Signed by M Hussein

Erbil Polytechnic University (EPU)
زانکۆی پۆلیتەکنیکی هەولێر — جامعة بوليتيكنيك أربيل
Department of Technology Engineering | Academic Year 2024 – 2025

SECOND STAGE GRADUATION PROJECT

FINAL TECHNICAL REPORT

Scalable IoT-Based Attendance Management System

A High-Concurrency Approach using ESP32-S3 and Golang

Grade 1: 23/25

Grade 2: 23/25

Final Grade 44/50

PROJECT TEAM

Sahand Sami
Akar Ayar
Sara Swara
Derin Hazhar

SUPERVISOR

Mr. Hussein

Signature: _____

Date: ____ / _____ / 2025

EVALUATION COMMITTEE

لێژنەی هەڵسەنگاندنی پرۆژەکانی قۆناغی دووەم

#	Name / ناوی مامۆستا	Degree / بڕوانامە	Title / نازناو	Role / بەرپرسیارێتی
1	Dr. Farah Amer Abd / د. فرح عامر عبد	PhD / دکتۆرا	Professor / پرۆفیسۆر	Committee Chair / سەرۆکی لێژنە
2	M. Jwan Muhammad Ali / م. جوان محمد علی	Master / ماستەر	Assistant Lecturer / مامۆستای یاریدەدەر	Member / ئەندام
3	M. Beston Ahmed Mustafa / م. بێستون احمد مصطفی	Master / ماستەر	Assistant Lecturer / مامۆستای یاریدەدەر	Member / ئەندام

TABLE OF CONTENTS

1. Introduction

Attendance tracking is a fundamental administrative task in every academic institution. In universities and polytechnic colleges, accurate attendance data directly affects student grades, administrative decisions, and institutional compliance with regulatory requirements. Despite the critical importance of this task, the majority of academic institutions — including many departments at Erbil Polytechnic University — still rely on outdated manual methods: paper sign-in sheets, verbal roll calls, or simple spreadsheets.

This report presents the design, implementation, and evaluation of a Scalable IoT-Based Attendance Management System developed as a second-stage graduation project. The system leverages the ESP32-S3 microcontroller, NFC card technology, the Go (Golang) programming language, a SQL relational database, and the MQTT messaging protocol to deliver a professional-grade, real-time attendance solution tailored for high-density academic environments.

Unlike typical student projects that employ basic Arduino boards and simple HTTP requests, this system was designed from the ground up with production-level considerations: concurrency, data integrity, network resilience, and security. The system is entirely self-hosted on a Local Area Network (LAN), ensuring data sovereignty and eliminating dependency on external cloud services.

2. Problem Statement

A structured analysis of the existing attendance process at the department identified four critical failure points:

#	Pain Point	Impact
1	**Time Waste**	Manual roll calls and paper sign-ins consume 10–15% of each lecture's available time — equivalent to 5–9 minutes per 60-minute session. Across a full academic semester, this accumulates to several hours of lost teaching time per course.
2	**Data Loss & Integrity**	Paper records are susceptible to physical loss, accidental damage, forgery, and transcription errors. Transferring data from paper to digital systems introduces further opportunity for human error.
3	**Proxy Attendance / Fraud**	Manual systems cannot prevent one student from marking attendance on behalf of another, creating serious fairness and academic integrity violations.
4	**Lack of Real-Time Visibility**	Administrators, heads of department, and faculty have no immediate access to attendance data. Reports can only be produced after manual data entry, sometimes days after the lecture.

3. Project Objectives

The project was guided by five clearly defined technical objectives:

- Ultra-Low Latency: The end-to-end process from NFC card tap to database record confirmation must complete in under 500 milliseconds (0.5 seconds).
- High Concurrency: The backend server must handle simultaneous attendance events from multiple classrooms (clusters) without performance degradation or data loss.
- Data Integrity & Delivery Guarantee: Every attendance event must be recorded exactly once, even under poor network conditions, using MQTT QoS Level 2.
- LAN-Centric & Air-Gapped Design: All system components — server, database, and MQTT broker — must reside on the institution's internal network. No student data should ever leave the premises.
- Real-Time Administrative Dashboard: Faculty and department heads must be able to view live attendance data through a web-based interface without requiring specialist technical knowledge.

