htl donaustadt
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, 31. 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, 31. 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

Combining Nature and Technology: How Sensor-Based and Data-Driven Technologies Enhance Plant Monitoring and Care

1.1 Introduction

1.1.1 Motivation and Objectives

1.1.2 Research Question and Relevance

1.2 Environmental Factors in Plant Growth

1.2.1 Importance of Soil Moisture, Temperature, Light, and Humidity

1.2.2 Effects of Environmental Factors on Plant Growth

1.2.3 Necessity of Precise Measurements

1.3 Sensor Technologies for Plant Monitoring

1.3.1 Overview of Common Sensor Types

1.3.2 Limitations and Typical Error Sources

Kapitel 2

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

2.1 Introduction

2.1.1 Background and Context

To get user hooked onto ones application is one of the most important factors for software success. Especially with the overflow of applications competing, keeping users motivated to perform regular tasks, particularly those that do not offer immediate rewards, is a challenge. Getting users interested in a normal activity is where gamification can help.

This thesis analyzes the concept of gamification and how it is applied within “Plant Up!”, a smart gardening system designed to assist users in caring for their plants.

2.1.2 Research Question

To systematically address these challenges this thesis poses the following primary research question:

How can gamification be effectively applied to a gardening system to keep users more engaged in a normal activity?

2.2 Theoretical Foundations of Gamification

2.2.1 Definition of Gamification

Gamification defines how game design elements and mechanics are integrated into 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 [44, 45].

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

It is important to note that gamification is not the same as serious games or games for entertainment. Although the terms sound similar, they describe different concepts. 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. For example, a flight simulator used for pilot training or a surgery simulation used for medical training. Games for entertainment, on the other hand, are produced only for enjoyment, without any educational or functional objective [46].

2.2.3 Why Gamification Works in Non-Game Contexts

While many people question the effectiveness of gamification, research has 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 within the human psyche. It addresses the basic needs of autonomy, competence, and social relatedness, which form 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 actively supporting these needs.
- **2:** It redefines user interaction with a system. Instead of removing all difficult or 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 supported by immediate feedback mechanisms such as points and badges.
- **3:** Gamification uses social interaction to build a sense of community. Features such as cooperative challenges and leaderboards satisfy the desire to be noticed by others, increasing attachment to the activity through competitive or collaborative motivation [47].

2.2.4 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) [48].

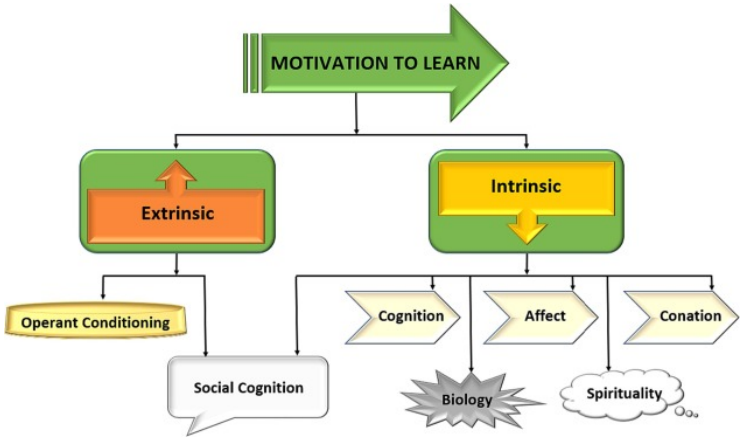

Abbildung 2.1: Classification of Motivation Types [48].

Intrinsic Motivation (The Engine of Engagement)

Intrinsic motivation refers to engaging in an activity because it is satisfying, enjoyable, or meaningful and is considered essential for long-term engagement [48]. According to Self-Determination Theory (SDT) [49], intrinsic motivation is supported when three basic psychological needs are fulfilled: Autonomy, Competence, and Relatedness.

When these needs are satisfied, individuals demonstrate increased curiosity, creativity, and persistence [48].

Self-Determination Theory (SDT) in Gamification

Self-Determination Theory 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 improved performance, persistence, and creativity. Failing to support them can significantly reduce motivation and well-being [48, 49].

Supporting Autonomy

Users should experience a sense of control over their actions. This is achievable with integrating optional tasks, multiple paths to success, and customization options [48].

Supporting Competence

Users require clear feedback that demonstrates progress and skill development, so that they can feel a sense of accomplishment and competence. It is mostly done by integrating experience points, progress indicators, adaptive difficulty, and immediate feedback loops [48].

Supporting Relatedness

Users are more engaged when they feel socially connected, hence integrating social feeds, cooperative goals, peer comparison, and mentorship features is a go to strategy [48].

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 [48].

- **Controlling Extrinsic Motivation:** When rewards are perceived as chore-like, they can reduce intrinsic motivation, resulting in short-term compliance but reduced long-term engagement. A typical example is a streak-based system that pressures users to perform regularly, which can lead to stress and burnout.[48].
- **Autonomy-Supportive Extrinsic Motivation:** When rewards provide informational feedback or signal competence, they can be internalized and reinforce intrinsic motivation. Feedback such as positive reinforcement for progress or improvement is more effective than obligation-based incentives.[48].

Effective gamification strategies use extrinsic rewards not as the primary objective, but as a means of supporting intrinsic motivation [47, 48].

The Role of Extrinsic Rewards

Gamification is not merely about adding rewards, but about how they are implemented. Extrinsic motivators can be powerful drivers of engagement but may reduce creativity and encourage mechanical behavior. This can result in retention driven by obligation rather than enjoyment [50]. When used carefully, extrinsic motivators can still provide short-term motivational boosts that foster intrinsic interests such as mastery or competition [50, 48].

Rewards that Support Intrinsic Motivation

Motivators that support intrinsic motivation are linked to a personal desire to master, explore, and enjoy an activity. They are the main motivator for long-term commitment, when implemented right, trying to keep them as informational feedback rather than controlling mechanisms is the key to succeeding [50] Often those are accomplished by integrating, badges, that indicate achievement and mastery, experience points, virtual currency for customization and titles [50].

