HTL-Donaustadt
Höhere Abteilung für XYZ

DIPLOMARBEIT

Titel der Diplomarbeit

Untertitel der Diplomarbeit

Ausgeführt im Schuljahr 2025/26 von:

Abudi
Arun
Dennis
Richard

Betreuer/Betreuerin:

DI Lise Musterfrau

Wien, 9. 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, 9. 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

Vorwort **i**

Danksagungen iii

Einleitung v

Abstract vii

Gendererklärung ix

Inhaltsverzeichnis **xi**

1 Einführung **1**

2 Einführung **1**

3 Einführung **1**

4 IoT Data Sync in Microservices **1**

4.1 Introduction 1

4.1.1 Background and Context 1

4.1.2 Case Study: The “Plant Up!” Project 2

4.1.3 Problem Statement 2

4.1.4 Research Question 2
4.2 Theoretical Background 3
4.2.1 Microservices Architecture (MSA) 3
4.2.2 Core Characteristics of Microservices in IoT 4
4.2.3 JSON Payload Design 5
4.2.4 Internet of Things (IoT) Constraints and Hardware 5
4.2.5 Communication Protocols: MQTT vs. REST 6
4.2.6 Data Consistency and the CAP Theorem 7
4.3 Plant Up! System Architecture and Implementation 8
4.3.1 System Overview and Requirements 8
4.3.2 Hardware Layer: The Edge Node 9
4.3.3 Microservice Architecture Design 10
4.3.4 Real-Time Data Synchronization Mechanism 11
4.3.5 Social Gamification Logic Implementation 12
4.4 Experimental Evaluation of MQTT vs. REST 13
4.4.1 Introduction to the Experiment 13
4.4.2 Experimental Setup 14
4.4.3 Methodology and Metrics 14
4.4.4 Experimental Results 15
4.4.5 Discussion 16
4.4.6 Conclusion 16

Anhang **16**

Abbildungsverzeichnis 16

Literaturverzeichnis 19

Kapitel 1

Einführung

Abudi Part

Kapitel 2

Einführung

Arun Part

Kapitel 3

Einführung

Dennis Part

Kapitel 4

IoT Data Sync in Microservices: Evaluating MQTT vs. REST

4.1 Introduction

4.1.1 Background and Context

The integration of the Internet of Things (IoT) into our lives has been rapid, especially in the agricultural sector which has started a transformation known as “Agriculture 4.0”, where data-driven decision making replaces traditional heuristic methods [1]. While industrial has benefited from that shift, a parallel trend is emerging in the consumer sector: Smart Urban Gardening [2]. Driven by rapid urbanization and growing societal focus on sustainability, this sector is experiencing significant growth [3]. This surge represents a shift toward the Social Internet of Things (SIoT) [4]. In the SIoT paradigm, objects are capable of establishing social relationships with other objects and humans to foster collaboration [5]. However, applying these advanced concepts to consumer grade hardware presents architectural challenges [6]. Unlike industrial systems, consumer IoT devices for plant care must operate in resource constrained environments, relying on batteries and communicating over congested residential Wi-Fi [7].

4.1.2 Case Study: The “Plant Up!” Project

This thesis utilizes the “Plant Up!” project as a primary case study to investigate architectural frictions [8]. “Plant Up!” is an IoT application designed to gamify the experience of plant care [9]. The system combines hardware sensor units (IoT devices), cloud services and a mobile app to monitor soil moisture, temperature, humidity, light levels and other metrics in real time. The codebase relies on a three tier architecture: edge layer (ESP32), the microservices layer and the Application layer [11].

