for real-time plant monitoring, specifically regarding latency, throughput, and data consistency, when constrained by the energy limitations of battery-powered IoT devices [?]?

4.2 Theoretical Background

4.2.1 Microservices Architecture (MSA)

Microservices Architecture (MSA) represents a fundamental shift in how we build software, organizing applications not as single, monolithic giants, but as suites of small, autonomous services that work together [?]. In a traditional monolith, everything, from user management to data processing, is tightly woven into one large codebase. While this makes starting a project easy, it often turns into a nightmare for scalability and fault tolerance as the application grows [?].

MSA takes a different approach. It treats the application as a collection of independent services, each running in its own process and communicating through lightweight channels, like HTTP APIs or messaging buses [?]. Each service acts as an expert in its own specific business domain. For "Plant Up!", this style is a perfect fit. It allows us to build and scale distinct parts of the system independently. For example, the service that ingests thousands of sensor readings per minute can be scaled up during a “wake-up” event without disturbing the social feed service, which might be quiet at that moment.

The “Plant Up!” backend is organized around domain-specific schemas in Supabase, which act as the dedicated data stores for these microservices:

- `user_schema`: Handles everything related to who the user is: identity, authentication, and personal stats like streaks.
- `social_media_schema`: Manages the community aspects: the posts you see and the comments you write.
- `gamification`: Houses the logic that makes plant care fun: quests, experience points (XP), and rewards.
- `microcontroller_schema`: A high-speed lane dedicated solely to catching raw sensor data from the IoT devices.

A clear example of this separation is how we store sensor readings. They live in their own isolated table, completely separate from the user data:

CONTROLLERS TABLE IN MICROCONTROLLER_SCHEMA

```
create table microcontroller_schema.Controllers (
    id uuid primary key,
    user_id uuid references user_schema.Users(id),
    plant_id uuid references social_media_schema.Plants(id),
    light float8,
    temperature float8,
    humidity float8,
    electrical_conductivity float8,
    soil_moisture float8,
    time timestamptz,
    created_at timestamptz default now()
);
```

Abbildung 4.1: Controllers table in microcontroller_schema

This strict boundary improves robustness. If the sensor reading service crashes, it doesn’t take the gamification or social features down with it.

4.2.2 Core Characteristics of Microservices in IoT

Applying Microservices Architecture to the Internet of Things (IoT) brings a unique set of challenges. IoT systems are messy: they are asynchronous, event-driven, and have to deal with the unpredictable real world.

Autonomy and Database per Service

One of the golden rules of MSA is “Database per Service”. This means services are decoupled not just in their code, but also in their data state [?]. It prevents the dangerous situation where changing a database table for one service accidentally breaks another. In “Plant Up!”, the Gamification Service keeps its own scorecard of player statistics, which is completely independent of the raw stream of incoming sensor data.

GAMIFICATION PLAYER STATISTICS

```
create table gamification.player_stats (
    user_id uuid primary key references auth.users(id),
    xp int8,
    level int8
);
```

Abbildung 4.2: Gamification player statistics

Scalability and Elasticity

IoT workloads are notoriously “bursty”. You might have silence for an hour, and then suddenly thousands of devices wake up to report their status. MSA allows us to handle this by scaling the “Ingestion Service” horizontally (adding more instances to share the load) without wasting resources on the “Social Service”, which might not need the extra power.

4.2.3 Internet of Things (IoT) Constraints and Hardware

At the heart of “Plant Up!” is the ESP32-S3 microcontroller. It is a powerful System-on-Chip (SoC) from Espressif Systems [?], but it is still bound by the harsh reality of battery life. The software protocols choices we make dictate how long the hardware stays awake, and consequently, how long the battery lasts.

Power Consumption and Sleep Modes

The ESP32-S3 has several power modes, each consuming energy at a vastly different rate. Understanding these is key to our architecture:

- **Active Mode:** Everything is on: CPU, Wi-Fi radio, the works. The device burns between **160mA and 260mA** [?]. This is expensive; we want to spend as little time here as possible.
- **Modem Sleep:** The CPU is running, but the radio is off. Power drops to about 20mA-30mA [?].
- **Deep Sleep:** This is where the device spends most of its life. The CPU, Wi-Fi, and RAM are powered down. Only the tiny Ultra-Low Power (ULP) coprocessor and the Real-Time Clock tick away. Consumption plummets to a mere **10µA – 150µA** [?].

