IoT

MDPI

Article

Modular IoT Architecture for Monitoring and Control of Office Environments Based on Home Assistant

Yevheniy Khomenko [1,†] and Sergii Babichev [1,2,*,†,‡]

[1] Department of Physics, Faculty of Computer Science, Physics and Mathematics, Kherson State University, 73000 Kherson, Ukraine; yekhomenko@ksu.ks.ua
[2] Department of Informatics, Faculty of Science, Jan Evangelista Purkyně University in Ústí nad Labem, 400 96 Ústí nad Labem, Czech Republic
* Correspondence: sergii.babichev@ujep.cz; Tel.: +420-777-843-785
† These authors contributed equally to this work.
‡ Current address: Pasteurova 3632/15, 400 96 Ústí nad Labem, Czech Republic

Abstract: Cloud-centric IoT frameworks remain dominant; however, they introduce major challenges related to data privacy, latency, and system resilience. Existing open-source solutions often lack standardized principles for scalable, local-first deployment and do not adequately integrate fault tolerance with hybrid automation logic. This study presents a practical and extensible local-first IoT architecture designed for full operational autonomy using open-source components. The proposed system features a modular, layered design that includes device, communication, data, management, service, security, and presentation layers. It integrates MQTT, Zigbee, REST, and WebSocket protocols to enable reliable publish–subscribe and request–response communication among heterogeneous devices. A hybrid automation model combines rule-based logic with lightweight data-driven routines for context-aware decision-making. The implementation uses Proxmox-based virtualization with Home Assistant as the core automation engine and operates entirely offline, ensuring privacy and continuity without cloud dependency. The architecture was deployed in a real-world office environment and evaluated under workload and fault-injection scenarios. Results demonstrate stable operation with MQTT throughput exceeding 360,000 messages without packet loss, automatic recovery from simulated failures within three minutes, and energy savings of approximately 28% compared to baseline manual control. Compared to established frameworks such as FIWARE and IoT-A, the proposed approach achieves enhanced modularity, local autonomy, and hybrid control capabilities, offering a reproducible model for privacy-sensitive smart environments.

Keywords: Home Assistant; IoT architecture; smart office; local automation; edge computing; open-source platforms

check for updates

Academic Editor: Amiya Nayak

Received: 5 October 2025
Revised: 6 November 2025
Accepted: 12 November 2025
Published: 17 November 2025

Citation: Khomenko, Y.; Babichev, S. Modular IoT Architecture for Monitoring and Control of Office Environments Based on Home Assistant. *IoT* **2025**, *6*, 69. https://doi.org/10.3390/iot6040069

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The rapid growth of the Internet of Things (IoT) has led to the widespread deployment of automation systems across office, industrial, and residential environments [1,2]. Although cloud-centric architectures currently dominate the field due to their ease of deployment and integration capabilities, they introduce notable challenges concerning data privacy, infrastructure control, and dependency on third-party services.

A review of modern automation systems reveals that most available solutions rely heavily on cloud infrastructure [3,4]. While cloud-based models offer clear advantages

in terms of scalability and integration availability, they also pose significant risks, including data leakage, limited user control over infrastructure, and susceptibility to external platform failures.

As demand grows for privacy, resilience, and service continuity, the development of local IoT systems capable of operating independently from external factors has become increasingly important. Such systems should provide users with full control over data, operational logic, and network security. Accordingly, the creation of flexible, secure, and cost-effective local IoT architectures—comparable in functionality to centralized models—has emerged as a key research direction. Essential requirements include local data storage and processing, operational autonomy, support for multi-vendor devices across various standards, and scalability without dependence on proprietary cloud stacks.

Among the available open-source platforms, *Home Assistant* holds a leading position due to its technical maturity, extensive integration ecosystem, and active developer community [5,6]. Compared to other solutions such as *Domoticz* [7] or *openHAB* (open Home Automation Bus) [8], *Home Assistant* provides superior scalability, high configurability through *YAML* (YAML Ain't Markup Language), regular updates without breaking backward compatibility, and native support for modern automation standards such as *Matter* and *Thread*. Matter is an open-source connectivity standard developed by the Connectivity Standards Alliance (CSA) to ensure secure and reliable communication among smart home devices, whereas *Thread* is a low-power, IPv6-based wireless mesh networking protocol designed for IoT applications. Furthermore, *Home Assistant* allows complete local deployment under an open-source license, ensuring independence from cloud services—making it particularly suitable for mission-critical installations that require security, reliability, and full user control.

Recent studies have explored the application of *Home Assistant* in various smart environment scenarios, highlighting both its strengths and limitations [9–13]. These works confirm the platform's versatility and cost-effectiveness but also expose persistent challenges, including the absence of standardized design principles for scalable local architectures, limited hybrid automation logic for heterogeneous devices, and insufficient support for non-technical users in complex configurations.

Despite progress in open-source IoT ecosystems, no standardized local-first architecture currently combines full operational autonomy, modular extensibility, hybrid automation logic, and verified fault resilience. Existing approaches often emphasize flexibility at the expense of privacy or prioritize local deployment without systematic evaluation of reliability and scalability.

This paper presents the design, development, and deployment of a modular IoT architecture for local monitoring and control of office environments, based on the *Home Assistant* platform. The proposed system is organized into three interconnected layers—physical, virtual, and logical—each responsible for functions ranging from sensor data acquisition to decision-making and automation. A core advantage of the architecture lies in its modularity and scalability, which enable adaptation to diverse office configurations and operational requirements with minimal reconfiguration. The system integrates environmental sensors, microcontrollers, actuators, and open-source communication protocols such as MQTT and REST API to support real-time monitoring of parameters including temperature, humidity, air quality, motion, lighting, and device activity. The architecture supports both manual and automated control modes, as well as secure local and remote access. A particular emphasis is placed on local data processing and low-latency operation, which enhance the system's reliability, autonomy from cloud infrastructure, and data privacy. The architecture has been implemented and validated in a real-world office setting, demonstrating measurable improvements in energy efficiency, indoor comfort, and workspace utilization. The

results confirm the feasibility of *Home Assistant*-based solutions for building cost-effective, customizable, and decentralized smart office infrastructures.

The main contributions of this research are summarized as follows:

- Design of a modular and scalable architecture for local IoT-based monitoring and control of office environments using the *Home Assistant* platform. The architecture integrates a three-layer model (physical, virtualization, and logical), providing a clear separation between sensor–actuator infrastructure, virtual nodes, and automation logic.
- Development of a fully autonomous control system featuring local data acquisition, processing, decision-making, and visualization independent of external cloud services. The communication subsystem is implemented using secure and open protocols (MQTT, REST API), ensuring flexibility and interoperability across heterogeneous devices.
- Real-world deployment of the system in a smart office scenario integrating multiple subsystems—climate control, lighting, security, energy management, and diagnostics—through a unified dashboard. The system supports real-time data exchange, cross-subsystem automation logic, and secure remote access, providing a practical foundation for decentralized smart environment control.

The remainder of this paper is organized as follows: Section 2 provides a comparative analysis of leading open-source and cloud-centric IoT platforms. Section 3 introduces the proposed modular IoT architecture and its seven functional layers—Device, Communication, Data, Management, Security, Service, and Presentation. Section 4 describes the decomposition into functional subsystems and integration of hybrid automation logic. Section 5 presents the technical implementation and performance evaluation metrics. Section 6 details the deployment setup, user interface, and evaluation results. Finally, Section 7 discusses the achieved outcomes, identified challenges, and directions for future development, focusing on scalability, usability, and adaptive optimization.

2. Related Work

Over the past decade, numerous smart home and IoT platforms have emerged, offering diverse architectural paradigms, integration capabilities, and operational models. These platforms aim to address growing demands for energy efficiency, comfort, security, and privacy in both residential and commercial automation systems. Among the most widely adopted are *Home Assistant* [6], *OpenHAB* [14], *Domoticz* [15], *Node-RED* [16], and *OpenRemote* [17]. In the academic domain, frameworks such as *FIWARE* and *IoT-A* have been proposed as reference models for large-scale IoT integration. However, these solutions remain primarily cloud-centric and entail considerable infrastructure overhead. Each approach exhibits specific strengths and limitations that influence its suitability for different deployment scenarios.

