UNIVERSITY DE SÃO PAULO
FACULTY OF PHILOSOPHY, SCIENCES AND LETTERS AT RIBEIRÃO PRETO
DEPARTMENT OF COMPUTER SCIENCE AND MATHEMATICS

GABRIEL CARVALHO SILVA

An IoT Smart Scale Proof of Concept for Smart Homes

Ribeirão Preto–SP

2025

A minha avó, Maria Daria Rocha, e ao Gabriel que escrevia códigos no papel.

Acknowledgements

I would like to express my deepest gratitude to my advisor, PhD. Cléver Ricardo Guareis de Farias, for their continuous support and patience.

My sincere appreciation also goes to the University of São Paulo for providing me with unique opportunities. All the experience I gathered while a student there cannot be measured.

I appreciate my mom, Rozirene Rocha de Carvalho for doing the best she could, she is a superhero, and I appreciate my grandparents and grandmothers, for teaching me affection. I appreciate my family as a whole for showing me what to be and what not to become.

I appreciate all the hardships encountered along the way, and all the good fortune that crossed my path until here. The good and the bad made me who I am, and I am specially grateful for the rejections and failures because they taught me my limits and they guide me to where I belong.

I am grateful for having wonderful friends, and I am specially grateful for Carla de Araújo Clementino Ribeiro and Matheus Sampaio Lopes, the two best friends anyone could ever hope for.

Thank you to all my teachers, professors, tutors, mentors, and any other sort of educator I found in life.

Thank you to my godfather and godmother for being as present in my life as it was humanly possible and thank you to my brother for never ever letting me give up.

Finally, I must express my very profound gratitude to my partner, Beatriz de Souza Silva, for providing me with unbelievable support and continuous encouragement daily, and to my father, MSc. Wilbert Carpi Silva, who taught me my very first lines of code through a telephone. This accomplishment would not have been possible without them.

“E ao vencedor, as batatas ”
(Quincas Borba)

Abstract

This document describes the development of an Internet of Things (IoT) smart scale proof-of-concept (PoC) for smart home integration. The motivation for this work is to address key barriers to adoption in the field, specifically high costs, privacy concerns, and system scalability. This project aims to build a custom architectural solution to demonstrate a cost-conscious, privacy-by-design approach. The developed system consists of an embedded scale with facial recognition capabilities, a service layer for data management, and a client dashboard for data visualization and configuration. The design emphasizes an event-driven architecture using Message Queuing Telemetry Transport (MQTT) to ensure efficient and adaptive communication. Ultimately, this PoC will serve as a practical demonstration of best practices in the design of IoT systems and of the application of novel approaches in edge computing, offering valuable insight into interoperability and performance trade-offs for smart home health devices.

Keywords: Internet of Things (IoT), Smart Homes, Embedded Systems, Facial Recognition, Adaptive Systems, Event-driven Architecture, MQTT, Proof-of-Concept

List of figures

Figure 1 – Generic IoT Architecture. 8
Figure 2 – System overview and communication routes . . . 16
Figure 3 – Physical layout of the scale 17
Figure 4 – Polynomial Regression resulting curve 19
Figure 5 – State Machine Diagram of the ESP32 workflow . 20
Figure 6 – Event flows 22
Figure 7 – MQTT Broker Topics and workflow 23
Figure 8 – Entity-Relationship Diagram of the Database . . 24
Figure 9 – Hexagonal Architecture of the HubService 25
Figure 10 – Dashboard Sequence Diagram for Retrieving Measurements of a given Profile 26
Figure 11 – Dashboard Profile creation 28
Figure 12 – Subject weighin themselves on scale 29
Figure 13 – Dashboard Profile measurement 30

List of tables

Table 1 – Raw ADC Readings vs. Known Weights 18
Table 2 – HubService API Endpoints 24

List of abbreviations and acronyms

ADC	Analog to Digital Converter
BMI	Body Mass Index
DBMS	Database Management System
EDA	Event-Driven Architecture
FR	Functional Requirement
GUI	Graphical User Interface
HePA	Hexagonal Platform Architecture
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
KPI	Key Performance Indicator
LAN	Local Area Network
mAP	mean Average Precision
MQTT	Message Queuing Telemetry Transport
NFR	Nonfunctional Requirement
PoC	Proof-of-Concept
REST	Representational State Transfer
TCP/IP	Transmission Control Protocol/Internet Protocol
UUID	Universally Unique Identifier

Summary

1 INTRODUCTION 1
1.1 Background 1
1.2 Objective 2
1.3 Methodology 2
1.4 Structure of the document 3

2 THEORETICAL FOUNDATION 5
2.1 Internet of Things 5
2.2 IoT Sensors 6
2.3 IoT System Architectural Styles 7
2.4 Smart Homes 8
2.5 Smart Scales 9
2.6 Privacy in IoT and Face recognition 10

3 THE PROOF OF CONCEPT (POC) 13
3.1 Project specification 13
3.2 PoC Overview 14
3.3 PoC edge layer 16
3.4 PoC fog layer 20
3.5 PoC cloud layer 23
3.6 Smart Scale Face Recognition 26
3.7 Project Evaluation 26

4 CONCLUSION 31
4.1 Contributions 31
4.2 Discussion 32
4.3 Future work 33

References 35

1

Introduction

1.1 Background

The history of the Internet is a story of continuous expansion and integration. Starting in the late 1960s with ARPANET, a network for government and academic use, it was a tool for sharing information across an enclosed group of institutions and people. The adoption of the TCP/IP protocol in the early 1980s and the birth of the World Wide Web in the 1990s democratized this connectivity, paving the way to a modern digital world. Moreover, with the rise of affordable sensors, ubiquitous wireless technologies like Wi-Fi and 5G, and the vast processing power of cloud computing, the Internet's reach extended beyond traditional computers to encompass everyday objects. This transformation allowed devices to collect, share, and act on data autonomously, laying the groundwork for a "network of things", or Internet of Things (IoT). This transformation has been particularly impactful in the healthcare domain, where everyday devices have evolved from simple measurement tools into interconnected monitoring systems.

Internet-connected things include thermostats that can be controlled remotely from smartphones and smart body scales that allow one to graphically review the progress of diets using smartphones, for example. Moreover, smart scales detect gradual weight changes and, when integrated with smart home systems, create comprehensive health monitoring environments. This continuous data stream allows, for example, for healthcare agents to intervene earlier and more precisely, potentially reducing hospital admissions and healthcare costs (PANJA et al., 2025).

However, despite the increasing interest in IoT, the development of cost-effective IoT solutions currently face many different challenges. For instance, privacy features in many existing IoT development frameworks are relatively limited (JIN; KUMAR; HONG, 2020a), which affects, for example, smart scale solutions reliability. Besides that, handling IoT sensor data, especially in terms of processing and integration with other data

sources, has its own setbacks (SATHYAMOORTHY et al., 2024). Likewise, there is no ground truth for project design and architecture, which raises the question of which service composition mechanism best fulfills the functional scalability requirements of IoT systems (ARELLANES; LAU, 2020). On top of that, buying a scale can be financially challenging, particularly for low-income individuals, and scales with advanced features cost significantly more than scales without these features (PARK et al., 2021).

1.2 Objective

Within this context, and specifically to address the barriers of cost, privacy, and scalability identified above, this project aims to deliver a working prototype of a smart scale with face recognition capability. Therefore, it consists of an embedded system for weighing a subject and capturing their face image, a web service for managing weight records and performing face recognition, and a dashboard for measured data visualization.

The development of the Proof-of-Concept (PoC) system takes into account cost and privacy concerns, as well as system scalability, considering different possible architecture styles (MARTINO et al., 2018). It also explores and validates different aspects of such a system and their specific challenges following novel approaches from literature.

1.3 Methodology

This work followed the literature review to guide the understanding of current state of the art regarding IoT Systems, specially related to smart scales and privacy-aware scopes. Therefore, in order to follow up with the design and implementation of the system and subsequent evaluation of it, we opted for the ESP32 (Espressif Systems, 2024) (the ESPCAM model specifically) as an accessible option for hardware solution and set a requirement for ESP32 compliant face embeddings extraction methods. Not only that, the architecture was defined by considering simplicity and extensibility capabilities besides functional requirements.

Several architectural styles were considered from current state of the art (MARTINO et al., 2018), leading to the adoption of a reactive event-driven architecture. This decision is aligned with the nature of the expected functionality of a smart scale, in which retry policies were implemented for when face recognition failed, and in which system components can be decoupled for extensibility.

Therefore, the ESP32 was calibrated for weight measurement and implemented the face embedding extraction using pre-trained models, an event queue was used to bridge

system components, a web service was developed to manage data acquisition and retrieval and a dashboard was also created to visualize weight measurements.

Finally, the usage of the system as a whole was put under daily tests leading to conclusions regarding the actual feasability of the solutions for the challenges concerning cost, privacy and performance.

1.4 Structure of the document

The following chapters explore in more depth the various aspects of this project: Chapter 2 reviews literature and related work; Chapter 3 details the PoC development; and Chapter 4 brings to light the findings of this project.

2

Theoretical Foundation

2.1 Internet of Things

The Internet of Things (IoT) can be defined as an open and comprehensive network of intelligent objects that have the capacity to organize themselves, share information, react and act in situations faced and changes in the environment (MADAKAM, 2015). IoT consists of an inter-network of physical devices like vehicles, buildings, and other items embedded with electronics, sensors, actuators, software, and network connectivity that allow these objects to collect and exchange data (JAIN; TANWAR; MEHRA, 2019).

Naturally, communication is a key layer upon which any IoT solution is built and it involves a careful consideration of both the physical infrastructure and the logical architecture. At the physical layer, devices need to communicate wirelessly, and the choice of technology depends heavily on the specific application's requirements for bandwidth, range, and power consumption. For example, Wi-Fi is a common choice for many smart home devices due to its high bandwidth, which is essential for data-intensive tasks like streaming video from a security camera, and its widespread availability. However, Wi-Fi is also relatively power-intensive, which is a major drawback for battery-operated devices that need to run for months or years without a charge (MOCRII; CHEN; MUSILEK, 2018).

In contrast, Bluetooth, and its more energy-efficient variant, Bluetooth Low Energy (WOOLLEY, 2019), is optimized for low-power, short-range communication, making it an ideal choice for wearable devices, fitness trackers, and other battery-powered sensors. Still, other emerging technologies such as LoRa (SORNIN et al., 2015), which is specifically designed for long-range, low-power communication, are more suited for applications in smart cities or large-scale industrial IoT, as highlighted by Kane *et al.* (KANE et al., 2022).