4. System Overview & Architecture

The system follows a layered, event-driven Client-Server architecture composed of three tiers:

4.1 Hardware Layer (Edge)

Each classroom is equipped with one ESP32-S3 microcontroller unit connected to an MFRC522 NFC reader module. When a student taps their NFC card, the ESP32-S3 reads the unique card UID, timestamps the event, and publishes the data to the MQTT broker over the institution's Wi-Fi network. The dual-core architecture of the ESP32-S3 is exploited: Core 0 handles Wi-Fi and MQTT communication while Core 1 is dedicated to NFC polling, ensuring zero interference between the two operations.

4.2 Messaging Layer (MQTT Broker)

An MQTT broker (Mosquitto) runs on the central server and acts as a message intermediary. All ESP32-S3 devices publish attendance events to structured topic channels (e.g., attendance/classroom-A/tap). The Golang backend subscribes to these topics and processes incoming messages asynchronously using goroutines. This decoupled architecture means adding a new classroom only requires deploying an additional ESP32-S3 unit — no changes to the server code are required.

4.3 Application Layer (Golang Backend + SQL Database)

The Golang backend server exposes both MQTT consumer logic and a REST API. Upon receiving an attendance event, the server validates the student UID against the database, writes a timestamped attendance record, and responds to the ESP32 with a confirmation signal (triggering an LED or buzzer on the device). Attendance records are stored in a normalized SQL relational database with tables for Students, Courses, Sessions, and AttendanceRecords.

4.4 Presentation Layer (Web Dashboard)

A web-based administrative dashboard allows faculty to view live session attendance, generate per-student and per-course reports, and export data. The dashboard communicates with the Golang backend via REST API calls over the LAN.

5. Hardware Design

5.1 ESP32-S3 Microcontroller

The ESP32-S3 (Espressif Systems) was selected as the edge processing unit for the following technical reasons:

- Dual Xtensa LX7 cores running at up to 240 MHz — enables true parallel task execution.
- Built-in Wi-Fi 802.11 b/g/n and Bluetooth 5 — eliminates the need for external wireless modules.
- 512 KB SRAM and 8 MB Octal-SPI PSRAM — sufficient memory for MQTT client libraries, NFC buffers, and OTA update support.
- Low power modes (deep sleep < 10 µA) — suitable for battery-backed deployments.
- Native USB support — simplifies firmware flashing and serial debugging.

5.2 MFRC522 NFC Reader Module

The MFRC522 communicates with the ESP32-S3 over SPI at up to 10 Mbps. It supports ISO/IEC 14443-A cards (MIFARE Classic, MIFARE Ultralight), which are widely available as low-cost student ID cards. The read range is 0–5 cm, providing a deliberate tap gesture that prevents accidental or passive reads. Each student's card stores a globally unique 4-byte or 7-byte UID that is used as the primary identifier in the database.

5.3 User Feedback

Each unit includes a bi-color LED and a buzzer. A green flash and single beep indicate successful attendance. A red flash and double beep indicate an unrecognised card or a card already marked for the current session, preventing duplicate records at the hardware level as a secondary safeguard.

6. Software Stack

6.1 Firmware (ESP32-S3 — C++ / Arduino Framework)

The ESP32-S3 firmware is written in C++ using the Arduino framework for ESP-IDF compatibility. Key libraries used are:

- MFRC522 library — SPI communication with the NFC reader.
- PubSubClient / arduino-mqtt — MQTT client with QoS 2 support.
- FreeRTOS — task scheduling across the two cores.

On boot, the device connects to the institution's Wi-Fi access point using stored credentials, establishes a persistent MQTT connection to the broker, and enters a continuous NFC polling loop on Core 1.

6.2 Backend Server (Golang)

The backend is written entirely in Go (Golang 1.22). Go was chosen for three specific reasons:

- Native concurrency model: Go's goroutines and channels allow the server to process thousands of simultaneous MQTT messages with minimal memory overhead — each goroutine consumes only ~2–8 KB of stack space compared to megabytes for OS threads.
- Performance: Go compiles to a native binary with no virtual machine or interpreter overhead, yielding sub-millisecond message processing latency.
- Strong standard library: Go's net/http package provides a production-grade HTTP server out of the box, reducing third-party dependencies.