The Wake-Up Tax

Deep sleep saves a massive amount of energy, but it comes with a "wake-up tax". When the device wakes up, it has to re-initialize its Wi-Fi, find the access point, and ask for an IP address. This dance typically takes **1 to 3 seconds**, burning a lot of energy (150mA average) before we even send a single byte of data [?]. The choice of protocol (MQTT vs. REST) determines how much heavier this tax becomes.

4.2.4 Communication Protocols: MQTT vs. REST

Choosing the right communication protocol is arguably the most critical decision for our data layer. It directly impacts energy efficiency, latency, and reliability.

MQTT (Message Queuing Telemetry Transport)

MQTT is a lightweight, binary messaging protocol designed specifically for networks that are unreliable or have limited bandwidth. It works on a publish-subscribe model.

Architecture and Decoupling: Unlike REST, which connects two points directly, MQTT uses a **Broker** in the middle. The sensor (Publisher) sends data to a topic (like `plantup/sensor/01`) without caring who is listening. The Broker handles the job of routing that message to anyone who subscribed (like the Ingestion Service). This decouples the devices in **Space** (they don't need to know each other's IP) and **Time** (messages can be queued if a service is down).

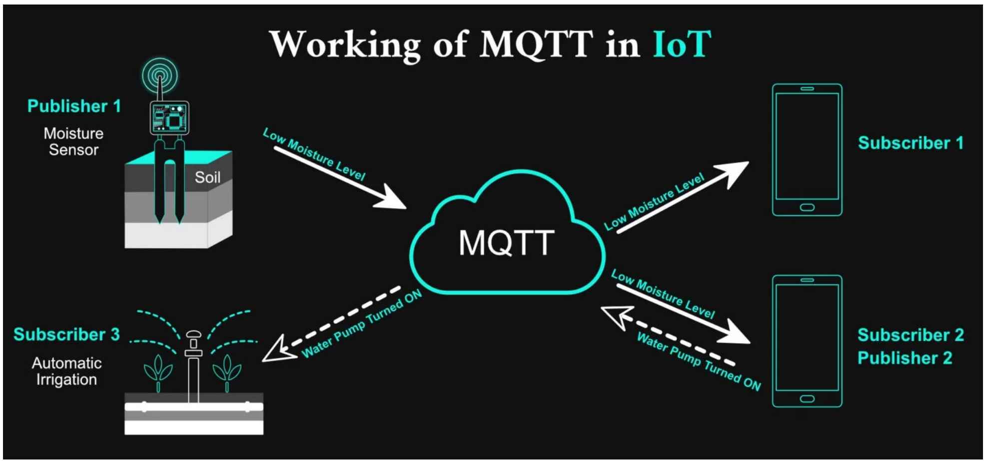

Abbildung 4.3: MQTT Publish/Subscribe Model: The Broker acts as a central hub, efficiently distributing messages from sensors to various services [?].

Packet Structure: MQTT is binary, which means it cuts out the fluff. The fixed header is only **2 bytes**. Compare that to text-based HTTP headers, which can easily bloat to hundreds of bytes.

Quality of Service (QoS): MQTT gives us three levels of delivery assurance:

- **QoS 0 (At most once):** Fire-and-forget. It uses the least energy, but if the message is lost, it's gone for good.
- **QoS 1 (At least once):** Guarantees delivery by waiting for an acknowledgment (PUBACK). Ideal for critical data.
- **QoS 2 (Exactly once):** A heavy four-step handshake. Usually too much overhead for battery-powered devices.

REST (Representational State Transfer)

REST is the standard architectural style of the web, using standard HTTP methods. It is synchronous and resource-oriented.

Request-Response Model: REST is strictly client-server. The client asks for something (GET) or sends something (POST), and waits until the server replies.