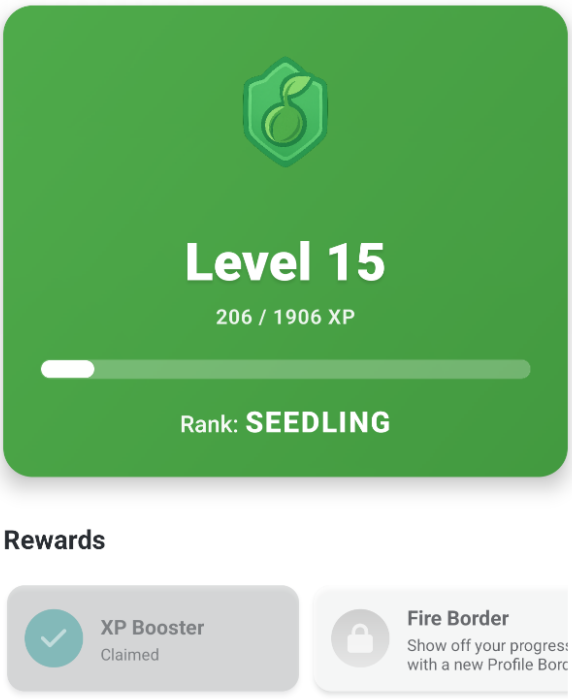

Abbildung 2.2: Level System and Rewards in Plant Up!.

Rewards that Undermine Intrinsic Motivation

Some rewards can be counterproductive when they seem controlling or worthless. [50] Such as over-rewarding insignificant actions,excessive penalties that induce stress, rankings without fairness or relevance, and when rewards replace genuine interest in the activity itself [50].

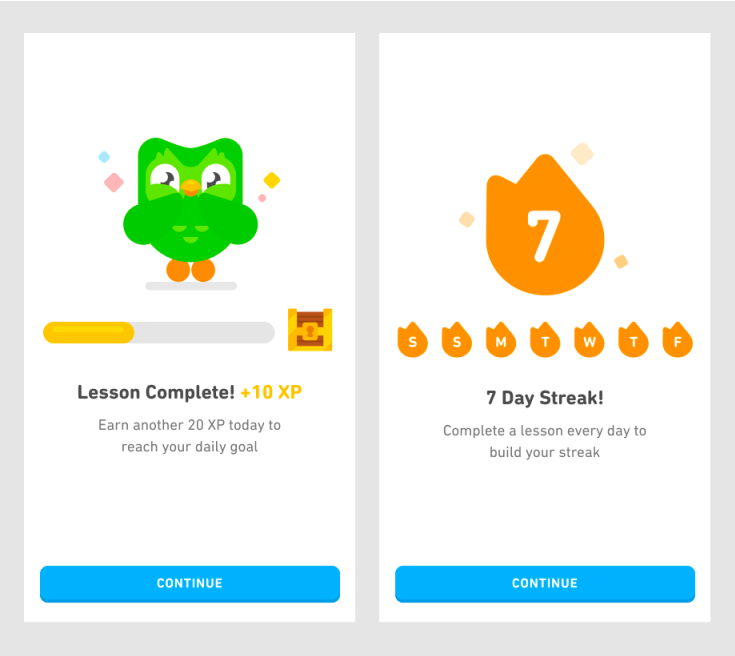

Abbildung 2.3: Duolingo Streak [54].

Attribution Theory

Attribution theory describes how individuals explain success and failure. In gamified systems, it is important that users get rewarded for their own effort and strategies rather than to luck or fixed ability. When implementing effort-based feedback it can increase motivation and encourage continued user engagement. Additionally, designing penalties and failure in such a way that it only seems temporary and improvable can lead to less frustration, hence making the user more likely to continue using the app [52].

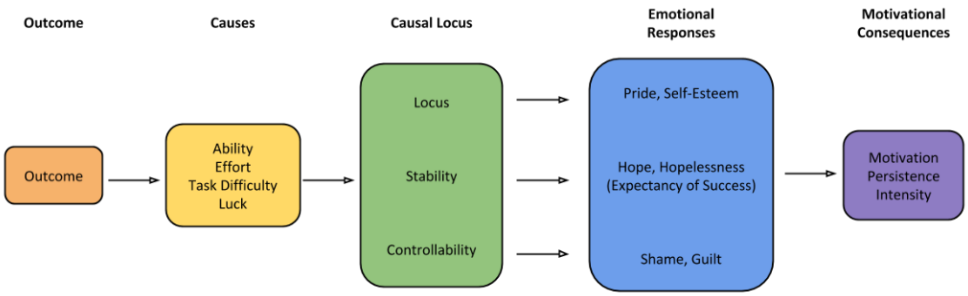

Abbildung 2.4: Attributional Process of Motivation.

Expectancy-Value Theory

Expectancy-value theory explains how what can motivate a user before engaging in an activity, those being: the expectation of success and the subjective value of the task. When a user thinks that they can succeed in an activity and values it, they are more likely to engage in it.

1. **Expectancy for Success:** Expectance for success can be defined with the question: "How likely is it that I will succeed in this activity?Ïn games and gamified systems, this belief is answered with a clear definition of goals, transparent rules, the difficulty of the task, and feedback provided during the activity. To make the user have a clear vision of their success, it is important to provide clear objectives, transparent mechanics, and visible indicators of progress. When the criterias of a Ïs this challenge achievable?"question is met, the user is more likely to invest effort in it [53].
2. **Subjective Task Value:** Subjective Task Value, describes if the task is worth the effort, which is subjective for each user. This value consists of several components, including intrinsic value, the importance of performing well, usefulness for future goals, and effort, such as time investment. In games a value can be increased by making the user progress faster, reducing frustration or making those tasks relevant for future goals.[53].

2.2.5 Engagement and Habit Formation

Feedback Loops

A feedback loop is a important component of gamification, as it provides the users with immediate feedback on their progress and achievements.