The backend is structured in three packages: mqtt (broker subscriber), api (REST handlers), and db (data access layer using database/sql with a PostgreSQL or SQLite driver).

6.3 Database (SQL)

The relational database schema consists of the following core tables:

- students(id, nfc_uid, full_name, stage, department)
- courses(id, name, lecturer_id, schedule)
- sessions(id, course_id, started_at, ended_at, classroom)
- attendance_records(id, session_id, student_id, tapped_at, status)

Unique constraints on (session_id, student_id) at the database level provide a final safeguard against duplicate records, independent of the MQTT QoS guarantee.

6.4 Web Dashboard

The administrative dashboard is a lightweight web interface served by the Golang backend. It provides real-time session views via polling or Server-Sent Events (SSE), per-course and per-student attendance summaries, CSV export, and session management (open/close a session for a given classroom).

7. Network Strategy — LAN Architecture

A deliberate design decision was made to host all system components exclusively on the institution's Local Area Network. This LAN-centric strategy provides two critical advantages:

7.1 Security & Data Sovereignty

Student biometric-adjacent data (NFC UIDs linked to identities) never leaves the college premises. There is no exposure to external internet threats, no dependency on third-party cloud providers, and no risk of data breaches originating outside the institution. This approach is aligned with best practices for personally identifiable information (PII) management.

7.2 Reliability & Independence

The system operates entirely independently of internet connectivity. A university-wide internet outage has zero impact on attendance recording. The MQTT broker and backend server can run on a standard low-power server or even a Raspberry Pi 4, making deployment cost-effective. Static IP addressing is configured for the server, and the ESP32-S3 devices use DHCP reservation to ensure consistent connectivity.

8. Communication Protocol — MQTT

8.1 Why MQTT over HTTP?

While HTTP/REST is the default choice for web applications, it is ill-suited for constrained IoT devices. MQTT (Message Queuing Telemetry Transport) was designed specifically for low-bandwidth, low-power, high-reliability machine-to-machine communication. Key advantages in this context:

Attribute	HTTP/REST	MQTT
Protocol overhead	High (full HTTP headers per request)	Minimal (2-byte fixed header)
Connection model	Request-response (stateless)	Persistent TCP connection
Delivery guarantee	None (application layer)	Built-in QoS 0 / 1 / 2
Broker/fan-out	Not native	Native publish-subscribe
Power consumption	Higher (new TCP handshake each time)	Lower (keep-alive pings)

8.2 QoS Level Selection

MQTT defines three Quality of Service levels. This system uses QoS 2 for all attendance events:

- QoS 0 — At most once (fire-and-forget): fastest but messages may be lost. Unsuitable for attendance.
- QoS 1 — At least once: guaranteed delivery but possible duplicates. Requires application-level deduplication.
- QoS 2 — Exactly once: uses a four-part handshake (PUBLISH → PUBREC → PUBREL → PUBCOMP) to guarantee the message is delivered exactly once. This is the correct choice for attendance, where duplicate or lost records have direct academic consequences.

QoS 2 adds approximately 2–4 round-trip messages of overhead, which on a LAN adds only 1–3 ms of latency — well within the 500 ms target.

8.3 Topic Structure

Topics follow the convention: attendance/{building}/{room}/tap

Example: attendance/block-A/room-201/tap. The Golang subscriber uses a wildcard subscription (attendance/#) to receive events from all classrooms with a single subscription, then extracts location context from the topic string.

9. Security Considerations

- NFC UID is not directly stored as a password. UIDs are hashed and mapped to student records — a raw card read cannot directly expose a student's identity if the database is accessed without application context.
- MQTT broker is configured with username/password authentication. Each ESP32-S3 device has unique credentials. TLS can be enabled over the LAN for additional protection.
- The Golang REST API uses token-based authentication for dashboard access, preventing unauthorised users from querying attendance data.
- Database access is restricted to the local server process. No database ports are exposed on the LAN.
- Anti-replay: The server maintains a per-session set of already-recorded UIDs. Even if a duplicate QoS 2 message were somehow delivered, the application layer rejects it and returns a 'already marked' response to the device.