Beyond the physical layer, the logical architecture of communication is crucial to

ensuring scalability, efficiency, and responsiveness in a heterogeneous IoT environment. The traditional request-response model, often implemented via HTTP/REST API calls, is a synchronous pattern in which a client sends a request to a server and waits for a response. This model is well-suited for many web applications but can be inefficient in an IoT context where devices may be intermittently connected and need to send data proactively rather than waiting for a request.

A more robust and scalable solution for IoT is an event-driven architecture (EDA). According to this paradigm, devices and services operate asynchronously, communicating through the publication and subscription of "events." For example, when a user steps on the scale, the embedded system publishes a "weight-measured" event to a central message broker. Other services, such as a data processing worker or the face recognition service, that are interested in this event are automatically notified.

This publish-subscribe model, which is often facilitated by a lightweight messaging protocol such as Message Queuing Telemetry Transport (MQTT) (OASIS, 2019), is highly advantageous for IoT systems. MQTT is designed for resource-constrained devices and low-bandwidth, high-latency networks, making it an ideal fit for the project. An EDA allows for loose coupling between components, meaning a new device or service can be added to the system without requiring changes to existing components. This modularity is a key factor for scalability and adaptability. In addition, it enables adaptive, context-aware data acquisition strategies.

In a system with low-frequency sensor data (weight) and high-latency operations (image processing for facial recognition), an event-driven model can optimize bandwidth and energy consumption. The system can be configured to only publish data when a significant change occurs (e.g., a weight measurement crosses a certain threshold) or when a specific condition is met, avoiding the need for continuous, wasteful polling. This adaptive approach is central to the proposed system's design, aiming to improve responsiveness and reduce resource consumption in a real-world smart home environment.

2.2 IoT Sensors

While the communication infrastructure enables IoT devices to exchange information, the actual data acquisition depends on the sensors embedded within these devices. IoT devices are also called Smart Things, which can also be perceived as physical objects connected to the web with some sensing capabilities (MADAKAM, 2015). Sensing is made possible by pairing embedded devices with sensors, actuators and other sorts of appendixes. Sensor are crucial for acquiring data and providing context for the interoperability of different IoT devices (PANJA et al., 2025).

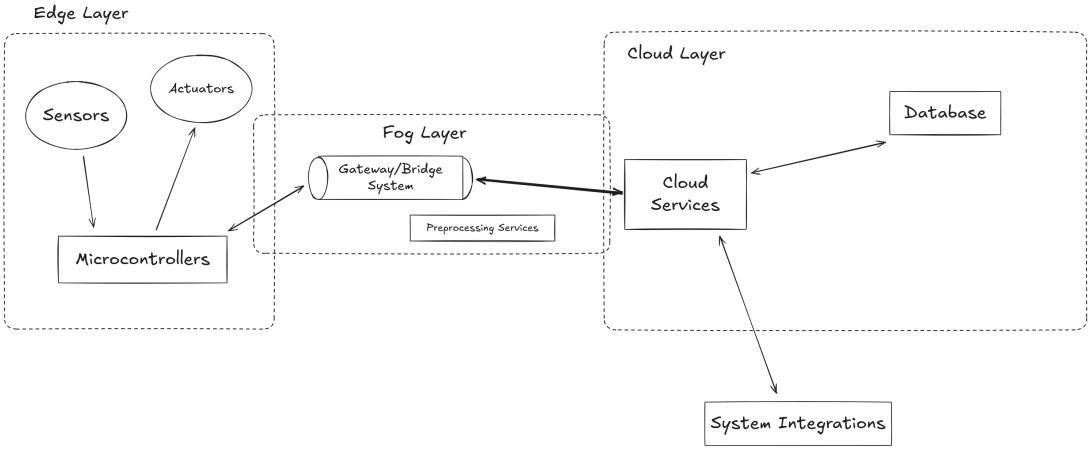
Sensors can be of many kinds and serve many purposes. Temperature, umidity and proximity sensors are some examples of it. Weight sensors specifically are commonly a composiiton of load cells. The most widely used load cell is the strain gauge which is a thin foil resistor that alters the voltage as it is strained, hence its name. Strain gauges measure variations in voltage once they are deformed, because the intensity of the electrical resistance variation in the strain gauges is proportional to the intensity of the applied force that deform them (MULLER et al., 2010). This is paramount for IoT systems due to its simplicity and several application possibilities.

2.3 IoT System Architectural Styles

Figure 1 shows a simplification of an IoT architecture. Each and every component of this diagram is an essential part of most IoT systems. In the edge layer, sensors (devices that transforms physical action in data), actuators (devices that transform energy into a physical action) and other smart appliances work together to gather important data and react to what is perceived in the environment. In order to scale that, however, and provide even more possibilities, cloud solutions integrate with the edge devices to achieve an ubiquitous experience. This integration is only possible through the implementation of bridges, such as network gateways, queue services, intermediate services, all those which constitute what is called a fog layer (MOCRII; CHEN; MUSILEK, 2018).

Still, different architectural approaches shift focus through the layers, putting more emphasis in one or another depending on the goal in mind and the applications desired. The cloud computing approach offers high processing power and storage option, while edge computing is more limited when it comes to computing power. However, edge computing achieves privacy measures more easily than the cloud. Nevertheless, fog computing is another approach that takes processing power closer to embedded devices on the edge, while still keeping the cloud layer unware of sensitive information. Therefore, depending on the goals for a project, the placement of system components and tasks should be placed accordingly to what every computing layer offers and limits. This PoC focuses on the edge layer as an enabler for private interactions with smart gadgets.

Figure 1 – Generic IoT Architecture.



Source: Author

2.4 Smart Homes

A smart home system forms when interconnected devices, embedded with electronics, sensors, software, and network connectivity, work within a household. Mocrii *et al.* (MOCRII; CHEN; MUSILEK, 2018) define this system as having complementary user and system functions built upon a general IoT-based architecture. The devices themselves carry embedded intelligence, identification, and automation capabilities designed to assist human life.

Seo *et al.* (SEO et al., 2015) proposed the Hexagonal Platform Architecture (HePA) as a reference architecture specifically designed for the complex, interconnected nature of the IoT era, including smart homes. Their implementation was a platform architecture model that aimed for extreme scalability while maintaining required performance. The authors acknowledges the primary challenge as the advent of the complex IoT era, where all devices are interconnected, requiring enormous amounts of interaction. Further, they showcase a generic application, making extensive use of ports and adapters (hexagonal pattern) to integrate different data sources and data destinations, as well as third-party services, and by doing so, it presents solid foundation for a reference architecture pattern to be followed.

Jin *et al.* (JIN; KUMAR; HONG, 2020b) proposed the Peekaboo framework to provide architectural support for building privacy-sensitive smart home applications following homomorphic (a structure-preserving map between two algebraic structures of the same type) encryption of sensitive data. Their implementation philosophy moves preprocessing tasks (e.g., face detection) from the cloud onto a user-controlled hub. They achieved this by extracting image embeddings before sending data to the cloud. The authors primarily addressed the challenge of reducing data egress and minimizing potential

privacy risks by preventing raw data from leaving the user's control.

2.5 Smart Scales

The evolution of personal weighing devices from mechanical to electronic allowed for better precision, ease of use and extra functionalities. The foundational technology of a modern electronic scale is a load cell, a transducer that converts mechanical force into an electrical signal. When an individual stands on the scale, a strain gauge undergoes a slight deformation. This deformation alters the electrical resistance of the gauge in a measurable way. An analog-to-digital converter processes this change, translating it into a precise digital weight value for display. While this technology significantly improved accuracy and usability over mechanical scales, its utility was confined to providing a single, instantaneous weight measurement.

The smart scale expands upon this foundation by integrating additional sensors and a communication module. These supplementary means provide a way to reshape the usage of a scale, or even add new functionality to it.

The connectivity of smart scales, typically through wireless protocols such as Bluetooth or Wi-Fi, is what defines their "smart" functionality. This capability facilitates the automatic and seamless transmission of collected data to a companion application or cloud-based service. This automated data flow eliminates the need for manual record-keeping, thereby supporting long-term, continuous health tracking. The compiled data can be visualized and analyzed over time, which supports a shift from reactive to preventive healthcare.

The adoption of Smart Scales provide a number of benefits, one of which being the possibility for health professionals to remotely track health indicators of their patients. Allied with other smart gadgets, severe health issues can be avoided such as heart attacks (PANJA et al., 2025). Neverheless, the adoption of this technology faces challenges. Research by Mafong *et al.* (MAFONG; KIM; FUJIOKA, 2020) indicates that while there is a general willingness to use smart scales, affordability remains a significant barrier for many consumers. The study found that a notable portion of potential users were unwilling or unable to purchase such a device, highlighting the need for cost-conscious development.

Hasti *et al.* (HASTI; PERMATA; ARIBOWO, 2025) developed an IoT-based digital weighing scale prototype to address the growing health concern of obesity. Their implementation utilized load cell sensors, an HX711 (Avia Semiconductor, 2010) amplifier, and an ESP8266 microcontroller for weight measurement. A companion Flutter-based application (MyWeightApp) connected to Firebase provided data storage, visualization, and real-time tracking of Body Mass Index (BMI) calculation. The primary implementation

challenge involved ensuring hardware accuracy; however, the authors' testing demonstrated a low 0.78 percent error rate, well within the tolerance threshold.

Jaiteh *et al.* (JAITEH et al., 2019) designed a multipurpose smart tracking system that functions as both a weighing scale and a human tracking system using gait analysis. The implementation featured a sensing platform built with eight load cells and an amplifier, which fed analog signals to an Arduino microprocessor for data processing, analysis, and representation. The authors powered the system with either a 9V battery or solar energy, indicating a focus on power efficiency and standalone operation. Their implementation focused on calibrating the load cells and testing the sensing platform's precision and accuracy against various static weights and different individuals.

Zargham *et al.* (ZARGHAM et al., 2023) introduced an Intelligent IoT-based Scale to automate the sales process for fruits and vegetables in small-scale retail. The implementation used a load cell with an HX711 amplifier for accurate weighing and integrated advanced computer vision using fine-tuned YOLOv5n and YOLOv7 models for item detection and identification. A Python script handled the pricing logic, and the authors developed a Graphical User Interface (GUI) for customer display and bill generation. The main implementation challenge was achieving high accuracy and efficacy in real-time processing. Still, their models achieved high mean Average Precision (mAP) scores (0.98 and 0.987) and high processing speeds.