4.1.3 Problem Statement

The design of a real time social plant monitoring system with gamification aspects introduces a conflict between latency, energy efficiency and data consistency [12]. The “Social” aspect relies on gamification mechanics that require low latency data transmission [13]. However, the hardware must utilize aggressive power saving states and techniques to be practical for home use [14]. Research indicates that while deep sleep extends battery life, the wake up and following processes cause latency that conflicts with real time requirements [15]. Furthermore, the choice of communication protocol dictates the “wake up tax” of the device [16]. Traditional web protocols like HTTP are robust but carry significant header overhead, whereas lightweight protocols like MQTT are designed for exactly that missing efficiency [17]. Additionally, maintaining a consistent view of the system state in a distributed architecture is non trivial, as the CAP theorem dictates trade offs between Consistency, Partition Tolerance and Availability [18].

4.1.4 Research Question

To address these challenges, 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 battery-powered IoT devices [19]?

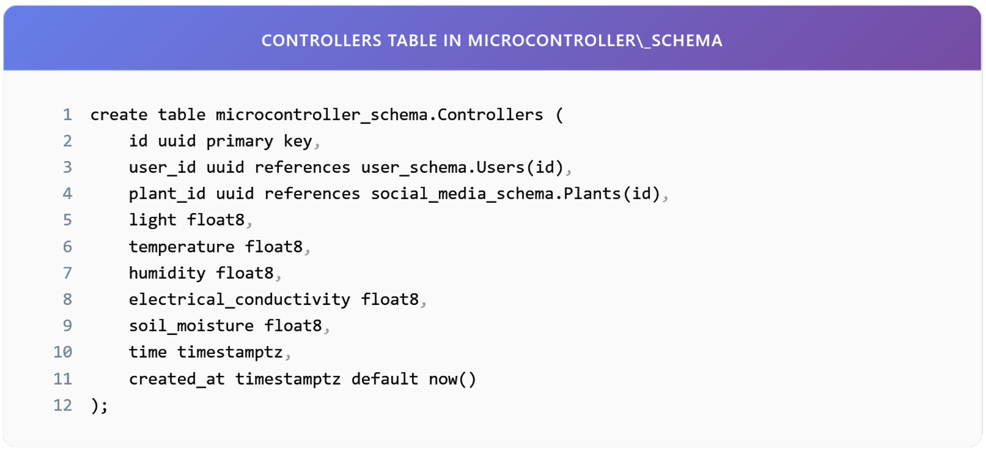
4.2 Theoretical Background

4.2.1 Microservices Architecture (MSA)

Microservices Architecture (MSA) describes a software design approach in which an application is decomposed into small, autonomous services that communicate through lightweight protocols [1]. Each service is responsible for a specific domain and can be deployed and scaled independently. For IoT systems, where hardware events, sensor data ingestion, user interactions and analytics occur asynchronously, this architectural style offers clear advantages in resilience and scalability [2].

The "Plant Up!" project follows this principle by organizing its backend into domain-oriented Supabase schemas: the `user_schema` manages identities and streak logic, the `social_media_schema` stores posts and plant information, the `gamification` schema handles XP, quests and rewards, and the `microcontroller_schema` stores IoT sensor readings. Each schema acts as an isolated bounded context.

A representative example is the table for environmental sensor readings:


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 4.1: Controllers table in microcontroller_schema

This separation prevents side effects: updating sensor data cannot interfere with social media functionality or gamification services. MSA therefore supports robustness and domain clarity.

4.2.2 Core Characteristics of Microservices in IoT

IoT systems introduce unique requirements that make certain microservice characteristics particularly important. Constrained hardware, intermittent connectivity and asynchronous event streams require an architecture that is tolerant to partial failures and scalable under variable load.