Home Assistant is an open-source Python-based platform emphasizing privacy, modular integration, and full local autonomy. The platform benefits from a large and active developer community and supports advanced automation through YAML scripting or GUI editors. As shown in [5,9], *Home Assistant* is particularly effective for edge-based deployments where low latency, data security, and user sovereignty are priorities.

OpenHAB provides a flexible Java-based architecture enabling vendor-neutral integration of diverse IoT devices and protocols via the Eclipse SmartHome framework. It supports features such as semantic tagging, persistence layers, RESTful APIs, rule-based automation, and modular add-ons. *OpenHAB* demonstrates strong interoperability across Z-Wave, ZigBee, Wi-Fi, and LTE networks, as validated in recent experimental testbeds. Despite its

scalability and extensibility, its main drawbacks include a steep learning curve and limited GUI customization, which may hinder adoption in user-centric environments [8,18,19].

Domoticz is a lightweight C++ platform optimized for low-resource devices such as Raspberry Pi. It provides essential automation features but lacks extensibility and modern interface design compared to competitors. Studies such as [20] highlight its ease of installation but limited advanced automation capabilities.

Node-RED, while not a traditional smart home platform, serves as a powerful flow-based development tool for IoT systems. It enables visual programming of automation workflows and is often used alongside other platforms [21]. However, it lacks native device abstraction and user-focused dashboards, making it more suitable for technically experienced users.

OpenRemote offers a more enterprise-oriented perspective, supporting hybrid cloud–local deployments and granular access control. Although it provides a flexible asset management model, its community support and ease of use remain limited [17].

Beyond these open-source ecosystems, a growing body of research has focused on hybrid and edge-centric IoT architectures emphasizing privacy, scalability, and energy efficiency. Yar et al. [22] introduced an IoT-enabled edge-computing paradigm for smart homes based on a Raspberry Pi controller, which reduces cloud dependency and latency by storing sensor data locally. Their five-layer model integrates MQTT-based communication, Node-RED dashboards, and local data aggregation, achieving improved energy efficiency and response time compared with cloud-only solutions. Similarly, Taiwo and Ezugwu [23] developed a cloud-integrated home automation system that incorporates a machine learning component (Support Vector Machine) for intruder detection. Their results show that intelligent decision-making can enhance safety, albeit with continued reliance on cloud storage and remote control.

Recent advances have also highlighted the importance of hardware-level and energy-aware optimizations for next-generation IoT systems. Antolini et al. [24] presented a comprehensive overview of Phase-Change Memory (PCM) as a foundational technology for analog in-memory and edge computing, demonstrating its potential to improve computational density and reduce latency in local inference tasks. Their study underscores how hardware innovation can enhance the decentralization and sustainability of IoT frameworks. In a complementary contribution, Xie and Fang [25] proposed an energy-aware generative AI edge inference framework for low-power IoT devices. Their architecture integrates adaptive quantization and lightweight model compression, achieving significant reductions in both latency and power consumption. Together, these studies reinforce the paradigm shift toward intelligent, energy-optimized edge systems capable of autonomous operation without continuous cloud dependence.

Stolojescu-Crisan et al. [20] proposed an IoT-based automation framework combining RESTful APIs and MQTT for device interoperability, offering a reference middleware model that highlights the trade-off between interoperability and complexity. Hamdan et al. [26] further advanced this direction by developing a hybrid IoT–cloud architecture aimed at optimizing communication latency and improving reliability in multi-sensor environments. Addow et al. [27] emphasized sustainability through a low-cost IoT-based automation system for urban housing, illustrating how energy-efficient designs can be achieved with limited hardware resources in developing regions.

Collectively, these studies demonstrate a clear evolution from cloud-dependent to decentralized and hybrid IoT paradigms. However, most existing approaches still retain partial reliance on remote infrastructure or lack integrated support for combining rule-based and data-driven automation under a unified local-first architecture. These limitations motivate the present work, which introduces a fully autonomous, privacy-preserving

IoT framework that unifies modular rule-based control and statistical automation while ensuring local data ownership and low-latency operation.

Overall, while recent frameworks have made significant progress in interoperability and cloud–edge integration, persistent challenges remain. These include reliance on remote infrastructures, limited transparency of proprietary middleware, and insufficient mechanisms for privacy-preserving local analytics. Consequently, the field continues to lack a fully decentralized, open, and verifiable architecture that ensures both autonomy and scalability under real-world operational conditions.

Comparative Analysis

To systematically compare these platforms, Table 1 summarizes key dimensions including architecture, local/cloud dependency, ease of integration, scalability, security, and user interface flexibility. The analysis is grounded in previous scientific evaluations [20,28,29] and direct experimentation conducted in this study. In addition, comparative insights from recent academic works [20,22–27] were integrated to reflect the evolution toward hybrid, edge-centric, and energy-aware IoT solutions emphasizing privacy, scalability, and sustainability.

Table 1. Comparative Analysis of Selected Smart Home Platforms and Research Frameworks.

Platform/Framework	Local Control	Integration Richness	User-Friendliness	Security/Privacy	Extensibility/Scalability	Ref.
Home Assistant	✓	**High**	Medium–High	**High**	**High**	[5,9]
OpenHAB	✓	High	Medium	High	High	[8,18,19]
Domoticz	✓	Medium	High	Medium	Low	[20]
Node-RED	✓	Medium	Low–Medium	Medium	High	[21]
OpenRemote	✓/cloud	Medium	Medium	High	Medium	[17]
Yar et al. (2021)	✓(Edge)	High	Medium	High (Local data storage)	High	[22]
Taiwo and Ezugwu (2021)	Cloud-based	Medium	High (Mobile App)	Medium–High (ML-enhanced)	Medium	[23]
Hamdan et al. (2020)	Hybrid (Cloud–Edge)	High	Medium	High	High (Multi-sensor)	[26]
Stolojescu-Crisan et al. (2021)	Cloud	High	Medium	Medium	Medium–High	[20]
Addow et al. (2025)	✓(Local/Low-cost)	Medium	High	Medium	Medium (Low-power)	[27]
Antolini et al. (2025)	✓(Edge Hardware)	High (PCM-based)	Medium	High (Energy-efficient)	High (Analog in-memory)	[24]
Xie and Fang (2025)	✓(AI Edge)	Medium–High	High	High (Generative AI)	High (Energy-aware)	[25]

Note. The table compares open-source, hybrid, and edge-oriented IoT frameworks across five major dimensions. High ratings indicate advanced feature support verified through published studies and experimental validation. The proposed Home Assistant–based architecture uniquely combines full local autonomy, modular scalability, and privacy-by-design compliance.

As shown in Table 1, many open-source platforms prioritize either simplicity (e.g., Domoticz), enterprise-level functionality (e.g., OpenRemote), or programmability (e.g., Node-RED). Meanwhile, frameworks such as *FIWARE* and *IoT-A* serve as reference architectures but remain dependent on centralized infrastructures and require considerable configuration effort.

Recent academic implementations, including those by Yar et al., Hamdan et al., Antolini et al., and Xie and Fang, illustrate the transition toward edge intelligence and energy-optimized IoT frameworks. Yar et al. demonstrated that local data aggregation significantly improves latency and reliability, while Hamdan et al. introduced a scalable hybrid edge–cloud model enhancing communication efficiency. Antolini et al. highlighted hardware-level optimization using phase-change memory for analog in-memory computing, and Xie and Fang proposed an AI-driven, energy-aware framework reducing power consumption in low-resource devices. In contrast, Taiwo and Ezugwu focused on ML-enhanced security in a cloud-centric design, Addow et al. prioritized sustainability in low-cost environments,

and Stolojescu-Crisan et al. explored middleware interoperability through MQTT and RESTful APIs.

Collectively, these studies reveal complementary strengths—hardware-assisted edge systems offer latency and efficiency benefits, while hybrid and cloud models provide extended functionality and computational flexibility. Nonetheless, most implementations continue to rely on remote processing, which constrains local autonomy and compromises privacy.