2.6 Privacy in IoT and Face recognition

The explosive growth of the Internet of Things, particularly within the intimate setting of the smart home, has introduced a new and complex set of privacy and security challenges. Unlike traditional computing devices, IoT devices are often embedded into everyday objects, collecting vast amounts of granular, and often highly sensitive, personal data without the user's continuous, conscious interaction. This data can range from health metrics and daily routines to audio and video recordings captured by devices like smart speakers and cameras. The collection, transmission, and storage of this sensitive information create a vast surface area for potential security vulnerabilities and privacy breaches.

A central issue is the lack of privacy-by-design principles in many commercially available IoT devices and their corresponding development frameworks. This can lead to a host of security weaknesses, including weak authentication mechanisms, the transmission of unencrypted data, and an absence of user controls for managing personal information. The decentralized and heterogeneous nature of IoT ecosystems further complicates matters. A smart home can consist of devices from multiple manufacturers, each with its own security standards and data handling policies, making it difficult for a user to have

a complete understanding and control over their data.

The use of biometric data, such as facial recognition in the context of the proposed Proof-of-Concept (PoC) smart scale, introduces a particularly acute privacy risk. If a biometric database is compromised, the user's identity is permanently at risk. This is a critical area that requires advanced security solutions. The work of Elordi *et al.* (ELORDI et al., 2021) offers a compelling example of how to address this challenge. They propose a system that uses homomorphic encryption to protect this sort of data securely for elderly care applications. Homomorphic encryption allows computations to be performed on encrypted data without the need to decrypt it first. In the context of facial recognition, this means that the face matching process can occur on a server without the server ever having access to the unencrypted biometric template. This approach provides a powerful layer of privacy protection, as even if a database were to be breached, the data would remain encrypted. This PoC should build upon this by exploring secure data handling for facial recognition within a cost-conscious, smart home-oriented architecture, aiming to demonstrate how such advanced privacy measures can be integrated into a practical PoC.

3

The Proof of Concept (PoC)

This PoC consists of an embedded system with integrated sensors capable of weighing and identifying a subject alongside complementary services for face recognition and data storage. It also offers a simple dashboard interface for managing subject registration (referred as profiles) and visualizing available weight measurements of them realized with the sensor mentioned. This section unfolds the details of designing and implementing the whole system and presents the decision process for every step and piece of it.

3.1 Project specification

Functional Requirements

The functional requirements (FRs) for the system define what it does from certain actor perspectices. The following functional requirements were identified:

1. The user shall be able to create profiles through a frontend client
    a) When creating a profile the user shall be able to name the profile
    b) When creating a profile the user shall be able to send a picture to be used for face recognition of the subject related to that profile
2. The user shall be able to access the available profiles and their available measurements
3. The user shall be able to measure their weight by stepping up on the scale when stepping up on the scale the system should then use its camera to identify the subject.

Nonfunctional Requirements

Nonfunctional Requirements (NFRs) layout the qualities of a system, or *how* it should perform its functionalities. For this PoC, the following NFR were set:

1. The weight measurement must be accurate within an 100 grams margin of error;
2. The system shall account for measurement or face matching errors and support retry policies;
3. The system must be easily extensible to consider the possible adition of extra sensors and connection to outside systems;
4. Privacy is paramount and sensitive data such as biometric data shall never be saved in its raw format.

3.2 PoC Overview

With the functional and nonfunctional requirements established as design constraints, this section presents the architectural approach developed to satisfy them.

The system consists of three layers: the edge, the fog, and the cloud layer. This layered architecture directly addresses the project's core requirements: the edge layer ensures privacy through local biometric processing (NFR 4), the fog layer enables extensibility through event-driven messaging (NFR 3), and the cloud layer provides data management and visualization capabilities (FR 2). The edge layer consists of a scale, an HX711 signal amplifier and an ESP32. The scale itself consists of a Wheatstone bridge built with straing gauge load cells, connected using standard copper wires. The load cells are then connected to an HX711 signal amplifier that is then connected to the ESP32.

The ESP32 is callibrated using known weights and programmed to poll variations in voltage caused by deformation of the strain gauges when a weight is placed on top of them. The ESP32 module comes with a camera that captures image once a given weight threshold is beated. The microcontroller calibration was done through the polynomial regression of the acquired ADC (voltage) value and respective known weight values, resulting in a curve that is used to map ADC values to weight measurements.

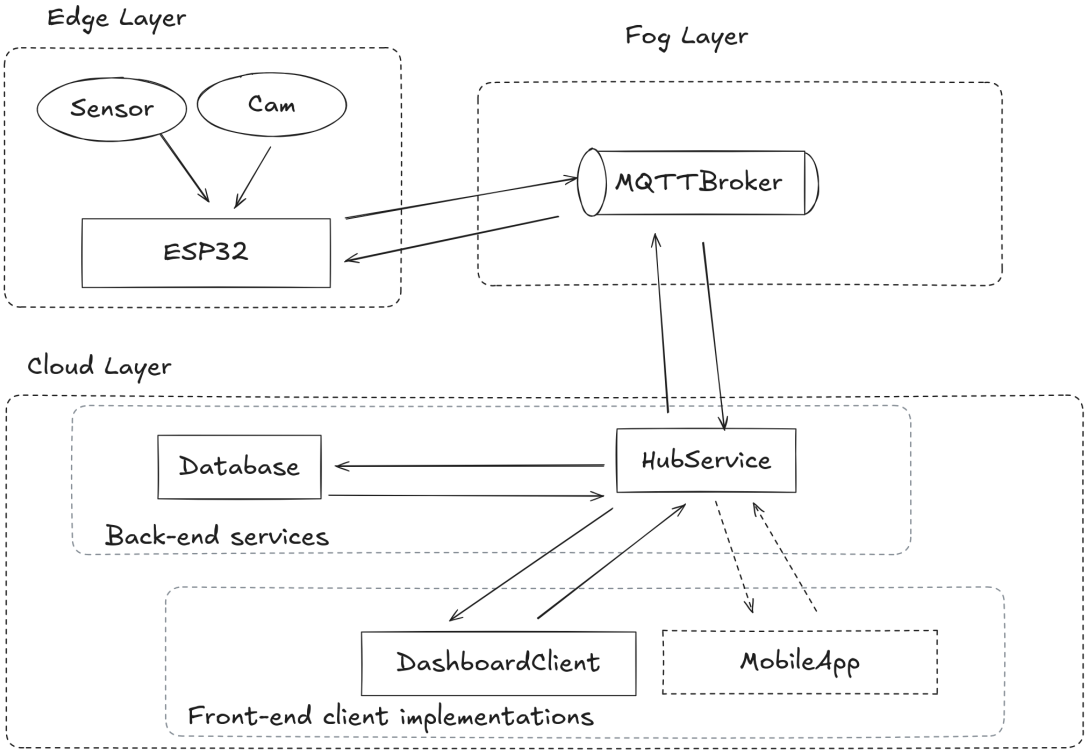
Both image feature vector and weight value are sent to an MQTT broker, which serves as a queue service. For the purpose of this PoC, the broker was set up in a personal computer and executed under a **docker container** for abstraction (Docker Inc., 2025), but it could be deployed on a RaspberryPi or even another ESP32 for its lightweight nature.

The fog layer consists of the MQTT broker which acts as the gateway between the embedded system and its gathered data users (cloud layer services). Nevertheless,

the cloud layer includes both a Hub Service for managing measurements in the database and a dashboard for visualizing available measurements by profile. Still, the broker could arguibly be moved to either the edge or the cloud layers, however, it was kept in the fog layer since it represents a hub for any other smart things in the household, meaning that before sharing their data to the cloud, they would pass through the broker, which then acts as a Gateway.

In order to accomodate the FRs and NFRs, many engineering decisions were made based on literature review and good practices in software engineering and architecture. The embedded system is an ESPCAM (an ESP32 with built-in camera and SD card support) and its software was written in C, using Espressif libraries for MQTT and TensorFlow Lite pre-trained models for extracting face feature vectors from camera imaging (Google Developers, 2025). The MQTT broker is the Mosquitto by Eclipse implementation (Eclipse Foundation, 2025). Within the cloud layer, the data storage is implemented using PostgreSQL (The PostgreSQL Global Development Group, 2025) running pgvector (KANE, 2025) plug-in for ease querying and storage of feature vectors. Still within the cloud layer, the Hub Service is implemented following the hexagonal architectural pattern and uses Kotlin (JetBrains, 2025) with the Spring Boot framework (VMware Tanzu, 2025) for providing a web API and an MQTT client. Finally, we implemented a dashboard in React for easier visualization of the measurements (Meta Open Source, 2025). Figure 2 shows an overview of the whole solution. Notice in the figure **MobileApp** is not part of this PoC, but it exemplifies how the system could be easily extended.