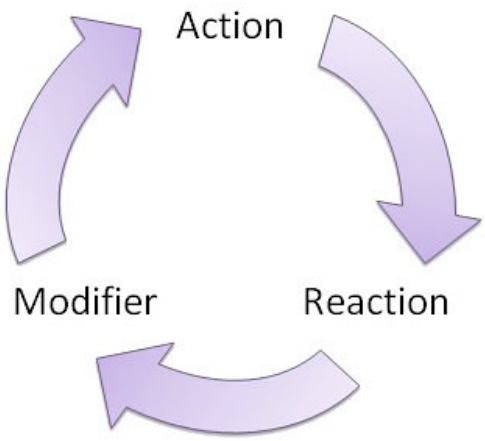

Abbildung 2.5: Feedback Loop [56].

It is important to take note of, that the feedback loop should consist of both positive and negative feedback.

Most people only like to receive positive feedback, such as: badges, experience points, virtual currency, titles, but it is important to take note of, that negative feedback can also be a motivator. Not only does a negative feedback, such as: penalties, warnings, or punishments, can be a wake up call for the user, to start improving their behavior, but it can also be a motivator to avoid those penalties. [57].

Short-term versus long-term engagement

Short-term engagement refers to the immediate satisfaction that users experience when they engage in an activity. Long-term engagement, on the other hand, refers to the sustained interest and motivation that users have for an activity over time. Short-term engagement is often driven by extrinsic motivators, such as rewards or recognition, while long-term engagement is driven by intrinsic motivators, such as the enjoyment of the activity itself.

Daily routines and behavioral reinforcement

Daily routines and behavioral reinforcement are important components of gamification, as they help users to develop and maintain healthy habits. Daily routines refer to the

regular patterns of behavior that users engage in on a daily basis. Behavioral reinforcement refers to the process by which users are motivated to engage in a behavior because it has been reinforced in the past. Daily routines and behavioral reinforcement can be used to reinforce desired behaviors or to discourage undesired ones.

2.3 Gamification Design Patterns

2.3.1 Core Game Mechanics

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

2.3.2 Advanced Engagement Patterns

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

2.3.3 Social Gamification

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

2.3.4 Failure States & Ethical Design

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

2.4 Why Gamification Works for “Normal” Activities

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

2.4.1 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.

2.4.2 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.

2.4.3 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.

2.5 Case Study: Gamification in Plant Up!

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

2.5.1 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.

2.5.2 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 ('Sprig'):** A character that provides tutorials, hints, and narrative context.

2.5.3 Mapping Design Patterns to Motivation

Each gamification element is designed to target specific motivational drivers:

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

2.6 Evaluation & Discussion

2.6.1 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.

2.6.2 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.

2.6.3 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 3

Edge vs. Cloud Processing in Smart Gardening: A Comparative Study for Environmental Sensor Data

3.1 Introduction

3.1.1 Background

Microservices in Gardening

- Paradigm shift, individual plant care, community interaction, shared data, digital transformation, nature meets tech, social platforms, sensor integration, remote monitoring, enthusiast community, collaborative gardening, data-driven plant care, IoT connectivity
- **[INSERT FIGURE: Concept Art of Social Gardening with IoT]**

The Latency Challenge: Real-time Notification vs. Cloud Round-Trips

- Technical hurdles, delay management, sensor reading, user notification delay, round-trip time (RTT), network jitter, cloud infrastructure, internet dependency, timely intervention, timing criticality, processing lag, signal propagation, feedback loops

- **[INSERT DIAGRAM: Round-Trip Time Visualization (Sensor → Cloud → Push Notification)]**

Dependency Risks: What happens to the plant when the Microservice is down?

- Centralized architecture, single point of failure, microservice downtime, plant health risks, missed alerts, service outages, server crashes, connectivity loss, maintenance windows, failover mechanisms, contingency planning, disaster recovery

3.1.2 Research Question

Primary Question

- **"What are the trade-offs between edge and cloud processing for real-time plant monitoring in consumer gardening systems?"**
- Edge computing benefits, Cloud computing benefits, latency vs. throughput, cost vs. complexity, data integrity, system reliability, user experience impact

3.1.3 Scope and Limitations

Focus on Consumer Hardware (ESP32/ESP8266)

- Consumer-grade electronics, ESP32 capabilities, ESP8266 limitations, cost-effectiveness, accessibility, hobbyist market, maker community, hardware constraints, memory limits, processing power, WiFi modules, microcontroller adoption
- **[INSERT PICTURE: ESP32/ESP8266 Microcontroller Setup]**

Comparison limited to Processing Location (Edge vs. Cloud), not Transport Protocols (MQTT assumed)

- Scope boundary, processing location focus, edge computing, cloud computing, transport layer abstraction, MQTT protocol, standard compliance, exclusion of

CoAP/HTTP, architectural comparison, logic placement, fixed communication variables

- **Protocol Discussion:** Detailed comparison and trade-offs between MQTT and REST communication protocols will be addressed in the following chapter Evaluating MQTT vs. REST.