The proposed architecture presented in this paper integrates the advantages of modular edge processing, hardware-assisted optimization, and complete local data sovereignty within a decentralized ecosystem. This approach bridges existing gaps in scalability, energy efficiency, and usability, demonstrating that privacy-preserving, local-first IoT systems can achieve performance comparable to cloud-based infrastructures while maintaining independence and user control.

In summary, the comparison underscores the need for a modular, local-first IoT architecture that ensures privacy, reliability, and fault tolerance while integrating both rule-based and data-driven automation for real-world applications. The next section introduces the proposed architecture, designed to address these challenges and validated through deployment in a smart office environment.

3. Architecture of the Local Automation System Based on Home Assistant

Among the available open-source platforms, the proposed system adopts *Home Assistant* as its foundational framework. *Home Assistant* is a widely used Python-based automation platform that supports full local control, modular integration, and secure deployment without reliance on cloud services. Its extensible architecture and extensive integration ecosystem make it particularly suitable for implementing decentralized and privacy-preserving IoT environments.

The proposed local automation system architecture, shown in Figure 1, follows a modular, scalable, and functionally decomposed design strategy built upon the *Home Assistant* platform. This structure enables seamless adaptation to diverse operational requirements and user-defined automation scenarios while maintaining high reliability, interoperability, and integration flexibility.

The architecture is logically organized into seven functional layers, spanning from physical device interaction to user-facing visualization and control. Each layer operates as an independent yet interoperable module with standardized interfaces and well-defined responsibilities, ensuring system stability, transparency, and ease of future expansion. These layers are: Device, Communication, Data, Management, Security, IoT Service, and Presentation. This layered structure reflects established design principles from modern control and automation systems, promoting localized data processing, minimal cloud dependency, and strong interoperability across heterogeneous devices and communication protocols.

The following subsections provide a detailed description of each layer, outlining its purpose, structure, and functional contribution to the overall Home Assistant-based automation ecosystem.

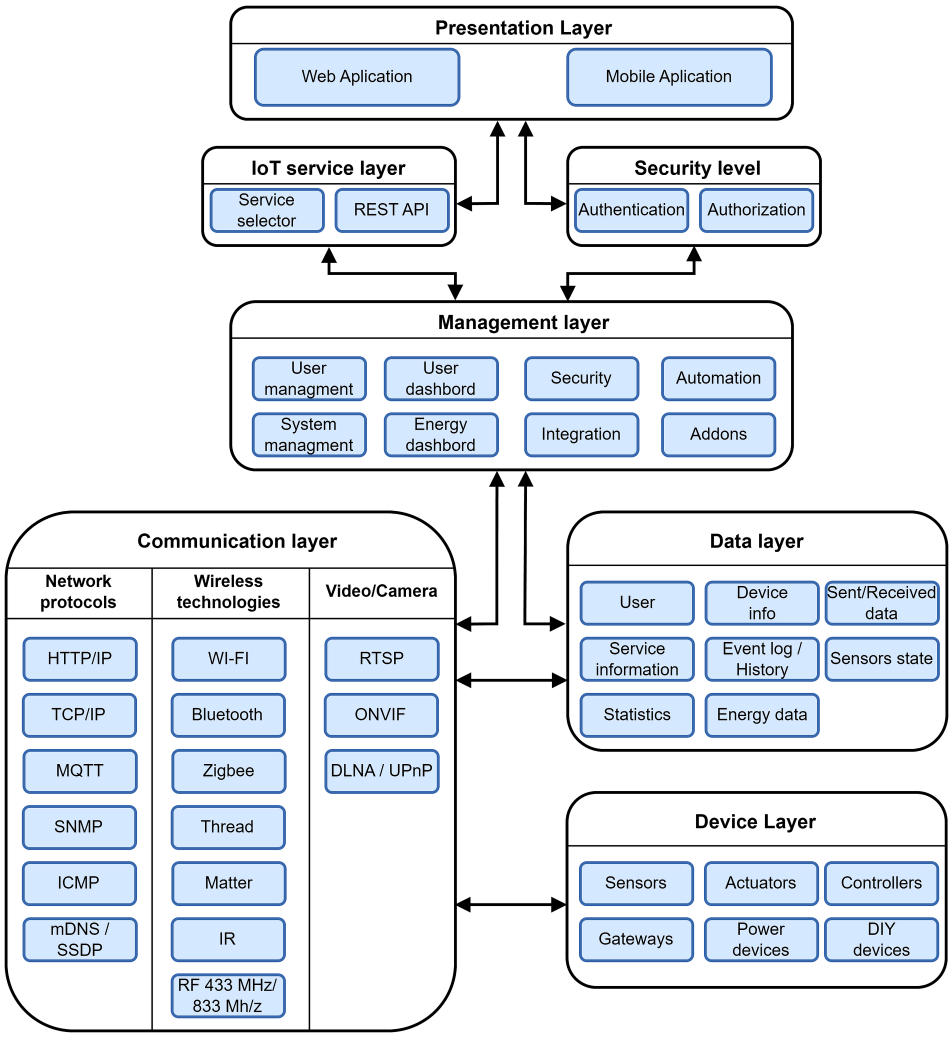


Figure 1. General architecture of the local automation system based on Home Assistant. The diagram depicts seven functional layers of the proposed architecture, ranging from device interaction to user interface. Arrows indicate the direction of data and control flow between layers, emphasizing modular separation and interoperability for localized automation.

3.1. Functional Layers of the Local IoT System Architecture

This section revisits the proposed architecture from a structured and comparative perspective, highlighting the key design decisions that distinguish our implementation from existing approaches. Unlike general-purpose frameworks such as FIWARE or IoT-A, the presented architecture emphasizes localized autonomy, minimal cloud dependence, and integration simplicity, using *Home Assistant* as the programmable core environment. Each layer described below is integrated within this specific context and collectively supports a cohesive, privacy-oriented automation ecosystem.

3.1.1. Device Layer

The device layer constitutes the physical interface between the system and its environment [30,31]. It is responsible for sensing and actuation [32,33], with detected events propagated upward to the higher logic layers for control decisions [34]. The deployment includes sensors for temperature, humidity, illumination, CO_2, and motion, as well as actuators such as smart relays, dimmers, thermostats, and HVAC modules. Devices communicate using Wi-Fi, Zigbee, Z-Wave, Bluetooth Low Energy (BLE), and Thread protocols. Compared with cloud-oriented systems that rely heavily on proprietary hubs, our solution integrates open-standard gateways (*Zigbee2MQTT*, *SkyConnect*, etc.) directly with the Home

Assistant core. The system supports both unidirectional and bidirectional communication streams and abstracts device behavior through entity modeling. Protocol selection (e.g., Zigbee for mesh, Wi-Fi for high-bandwidth connections) is aligned with energy, latency, and deployment constraints. The novelty lies in the adaptive multi-gateway configuration embedded within a locally orchestrated Home Assistant environment, offering plug-and-play capability without external cloud dependence.

3.1.2. Communication Layer

This layer ensures reliable message exchange between devices and higher-level logic. Unlike distributed IoT platforms that rely on internet-hosted brokers, our implementation employs a local MQTT broker (Mosquitto) complemented by native WebSocket integrations. MQTT guarantees minimal network overhead and supports multiple QoS levels, ensuring reliable propagation of state updates and event triggers in low-bandwidth environments. Auto-discovery mechanisms and network services such as DHCP, mDNS, and Zeroconf enable zero-configuration deployment. Protocol translation through multi-protocol gateways allows seamless integration across heterogeneous device ecosystems. In contrast to cloud-centric architectures, communication remains fully operational in isolated local networks. This design departs from traditional MQTT topologies by maintaining a self-hosted, resilient messaging backbone that ensures unified data exchange and application-level consistency.