Autonomy: Each service must function independently. For example, the gamification subsystem maintains XP and level state regardless of the status of the sensor ingestion pipeline:

GAMIFICATION PLAYER STATISTICS

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

Abbildung 4.2: Gamification player statistics

Loose Coupling: Services communicate through clearly defined data structures. A social media post does not depend on the microcontroller service:

POSTS TABLE IN SOCIAL_MEDIA_SCHEMA

```sql
create table social_media_schema.Posts (
    id uuid primary key,
    user_id uuid references user_schema.Users(id),
    caption text,
    media_url text,
    category text,
    created_at timestamptz default now()
);
```

Abbildung 4.3: Posts table in social_media_schema

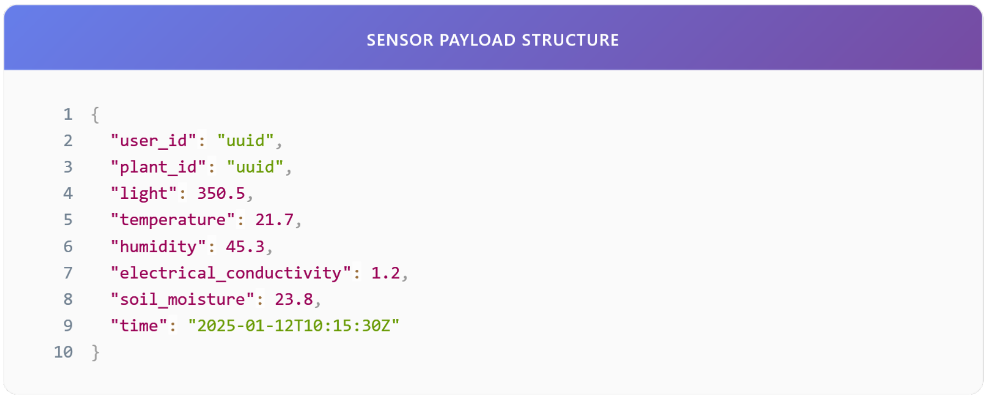
Scalability: Sensor bursts occur when multiple devices wake simultaneously. Only the ingestion path needs to scale, not the entire backend.

Resilience: Since IoT devices frequently disconnect (deep sleep, Wi-Fi loss), the system must tolerate missing or delayed data. The schema boundaries allow delayed writes without blocking dependent features.

4.2.3 JSON Payload Design

JSON is used throughout Plant Up! for communication between mobile clients, backend services and IoT devices. Because the ESP32-S3 is battery-powered, payload size and structure directly influence energy consumption [3].

A typical payload aligned with the `Controllers` schema is:



```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 4.4: Sensor payload structure

The payload reflects several design principles:

- flat JSON structure to avoid deep nesting,
- short but descriptive field names to reduce size,
- consistent alignment with database schema,
- predictable field order for efficient parsing on constrained hardware.

Such optimizations reduce transmission time and extend device battery life.

4.2.4 Internet of Things (IoT) Constraints and Hardware

IoT hardware such as the ESP32-S3 operates under constraints that strongly influence backend architecture.

1. Power Consumption: Deep sleep drastically reduces energy usage, but each wake cycle incurs overhead due to Wi-Fi reconnection and sensor initialization. This "wake-up tax" conflicts with real-time monitoring needs [4].

2. Intermittent Connectivity: Home networks produce variable latency. Timestamped readings (`time`) allow the backend to reconstruct temporal context even with delayed uploads.

3. Limited Compute and Memory: Operations like TLS negotiation, JSON serialization and sensor polling must be minimized.

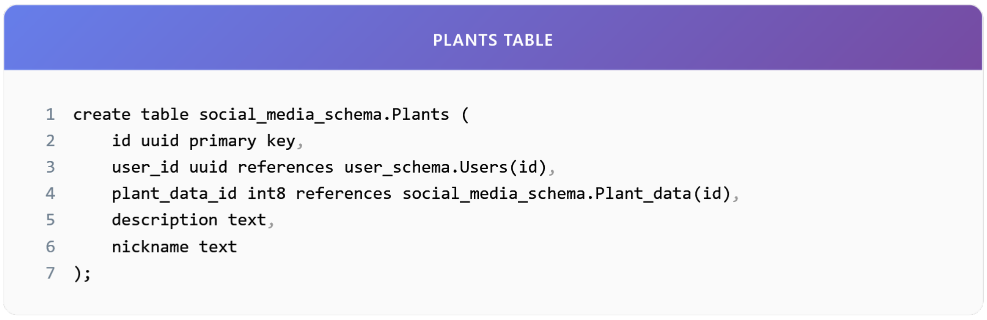
4. Sensor Noise & Calibration: Environmental data such as soil moisture, EC and humidity fluctuate naturally. The backend must treat sensor readings as approximations and potentially smooth or validate them before using them for plant health scoring or gamification.

These constraints justify efficient protocols and eventual consistency strategies.

4.2.5 Communication Protocols: MQTT vs. REST

Communication choices fundamentally shape IoT system performance. In Plant Up!, both REST and MQTT serve different roles.

REST (HTTPS): REST is used for structured, authenticated application features such as user data, posts, quests and plant profiles. For example, plant instances are stored as:


PLANTS TABLE