10. Testing & Performance

10.1 Latency Testing

End-to-end latency was measured from the moment of NFC tap to the moment the ESP32 received the server acknowledgement (LED/buzzer trigger). Results across 100 test taps on the LAN:

Metric	Value	Target
Average latency	112 ms	< 500 ms
Minimum latency	78 ms	< 500 ms
Maximum latency	247 ms	< 500 ms
99th percentile	310 ms	< 500 ms

10.2 Concurrency Testing

A load test was conducted by simulating 50 simultaneous MQTT publishes (representing 50 classrooms tapping at the same instant) to the Golang backend. All 50 messages were processed and confirmed within 380 ms, with zero data loss or duplicate records. The Go runtime's goroutine scheduler handled all 50 concurrent handlers using fewer than 15 MB of additional RAM.

10.3 Data Integrity

A network interruption test confirmed QoS 2 resilience: the server was shut down mid-session, the network was restored, and the server was restarted. All pending messages held by the MQTT broker were delivered exactly once upon reconnection, with no duplicates in the database. The unique constraint on (session_id, student_id) rejected one artificially injected duplicate in testing, confirming the multi-layer integrity design.

11. Future Work & Scalability

- Cloud Hybrid Mode: An optional synchronisation daemon could push anonymised attendance statistics to a cloud analytics platform (e.g., AWS IoT Core) at end-of-day, enabling institution-wide dashboards without real-time cloud dependency.
- Mobile Application: A companion mobile app for students to view their own attendance records and receive absence alerts via push notification.
- Biometric Integration: Integration with a fingerprint module (e.g., R307) as a secondary authentication factor on high-security sessions, in addition to NFC.
- Machine Learning Anomaly Detection: Analysis of attendance patterns to detect anomalies (e.g., a student whose NFC card is used in two different classrooms simultaneously) and flag potential proxy attempts.
- Containerisation: Packaging the Golang backend, MQTT broker, and database in Docker containers for simplified deployment and reproducibility across different server hardware.
- OTA Firmware Updates: Implementing over-the-air firmware update capability for the ESP32-S3 fleet, enabling security patches and feature updates without physical access to each device.

12. Conclusion

This project has successfully demonstrated that a professional-grade, scalable, and secure IoT attendance management system can be designed, built, and validated within an academic setting. By combining the ESP32-S3's dual-core edge processing capability, Go's exceptional concurrency model, MQTT's reliable publish-subscribe protocol, and a LAN-first network philosophy, the team has produced a system that measurably outperforms traditional attendance methods on every relevant dimension: speed, reliability, security, and administrative utility.

The system consistently achieved end-to-end attendance confirmation in under 300 ms on average — well below the 500 ms target — and demonstrated the ability to process 50 simultaneous events without data loss. All attendance records were delivered with exactly-once semantics, ensuring academic fairness. Student data remained entirely within the institution's network perimeter throughout all testing.

This project provides a practical, deployable blueprint for modernising academic administration at Erbil Polytechnic University and similar institutions across the Kurdistan Region. The modular architecture ensures that future enhancements — from biometric integration to cloud analytics — can be added incrementally without requiring a system redesign.

13. References

[1] Espressif Systems. (2023). ESP32-S3 Technical Reference Manual. https://www.espressif.com/

[2] OASIS Standard. (2019). MQTT Version 5.0 Specification. https://docs.oasis-open.org/mqtt/

[3] Eclipse Foundation. (2023). Eclipse Mosquitto: An Open Source MQTT Broker. https://mosquitto.org/

[4] The Go Programming Language Authors. (2024). The Go Programming Language Specification. https://go.dev/ref/spec

[5] NXP Semiconductors. (2016). MFRC522 Standard Performance MIFARE and NTAG Frontend Datasheet.

[6] Banks, A., & Gupta, R. (2014). MQTT Version 3.1.1 — OASIS Standard. https://docs.oasis-open.org/

[7] Grigorik, I. (2013). High Performance Browser Networking. O'Reilly Media.