Figure 2 – System overview and communication routes

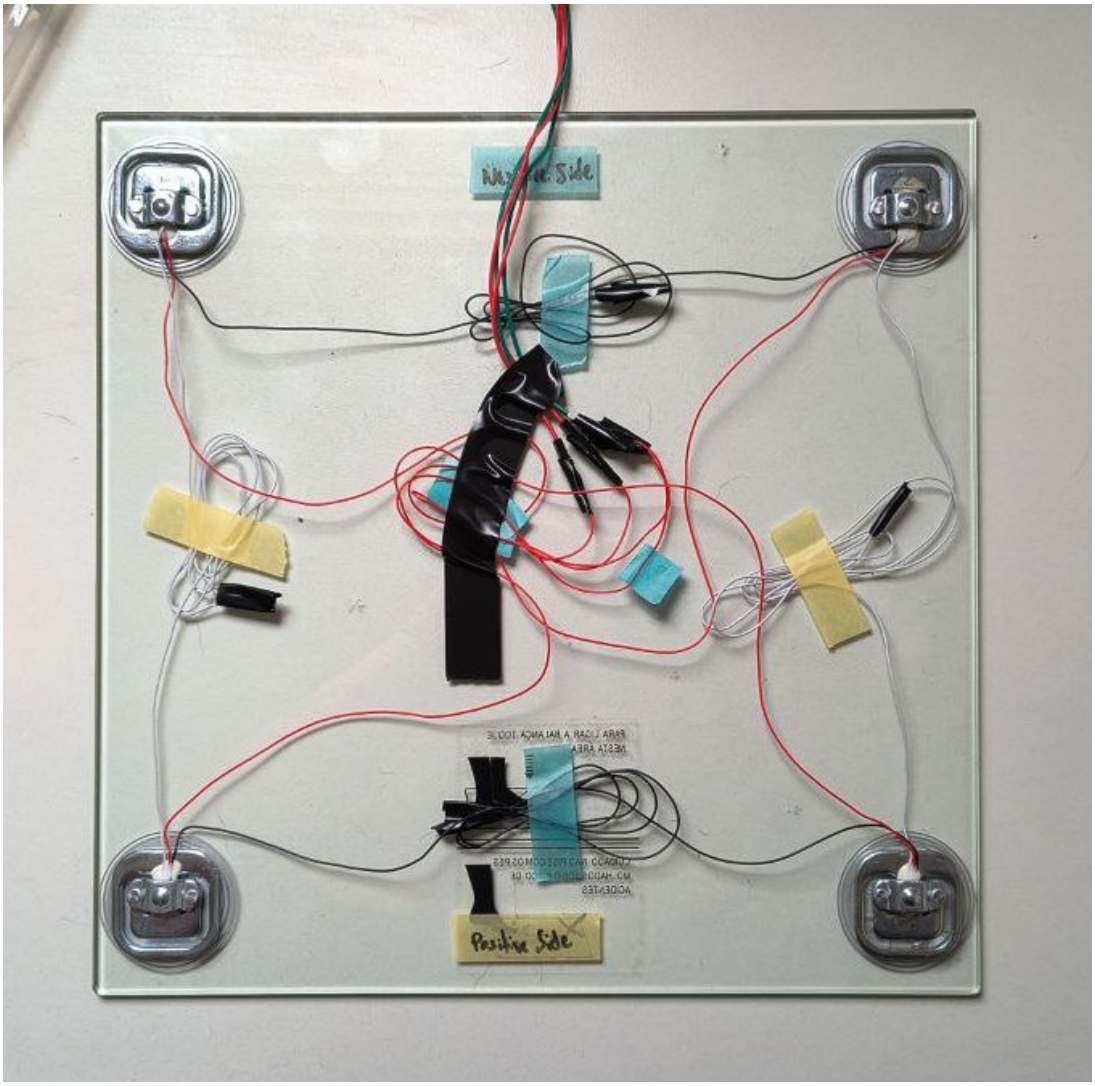


Source: Author

3.3 PoC edge layer

The **edge layer** contains the embedded solution with a weight sensor and a camera attatched to it. The solution was developed using an ESP32 microcontroller embedded with a 2 megapixel camera (ESPCAM), alongisde an HX711 signal amplifier chip. Figure 3 shows the actual load cell layout in the PoC, which contains 4 resistors wired together to measure up to 200Kg.

Figure 3 – Physical layout of the scale



Source: Author

The ESP32 is aware of voltage variation to infer weight values based on callibrated reference values. Once a threshold is perceived, the camera is activated and an image is taken of the subject being weighed. Both weight and image information are sent to the MQTT broker.

The callibration of the system was made through the measurement of voltage for known weight values by gradatively adding water to a bottle, measuring its weight with a kitchen scale and then measuring the difference in voltage perceived by the microcontroller. Those values are then fit to a curve through polynomial regression. The resulting sum of powers defines the function that maps Analog to Digital Converter (ADC) voltage read values to weight values. Table 1 list 9 samples of ADC readings and their respective already known weights.

Table 1 – Raw ADC Readings vs. Known Weights

ADC Read Voltage	**Known Weight (in grams)**
46 623	0
46 680	34
47 220	310
47 175	427
48 030	630
47 300	763
48 500	973
48 790	1284
48 860	1579

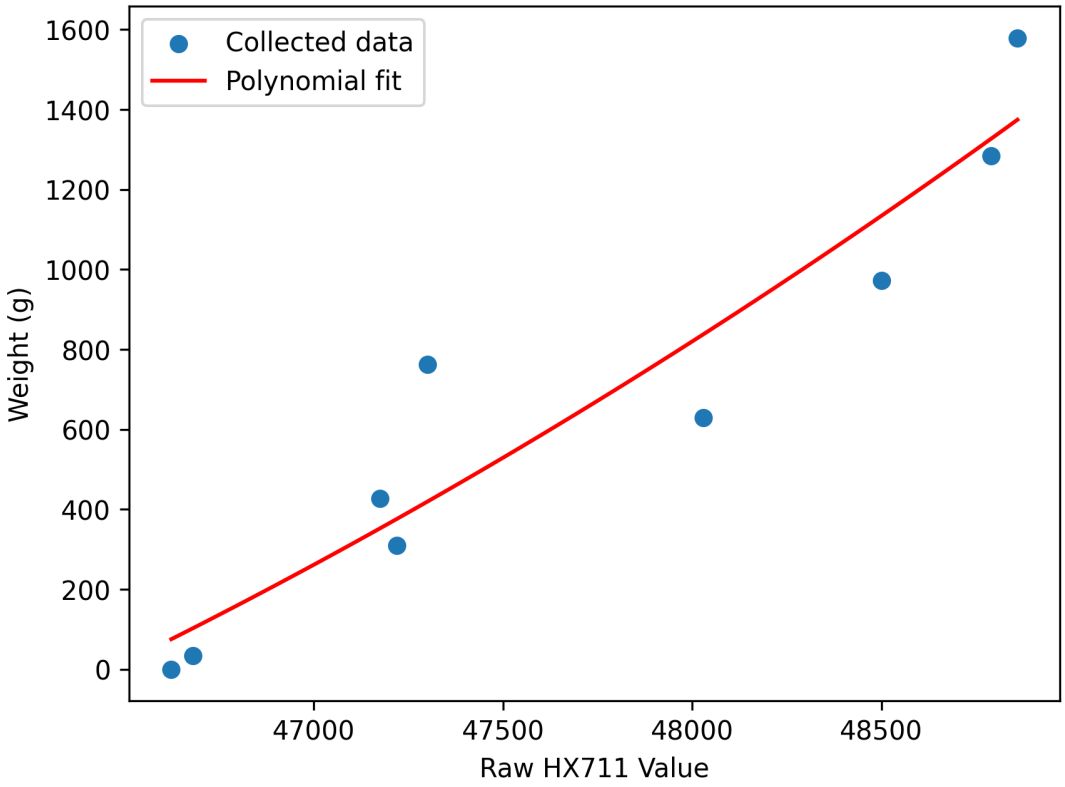
Furthermore, a polynomial regression script reduced the sample data into a curve function that could adjust the voltage values to weight measurements. See listing 3.1. Figure 4 shows the resulting curve.

Listing 3.1 – Python Script for data calibration

```
# Polynomial regression
coefficients = np.polyfit(raw_values, weights, 2)
print("Coefficients␣(highest␣degree␣first):", coefficients)

# This gives you: weight = c[0]*x² + c[1]*x + c[2]
# For degree 2: [a2, a1, a0]
```

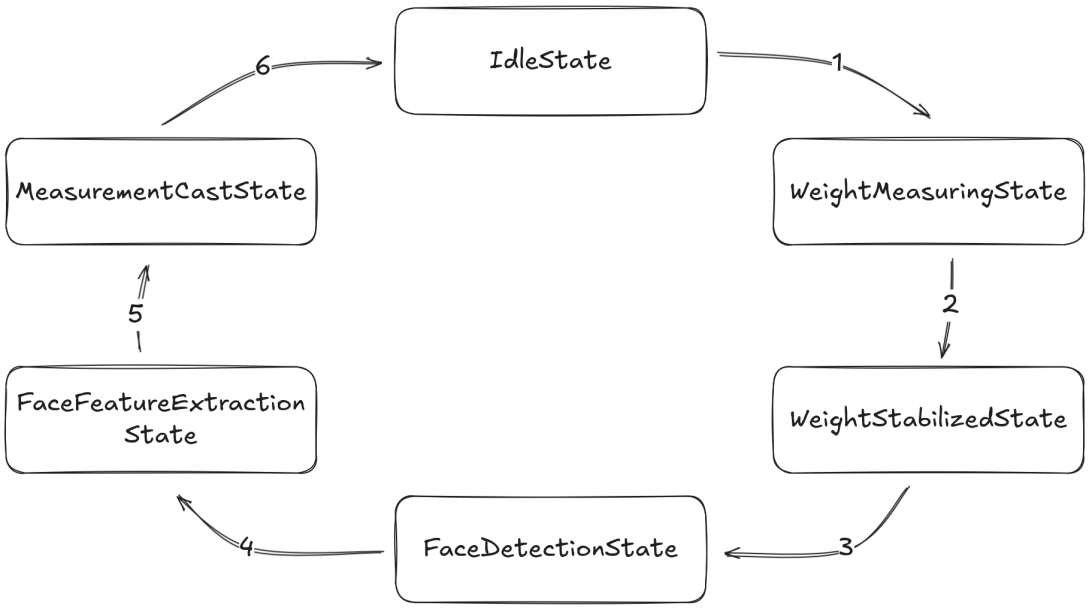
Figure 4 – Polynomial Regression resulting curve



Source: Author

Once the weight beats a given threshold of 5 grams the scale is set to retire from its idle state and take a picture of the subject on it. The microcontroller makes use of the pre-trained MobileFaceNet model from TensorFlow Lite library to extract the feature vector from the image (CHEN et al., 2018). By doing so, the subject image never leaves the (edge layer) and thus their privacy is protected by avoiding the transmission of possible sensitive biometric data to the **cloud layer**. Still, this feature vector is enough to query registered profiles in the database and follows the homomorphic encryption approach (JIN; KUMAR; HONG, 2020b). Figure 5 shows the state machine representation of the ESP32 microcontroller workflow.

Figure 5 – State Machine Diagram of the ESP32 workflow



Source: Author

At IdleState The system is in standby mode, waiting for the weight threshold crossing trigger. At WeightMeasuringState, the system actively reads data from the scale until the value becomes stable when sampling and the system reaches the WeightStabilizedState. Once weight is stable, the microcontroller achieves the FaceDetectionState in which the camera is activated to detect and capture a humance face. With the image captured, its feature vector is built during FaceFeatureExtractionState and then the system enters into the MeasurementCastState, when all data is packaged and sent to the MQTT broker. The system goes back to IdleState once no weight is measured and/or it is notified of success or failure in face matching.

3.4 PoC fog layer

The **fog layer** consists of the intermediate services and tools for the system to work, however it still remains within the household domain. This means information here is still private to the local network (LAN). This layer can be deployed to serve as an IoT Hub for smart things in the house to communicate between themselves.