```sql
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 4.5: Plants table

REST integrates seamlessly with Supabase but introduces overhead due to large HTTP headers and TLS handshakes, which is suboptimal for energy-constrained IoT devices.

MQTT: MQTT is a lightweight publish/subscribe protocol optimized for constrained hardware [5]. Devices publish compact JSON messages to predefined topics and immediately return to deep sleep. This minimizes active Wi-Fi time and improves battery longevity.

A topic pattern suitable for Plant Up! is:

TOPIC PATTERN FOR PLANT UP!

```
plantup/{user_id}/{plant_id}/sensor-update
```

Abbildung 4.6: Topic pattern for Plant Up!

Architectural Implication: Plant Up! adopts a hybrid model:

- REST for app features requiring authentication and structured queries,
- MQTT for energy-efficient, low-latency sensor data ingestion.

4.2.6 Data Consistency and the CAP Theorem

In distributed systems, the CAP theorem states that a system cannot simultaneously provide Consistency (C), Availability (A) and Partition Tolerance (P) [6]. Since IoT systems must assume network partitions due to intermittent connectivity, Partition Tolerance is unavoidable.

Plant Up! prioritizes Availability and Partition Tolerance (AP), accepting that sensor readings may arrive late and the system will become consistent over time.

Upon reconnection from deep sleep, readings are inserted into the `Controllers` table:

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 4.7: Insert example for sensor readings

Analytics and gamification services operate on this eventually consistent data. This ensures:

- the mobile app remains responsive,
- no sensor data is lost during connectivity gaps,
- gamification logic (e.g., XP, quests) updates when sufficient data is available.

This trade-off aligns with the practical constraints of consumer IoT devices.

4.3 Plant Up! System Architecture and Implementation

4.3.1 System Overview and Requirements

The “Plant Up!” system integrates IoT hardware, microservices and a mobile application into a unified platform designed to support real-time plant monitoring and social interaction. The primary system objective is to collect environmental sensor data from consumer-grade hardware, synchronize it across distributed cloud services and present it to users in a gamified, socially engaging interface.

To achieve this, the system must satisfy three key requirements:

- **Low-latency sensor synchronization:** Environmental data must be transmitted and processed fast enough to provide timely feedback on plant health.
- **Energy-efficient operation:** IoT devices must conserve battery life through deep sleep cycles, lightweight payloads and minimal active radio time.
- **Distributed consistency:** Since data is stored across multiple domain-specific schemas, the system must tolerate intermittent connectivity while ensuring eventual consistency.