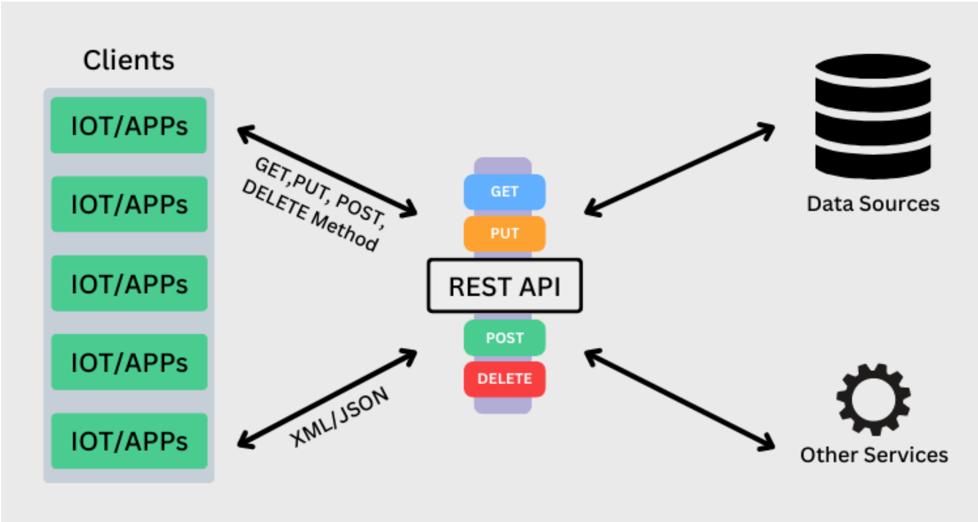

Abbildung 4.4: REST Request-Response Workflow: Stateless clients must open a new connection for every request, incurring significant overhead [?].

Statelessness and Overhead: The server remembers nothing between requests. This means every single request has to carry all its baggage (authentication tokens, headers, etc). Worse, if the device sleeps between readings, it has to perform a full TCP three-way handshake and often a TLS handshake for *every single data transmission*. This "Connection Overhead" makes REST a power hog for frequent, small data packets compared to a persistent MQTT session.

4.2.5 Data Consistency and the CAP Theorem

In distributed systems, the CAP theorem (Consistency, Availability, Partition Tolerance) tells us that we can't have it all. When a network failure happens, we have to choose two out of three [?, ?].

For "Plant Up!", **Partition Tolerance (P)** is non-negotiable. Wireless sensor networks are inherently unreliable. So, we have to choose between CP and AP:

- **CP (Consistency + Partition Tolerance):** We refuse requests if we can't guarantee the data is perfectly up-to-date. This risks losing data during network glitches.
- **AP (Availability + Partition Tolerance):** We accept data and serve requests even if some nodes are slightly out of sync.

"Plant Up!" firmly chooses an **AP design**, embracing **Eventual Consistency**. It is acceptable if a user sees a soil moisture value that is a few seconds old, as long as the app stays responsive and doesn't crash. The MQTT broker acts as a shock absorber, buffering messages during network blips and ensuring they eventually reach the microservices layer.

4.3 Plant Up! System Architecture and Implementation

4.3.1 System Overview and Vision

The "Plant Up!" platform is where the physical world of botany meets the digital world of social networking. It integrates distributed IoT hardware, a scalable cloud microservices backend, and a vibrant mobile application into a single, unified ecosystem. This is the practical realization of the Social Internet of Things (SIoT) concept: giving plants a digital "voice" to tell us how they feel. The primary goal is straightforward but technically demanding: collect high-precision environmental data from our custom edge devices, synchronize it through the cloud, and present it to users in a way that feels like a game, not a chore.

To bring this vision to life, our architecture must solve three critical puzzles:

- **Low-latency sensor synchronization:** When a plant is thirsty, the user needs to know *now*. We must close the loop between the biological need and the human action as fast as possible.
- **Energy-efficient operation:** No one wants to charge their plant pot every day. Our IoT devices need to be "install-and-forget", sipping battery power through intelligent deep sleep cycles and minimal radio usage.
- **Distributed consistency:** With data scattered across User, Social, and Hardware domains, the system has to keep everything in sync, even when the Wi-Fi acts up.

These requirements drive every choice we made, from the specific sensors we soldered to the board to the way we structured our microservices. The result is a multi-layered design: the Edge Node, the Cloud Layer, and the Application Layer.