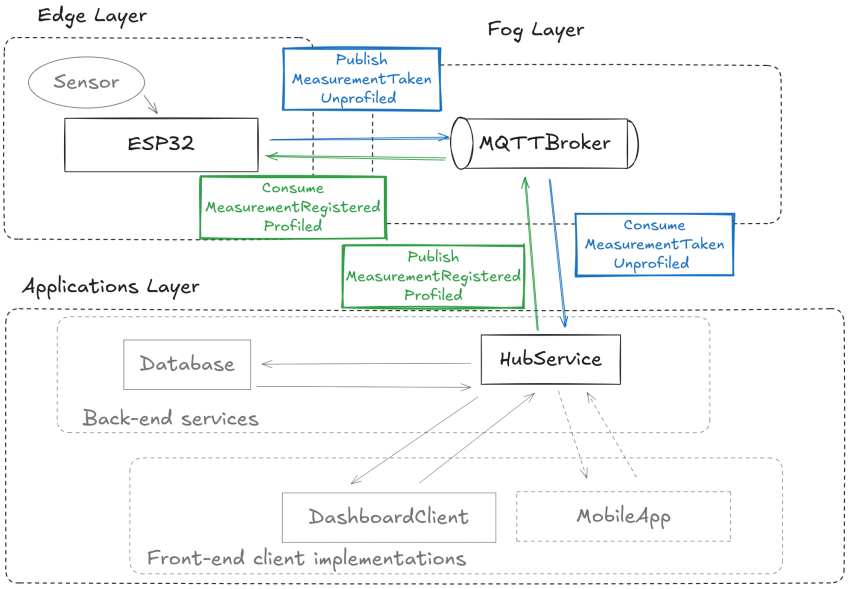
Within this project, this layer contains the **MQTT Broker**, which acts as a queue service bridging the edge and cloud layers. This is done by representing tasks as events to be parsed and reacted upon by the other layers. Therefore, the MQTT Broker serves as a queue service for bridging the edge layer to the services that depend on its output. Furthermore, the MQTT Broker and the fog layer as a whole (which could contain auxiliary services, for example), act as a hub for any other embedded devices in

the environment.

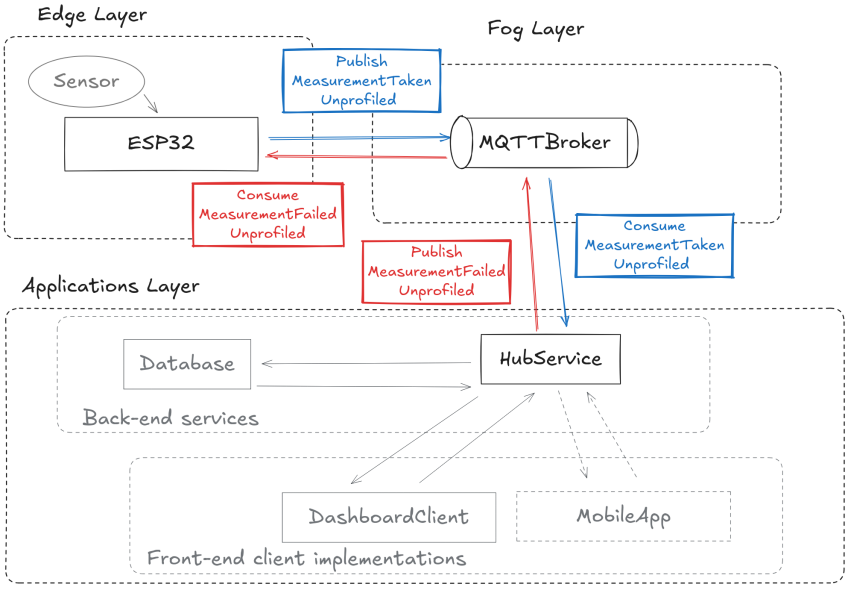
In this schema, the ESP32 *publishes* a message to the broker as a **MeasurementTakenUnprofiled** event, which contains a **topic** name, a weight value and a vector of embeddings. The topic represents something similar to a table in a Database Management System (DBMS) and is used to separate event contexts. Consequently, the HubService *consumes* the message and attempts to find the closer vector in the database and sends back a **MeasurementRegisteredProfiled** event so that the EPS32 is made aware everything went as expected (see Figure 6a). If the image does not match any registed profile, then the ESP32 is notified through the publishing of an **MeasurementFailedUnprofiled** event in the MQTT broker (see Figure 6b).

This workflow follows and Event-Driven Architecture (EDA) and is made possible through the definition of topics within the MQTT broker. Topics are similar to tables in which they represent a specific concept or event, however in practice topics represent a queue or a stream within a broker that handles many topics. User services publish or consume from specific topics within the broker. Each event is always sent to its own topic, and services subscribe and publish to their topics of concern. Figure 7 illustrates the MQTT broker topics and related publishers and subscribers.

Figure 6 – Event flows



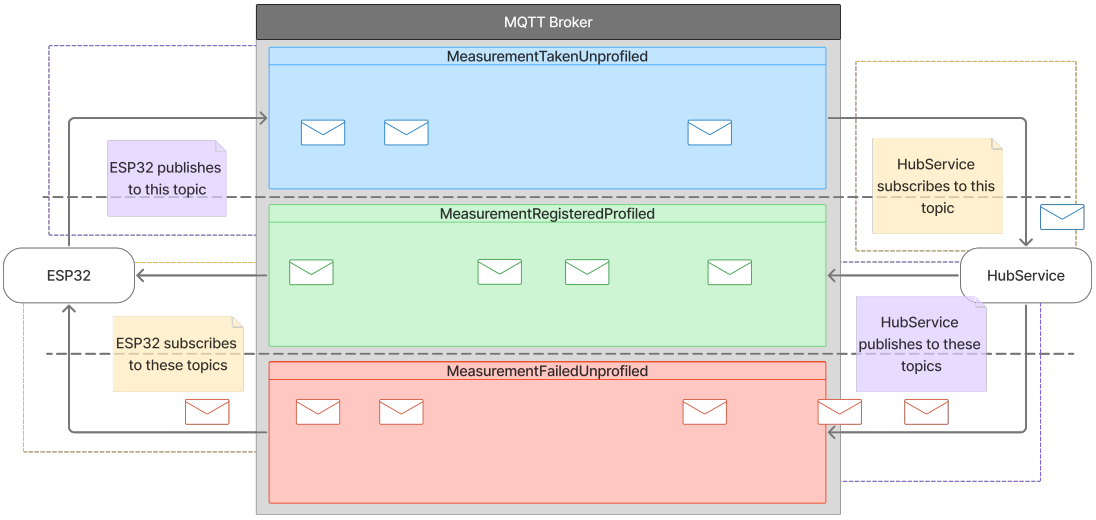
(a) Event Flow when face is recognized



(b) Event Flow when face is not recognized

Source: Author

Figure 7 – MQTT Broker Topics and workflow



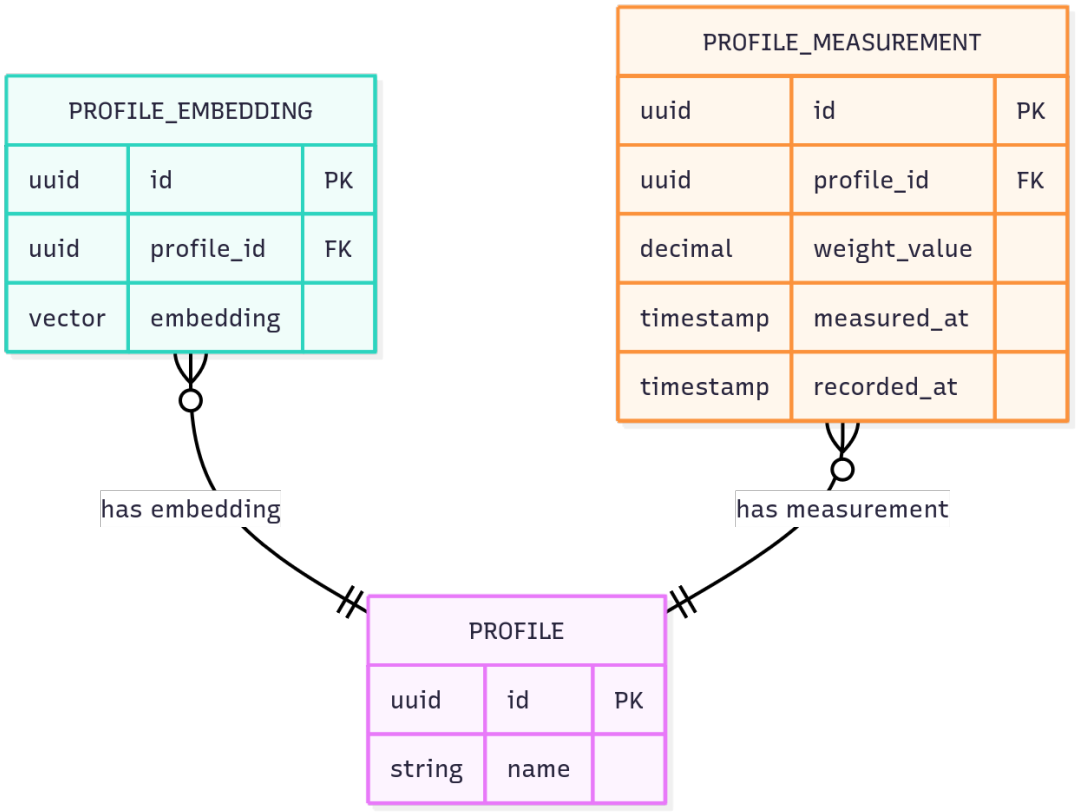
Source: Author

3.5 PoC cloud layer

The **cloud layer** is comprised of the HubService and a PostgreSQL database for storing profiles, their face embeddings and their measurements. The HubService is implemented following the Hexagonal Architectural Pattern and consists of an API for managing accessing database records and an MQTT client to consume and publish events to the MQTT broker in the **fog layer**. For this PoC, a dashboard serves as an example application of how the system could be used. The dashboard serves for managing profiles and visualizing the measured weights for them.

The database is designed to be simple and yet efficient, storing data in three tables: **profile**, **profile measurement**, and **profile embedding**. The **profile** table stores scale users information and identifies them with an Universally Unique Identifier (UUID). The **profile measurement** records holds a reference identifier to the profile they belong to and the weight measurement value. Finally, the **profile embedding** table stores available feature vectors for a profile to be used by face matching algorithms. PostgreSQL is set up with the **pgvector** plug-in that allows for the storage and retrieval of vector types in an efficient manner. Figure 8 shows the Entity-relationship Diagram of the database. This schema places profiles as owners of measurements and embeddings. From this perspective, queries remain simple and dedicated API endpoints make it easy to manipulate each entity separately or as collections. For example, given a profile identifier, it is possible to query its related embeddings and measurements directly. This is very convenient for the construction of data visualization tools, such as the dashboard.

Figure 8 – Entity-Relationship Diagram of the Database



Source: Author

The HubService was developed in the Kotlin programming language and used the Spring Boot framework. It contains a REST controller for receiving HTTP requests from web clients and an MQTT adapter for publishing and subscribing to a broker.