3.2 Theoretical Framework

3.2.1 Modern IoT Backends

Backend-as-a-Service (BaaS): The Role of Supabase in IoT

- Backend-as-a-Service, BaaS, Supabase architecture, real-time subscriptions, websockets, rapid development, managed infrastructure, scalability, postgres-native, authentication, database APIs, serverless paradigm, unified backend
- **[INSERT DIAGRAM: Supabase Realtime Architecture Flow]**
- Source: `https://metadesignsolutions.com/supabase-vs-firebase-when-to-ch#:~:text=Supabase%20is%20an%20open%2Dsource,power%20of%20a%20relation20database.`

Time-Series Databases: Why standard SQL fails for sensor streams

- Relational database limitations, high-velocity ingestion, storage bloat, index inefficiency, B-tree performance, standard SQL constraints, sensor data volume, continuous streams, data retention issues, query degradation
- Source: `https://milvus.io/ai-quick-reference/what-are-the-limitations-c`

The TimescaleDB Extension: Hypertables and Chunking explained

- PostgreSQL extension, TimescaleDB, hypertables, automatic partitioning, data chunking, temporal locality, high-performance inserts, time-series optimization, compression algorithms, continuous aggregates, SQL compatibility, retention policies

- **[INSERT DIAGRAM: Hypertable vs. Normal Table Storage]**
- Source: `https://www.tigerdata.com/learn/the-best-time-series-databases-c`

Object Storage and Media Handling in Microservices

- Unstructured data storage, Supabase Storage, S3 compatibility, scalable object storage, media asset management, separation of concerns (DB vs. Storage), CDN caching strategies, secure file access policies, handling large binary files (blobs)
- Source: `https://github.com/supabase/storage`

3.2.2 Microservices in IoT

The "Gateway Pattern: Using a C# Service to bridge MQTT and Database

- Gateway design pattern, architectural patterns, C# middleware, .NET Core, MQTT broker, database bridge, decoupling services, protocol translation, message buffering, security boundary, data normalization, traffic throttling
- **[CODE SNIPPET: C# Worker Service Basic Structure]**
- Source: `https://microservices.io/patterns/apigateway.html`

Service Inter-communication: Synchronous vs. Asynchronous patterns

- Inter-service communication, synchronous REST, asynchronous messaging, message queues, event-driven architecture, request-response, immediate consistency, eventual consistency, system coupling, latency impact, blocking operations
- Source: `https://www.geeksforgeeks.org/system-design/microservices-commun`

3.2.3 Edge Computing Patterns

Cloud Computing, Fog Computing, and Edge Computing

- **Cloud Computing:** Centralized processing, high computational power, high latency, infinite storage, global access.
- **Fog Computing:** Intermediate layer, gateway processing, local area network, bridging Edge and Cloud, reduced latency compared to Cloud.
- **Edge Computing:** Source processing, on-device logic, microcontroller level, lowest latency, real-time response, limited resources.
- **[INSERT FIGURE: Cloud vs. Fog vs. Edge Architecture Pyramid]**

"Thick Edge"vs. "Thin Edge: How much logic should the microcontroller hold?

- Edge architecture, Thick Edge, Thin Edge, logic distribution, processing offload, local autonomy, bandwidth conservation, resource constraints, computational power, firmware complexity, distributed intelligence, architectural trade-offs, remote management

3.3 The PlantUp Architecture

3.3.1 System Overview

High-Level Diagram: From Soil Sensor to React Native App

- Full-stack architecture, system topology, soil moisture sensors, analog-to-digital conversion, capacitive sensing, React Native framework, mobile application, user interface, cloud synchronization, end-to-end data flow, interactive dashboard
- **[INSERT FIGURE: High-Level System Architecture Diagram]**
- Source: `https://www.geeksforgeeks.org/postgresql/postgresql-schema/` `https://www.geeksforgeeks.org/system-design/architecture-of-a-system/`

3.3.2 The Microservices Layer (Cloud)

User & Social Media Services: Gamification and Content Sharing

- Social microservices, gamification engine, PostgreSQL schemas, user profiles, experience points (XP), leaderboards, achievement system, social graphs, user interactions, relational data modeling, reward logic, competition mechanics
- **Social Media Features:** Video upload pipeline, automatic thumbnail generation, post creation, Supabase Storage buckets, media metadata, content delivery network (CDN) integration, secure signed URLs, social feed algorithms
- **[INSERT DIAGRAM: Gamification Database Schema (ERD)]**
- **[INSERT DIAGRAM: Video Upload and Thumbnail Generation Flow]**
- Source: `https://www.cloudflare.com/learning/cdn/what-is-a-cdn/` `https://supabase.com/docs/guides/storage/cdn/fundamentals`

Webshop Service: Shopify Integration

- Shopify platform, SaaS e-commerce, API integration, inventory management, external service, seamless user experience, checkout process, webhook sync, product catalog, managed hosting, third-party plugin ecosystem
- **[INSERT SCREENSHOT: PlantUp Shop Concept on Shopify]**
- Source: `https://www.cloudflare.com/de-de/learning/cloud/what-is-saas/` `https://www.techtarget.com/searchcloudcomputing/definition/Software-as`

Microcontroller Management Service: The C# .NET Core Worker

- Background worker, .NET ecosystem, device management, command orchestration, state synchronization, keep-alive monitoring, message processing, centralized control, device registry, firmware updates (OTA), scalable architecture
- **[CODE SNIPPET: C# Device State Manager Class]**
- Source: `https://learn.microsoft.com/en-us/aspnet/core/fundamentals/host/hosted-services?view=aspnetcore-10.0&tabs=visual-studio`

3.3.3 The Data Persistence Layer (Supabase)

Unified Database Strategy: Single Postgres instance with logical schema separation

- Database consolidation, single instance, logical isolation, schema-based multi-tenancy, operational simplicity, reduced overhead, unified backup, cross-schema querying, management efficiency, Postgres roles, access control

The microcontroller_data Schema: Utilizing the TimescaleDB Extension

- Dedicated schema, TimescaleDB integration, sensor data storage, time-series optimization, optimized querying, high-volume tables, data partitioning, specialized indexing, analytical queries, storage efficiency, historical data
- **[CODE SNIPPET: SQL Creation Script for TimescaleDB Hypertable]**
- Source: `https://www.influxdata.com/time-series-database/` `https://github.com/timescale/timescaledb`

Hypertable Design: Partitioning strategy for millions of sensor readings

- Hypertable architecture, data partitioning, time intervals, chunk management, scalable storage, query performance, massive datasets, millions of rows, retention management, automated maintenance, disk I/O optimization
- **[INSERT DIAGRAM: Visualizing Hypertable Chunks based on Time]**
- Source: `https://www.cloudthat.com/resources/blog/scaling-time-series-da`