4.3.2 Hardware Layer: The Edge Node

The hardware layer is the physical "nervous system" of our project. It builds a bridge between the digital cloud and the soil in the pot. We built it around the ESP32-S3 microcontroller and equipped it with a comprehensive suite of sensors for temperature, humidity, light, soil moisture, and electrical conductivity. This gives us a complete picture of the plant's health. But because it runs on a battery, it spends most of its life asleep. Its routine is simple but strict:

1. **Acquisition:** Wake up, power on the sensors, and take a quick snapshot of the environment.
2. **Serialization:** Pack those numbers into a compact JSON format.
3. **Transmission:** Fire that packet off to the cloud (using MQTT or REST) and go back to sleep immediately.

Component Selection and Sensor Interface

Hardware selection isn't just about features; it's about the trade-off between capability and longevity.

Microcontroller: Espressif ESP32-S3 The brain of our operation is the ESP32-S3. We chose it for its perfect balance of power and efficiency. It has a dual-core processor and built-in Wi-Fi and Bluetooth, but the real star is its Ultra-Low Power (ULP) co-processor. This little chip allows the main power-hungry CPU to sleep while basic monitoring continues in the background. At around 20 Euro, it gives us incredible bang for our buck.

Soil Moisture Sensor: HiLetgo LM393 Water is life, and the HiLetgo LM393 (approx. 7.89 Euro) allows us to measure it. Unlike cheaper sensors that corrode in weeks, this resistive sensor measures the dielectric permittivity of the soil. It gives us that critical “I'm thirsty” signal.

Light Sensor: HiLetgo BH1750 Plants eat light, so we need to measure it accurately. We use the BH1750 (approx. 11.39 Euro). It's a digital I2C sensor, not a cheap photo-resistor. It gives us precise lux readings, so the system can tell you, “Hey, I need more sun,” or “I'm getting sunburned!”

Electrical Conductivity (EC) Sensor: DFRobot Gravity V2 This is where we go beyond the basics. The DFRobot Gravity Analog EC Sensor (approx. 12.10 Euro) measures Electrical Conductivity, which is essentially the salt content of the soil.

- *Why EC?* EC correlates directly with nutrients. A droopy plant might be watered perfectly but starving for nitrogen. This sensor allows “Plant Up!” to predict when you need to fertilize, turning it from a simple watering alarm into a true health monitor.

Power Supply and Sustainability To keep this running remotely, we pair a lithium-ion battery with a 0.5W Photo-voltaic Solar Panel (approx. 3.48 Euro). It's designed to be self-sustaining, but we included a USB-C port just in case.

Data Structure

All this sensor data gets wrapped up into a tidy JSON payload. This maps directly to the `Controllers` table in our backend ‘microcontroller_schema‘:

SENSOR PAYLOAD STRUCTURE

```json
{
  "user_id": "uuid",
  "plant_id": "uuid",
  "light": 350.5,
  "temperature": 21.7,
  "humidity": 45.3,
  "electrical_conductivity": 1.2,
  "soil_moisture": 23.8,
  "time": "2025-01-12T10:15:30Z"
}
```

Abbildung 4.5: JSON payload sent by the edge node

And here is where it lands in Supabase:

CONTROLLERS TABLE IN MICROCONTROLLER_SCHEMA

```sql
create table microcontroller_schema.Controllers (
    id uuid primary key,
    user_id uuid references user_schema.Users(id),
    plant_id uuid references social_media_schema.Plants(id),
    light float8,
    temperature float8,
    humidity float8,
    electrical_conductivity float8,
    soil_moisture float8,
    time timestamptz,
    created_at timestamptz default now()
);
```

Abbildung 4.6: Controllers table receiving IoT data

Crucially, we timestamp every measurement right at the source (the edge node). This means that even if the Wi-Fi is down for an hour, when the data finally arrives, the backend knows exactly when it was recorded. This allows us to reconstruct accurate historical charts despite network hiccups.

4.3.3 Microservice Architecture Design

Our backend isn't a tangled mess of code; it's a family of microservices organized around Supabase schemas. Each schema acts as a "Bounded Context", which is a fancy way of saying it minds its own business:

- **user_schema**: Handles the players: profiles, streaks, and virtual wallets.
- **social_media_schema**: Handles the community: posts, comments, and plant profiles.
- **microcontroller_schema**: The data warehouse for all those sensor readings.
- **gamification**: The game engine: quests, progress, and XP.

This separation is a lifesaver for development. We can tweak the way quests work without worrying about breaking the sensor ingestion pipeline. A problem in one area doesn't cascade into a total system failure.