3.1.3. Data Layer

This layer manages real-time telemetry and historical records. All devices are modeled as entities with persistent attributes, state histories, and unique identifiers. While many IoT frameworks rely on external data lakes, *Home Assistant* incorporates an internal SQLite engine, with optional extensions to PostgreSQL and InfluxDB for large-scale time-series analytics. The modularity of the proposed architecture allows dynamic reconfiguration of the data backend to meet scalability requirements while maintaining compatibility with visualization tools such as Grafana. Templating, filtering, and event-trigger mechanisms are supported through Jinja2 logic and the native Recorder, Logbook, and History modules. Unlike conventional IoT platforms, the data layer in our implementation supports real-time feedback loops and low-latency decision-making directly at the edge.

3.1.4. Management Layer

This layer represents the decision-making core of the system, implemented through YAML-defined automations and UI-driven flow editors. The trigger–condition–action paradigm, enhanced by a high-level templating engine, allows for granular control over automation responses. Each automation can be represented as a finite-state reactive process. The use of composable YAML logic and complete offline operability differentiates our system from commercial cloud solutions that typically introduce round-trip latency. Central orchestration, local script execution, and synchronization across device clusters are supported natively. In contrast to FIWARE-based rule engines that depend on centralized orchestrators, this layer enables fully decentralized control logic at the edge.

3.1.5. Security Layer

Security is embedded across all architectural layers. Transport Layer Security (TLS), role-based access control (RBAC), VLAN-based network segmentation, and least-privilege integration policies are enforced throughout the system. Local-only processing is used by default, ensuring that no external APIs are exposed unless explicitly authorized by the user. This contrasts with remote API-based architectures, where security is typically delegated to third-party providers. Integration with `Let's Encrypt` enables automatic certificate

renewal, while audit logging and multi-user management ensure operational traceability. The entire stack can be deployed within isolated VLANs, providing both physical and logical separation for enhanced security.

3.1.6. IoT Service Layer

This layer provides interfaces for automation, analytics, and third-party integrations. REST and WebSocket endpoints enable bidirectional interactions—both for internal automations and for external systems such as Node-RED or mobile applications. The architecture supports stateless service calls using JSON-formatted payloads, promoting compatibility with diverse IoT ecosystems. Compared to monolithic broker-based designs, our microservice-compatible service layer facilitates distributed deployment, modular scaling, and fault isolation.

3.1.7. Presentation Layer

The Presentation Layer implements the user interface through the *Lovelace UI*, which provides reactive dashboards, customizable widgets, and role-based visualization modes. All dashboards update entity states in real time without requiring page reloads. This layer is fully decoupled from the automation logic, ensuring system resilience even in the event of UI failure. In contrast to web-hosted portals in other architectures, the interface remains fully operational without internet access. The use of local-only HTTPS communication, encrypted storage, and persistent caching further enhances reliability and privacy.

Each layer of the proposed architecture is grounded in well-established design paradigms yet implemented in a novel configuration that supports full local autonomy, modular extensibility, and cross-platform interoperability.

Although the high-level overview presented in Figure 1 illustrates the system's structural organization, each module is explicitly detailed in the subsections that follow, outlining its function, interaction mechanisms, and role within the complete deployment. This alignment between conceptual design and implementation ensures reproducibility and transparency of the proposed architecture, enabling direct mapping to real-world configurations.

4. Functional Subsystems of the Local Automation Server

Building upon the layered architecture introduced in the previous section, this part focuses on the decomposition of the automation platform into practical subsystems that operate within a *Home Assistant*–based deployment. Each subsystem addresses a specific domain of smart environment management, integrating sensors, actuators, logic processors, and user interfaces for monitoring and control.

Unlike conventional cloud-dependent platforms, the proposed system ensures complete local autonomy, enabling continuous operation even in the absence of internet connectivity. Figures 2 and 3 provide architectural overviews from subsystem and software perspectives, respectively.

Each subsystem is self-contained yet interoperable, supporting both simple automations and complex multi-domain control workflows. The modular implementation approach allows incremental deployment and domain-specific scaling, facilitating system evolution without architectural redesign.

The subsystems include:

- Security subsystem–detects and mitigates physical and environmental threats in real time.
- Climate control subsystem–regulates temperature, humidity, and air quality through automated ventilation management.

- Lighting subsystem–optimizes illumination levels based on environmental and occupancy data.
- Power management subsystem–monitors energy use and executes adaptive energy-saving strategies.
- System services subsystem–ensures runtime integrity, notifications, and automated data backups.

The overall configuration promotes interoperability through open standards, hybrid device integration, and modular scalability. A comparative performance review against existing IoT architectures is presented in Section 6.

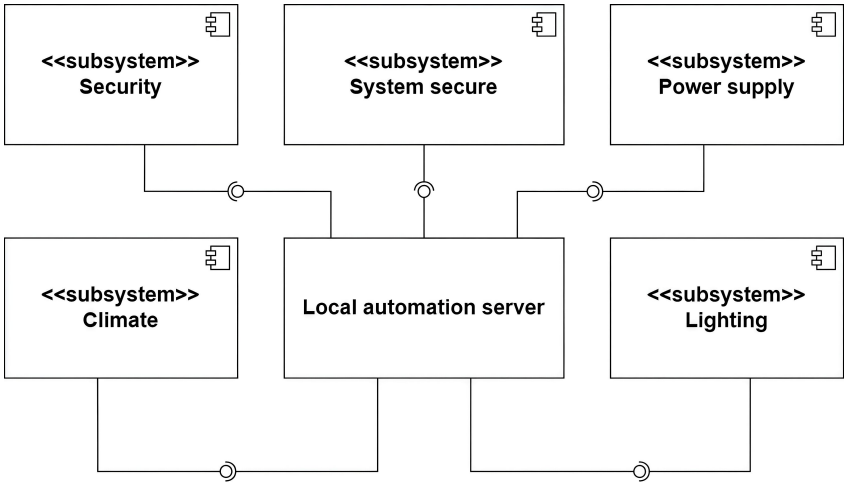


Figure 2. General model of functional subsystems of the local automation server based on Home Assistant. The figure illustrates the decomposition of the automation platform into five interoperable subsystems—security, climate, lighting, power management, and system services—highlighting bidirectional data exchange and event-driven coordination among them.

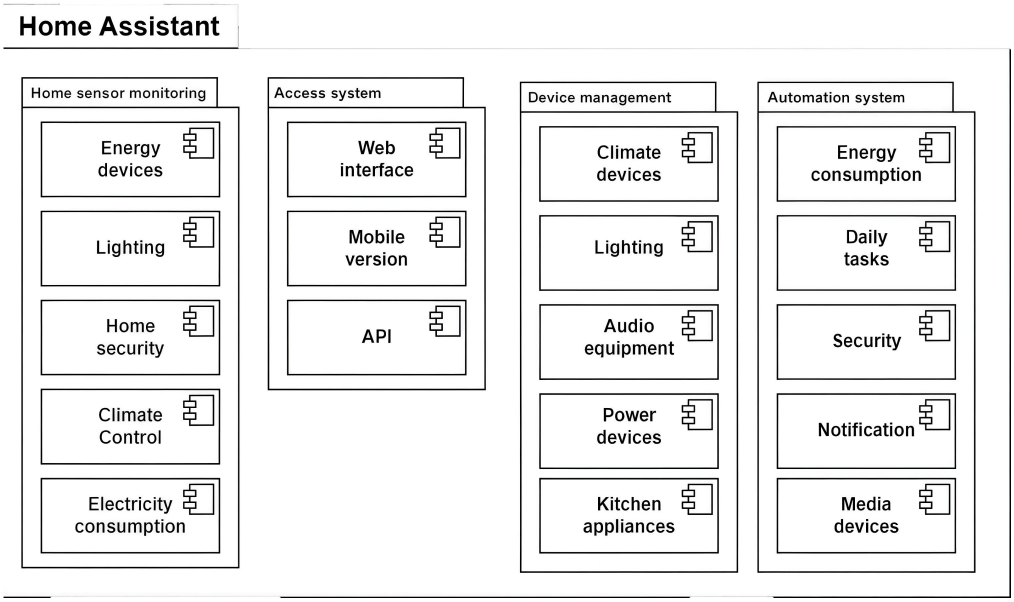


Figure 3. Logical architecture of software modules in the Home Assistant system. The schematic represents the software architecture of the Home Assistant instance, showing module-level communication, data persistence, and interaction with add-ons such as MQTT, InfluxDB, and Grafana.