3.4 Methodology

3.4.1 Experimental Architectures

Architecture A: Cloud-Centric (The "Connected"Plant)

- Cloud-centric design, connected model, centralized decision making, MQTT transmission, C# service logic, historical querying, time-weighted averages, notification latency, server-side processing, dumb terminal, remote monitoring, bandwidth dependence
- **[INSERT DIAGRAM: Architecture A Data Flow (Sensor → Cloud → Push Notification)]**
- Source: `https://www.ibtimes.com/architecting-connected-retail-future-clo`

Architecture B: Edge-Centric (The SSmart"Plant)

- Edge-centric design, smart device, distributed intelligence, local processing, independent operation, C++ logic, asynchronous logging, eventual consistency, disconnect tolerance, local status indication, autonomous state monitoring, bandwidth efficiency
- **[INSERT DIAGRAM: Architecture B Data Flow (Sensor → Edge → Local Indication/Async Log)]**
- Source: `https://www.mdpi.com/1424-8220/20/3/892`

3.4.2 Implementation Details

Edge Firmware (C++): Sensor Calibration and Power Management

- Firmware development, C++ programming, embedded systems, capacitive moisture sensing, signal smoothing, debouncing, deep sleep modes, battery optimization, data packetization, interrupt handling, robust error recovery
- **[CODE SNIPPET: C++ ESP32 Deep Sleep & Sensor Reading]**

Cloud Service (C#): Implementation of the Supabase.Client and Supabase.Realtime listeners

- Cloud service implementation, C# coding, Supabase SDK, real-time listeners, websocket subscription, event handling, state synchronization, database triggers, remote procedure calls, responsive backend, client library integration
- **[CODE SNIPPET: C# Supabase Realtime Listener Setup]**
- Source: `https://github.com/supabase/realtime`

Database Schema: Definition of sensor_readings hypertable

- Schema definition, DDL statements, hypertable creation, column types, indexing strategy, partition keys, compression settings, data persistence, storage optimization, analytical capabilities, query planning
- **[CODE SNIPPET: SQL for Hypertable Creation and Compression Policy]**
- Source: `https://gunesramazan.medium.com/managing-iot-heartbeat-data-wit`

Power Supply Design and Consumption Analysis

- **Worst Case Scenario:** Continuous operation, all sensors active, WiFi transmission 24/7, 168 mA continuous draw, 4032 mAh daily consumption, theoretical maximum, stress testing benchmarks
- **Realistic Strategy (Deep Sleep):** Cyclic operation, 10-minute reporting interval, 2-5 mA average consumption, user-configurable frequency, deep sleep (0.1 mA), active burst (150 mA for 3s), efficient power management
- **Power Architecture:** Rechargeable LiPo battery (3.7V), Dual charging path (USB-C + Solar), BQ24074 Power Management Module, automatic source switching, stable voltage regulation, outdoor autonomy
- **Component Specifications:** LiPo Capacity (3000-5000 mAh), Solar Panel (5-6V, 1-2W, outdoor-proof), Voltage Regulator (Integrated)
- **Battery Life Projections:**
  - 3000 mAh Battery: 3 mA avg $\rightarrow$ 1000 hours $\rightarrow$ approx. 41 days

  - 5000 mAh Battery: 3 mA avg → 1666 hours → approx. 69 days (>2 months)

- **[INSERT GRAPH: Power Consumption Profile (Deep Sleep vs Active Burst)]**
- Source: `https://running-system.atlassian.net/wiki/spaces/RS/pages/126550051/Power+Management+Research`

3.4.3 Test Scenarios

- Source: `https://www.geeksforgeeks.org/software-testing/system-testing/`
  `https://github.com/timescale/tsbs`
  `https://www.bytebase.com/blog/postgres-row-level-security-limitations-`
  `https://supabase.com/docs/guides/realtime/benchmarks`

Scenario 1: Ideal Conditions

- Baseline testing, ideal environment, low latency, zero packet loss, stable connectivity, theoretical maximums, benchmarking, optimal performance, clean signal, control group, reference measurements

Scenario 2: Database Load (Simulating 1,000 concurrent writes to TimescaleDB)

- Load testing, stress test, concurrent users, high ingestion rate, database bottlenecks, resource contention, write blocking, 1000 connections, performance degradation, scalability limits, system stability, latency spike
- **[INSERT PICTURE: Load Testing Setup / Console Logs]**

Scenario 3: Service Outage (Simulating a crash of the C# Microservice)

- Resilience testing, fault injection, service crash, chaos engineering, availability test, outage simulation, watchdog timers, system recovery, safety verification, data loss analysis, fail-safe behavior

3.5 Results and Data Analysis

3.5.1 Latency Analysis

Edge Response Time (<50ms) vs. Cloud Round Trip (>200ms + DB Query Time)

- Quantitative analysis, edge speed, sub-50ms response, cloud delay, >200ms latency, network overhead, database query time, round-trip metrics, comparison charts, performance penalty, real-time constraints, user experience impact
- **[INSERT GRAPH: Comparison Bar Chart of Alert Generation Time (Edge vs. Cloud)]**
- Source: `https://ifactoryapp.com/blog/real-time-ai-cloud-latency-manufac`

3.5.2 Database Performance

Impact of High Ingestion Rates on C# Service Query Performance

- Performance impact, ingestion throughput, query latency, lock contention, C# service load, resource usage, CPU/Memory metrics, read-write conflict, optimization analysis, database tuning, concurrency issues
- **[INSERT GRAPH: Query Latency vs. Ingestion Rate]**

Storage Efficiency: Raw Data (Cloud Decision) vs. Batched Aggregates (Edge Decision)

- Storagemetrics, raw data volume, aggregation efficiency, payload size, storage costs, bandwidth consumption, cloud storage, batched transmission, data compression, economy of scale, archival strategy
- **[INSERT CHART: Projected Storage Costs Over Time]**
- Source: `https://metadeskglobal.com/from-raw-sensor-data-to-intelligent-`
  `https://stonefly.com/blog/iot-data-storage-architectures-databases/`