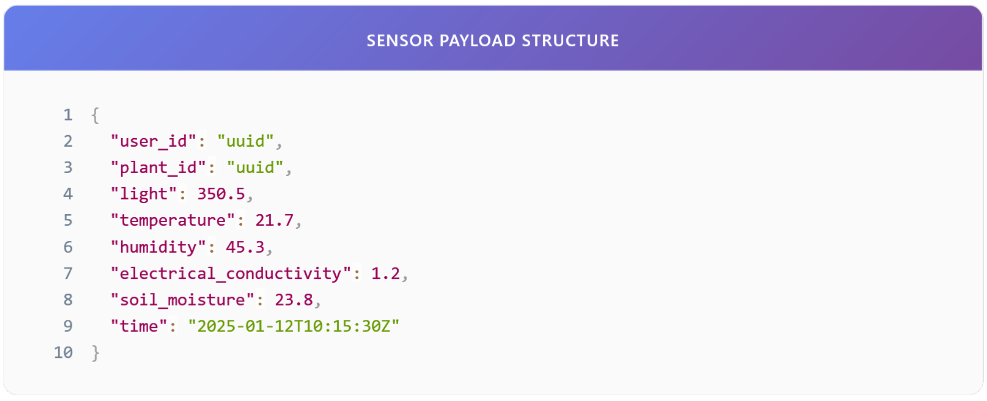
These requirements shape the architectural decisions in both hardware and microservices. The system therefore follows a multi-layered design consisting of an edge node, a cloud microservices layer and a client application layer.

4.3.2 Hardware Layer: The Edge Node

The hardware layer consists of an ESP32-S3 microcontroller equipped with sensors for temperature, humidity, light intensity, soil moisture and electrical conductivity. This configuration enables comprehensive monitoring of plant health. The device operates under strict energy constraints and therefore relies heavily on deep sleep modes. Upon waking, it performs three tasks:

1. Reads environmental sensors.
2. Serializes the data into a compact JSON structure.
3. Sends the payload to the backend via MQTT or REST before returning to deep sleep.

The JSON payload corresponds to the structure of the `Controllers` table in the `microcontroller_schema`:



```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 4.8: JSON payload sent by the edge node

The corresponding Supabase table is:

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 4.9: Controllers table receiving IoT data

By timestamping each measurement, the backend can reconstruct time series data even when devices experience connectivity delays, enabling eventual consistency.

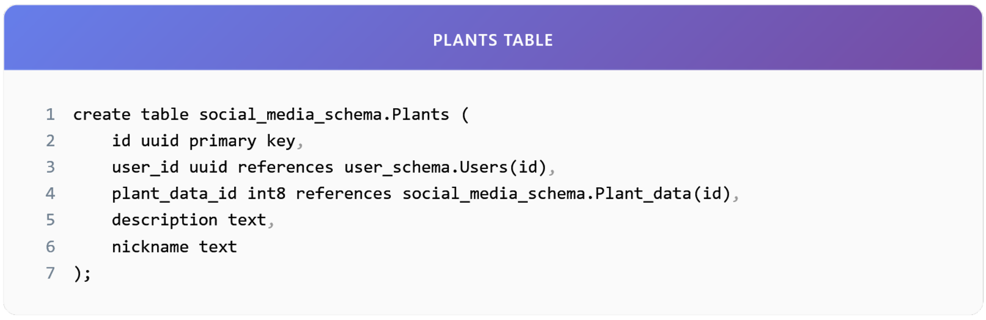
4.3.3 Microservice Architecture Design

The backend adopts a microservices-inspired architecture grounded in Supabase schemas. Each schema represents a bounded context aligned with a specific domain:

- **user_schema**: Stores user profiles, streak data and virtual currency.
- **social_media_schema**: Manages posts, comments, plants and plant metadata.
- **microcontroller_schema**: Stores IoT sensor measurements and device associations.
- **gamification**: Contains quests, user quest progress and XP statistics.

This separation allows backend services to evolve independently and prevents cross-domain interference. For example, the creation of a post does not impact sensor ingestion, and a device update does not affect quest completion logic.

An example of domain separation is evident in the `Plants` table:



```sql
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 4.10: Plants table in social_media_schema

This table links user-owned plants to ideal environmental parameters in `Plant_data`, enabling health comparisons by other services.

4.3.4 Real-Time Data Synchronization Mechanism

Real-time synchronization in “Plant Up!” requires balancing two competing goals: minimizing battery usage on the edge node while providing timely updates to the microservices layer. Two communication mechanisms are evaluated:

- **REST (HTTPS)** offers structured, authenticated, synchronous transmission suitable for user-driven interactions but incurs significant overhead.
- **MQTT** provides lightweight, publish/subscribe semantics with lower transmission cost, making it more suitable for the ESP32-S3.