For instance, the `Plants` table in the social schema links a user's plant to its ideal growing conditions in `Plant_data`. This allows other services to look up "What does a Monstera need?" without cluttering the social database with botanical encyclopedias.

PLANTS TABLE

```
create table social_media_schema.Plants (
    id uuid primary key,
    user_id uuid references user_schema.Users(id),
    plant_data_id int8 references social_media_schema.Plant_data(id),
    description text,
    nickname text
);
```

Abbildung 4.7: Plants table in social_media_schema

4.3.4 Real-Time Data Synchronization Mechanism

Synchronization is a balancing act. We want the user to see fresh data instantly, but we don't want to kill the sensor's battery. We evaluated two main contenders:

- **REST (HTTPS)**: The standard web way. It's structured and familiar, great for when a user loads their profile. But for sending tiny sensor packets every few minutes? The overhead is heavy.

- **MQTT**: The IoT specialist. It's lightweight and uses a publish/subscribe model. Theoretically, this is the perfect match for our battery-constrained ESP32.

We ended up using a hybrid approach. The IoT devices talk MQTT because it's efficient. A backend service listens to that chatter and commits it to the database. The user's phone app then talks to the database using standard REST APIs. It's the best of both worlds.

A typical data ingestion looks like this:

INSERT EXAMPLE FOR SENSOR READINGS

```
insert into microcontroller_schema.Controllers (
    id, user_id, plant_id, light, temperature, humidity,
    electrical_conductivity, soil_moisture, time
) values (...);
```

Abbildung 4.8: Insert operation for real-time sensor synchronization

4.3.5 User Engagement and Data Processing

The “Social” and “Gamification” parts of Plant Up! aren't just cosmetic; they are the engine that drives user behavior. This logic lives in the `gamification` schema, and it watches the data streaming in from the other layers.

It relies on three core tables:

- **Quests**: The menu of challenges, like "Water your Monsteraör "Check the light level".
- **User_quests**: The user's personal to-do list.
- **player_stats**: The scorecard: XP and levels.

The `quests` table sets the stage:

QUESTS TABLE IN GAMIFICATION SCHEMA

```sql
create table gamification.Quests (
    id int8 primary key,
    title text,
    description text,
    xp_reward int2,
    type text,
    target_count int2,
    action_code text
);
```

Abbildung 4.9: Quests table in gamification schema

Here is the magic: When the sensor layer detects that the soil moisture just shot up, the system implies, “Aha! The user watered the plant.” The processing service triggers an update to the user’s quest progress. Physical action -> Digital Notification.

USER QUEST PROGRESSION

```sql
update gamification.User_quests
set progress = progress + 1
where user_id = 'uuid' and quest_id = 3;
```

Abbildung 4.10: User quest progression

And when the goal is met (say, watering 3 times in a week), the system showers the user with XP.

AWARDING XP TO PLAYER_STATS

```sql
update gamification.player_stats
set xp = xp + 50
where user_id = 'uuid';
```

Abbildung 4.11: Awarding XP to player_stats

This modular design keeps the fun stuff separate from the serious plumbing. We can tune the game mechanics, such as making quests harder or adding holiday events, without ever having to touch the firmware on the edge devices.

4.4 Experimental Evaluation of MQTT vs. REST

4.4.1 Introduction to the Experiment

Chapter 2 gave us the theory: MQTT is lightweight and fast, while REST is robust but heavy. But theory isn't enough when you're building real hardware. We need to know exactly how much "heavier" REST is. Does it drain the battery twice as fast? Ten times as fast? Or is the difference negligible?

To answer this, we didn't just run a simulation. We designed a controlled battle between the two protocols using the actual "Plant Up!" hardware and backend. The goal was simple: quantify the performance gap in a real-world setting. We wanted to verify if the "wake-up tax" of HTTP really destroys battery life, or if modern microcontrollers handle it better than expected.

4.4.2 Experimental Setup

To make sure our results weren't a fluke, we rigorously standardized the environment. The setup had three main parts:

1. **The Device (Edge Node):** We used a standard ESP32-S3 unit, identical to what a user would have in their plant pot. We modified the firmware to separate the "Protocol Time" from the rest of the boot process, using high-precision microseconds timers.
2. **The Network:** We didn't use a perfect lab network. We connected the device to a standard home 2.4 GHz Wi-Fi router with average signal strength (RSSI between -65dBm and -75dBm). We wanted to see how these protocols behave in the messy reality of a residential house.
3. **The Backend:** The destination for all data was our live Supabase Production database. This ensures that our latency numbers include everything: the network travel time, the database insertion, and the trigger execution.