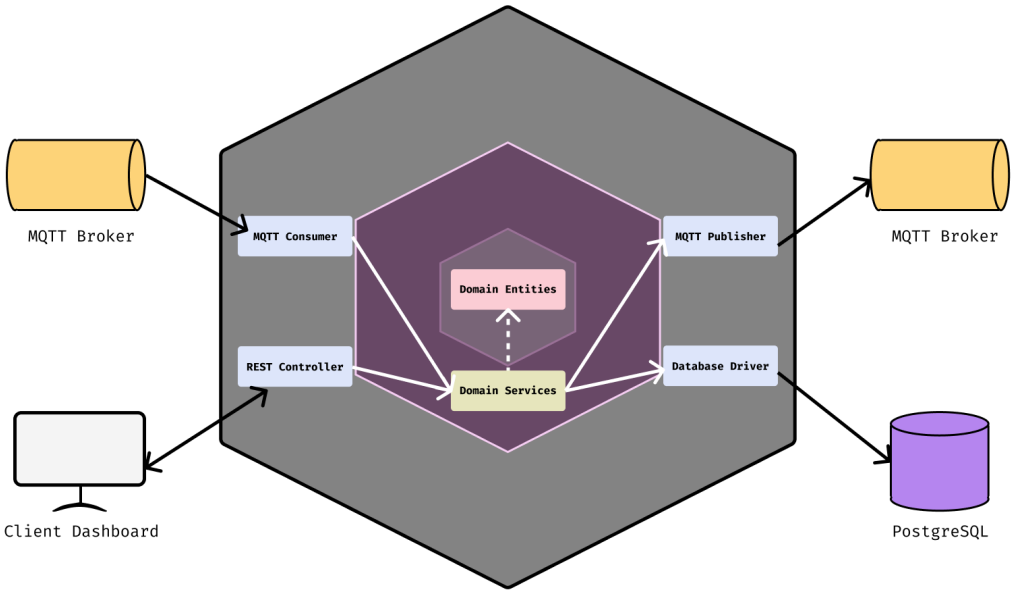
The REST API serves endpoints for managing profiles, which includes creating, updating, and deleting profiles, as well as reading the available ones in the database. The API also contains two extra endpoints for adding embeddings to a profile and for retrieving the measurements of a profile. Table 2 the available endpoints.

Table 2 – HubService API Endpoints

Method	**Endpoint**	**Description**
GET	/api/profiles	Retrieve all available profiles.
POST	/api/profiles	Create an instance of profiles.
GET	/api/profiles/{{id}}	Retrieve an instance of profiles.
PUT	/api/profiles/{{id}}	Update the information of a profile.
DELETE	/api/profiles/{{id}}	Destroy an instance of profiles given its UUID.
GET	/api/measurements	Retrieve all available measurements.
POST	/api/measurements	Create an instance of measurements.
GET	/api/measurements/by-profile/{{profileId}}	Retrieve all measurements that belong to the same profile, given the profile UUID.
GET	/api/measurements/{{id}}	Retrieve a measurement given its UUID.
DELETE	/api/measurements/{{id}}	Destroy an instance of measurements given its UUID.
GET	/api/embeddings	Retrieve all available embeddings.
POST	/api/embeddings	Create an instance of embeddings.
GET	/api/embeddings/by-profile/{{profileId}}	Retrieve all embeddings that belong to the same profile, given the profile UUID.
GET	/api/embeddings/{{id}}	Retrieve an embedding given its UUID.
DELETE	/api/embeddings/{{id}}	Destroy an instance of embeddings given its UUID.

Notice that even though the endpoint **POST /api/measurements** is available in the API, it was used solely for testing. Real measurements should come from the smart scale only. This is done through the implementation of an MQTT client placed as an incoming adapter within the Hexagonal Architecture of the HubService. Figure 9 showcases why this architecture is so versatile. Many different types of service can be moderately easily coupled into the architecture without disrupting what already exists (SEO et al., 2015) and keeping decoupling between classes high. The domain model holds not only the data structure of the base entities, but also the business model of the system. The domain however is not concerned with is use cases, therefore its code remain clean and agnostic. The upper layers in the architecture give context and purpose to these definitions, in such a way that dependencies never goes inwards (VERNON, 2013).

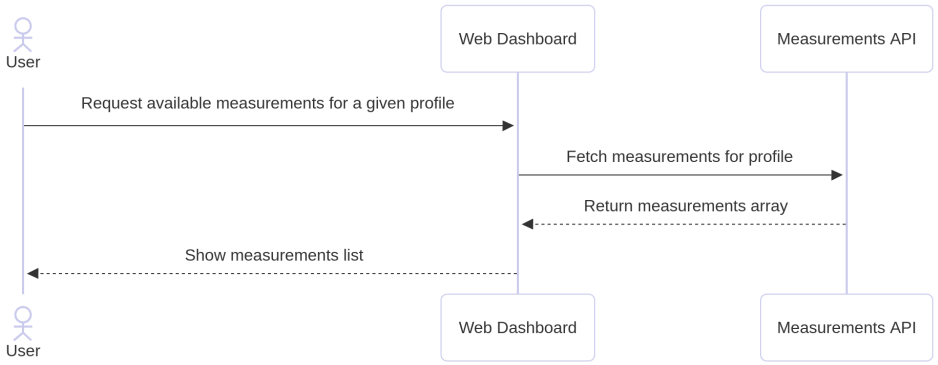
Figure 9 – Hexagonal Architecture of the HubService



Source: Author

Alongside the REST API, the Dashboard allows for using the API through a graphical user interface (GUI). Even though it is simple, it lets the smart scale users follow up all measurements available per profile. Figure 10 shows a sequence diagram for this feature. The user of the dashboard selects a profile and then the client logic fetches measurements for that profile by reaching to the responsible API endpoint in the HubService, which connects to the database and finally produces the response to the dashboard.

Figure 10 – Dashboard Sequence Diagram for Retrieving Measurements of a given Profile



Source: Author

3.6 Smart Scale Face Recognition

The face recognition implementation represents the convergence of multiple architectural decisions described in previous sections, particularly the privacy-by-design approach (NFR 4) and the event-driven architecture that supports retry mechanisms (NFR 2). Specially, the set privacy concerns required novel solutions. The images used for profile matching are never stored in order to keep privacy untouched. The project uses an homomorphic encryption approach in which the feature vector of the system is used to represent it. This method allows for searching and comparing images without the raw data.

This is made possible through the usage of the pgvector plugin within the PostgreSQL database used. It allows for storage and query of vector, which eases the process of matching profiles to weighing subjects. The one-to-many relationship between profile and profile embedding database entities allow for more chances of matching a scenario in which the machine learning models deployed do not take advantage of top-notch hardware, and are in the contrary very limited in resources.

Therefore, both during the registration of a profile and the taking of an image by the ESP32, the image embeddings are taken and used in place of the raw image data. This way biometric information never reaches the **cloud layer**.

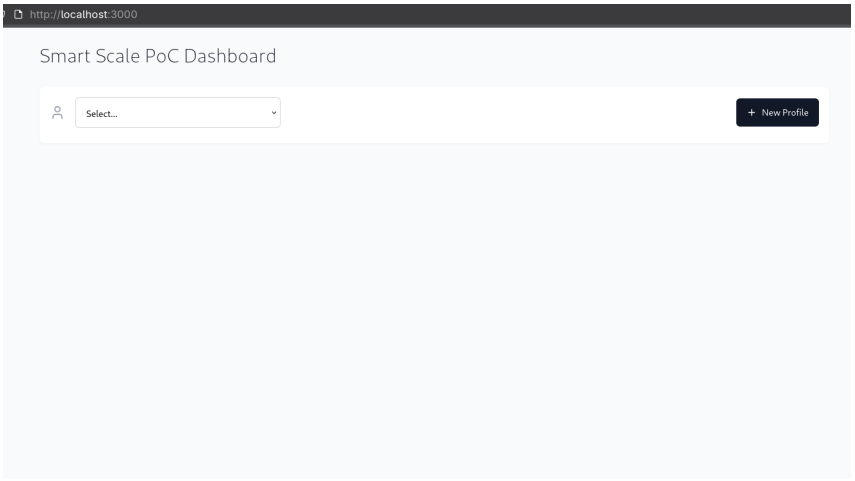
3.7 Project Evaluation

The PoC was evaluated in terms of how well the novel approaches proposed in literature could actually be implemented. More than that, we evaluate the face matching performance when using pre-trained models that can run on an ESP32 and the overrall performance of the system by measuring latency of data flow across the whole solution.

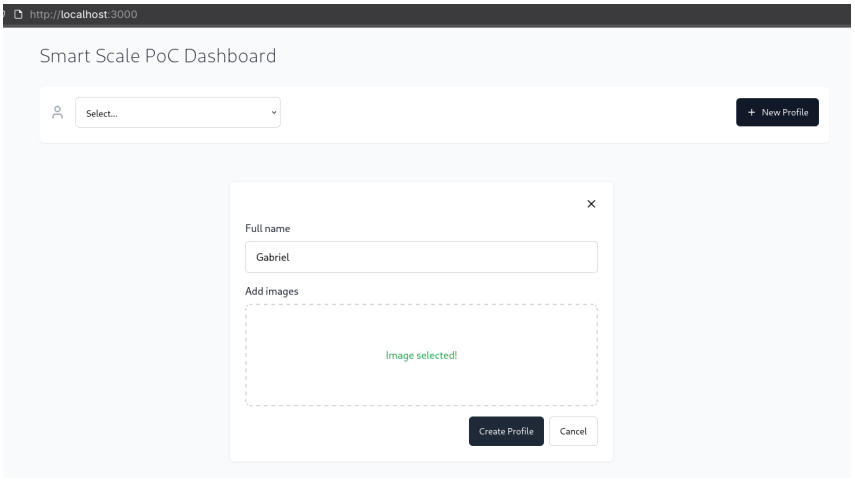
In order to measure face matching performance we compare different lighting conditions and percentage of retry request events in the communication between edge layer and cloud layer through the MQTT broker. Besides that, for performance we consider the different time values that are created for measurements, being those: measurement time (the moment the ESP32 sends the event with the measured value); and record time (the moment the measurement is recorded in the database). Nevertheless, those Key Performance Indicators (KPIs) combined serve the purpose of evaluating the feasability and aplicability of the novel approaches gathered in the theorical foundation.

A subject profile is created through the Dashboard. Figure 11a shows the interface in which you click the button to create a new profile, set a name and send a picture for face recognition reference, such as Figure 11b shows. No entries will show up for the profile until measurements are taken (see Figure 11c). Then, to measure their weight the person stands on the scale and holds the ESPCAM to their face as shown in Figure 12. Once face recognition is done successfully, the Dashboard will show a new entry (see Figures 13a and 13b).