4.1. Security Subsystem

The security subsystem is a critical component of the local automation server, providing real-time monitoring, threat detection, and automated response execution according to

predefined rules. Its architecture supports flexible integration of commercial off-the-shelf (COTS) and custom-built (DIY) sensors via standards such as ESPHome, Zigbee2MQTT, Bluetooth Low Energy (BLE), and MQTT. An overview of the subsystem's internal structure, implemented in *Home Assistant*, is shown in Figure 4. The diagram visualizes the interaction among sensors, actuators, communication modules, and event-processing services, forming a complete pipeline for real-time threat detection and mitigation.

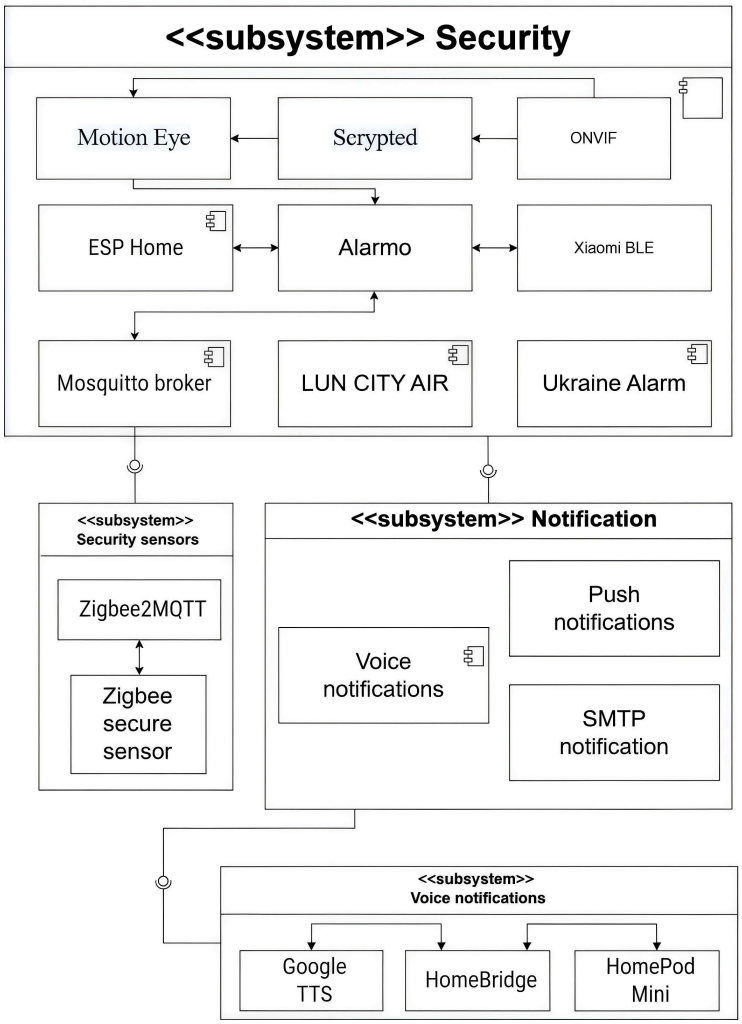


Figure 4. Structure of the Security Subsystem in the Home Assistant System. The figure outlines the components responsible for threat detection and mitigation, including sensors, communication modules, event-processing logic, and alert channels. Arrows represent the real-time flow of information from sensor inputs to user notifications.

The subsystem supports multiple safety mechanisms, including detection of water leaks, door/window openings, vibration, carbon monoxide, and other environmental hazards. Captured events are processed by the Home Assistant's event-driven core, which dynamically triggers predefined automation routines. Integration with video surveillance is achieved through ONVIF support and the *MotionEye* module, enabling motion detection, automated recording, event-based alerts, and live streaming via web or mobile interfaces. Environmental threat monitoring is also supported through integrations such as LUN City Air and Ukraine Alarm, delivering real-time data on air quality, pollution, and chemical or radiological alerts. This feature provides added value for high-risk or industrial environments. User notifications are issued via a multichannel system—push alerts, email

(SMTP), voice announcements (Google TTS, HomePod, HomeBridge), and local alarm signals—allowing fully configurable and context-aware responses.

To further validate the robustness of the security mechanisms, Table 2 summarizes the STRIDE-based threat model and corresponding mitigation strategies implemented in the proposed architecture.

Table 2. Summary of STRIDE Threats and Corresponding Mitigation Strategies.

Threat Category	Mitigation Strategy
Spoofing (S)	Multi-factor authentication, unique API tokens for device pairing, and encrypted MQTT communication (TLS 1.3).
Tampering (T)	File integrity verification, version-controlled configuration backups, and container-level isolation within Proxmox virtualization.
Repudiation (R)	Cryptographically signed audit logs with timestamping and user attribution.
Information Disclosure (I)	Local-only data processing, end-to-end TLS encryption, and explicit restriction of external API exposure.
Denial of Service (D)	MQTT rate limiting, container resource quotas, and automatic service restart under overload conditions.
Elevation of Privilege (E)	Role-Based Access Control (RBAC) enforced through Home Assistant Supervisor and Linux permission management.

Note. STRIDE mapping demonstrates how each identified threat is mitigated at both the system and container levels, ensuring confidentiality, integrity, and availability in local deployments.

From a regulatory and data protection standpoint, the proposed architecture adheres to the principles of the General Data Protection Regulation (GDPR, EU 2016/679). All sensor data, user profiles, and event logs are stored and processed exclusively within the local network, preventing any transfer of personal information to external servers or third-party platforms. User consent and purpose limitation are inherently enforced through local configuration files that define data retention policies. Each automation module maintains audit logs managed by the Home Assistant Supervisor, ensuring full traceability of configuration and access events. In multi-user environments, authentication and RBAC restrict administrative privileges to authorized users, while audit logs are timestamped and cryptographically signed to prevent tampering. Collectively, these measures ensure GDPR-compliant data minimization, transparency, and accountability.

4.2. Climate Subsystem

The climate subsystem is a key functional module of the local automation server, responsible for maintaining comfortable and safe indoor environmental conditions across residential, commercial, and educational settings. It automates the regulation of temperature, humidity, carbon dioxide concentration, and other air quality parameters. Figure 5 illustrates a typical implementation of this subsystem in *Home Assistant*, highlighting the core relationships among sensors, actuators, and control logic that form adaptive environmental feedback loops.

The subsystem follows a modular, vendor-agnostic design that supports both commercial hardware (*Sonoff, Shelly, Aqara, Tuya*) and custom DIY sensors, ensuring flexibility with respect to deployment scale, user preferences, and budget constraints. Sensor data are transmitted in JSON format, processed by the Home Assistant event-driven engine, and stored in time-series databases for long-term analysis. The main functions include controlling heating, ventilation, and air conditioning (HVAC) systems according to environmental readings and predefined automation rules. For instance, heating is activated when the

temperature falls below a target threshold, while cooling is engaged when it exceeds the set limit. Humidity is regulated using humidifiers or dehumidifiers to maintain comfort levels.