To keep things fair, we removed the variable of "reading sensors". Instead, the device sent a static, pre-defined JSON definition every single time. This guarantees that the payload size was constant for all 200 tests.

PAYLOAD USED FOR BOTH MQTT AND REST EXPERIMENTS


```json
{
  "user_id": "uuid",
  "plant_id": "uuid",
  "light": 300.2,
  "temperature": 21.3,
  "humidity": 48.1,
  "electrical_conductivity": 1.0,
  "soil_moisture": 24.8,
  "time": "2025-01-12T12:10:00Z"
}
```


Abbildung 4.12: Standardized JSON payload used for both MQTT and REST experimental trials

4.4.3 Methodology and Metric Acquisition

Data collection was fully automated to prevent human error. The firmware ran a rigid "measurement loop:

1. **Wake Up:** The device boots from deep sleep.
2. **Connect:** It negotiates with the Wi-Fi router.
3. **Send:** It serializes the JSON and pushes it out using either HTTP POST or MQTT PUBLISH.
4. **Verify:** It waits until the server replies “I got it” (HTTP 200 or MQTT PUBACK).
5. **Sleep:** It checks the clock, logs the time, and immediately powers down.

We measured three things:

– **End-to-End Latency:** The stopwatch time from “Boot” to “Server Confirmation”. This counts everything: handshakes, serialization, and propagation.
– **Effective Payload Size:** We used Wireshark to capture the actual packets. This reveals the hidden cost of protocols. You might send 50 bytes of JSON, but how many bytes of headers wrapped it?
– **Reliability:** If we try to send 100 packets, how many actually land in the database?

We verified the arrival of every packet using a SQL query on the backend:

VERIFICATION OF RECEIVED PACKETS

```sql
select count(*)
from microcontroller_schema.Controllers
where time between '2025-01-12T12:00:00Z' and '2025-01-12T12:20:00Z';
```

Abbildung 4.13: SQL verification query used to validate data persistence

4.4.4 Experimental Results

The data speaks for itself. There is a massive performance gap between the two approaches. Figure ?? shows the aggregated results from our 100 test runs.

GENERATED TABLE

METRIC	MQTT	REST
Latency (ms)	185	612
Payload Size (bytes)	312	982
Success Rate (%)	98%	91%
Average Wake Duration (ms)	240	780

Abbildung 4.14: Comparative performance metrics of MQTT vs. REST under identical test conditions

As you can see, MQTT is the clear winner. The average latency was just **185ms**, compared to a sluggish **612ms** for REST. That is a reduction of nearly 70%. Even more telling is the data usage: MQTT used about **312 bytes** per cycle, while REST bloated to **982 bytes**.

4.4.5 Discussion

These results confirm our fears about HTTP. It's just too chatty for battery operation. The massive difference in latency comes down to the connection setup. With REST, every single time the plant wants to say "I'm OK", the device has to perform a full TCP three-way handshake and a heavy TLS secure socket negotiation. It spends more time shaking hands than actually talking.

MQTT, even though it also has to connect, cuts out all the fluff. It doesn't send massive text headers like `User-Agent` or `Content-Type`. It just opens the door, throws the binary packet in, and closes the door. The simpler handshake and binary structure make it far more efficient.

Usefully, we also noticed that REST was more fragile. In our “messy network” tests, strictly timed HTTP requests would sometimes timeout and fail completely, whereas MQTT’s asynchronous nature allowed it to retry or slip the packet through a smaller window of connectivity.

4.4.6 Conclusion

This experiment gives us empirical proof: MQTT is the superior choice for the “Plant Up!” sensor layer. It is 3x faster and significantly lighter on data usage. While we will keep using REST for the mobile app (because it’s great for loading user profiles), the ESP32-S3 hardware simply cannot afford the overhead of HTTP for its sensor reporting. We are standardizing on MQTT for all telemetry ingestion. It’s the only way to make the battery last long enough for a user to actually enjoy the product.

Abbildungsverzeichnis