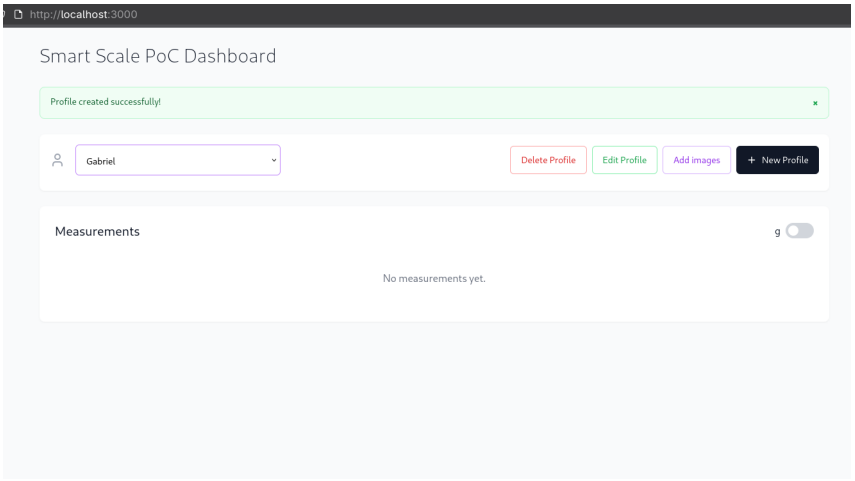
Figure 11 – Dashboard Profile creation



(a) Empty Dashboard



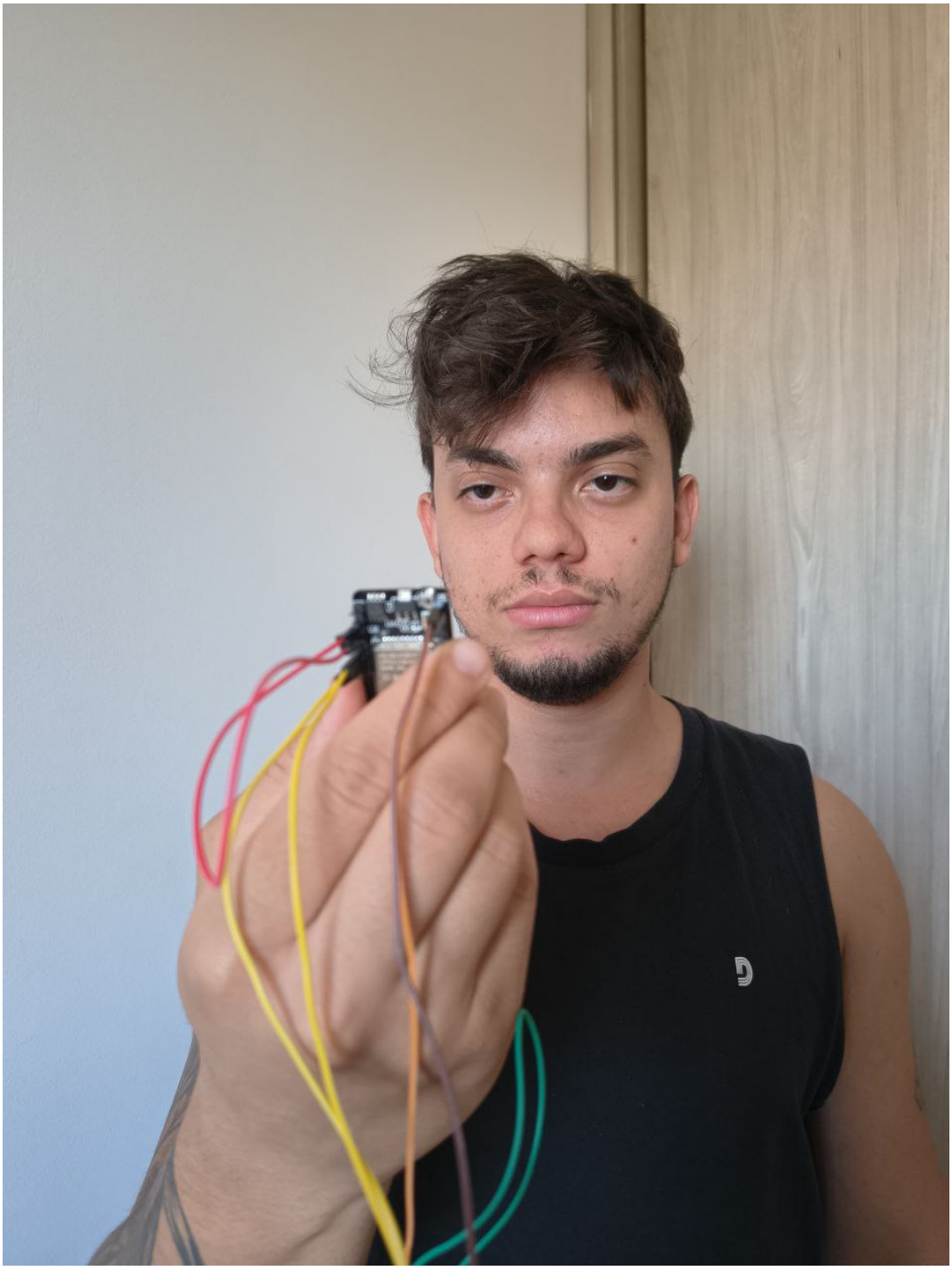
(b) Profile creation



(c) Empty Profile

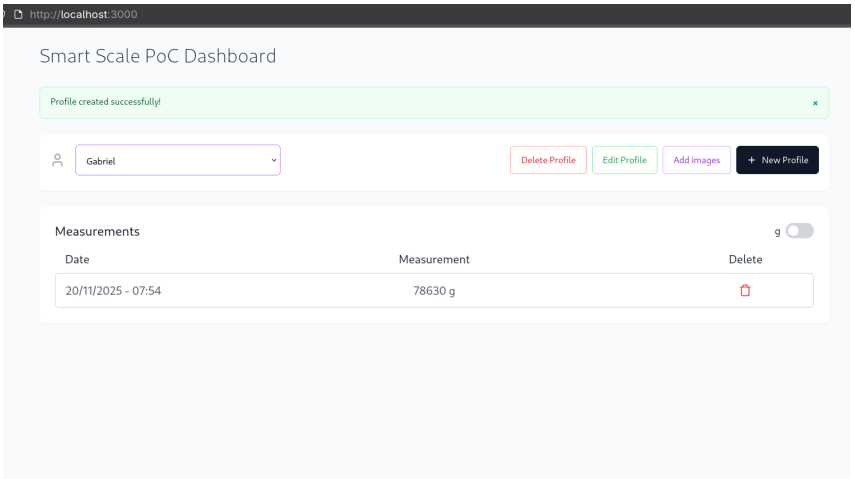
Source: Author

Figure 12 – Subject weighin themselves on scale

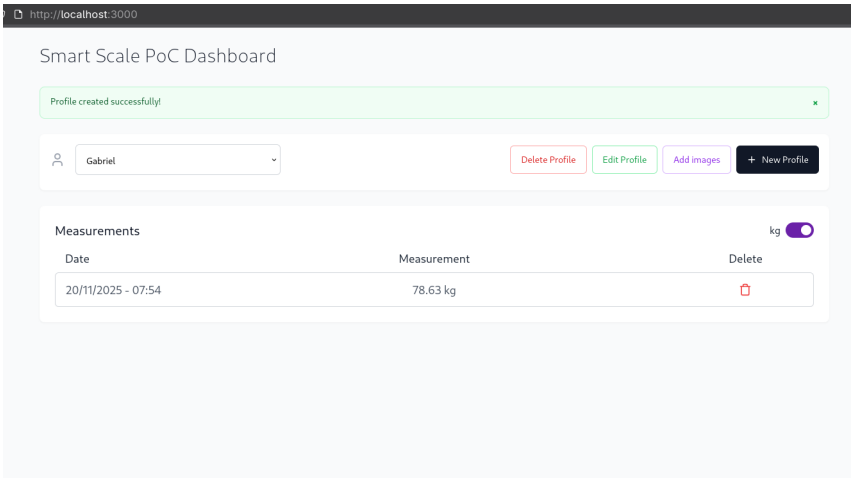


Source: Author

Figure 13 – Dashboard Profile measurement



(a) Measurement in grams



(b) Measurement in Kilograms

Source: Author

From repetitive testing and measurement of weights, the gathered information shows that face recognition on the edge layer takes some considerable time, leading to 1 to 2 seconds of delay when compared with the sending of the weight value only, which happens almost instantly. Overrall, the system is fast, but every component is running in the same network. In the ocasion of communicating with a real cloud service, performance could differ depending on internet connection quality.

4

Conclusion

4.1 Contributions

This work presents a comprehensive proof-of-concept (PoC) implementation of an IoT smart scale with facial recognition capabilities designed specifically for smart home integration. The project makes several significant contributions to the field of IoT systems and smart home technology starting by its Privacy-by-Design approach. The PoC demonstrates a practical implementation of homomorphic encryption principles in a resource-constrained embedded environment. By extracting facial feature vectors at the edge layer using the MobileFaceNet model from TensorFlow Lite, the system ensures that raw biometric data never leaves the user's control. This approach addresses one of the most critical concerns in IoT adoption—privacy and security of sensitive personal information.

The project achieves its goal within an affordable budget and accessible hardware components. The use of an ESP32 microcontroller (ESPCAM), HX711 signal amplifier, and standard strain gauge load cells provides a compelling alternative to expensive commercial solutions, addressing the affordability barrier identified by Mafong et al. (2020) as a significant obstacle to smart scale adoption.

The implementation also validates the effectiveness of an event-driven architecture (EDA) using MQTT as a lightweight messaging protocol for IoT systems. This architectural choice enables loose coupling between system components, facilitating easy extensibility and scalability. The three-layer architecture (edge, fog, and cloud) with clear separation of concerns provides a reference model for future IoT smart home applications. Aligned with that, the HubService implementation following the hexagonal architectural pattern demonstrates how domain-driven design principles can be effectively applied to IoT backend services. This approach ensures that the business logic remains agnostic to infrastructure concerns, allowing for multiple adapters (REST API, MQTT client) to coexist without compromising the core domain model. Still, the fog layer implementation,

centered around the MQTT broker, creates a foundation for a comprehensive smart home ecosystem. The architecture naturally accommodates the addition of new sensors and devices without requiring modifications to existing components, supporting the scalability requirements identified in the literature.