3.5.3 Reliability and Resilience

Alert Delivery Success Rates during a simulated 1-hour internet outage

- Reliability metrics, delivery success analysis, outage simulation, local data buffering, resync capability, data integrity, critical failure, connectivity loss, edge resilience, robust design, message queuing
- **[INSERT TIMELINE: Data Sync Recovery after 1-Hour Outage]**
- Source: `https://www.opal-rt.com/blog/5-essential-computer-simulation-tec`

3.5.4 Gamification Synchronization

The “Lag” between Action (Watering) and Reward (XP Update in Social Service)

- User experience, feedback delay, action-reward loop, gamification lag, sync latency, psychological impact, interface responsiveness, XP updates, social satisfaction, system consistency, near real-time
- **[INSERT DIAGRAM: Sequence Diagram of Action-to-Reward Delay]**
- Source: `https://supabase.com/docs/guides/functions`

3.6 Discussion

3.6.1 The "HybridSSweet Spot for PlantUp

Using Edge for Real-time Monitoring and Cloud for Engagement (XP/Leaderboards)

- Architectural synthesis, hybrid model, best of both worlds, edge reliability, cloud scalability, critical vs. non-critical, monitoring logic, engagement layer, strategic separation, optimal design, computing paradigm balance
- **[INSERT DIAGRAM: The Hybrid SSweet Spot"Logic Distribution]**
- Source: `https://ieeexplore.ieee.org/document/11158770`

3.6.2 Database Considerations

Benefits of Supabase's unified access for the React Native App

- Developer experience, unified API, frontend simplification, React Native integration, simplified auth, reduced complexity, rapid prototyping, coherent data model, streamlined access, architectural elegance
- Source: `https://taglineinfotech.com/blog/which-backend-is-best-for-reac`

Trade-offs of querying TimescaleDB for real-time control loops

- Design trade-offs, real-time query risks, database dependency, latency variability, connection overhead, system stability, control loop jitter, architectural decisions, load implications, performance vs. consistency
- Source: `https://www.tigerdata.com/blog/timescaledb-vs-influxdb-for-time`

3.6.3 Complexity vs. Cost

Development cost of maintaining logic in two places (C++ and C#) vs. centralized C# logic

- Cost-benefit analysis, maintenance overhead, code duplication, technological diversity, C++ vs C#, staffing requirements, development velocity, operational complexity, total cost of ownership, robustness justification, long-term viability
- **[INSERT TABLE: Cost/Complexity Comparison Matrix]**
- Source: Chapter 2.1 `https://flame-challenge.authorea.com/users/982330/articles/1348311/master/file/data/wileyNJDv5_AMA/wileyNJDv5_AMA.pdf`
  `https://www.baeldung.com/cs/distributed-systems-thin-vs-thick-clients`
  `https://www.einfochips.com/blog/understanding-risks-in-over-the-air-f`

3.7 Conclusion

3.7.1 Summary of Findings

- Research summary, key takeaways, latency findings, reliability confirmation, architectural verdict, performance benchmarks, hypothesis validation, data-driven conclusions, system evaluation, project goal achievement

3.7.2 Architectural Recommendations for PlantUp

Adopt Ëdge-First"Monitoring to guarantee data integrity

- Strategic recommendation, Edge-First philosophy, integrity priority, decentralized monitoring, robustness first, fail-safe architecture, local buffering, connectivity independence, critical path optimization

Use Supabase Edge Functions (optional) vs. C# Service for lighter tasks

- Alternative architecture, serverless functions, Supabase Edge Functions, lightweight logic, cost reduction, operational simplicity, scaling flexibility, C# service replacement, modernization, reduced maintenance

3.7.3 Future Work (e.g., Predictive Health Alerts using Machine Learning on TimescaleDB data)

- Future directions, predictive analytics, machine learning, AI integration, smart health alerts, user guidance, data mining, TimescaleDB analytics, proactive care, intelligent systems, next-generation IoT
- **[INSERT CONCEPT ART: Future AI Dashboard]**

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 [20]. 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 [21].

Driven by rapid global urbanization, shrinking living spaces, and a rising awareness of environmental sustainability, urban gardening has grown into a significant movement [22]. This is more than just a hobbyist trend; it signals a shift toward the Social Internet of Things (SIoT) [23]. 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 [24]. 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 [25]. 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 [26].

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 [27]. 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 [28]. 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 [29]. 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 [30]. 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 [31].

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 [31]. 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 [19].

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 [32]. 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 [33]?

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 [34]. 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 [35].

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 [36]. 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 [37]. 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 [39], 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** [40]. 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 [40].
- **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** [39].

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 [41]. 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).

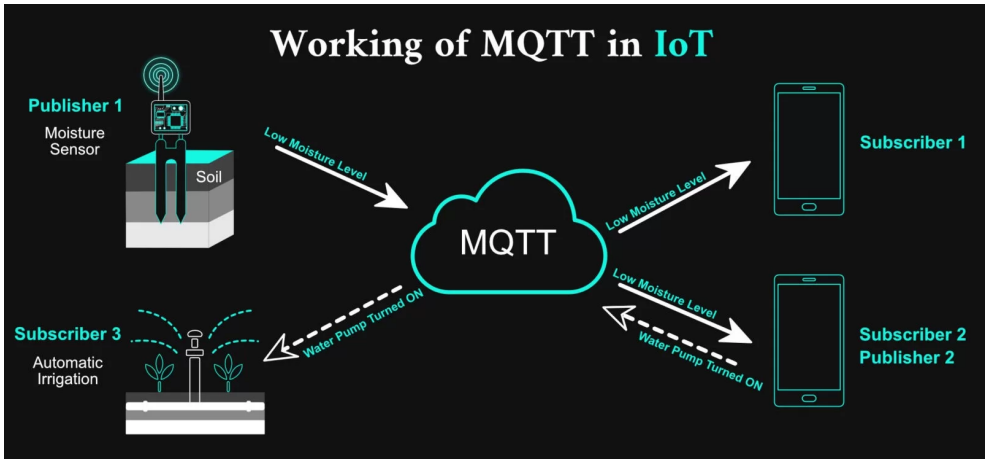

Abbildung 4.3: MQTT Publish/Subscribe Model: The Broker acts as a central hub, efficiently distributing messages from sensors to various services [19].