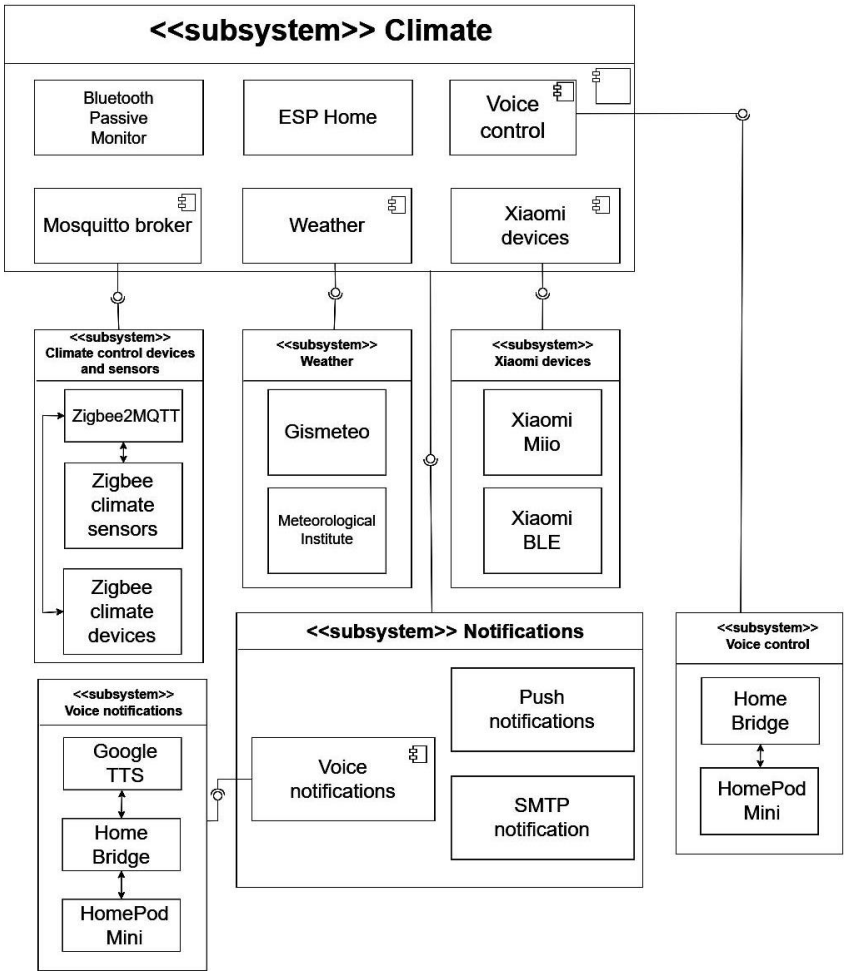


Figure 5. Structure of the Climate Subsystem in the Home Assistant System. The diagram visualizes the connections between environmental sensors, HVAC actuators, control logic, and data storage modules. It shows feedback loops for temperature and humidity regulation based on dynamic thresholds and contextual inputs.

In contrast to static HVAC configurations, the Home Assistant-based subsystem offers adaptive control through contextual inputs such as user schedules, geolocation, occupancy detection, and external weather data. This design enhances both comfort and energy efficiency by automatically adjusting HVAC operation to real-time conditions. The subsystem also integrates external data sources (e.g., OpenWeatherMap), occupancy indicators (BLE beacons, Wi-Fi tracking), and time-based routines to further optimize operation. Cross-subsystem coordination is supported—for example, triggering ventilation when CO_2 levels rise or pausing fresh air intake when outdoor air quality deteriorates. Feedback is provided through dashboards, trend graphs, mobile applications, and voice assistants, enabling user-informed, unobtrusive environmental management.

4.3. Lighting Subsystem

The lighting subsystem constitutes another core component of the local automation server, delivering a comfortable, functional, and energy-efficient illumination environment. Beyond its visual role, lighting contributes to user comfort, concentration, and circadian well-being. Its primary objective is to automate artificial lighting control by evaluating both external and internal factors such as ambient light, occupancy, time of day, and

personalized user scenarios. This data-driven approach reduces energy consumption while maintaining ergonomically optimized illumination conditions. For instance, in offices or classrooms, lighting is activated only when presence is detected or during scheduled sessions; in residential environments, brightness can be adjusted dynamically for relaxation or programmed to simulate presence while occupants are away.

The subsystem supports a wide range of capabilities—from simple on/off control to dynamic adjustments of brightness and color temperature based on time of day or user activity (e.g., work, study, leisure). Motion, light, and door sensors combined with smart switches create an interactive, responsive lighting ecosystem. For example, artificial lights are dimmed or turned off once daylight reaches sufficient levels, while gradual dimming in the evening supports circadian alignment. Figure 6 shows the subsystem configuration in the *Home Assistant* environment.

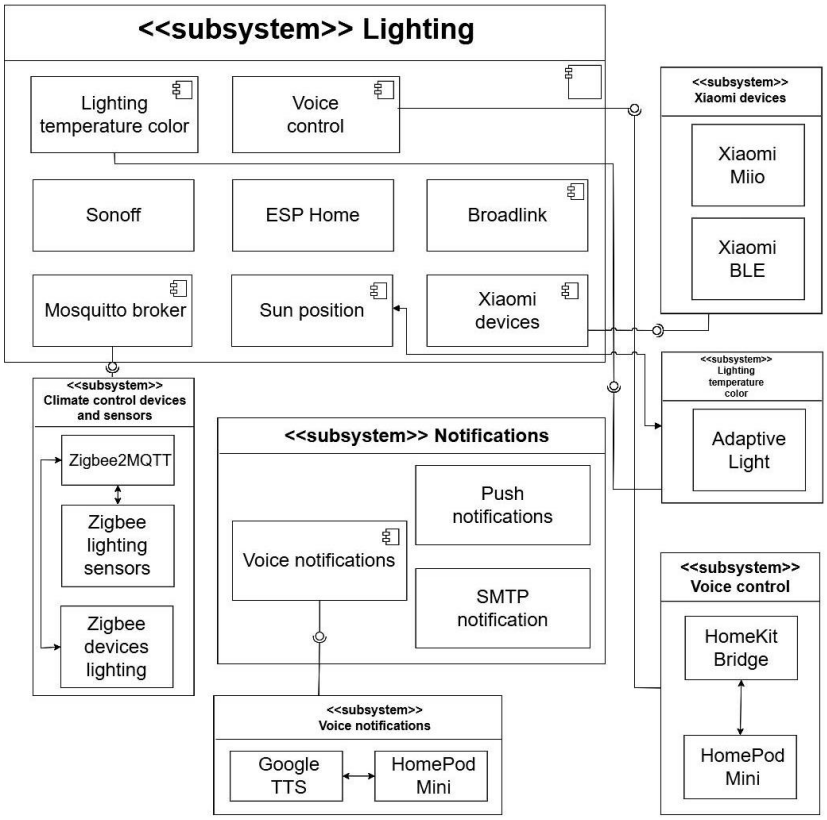


Figure 6. Structure of the Lighting Subsystem in the Home Assistant System. The illustration highlights integration among motion and light sensors, smart switches, dimmers, and automation rules that dynamically adjust brightness and color temperature. It demonstrates how lighting adapts to occupancy, daylight levels, and user-defined scenes.

Integration within the broader Home Assistant ecosystem enables lighting to react to contextual events across subsystems. For example, lights can be linked to presence detection (via motion or geolocation), window state (via contact sensors), or multimedia activity (e.g., dimming during video projection). Support for open protocols such as Zigbee, Z-Wave, MQTT, ESPHome, and Modbus ensures interoperability across heterogeneous lighting devices from multiple vendors. Users can define lighting scenes such as “Study,” “Break,” or “Meeting,” each activating customized configurations. Control can be exercised automatically or manually via mobile apps, wall panels, or voice assistants (Google Assistant, Amazon Alexa).

The subsystem also supports adaptive lighting, dynamically modifying both light intensity and spectral composition. Cool white light is used in the morning to enhance

alertness, while warm tones in the evening support relaxation and sleep regulation. In educational and office contexts, this adaptability contributes to improved focus and reduced visual fatigue. Overall, the lighting subsystem exemplifies the balance of modularity, scalability, and context-awareness achieved within the Home Assistant framework, merging energy efficiency, ergonomic lighting design, and user personalization.

4.4. Power Management Subsystem

The power management subsystem adopts a modular, standards-based design that supports open communication protocols, enabling interoperability with both commercial systems (e.g., *EcoFlow, Sonoff, Shelly, APC*) and custom solutions such as ESP32-based metering modules, NUT servers, or DIY backup power units. As shown in Figure 7, the subsystem architecture accommodates site-specific deployment requirements, balancing cost, infrastructure availability, and reliability.