4.2 Discussion

The development and implementation of this PoC revealed several important insights regarding the practical challenges and trade-offs inherent in building privacy-conscious IoT systems for smart homes.

One of the primary technical challenges encountered during implementation was achieving stable weight readings from the strain gauge sensors. The analog nature of load cell measurements introduces noise and variability, particularly when subjects first step onto the scale. The system requires a brief stabilization period before capturing an accurate measurement, which impacts the user experience. Future iterations should implement more sophisticated digital filtering techniques, such as signal filters, to reduce measurement latency while maintaining accuracy.

The decision to perform facial feature extraction at the edge layer rather than in the cloud exemplifies the fundamental tension between privacy and processing power in IoT systems. While this approach successfully protects user privacy by never transmitting raw images, it constrains the system to using lightweight models that can run on resource-limited hardware. More sophisticated facial recognition models that might offer better accuracy under challenging conditions require computational resources beyond what the ESP32 can provide. This limitation underscores the need for continued research into efficient edge computing algorithms specifically designed for privacy-sensitive applications. Nevertheless, other mixed architectures that place face recognition on the fog layer take privacy risks but might provide better user experience.

Regarding the cloud layer, the hexagonal architecture employed in the HubService proved highly effective in accommodating multiple interfaces (REST API, MQTT client) without compromising the core business logic. The use of ports and adapters facilitated the integration of the dashboard as a client application and could easily support additional clients, such as mobile applications or integration with other smart home platforms. This architectural pattern aligns well with the recommendations of Seo et al. (2015) for IoT platform architectures and validates their applicability to smart home contexts.

The integration of PostgreSQL with the pgvector plugin for storing and querying facial embeddings represents an elegant solution to the challenge of matching subjects to profiles without storing raw biometric data. The vector similarity search capabilities

enable efficient matching even with multiple embeddings per profile, which helps compensate for the lower accuracy of lightweight recognition models by allowing users to register multiple reference images under different conditions.

Finally, several limitations of the current PoC should be acknowledged. First, the system has been tested primarily in controlled conditions with a very limited number of users. Also, the facial recognition system does not currently implement measures to prevent spoofing attacks (e.g., using photographs), which would be essential for security-critical applications. Still, considering the scope of the work and the context of this application, those limitations do not affect the overall contribution of this work. Within a domestic use case, connection issues are not likely to happen in the Local Area Network (LAN) and for privacy, the homomorphic encryption keeps user identity anonymous as well as sensitive data away from the network as a whole. Not only that, the MQTT broker working as bridge also limits the direct connection to the camera in the ESP32.

4.3 Future work

This proof-of-concept demonstrates that it is possible to build sophisticated, privacy-conscious IoT systems for smart homes using affordable hardware and well-architected software. While challenges remain in optimizing performance and addressing real-world deployment complexities, the project validates key architectural principles and provides a foundation for future research and development in this rapidly evolving field. The insights gained from this work contribute to the broader understanding of the trade-offs between cost, privacy, and performance in IoT systems and demonstrate practical approaches to addressing the barriers to IoT adoption in smart home contexts. Nevertheless, large-scale testing with diverse user populations, varying environmental conditions, and extended time periods would be necessary to fully validate the system's robustness.

The PoC developed in this project establishes a foundation for numerous research directions and practical applications in the IoT and smart home domains. The architecture and privacy-preserving approaches demonstrated in this PoC can be extended to create comprehensive health monitoring systems within the Internet of Medical Things paradigm. Integration with additional biometric sensors (blood pressure monitors, glucose meters, BMI) following the same event-driven architecture would enable holistic health tracking while maintaining the privacy-by-design principles. Such systems could facilitate remote patient monitoring, enabling healthcare providers to track vital signs and intervene proactively when anomalies are detected. The MQTT-based messaging infrastructure is well-suited for aggregating data from multiple health monitoring devices, and the hexagonal architecture of the HubService could easily accommodate adapters for medical data standards.

References

ARELLANES, D.; LAU, K.-K. Evaluating iot service composition mechanisms for the scalability of iot systems. *Future Generation Computer Systems*, v. 108, 03 2020.

Avia Semiconductor. *HX711: 24-Bit Analog-to-Digital Converter (ADC) for Weigh Scales*. [S.l.], 2010. Datasheet.

CHEN, S. et al. Mobilefacenets: Efficient cnns for accurate real-time face verification on mobile devices. In: SPRINGER. *Chinese Conference on Biometric Recognition*. [S.l.], 2018. p. 428–438.

Docker Inc. *Docker: Accelerated Container Application Development*. 2025. <https://www.docker.com/>. Accessed: 2025-11-21.

Eclipse Foundation. *Eclipse Mosquitto: An open source MQTT broker*. 2025. <https://mosquitto.org/>. Accessed: 2025-11-21.

ELORDI, U. et al. Optimal deployment of face recognition solutions in a heterogeneous iot platform for secure elderly care applications. *Procedia Computer Science*, v. 192, p. 3204–3213, 2021.

Espressif Systems. *ESP32 Technical Reference Manual*. [S.l.], 2024. Disponível em: <https://www.espressif.com/en/support/documents/technical-documents>.

Google Developers. *TensorFlow Lite: ML for Mobile and Edge Devices*. 2025. <https://www.tensorflow.org/lite>. Accessed: 2025-11-21.

HASTI, L.; PERMATA, E.; ARIBOWO, D. Development of a Digital Body Weight Scale Prototype with IoT-Based BMI Calculation and Real-Time Weight Tracking. *Aviation Electronics, Information Technology, Telecommunications, Electricals, and Controls (AVITEC)*, v. 7, p. 31, fev. 2025.

JAIN, A.; TANWAR, P.; MEHRA, S. Home automation system using internet of things (iot). In: *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*. [S.l.: s.n.], 2019. p. 300–305.

JAITEH, S. et al. Smart Scale Tracking System Using Calibrated Load Cells. In: *2019 IEEE Conference on Sustainable Utilization and Development in Engineering and Technologies (CSUDET)*. [s.n.], 2019. p. 170–174. ISSN: 2473-3652. Disponível em: <https://ieeexplore.ieee.org/document/9214692>.

JetBrains. *Kotlin Programming Language*. 2025. <https://kotlinlang.org/>. Accessed: 2025-11-21.

References

JIN, H.; KUMAR, S.; HONG, J. Providing architectural support for building privacy-sensitive smart home applications. In: *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*. New York, NY, USA: Association for Computing Machinery, 2020. (UbiComp/ISWC '20 Adjunct), p. 212–217. ISBN 9781450380768. Disponível em: <https://doi.org/10.1145/3410530.3414328>.

JIN, H.; KUMAR, S.; HONG, J. Providing architectural support for building privacy-sensitive smart home applications. In: *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*. New York, NY, USA: Association for Computing Machinery, 2020. (UbiComp/ISWC '20 Adjunct), p. 212–217. ISBN 978-1-4503-8076-8. Disponível em: <https://dl.acm.org/doi/10.1145/3410530.3414328>.

KANE, A. *pgvector: Open-source vector similarity search for PostgreSQL*. 2025. <https://github.com/pgvector/pgvector>. GitHub Repository.

KANE, L. et al. Network architecture and authentication scheme for lora 2.4 ghz smart homes. *IEEE Access*, v. 10, p. 93212–93230, 2022.

MADAKAM, S. Internet of things: Smart things. *International Journal of Future Computer and Communication*, v. 4, p. 250–253, 05 2015.

MAFONG, K.; KIM, S.; FUJIOKA, K. Are patients willing to use a smart scale? *Current Developments in Nutrition*, v. 4, p. 063–055, 2020. ISSN 2475-2991. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2475299123096750>.

MARTINO, B. D. et al. Internet of things reference architectures, security and interoperability: A survey. *Internet of Things*, v. 1–2, p. 99–112, 2018.

Meta Open Source. *React: A JavaScript library for building user interfaces*. 2025. <https://react.dev/>. Accessed: 2025-11-21.

MOCRII, D.; CHEN, Y.; MUSILEK, P. Iot-based smart homes: A review of system architecture, software, communications, privacy and security. *Internet of Things*, v. 1-2, p. 81–98, 2018. ISSN 2542-6605. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2542660518300477>.

MULLER, I. et al. Load cells in force sensing analysis – theory and a novel application. v. 13, n. 1, p. 15–19, 2010. ISSN 1941-0123. Disponível em: <https://ieeexplore.ieee.org/document/5399212>.

OASIS. Mqtt version 5.0. *Retrieved June*, v. 22, n. 2020, p. 1435, 2019.

PANJA, S. et al. 1 - internet of medical things: architecture, trends, and challenges. In: NGUYEN, T. A. (Ed.). *Blockchain and Digital Twin for Smart Healthcare*. Elsevier, 2025. p. 1–17. ISBN 978-0-443-30300-5. Disponível em: <https://www.sciencedirect.com/science/article/pii/B9780443303005000026>.

PARK, J. S. et al. Affordability and features of home scales for self-weighing. *Clinical obesity*, v. 11, p. e12475, 06 2021.

References

SATHYAMOORTHY, S. et al. Advances and challenges in IoT sensors data handling and processing in environmental monitoring networks. *HAFED POLY Journal of Science, Management and Technology*, v. 5, p. 40–60, 05 2024.

SEO, S. et al. HePA: Hexagonal Platform Architecture for Smart Home Things. In: *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*. [s.n.], 2015. p. 181–189. ISSN: 1521-9097. Disponível em: <https://ieeexplore.ieee.org/abstract/document/7384294>.

SORNIN, N. et al. Lorawan specification. *LoRa alliance*, v. 1, p. 16, 2015.

The PostgreSQL Global Development Group. *PostgreSQL: The World's Most Advanced Open Source Relational Database*. 2025. <https://www.postgresql.org/>. Accessed: 2025-11-21.

VERNON, V. *Implementing Domain-Driven Design*. [S.l.]: Addison-Wesley, 2013. Google-Books-ID: X7DpD5g3VP8C. ISBN 978-0-13-303988-7.

VMware Tanzu. *Spring Boot*. 2025. <https://spring.io/projects/spring-boot>. Accessed: 2025-11-21.

WOOLLEY, M. Bluetooth core specification v5. 1. *Bluetooth Special Interest Group*, 2019.

ZARGHAM, A. et al. Revolutionizing Small-Scale Retail: Introducing an Intelligent IoT-based Scale for Efficient Fruits and Vegetables Shops. *Applied Sciences*, v. 13, n. 14, p. 8092, jan. 2023. ISSN 2076-3417. Publisher: Multidisciplinary Digital Publishing Institute. Disponível em: <https://www.mdpi.com/2076-3417/13/14/8092>.