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.

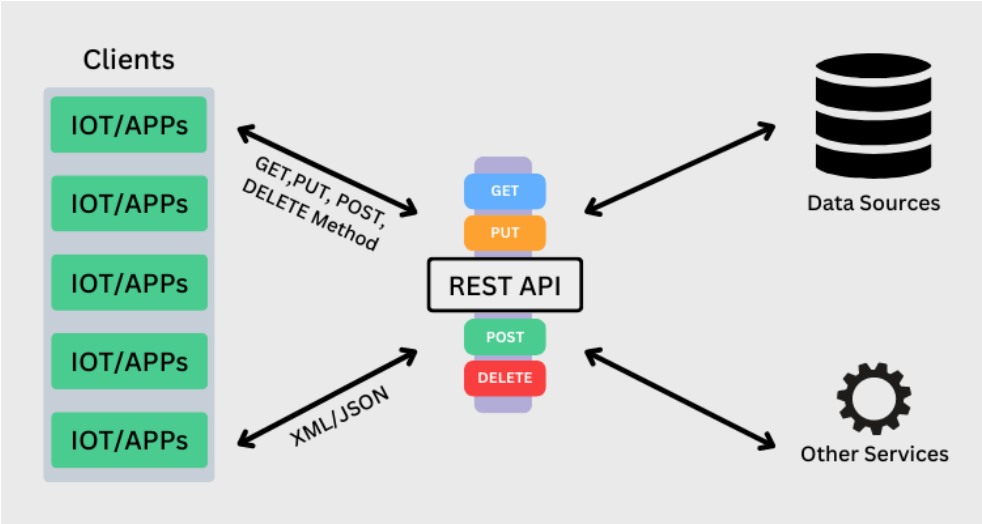

Abbildung 4.4: REST Request-Response Workflow: Stateless clients must open a new connection for every request, incurring significant overhead [18].

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 [38, 42].

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

```
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

```
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

```
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