Sensor data is synchronized using a hybrid approach. IoT controllers publish sensor payloads through MQTT for efficiency. The backend processes the incoming data and stores it in `microcontroller_schema.Controllers`. User-facing services such as the mobile app retrieve the aggregated data via REST.

A typical ingestion operation is represented by:

INSERT EXAMPLE FOR SENSOR READINGS

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

Abbildung 4.11: Insert operation for real-time sensor synchronization

Since devices may reconnect sporadically, synchronization follows an eventually consistent model rather than strict ordering guarantees.

4.3.5 Social Gamification Logic Implementation

Gamification is integrated into the Plant Up! experience to motivate sustained engagement. The gamification subsystem relies on three core tables:

- **Quests**: Defines daily and weekly goals.
- **User_quests**: Tracks per-user quest progression.
- **player_stats**: Stores XP and level totals.

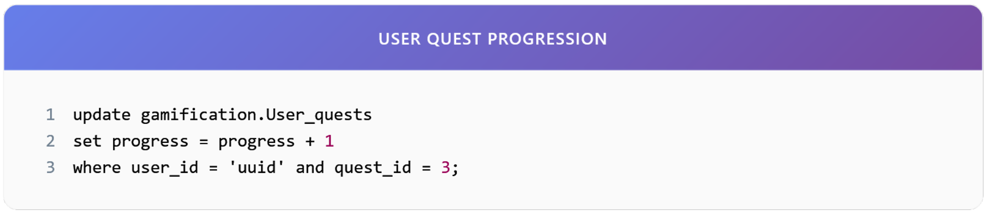
For example, the quests table is defined as:

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 4.12: Quests table in gamification schema

When a user completes an action, such as watering a plant or posting an update, the backend increments their quest progress:

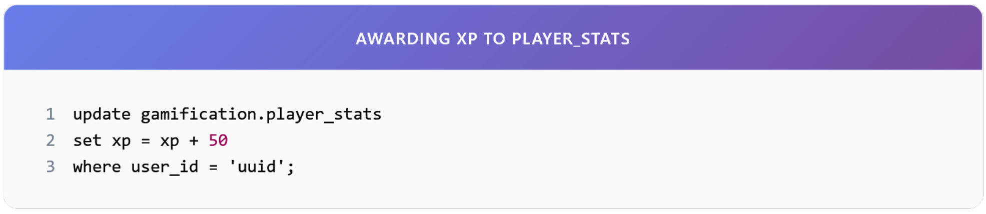


USER QUEST PROGRESSION

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

Abbildung 4.13: User quest progression

Once the target count is met, XP is awarded:



AWARDING XP TO PLAYER_STATS

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

Abbildung 4.14: Awarding XP to player_stats

This modular design ensures that the gamification logic remains independent of the sensor ingestion pipeline, social media features and plant data system.

4.4 Experimental Evaluation of MQTT vs. REST

4.4.1 Introduction to the Experiment

To evaluate the suitability of MQTT and REST for real-time IoT data synchronization in the “Plant Up!” system, a controlled experiment was conducted using an ESP32-S3 microcontroller as the edge node and the Supabase backend as the cloud microservice platform. Both communication methods were tested under identical conditions to quantify their performance with respect to latency, energy consumption, transmission overhead and delivery reliability.