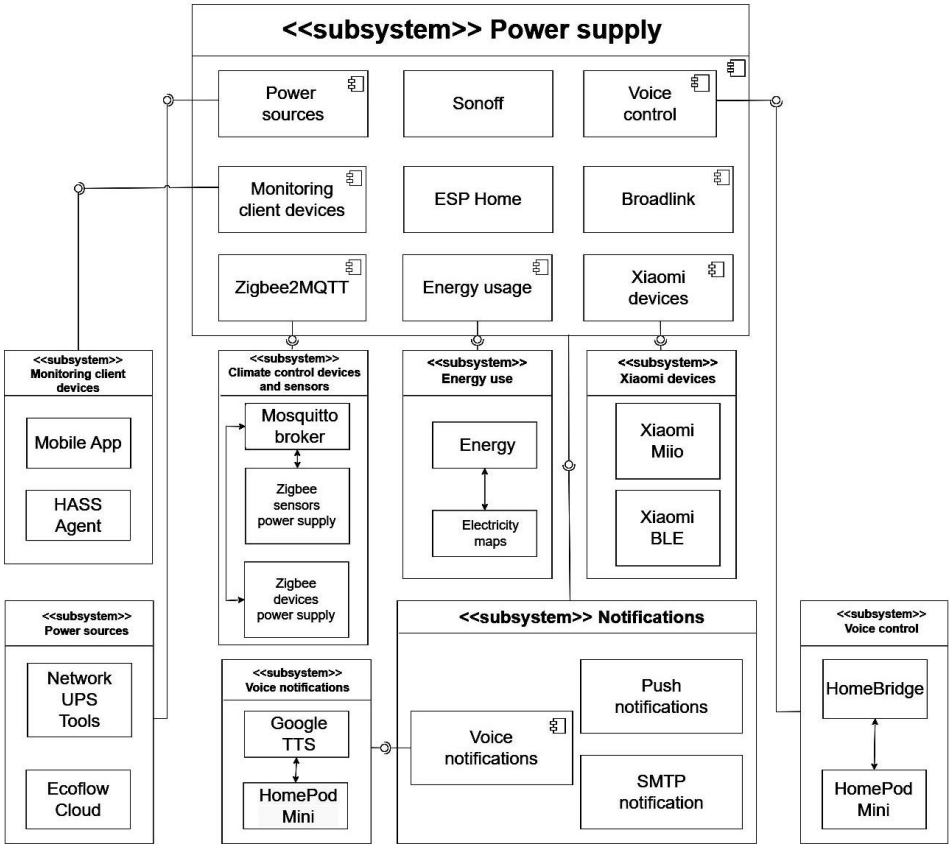


Figure 7. Structure of the Power Management Subsystem in the Home Assistant System. The figure illustrates energy flow and monitoring processes within the subsystem, including data acquisition from metering modules, storage in time-series databases, and automation routines for load balancing and power-failure recovery.

Subsystem components communicate via MQTT, Modbus, Zigbee, or REST API protocols, transmitting JSON-formatted telemetry to the system core for processing, visualization, and rule-based control. The subsystem's key functions include continuous monitoring of voltage, current, power, and frequency, as well as energy-use analytics at both the device and zone levels. Measurements collected from digital meters and sensors are stored in time-series databases, facilitating trend identification, load forecasting, and early anomaly detection. In case of a power failure or abnormal conditions, the system can automatically isolate non-critical loads or activate backup power systems such as UPS units or battery stations.

Unlike traditional energy management systems with static logic, the proposed power management subsystem enables context-aware, adaptive control strategies. These strategies may incorporate dynamic factors such as electricity tariffs, occupancy schedules, weather forecasts, and the operational state of other subsystems. This allows for the implementation of energy-saving techniques such as peak shaving, dynamic load shifting, and time-of-use optimization, all without external cloud dependency. Users access comprehensive visual analytics via dashboards and reports, while critical events—such as overloads, outages, or battery failures—generate immediate notifications through mobile apps or email.

Through its integration with other subsystems, the power management layer contributes to system-wide resilience and energy efficiency. For instance, during a blackout, the system can reduce HVAC operation, dim lighting, and shift the security subsystem into power-saving mode. Overall, the subsystem provides the foundation for sustainable, context-aware, and fault-tolerant operation within the local automated environment.

4.5. System Services Subsystem

The system services subsystem represents a fundamental layer of the monitoring and control infrastructure, ensuring operational continuity, reliability, and information security within the local automation server. As shown in Figure 8, its functionality extends beyond conventional telemetry and monitoring—providing an adaptive runtime environment capable of responding to both internal disruptions and external threats.

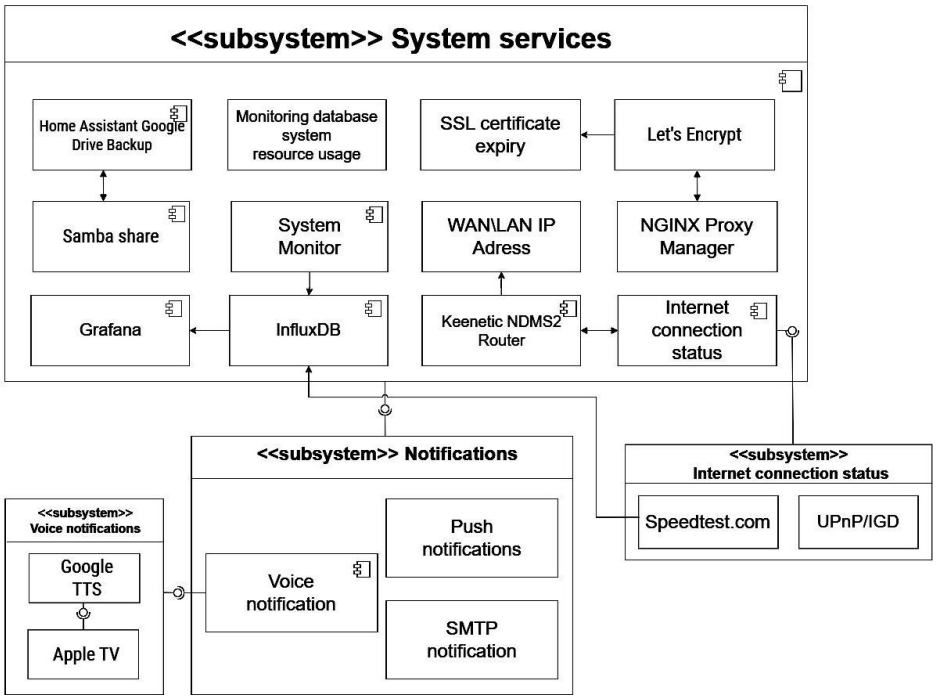


Figure 8. Structure of the System Services Subsystem in Home Assistant. The diagram presents the components responsible for diagnostics, network auditing, SSL certificate management, and alerting. It illustrates real-time monitoring and feedback channels that ensure high availability and operational resilience.

A key function of this subsystem is the dynamic monitoring of server performance and runtime processes. By continuously evaluating hardware and software states, the system detects anomalies such as resource overuse, service failures, or unexpected process behavior. Proactive diagnostics not only capture current issues but also enable prediction of potential failures and the initiation of preventive maintenance routines.

The subsystem also provides network auditing and activity analysis, covering both local communications and external access events. It logs details about connected clients, access points, session durations, and interaction frequency, enabling detection of anomalies that may indicate unauthorized access attempts. In this capacity, the subsystem functions as an integrated network audit and security monitor. System integrity is further ensured through automatic backup verification routines, which check the presence, completeness, and recoverability of stored archives. These measures guarantee full system restoration capability in the event of a service disruption or data loss.

Cryptographic management constitutes another key aspect of this subsystem. The system services layer oversees Secure Socket Layer (SSL) certificate management, automatically tracking expiration dates and validity status to prevent potential lapses in encrypted communication or administrative access. Timely renewal actions mitigate the risk of unplanned downtime. The subsystem also incorporates a multi-channel notification engine responsible for communicating system health, technical incidents, and administrative events. Alert delivery is tailored to severity levels and user preferences through channels such as email, mobile notifications, and synthesized voice prompts.

Collectively, these capabilities make the system services subsystem a cornerstone of reliability and observability within the automation framework, ensuring sustained operational resilience, data integrity, and administrative transparency.