```
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

```
1 select count(*)
2 from microcontroller_schema.Controllers
3 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 4.14 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

Literaturverzeichnis

[1] Z. Cao, T. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, “Openpose: Realtime multi-person 2d pose estimation using part affinity fields,” in *In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[2] K. Sun, B. Xiao, D. Liu, and J. Wang, “High-resolution representations for labeling pixels and regions,” in *In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[3] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele, “2d human pose estimation: New benchmark and state of the art analysis,” in *In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[4] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu, “Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments,” *In: IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2013.

[5] D. Pavllo, C. Feichtenhofer, D. Grangier, and M. Auli, “3d human pose estimation in video with temporal convolutions and semi-supervised training,” in *In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[6] P. F. Felzenszwalb and D. P. Huttenlocher, “Pictorial structures for object recognition,” *In: International Journal of Computer Vision*, vol. 61, no. 1, pp. 55–79, 2005.

[7] C. Lugaresi, J. Tang, H. Nash, and et al., “Mediapipe: A framework for building perception pipelines,” in *In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2019.

[8] M. Contributors, “Openmmlab pose estimation toolbox and benchmark,” https://github.com/open-mmlab/mmpose, 2020.

[9] A. Mathis, P. Mamidanna, K. M. Cury, and et al., “Deeplabcut: Markerless pose estimation of user-defined body parts with deep learning,” *In: Nature Neuroscience*, vol. 21, no. 9, pp. 1281–1289, 2018.

[10] A. Grinciunaite, A. Petrenas, and D. Navakauskas, “Pose estimation for human motion analysis: A survey,” *In: Sensors*, vol. 22, no. 17, p. 6501, 2022.

[11] Y. Xu, J. Wang, Y. Zhang, J. Wang, and Y. Wei, “Vitpose: Simple vision transformer baselines for human pose estimation,” *In: Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

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

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

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

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

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

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

[18] DreamFactory, “Rest vs graphql: Which api design style is right for your organization?” https://blog.dreamfactory.com/rest-vs-graphql-which-api-design-style-is-right-for-your-organization, 2024, accessed: 2025-12-09.

[19] PsiBorg, “Advantages of using mqtt for iot devices,” https://psiborg.in/advantages-of-using-mqtt-for-iot-devices/, 2024, accessed: 2025-12-09.

[20] Leher, “Exploring agriculture 4.0: Technology’s role in future farming,” https://leher.ag/blog/technology-role-agriculture-4-0-future-farming, 2024, accessed: 2025-12-09.

[21] StreetSolver, “The smart urban garden: Top tech & ai helping you grow more food at home in 2026,” https://streetsolver.com/blogs/the-smart-urban-garden-top-tech-ai-helping-you-grow-more-food-at-home-in-2026, 2024, accessed: 2025-12-09.

[22] GrowDirector, “Urban agriculture in 2025: A growing trend with deep roots,” https://growdirector.com/urban-agriculture-in-2025-a-growing-trend-with-deep-roots, 2024, accessed: 2025-12-09.

[23] BPB Online, “What is social internet of things (siot)?” https://dev.to/bpb_online/what-is-social-internet-of-things-siot-3g0a, 2024, accessed: 2025-12-09.

[24] J. Jung and I. Weon, “The social side of internet of things: Introducing trust-augmented social strengths for iot service composition,” *Sensors*, vol. 25, no. 15, p. 4794, 2025, accessed: 2025-12-09. [Online]. Available: https://mdpi.com/1424-8220/25/15/4794

[25] KaaIoT, “Iot device challenges – power & connectivity constraints,” https://kaaiot.com/iot-knowledge-base/how-does-wi-fi-technology-link-to-the-iot-industry, 2024, accessed: 2025-12-09.

[26] B. Hemus, “6 common iot challenges and how to solve them,” https://emnify.com/blog/iot-challenges, 2024, accessed: 2025-12-09.

[27] Smartico, “Growing green: How gamification is revolutionizing sustainable agriculture,” https://smartico.ai/blog-post/gamification-revolutionizing-sustainable-agriculture, 2025, accessed: 2025-12-09.

[28] B. Che, “Implementing iot with a three-tier architecture,” https://enterprisersproject.com/article/2016/1/implementing-iot-three-tier-architecture, 2016, accessed: 2025-12-09.

[29] Spinify, “The critical role of real-time feedback in gamification,” https://spinify.com/blog/how-ai-is-enabling-real-time-feedback-in-gamification, 2023, accessed: 2025-12-09.

[30] Programming Electronics, “A practical guide to esp32 deep sleep modes,” https://programmingelectronics.com/esp32-deep-sleep-mode, 2024, accessed: 2025-12-09.

[31] R. Community, “Esp32 deep sleep wake-up discussion,” https://reddit.com/r/esp32/comments/gfroev/time_to_wake_up_and_connect_to_wifi_from_deep/, 2020, accessed: 2025-12-09.

[32] DesignGurus, “Cap theorem explained,” https://designgurus.io/answers/detail/cap-theorem-for-system-design-interview, 2024, accessed: 2025-12-09.

[33] Nabto, “MQTT vs. REST in IoT: Which should you choose?” https://nabto.com/mqtt-vs-rest-iot, 2021, accessed: 2025-12-09.

[34] AWS, “Microservices architecture - key concepts,” https://docs.aws.amazon.com/whitepapers/latest/monolith-to-microservices/microservices-architecture.html, 2019, accessed: 2025-12-09.

[35] M. Fowler, “Monolithic vs. microservices,” https://martinfowler.com/articles/microservices.html, 2024, accessed: 2025-12-09.

[36] S. Newman, *Building Microservices*. O'Reilly Media, 2015, accessed: 2025-12-09.

[37] C. Richardson, "Microservices and data," https://microservices.io/patterns/data/database-per-service.html, 2024, accessed: 2025-12-09.

[38] E. Brewer, "Brewer's cap theorem," https://interviewnoodle.com/understanding-the-cap-theorem-a-deep-dive-into-the-fundamental-trade-offs-of-distributed-sys 2000, summary by InterviewNoodle. Accessed: 2025-12-09.

[39] Espressif Systems, *ESP32-S3 Series Datasheet*, 2024, accessed: 2025-12-09. [Online]. Available: https://espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf

[40] LastMinuteEngineers, "Esp32 power consumption - active mode," https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/, 2024, accessed: 2025-12-09.

[41] A. S. Forum, "Wi-fi reconnect overhead," https://forums.adafruit.com/viewtopic.php?f=57&t=163388, 2020, accessed: 2025-12-09.

[42] S. Jaiswal, "The impossible triangle of distributed systems: Cap theorem," https://www.linkedin.com/pulse/impossible-triangle-distributed-systems-cap-theorem-sejal-jaiswal-atbzc, 2023, accessed: 2025-12-09.

[43] R. Onuoha, "Diplomarbeit," 2025, internal Reference. Accessed: 2025-12-09.

[44] Gabler Wirtschaftslexikon, "Definition: Gamification," https://wirtschaftslexikon.gabler.de/definition/gamification-53874, 2024, accessed: 2025-12-28.

[45] G. Burtt, "Gamification, game-based learning and serious games – what's the difference?" https://grendelgames.com/gamification-game-based-learning-serious-games/, n.d., accessed: 2025-12-28.

[46] R. Damaševičius, "Serious games and gamification in healthcare: A meta-review?" https://www.mdpi.com/2078-2489/14/2/105, 2023, accessed: 2025-12-28.

[47] Haufe Akademie, "Gamification: Spielerisch zu mehr motivation," https://www.haufe-akademie.de/digital-suite/blog/ds-gamification, 2024, accessed: 2025-12-28.

[48] D. Bandhu, M. M. Mohan, N. A. P. Nittala, P. Jadhav, A. Bhadauria, and K. K. Saxena, "Theories of motivation: A comprehensive analysis of human behavior drivers," *Acta Psychologica*, 2024, accessed: 2025-12-28.

[49] Center for Self-Determination Theory, “Theory - self-determination theory,” https://selfdeterminationtheory.org/theory/, 2024, accessed: 2025-12-30.

[50] Gamify, “Extrinsic vs. intrinsic motivation in gamification marketing,” https://www.gamify.com/gamification-blog/extrinsic-vs-intrinsic-motivation-in-gamification-marketing, 2024, accessed: 2025-12-30.

[51] FutureLearn, “The psychology of games,” https://www.futurelearn.com/info/courses/game-psychology/0/steps/428462, n.d., accessed: 2025-12-31.

[52] Red White Console, “Motivation and games: Attribution theory,” https://www.redwhiteconsole.net/motivation-games-attribution-theory/, n.d., accessed: 2025-12-31.

[53] ScienceDirect, “Expectancy-value theory,” https://www.sciencedirect.com/topics/psychology/expectancy-value-theory, n.d., accessed: 2025-12-31.

[54] A. Yu, “Improving the streak: Forming habits one lesson at a time,” https://blog.duolingo.com/improving-the-streak/, n.d., accessed: 2025-12-31.

[55] D. Jin, “Diplomarbeit,” 2025, internal Reference. Accessed: 2025-12-31.

[56] A. Marczewski, “Feedback loops, gamification and employee motivation,” https://www.gamified.uk/2013/03/25/feedback-loops-gamification-and-employee-motivation/, 2013, accessed: 2025-12-31.

[57] Motivacraft, “Feedback loops in gamification: The key to engaging user experiences,” https://www.motivacraft.com/feedback-loops-in-gamification-the-key-to-engaging-user-experiences/, n.d., accessed: 2025-12-31.