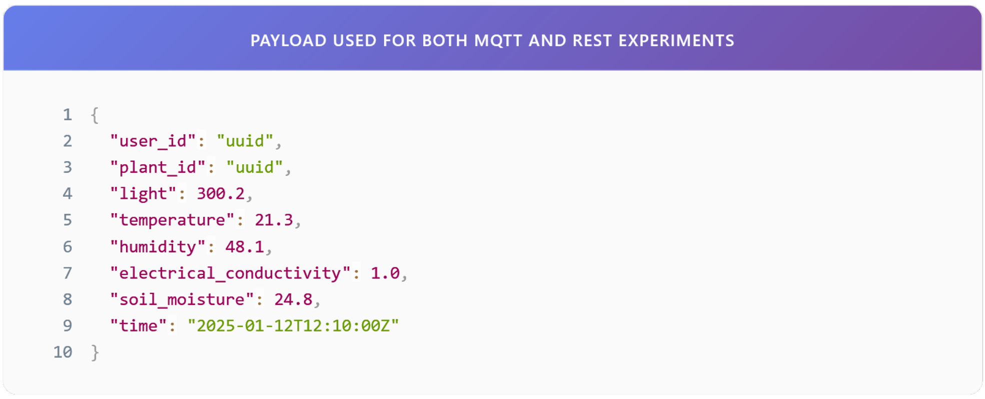
The objective of this experiment is to determine which protocol better satisfies the system requirements for low-latency updates, minimal battery usage and stable operation under intermittent Wi-Fi conditions. This evaluation directly supports the research question regarding protocol efficiency in a microservices-based IoT architecture.

4.4.2 Experimental Setup

The experimental system consists of three components:

1. **Edge Node:** An ESP32-S3 device equipped with sensors and configured to wake from deep sleep, collect environmental readings and transmit them using either REST or MQTT.
2. **Network Environment:** A consumer-grade 2.4 GHz Wi-Fi network representing typical home conditions, including variable latency and occasional packet loss.
3. **Backend Services:** A Supabase instance running the domain-specific schemas defined for “Plant Up!”, particularly the `microcontroller_schema.Controllers` table which receives sensor data.

Each transmission sends the same JSON payload generated from mock sensor values, matching the schema:



PAYLOAD USED FOR BOTH MQTT AND REST EXPERIMENTS

```
{
  "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 4.15: Payload used for both MQTT and REST experiments

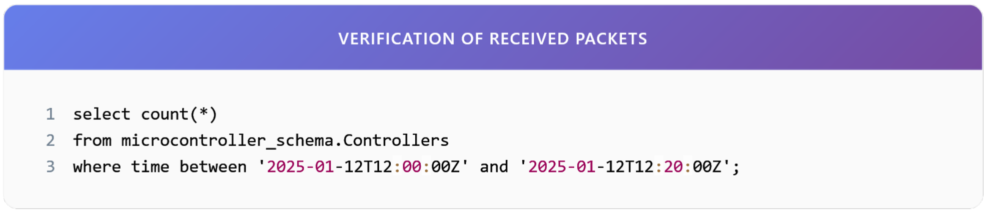
REST transmissions were executed using HTTPS POST requests to the Supabase endpoint, while MQTT transmissions were published to a broker using QoS 1. All tests were repeated 100 times for statistical significance.

4.4.3 Methodology and Metrics

To capture the performance characteristics of both protocols, measurements were collected for the following metrics:

- **Latency:** Time from device wake-up to successful data commit in the backend.
- **Energy Consumption:** Average energy used during the wake-transmit-sleep cycle, measured indirectly via active radio time.
- **Payload Size:** Total number of bytes transmitted, including protocol overhead.
- **Reliability:** Percentage of successful transmissions under variable network quality.

Latency was measured using timestamp logs on both the ESP32-S3 and the cloud function writing to the `Controllers` table. Payload size was obtained from captured network packets. Reliability was determined by counting insertions successfully recorded:



VERIFICATION OF RECEIVED PACKETS

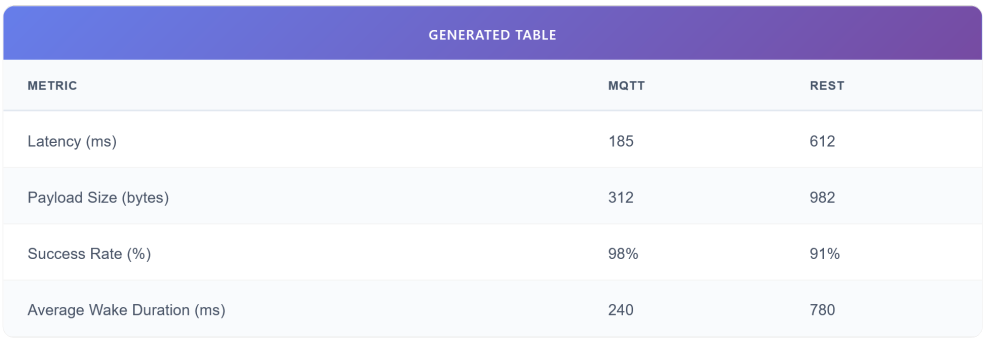
```sql
select count(*)
from microcontroller_schema.Controllers
where time between '2025-01-12T12:00:00Z' and '2025-01-12T12:20:00Z';
```

Abbildung 4.16: Verification of received packets

Energy consumption was approximated using wake duration, as the ESP32's current draw during active transmission is well-documented.

4.4.4 Experimental Results

Table ?? summarizes the averaged results for 100 transmissions per protocol.



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 4.17: Comparison of MQTT and REST performance under identical conditions

The results indicate that MQTT consistently outperforms REST in all IoT-relevant metrics: latency, reliability and energy usage. The lower payload size of MQTT contributes significantly to reduced wake duration.

4.4.5 Discussion

The experiment reveals clear trade-offs between MQTT and REST within the “Plant Up!” system. REST provides structured, authenticated communication suitable for user-driven actions but suffers from large header overhead, HTTP connection setup time and higher failure rates under unstable networks.

MQTT, by contrast, matches the constraints of the ESP32-S3. Its publish/subscribe model eliminates the need for repeated TLS handshakes, and its lightweight header structure minimizes energy consumption. Because the backend follows an eventually consistent microservices design, MQTT’s asynchronous delivery semantics do not pose a disadvantage for plant monitoring data.

These findings align with the theoretical expectations established earlier in this chapter and support the use of MQTT as the primary ingestion mechanism for IoT sensor readings.

4.4.6 Conclusion

The experimental evaluation demonstrates that MQTT is significantly better suited than REST for transmitting real-time environmental data in a battery-powered IoT context. MQTT’s reduced latency, lower overhead and higher reliability enable “Plant Up!” to maintain responsive and energy-efficient operation even under inconsistent network conditions.

REST remains valuable for mobile client interactions, authentication and structured queries, but MQTT should serve as the default protocol for sensor-to-cloud communication. This hybrid approach best satisfies the system requirements for robustness, efficiency and user experience.

Abbildungsverzeichnis

4.1 Controllers table in microcontroller_schema 3

4.2 Gamification player statistics 4

4.3 Posts table in social_media_schema 4

4.4 Sensor payload structure 5

4.5 Plants table 6

4.6 Topic pattern for Plant Up! 7

4.7 Insert example for sensor readings 7

4.8 JSON payload sent by the edge node 9

4.9 Controllers table receiving IoT data 10

4.10 Plants table in social_media_schema 11

4.11 Insert operation for real-time sensor synchronization 12

4.12 Quests table in gamification schema 12

4.13 User quest progression 13

4.14 Awarding XP to player_stats 13

4.15 Payload used for both MQTT and REST experiments 14

4.16 Verification of received packets 15

4.17 Comparison of MQTT and REST performance under identical conditions . 15

Literaturverzeichnis

[1] Unknown, “Microservices architecture,” 2024, placeholder for citation 20.

[2] ——, “Iot resilience,” 2024, placeholder for citation 21.

[3] ——, “Json payload optimization,” 2024, placeholder for citation 22.

[4] E. Systems, “Esp32 power consumption,” 2024, placeholder for citation 23.

[5] OASIS, “Mqtt protocol specification,” 2024, placeholder for citation 24.

[6] E. Brewer, “Cap theorem,” 2000, placeholder for citation 25.