4.6. Data-Driven Enhancement of Automation Logic

While the core automation engine in *Home Assistant* adheres to a rule-based paradigm—employing the trigger–condition–action model—the proposed architecture introduces a hybrid data-driven logic layer. This approach bridges deterministic control and adaptive intelligence by integrating statistical learning directly into the automation pipeline, enabling the system to adjust its behavior dynamically in response to environmental data.

To achieve this, a data-driven augmentation mechanism was deployed to complement existing manual rules. The mechanism continuously evaluates key contextual variables (e.g., temperature, CO_2 concentration, and occupancy) and infers optimal control thresholds based on historical patterns and environmental correlations.

This hybrid mechanism is implemented through three main components:

- Time-series analysis: Sensor data are aggregated in InfluxDB and visualized via Grafana to extract typical daily profiles for temperature and CO_2 concentration.
- Dynamic rule injection: Automation thresholds (e.g., HVAC setpoints) are calculated using rolling averages and statistical deviations (e.g., z-scores) and applied to YAML-based automations through templating.
- Anomaly-aware automation: In cases of abnormal environmental fluctuations (e.g., sudden CO_2 spikes), the automation logic adapts ventilation or climate parameters in real time.

This integration of learned behavioral patterns enhances system responsiveness, personalization, and operational efficiency—particularly in multi-zone dynamic environments—while preserving transparency and full local autonomy.

The data-driven automation module employs lightweight statistical learning algorithms seamlessly integrated into the automation workflow. In the current deployment, ridge regression and bagged decision tree regressors were implemented to predict short-term energy demand and detect anomalies in device operation. Ridge regression was chosen for its robustness to multicollinearity and noise in locally collected sensor data, providing stable and consistent predictions. Decision tree ensembles complement this by modeling nonlinear dependencies between temperature, occupancy, and power consumption variables.

To prevent model drift, a rolling evaluation and retraining mechanism was introduced. Model accuracy is re-assessed every seven days using the root mean squared error (RMSE) criterion. If the RMSE exceeds 15% of the baseline level, the model is automatically retrained using the most recent data stored in the local InfluxDB repository. This adaptive retraining cycle ensures continuous self-calibration of the hybrid automation engine without any reliance on cloud resources, thereby maintaining adaptability and statistical robustness over time.

The resulting hybrid architecture achieves a balance between deterministic rule-based logic and adaptive statistical control. Operating entirely within the Home Assistant environment, it combines modular extensibility, local execution, and advanced orchestration features through the Supervisor and integrated add-ons—ensuring intelligent, privacy-preserving, and fully autonomous automation.

5. Technical Implementation, Communication, and Data Protection in the Local Automation Environment

The successful deployment of a robust, scalable, and secure smart automation system depends on the design of a comprehensive communication infrastructure that integrates heterogeneous devices, multi-tier communication pathways, and distributed management services. Within increasingly complex digital ecosystems—particularly in industrial, academic, and commercial domains—the communication layer acts as the vital bridge between the physical sensing environment and the logical control modules, enabling coherent coordination, adaptability, and context-aware automation across all system components.

The proposed local automation framework adheres to the principles of modularity, decentralization, and functional isolation. This design ensures flexible scalability, optimized data routing, and a minimized risk of single points of failure. Virtualization, realized through a hypervisor-based environment, provides an abstraction layer between hardware and software resources, thereby simplifying deployment, maintenance, and version control processes. At the core of this infrastructure is a Home Assistant-based control node, responsible for event aggregation, scenario orchestration, and real-time human–machine interaction.

Information security represents a foundational design objective of the communication infrastructure. Strong security measures are embedded at multiple layers to guarantee communication reliability, data integrity, threat resilience, and controlled system access. These safeguards are implemented using a combination of hardware-level protections and software-based countermeasures that collectively reinforce the system's defense-in-depth architecture.

5.1. Infrastructure of the Local Automation Server

The backend architecture of the Home Assistant-based monitoring and automation environment [6] is structured around five key design principles: modularity, functional decomposition, decentralized processing, interoperability, and fault tolerance (Figure 9). This configuration enables autonomous system regulation, elastic scaling, efficient failure containment, and high-fidelity data management in real time [35]. By logically separating the physical and software domains, the architecture maintains granular control, transparency, and maintainability—qualities essential for modern cyber-physical and information-centric infrastructures.

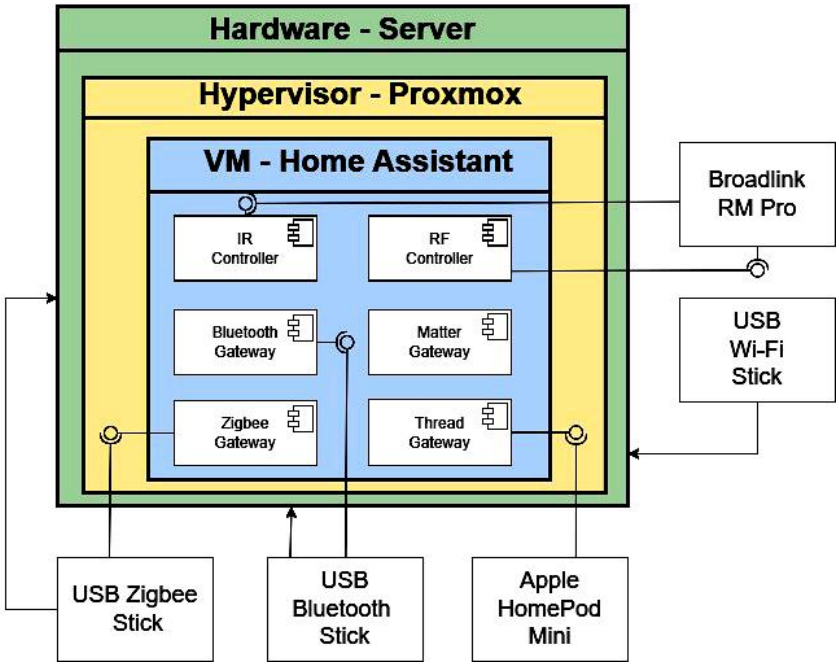


Figure 9. Architecture of the Automation System's Server Environment. The diagram provides a hierarchical overview of the Proxmox-based virtualization environment, illustrating three structural layers: the physical hardware base, the hypervisor level, and logical containers. It highlights the functional separation of services such as Home Assistant, MQTT, InfluxDB, and Grafana to ensure isolation, modularity, and security.

At the core of this architecture lies the Home Assistant platform [6], which functions as the intelligent coordination hub for aggregating, analyzing, and controlling distributed IoT resources. In contrast to traditional cloud-dependent frameworks, Home Assistant supports edge-based local event processing [35], thereby reducing network latency, eliminating cloud dependence, and enhancing data sovereignty and system resilience. This approach enables the automation engine to execute fully offline operations while maintaining real-time responsiveness and end-to-end data privacy.

5.1.1. Physical Layer

The physical layer forms the hardware foundation of the automation infrastructure and plays a pivotal role in ensuring reliability, performance, and continuous operation. It is implemented as a dedicated local server node equipped with a multi-core CPU, sufficient RAM, non-volatile SSD/HDD storage, and high-speed network interfaces supporting gigabit or multi-gigabit connectivity. An uninterruptible power supply (UPS) provides protection against power interruptions. The entire system operates within a closed local network without dependency on external cloud services, thereby ensuring full autonomy, information sovereignty, and complete user control over data.

5.1.2. Hypervisor Layer

This layer is realized using the Proxmox Virtual Environment [36], built on top of the KVM hypervisor [37] and LXC containerization framework [38]. Proxmox provides strong virtualization features, including instance isolation, live migration, high availability (HA) clustering, and snapshot-based backup. It also offers a RESTful API and comprehensive command-line interface for administrative automation. The architecture supports a microservice-based design, where functional modules such as Home Assistant, TimescaleDB, InfluxDB, the MQTT broker [39], Grafana, and Node-RED operate in

independent, resource-isolated containers. The use of ZFS and BTRFS file systems adds further resilience through native deduplication, checksumming, and integrity verification. Integration with Prometheus and the Proxmox Backup Server enhances reliability and disaster recovery.

5.1.3. Logical Layer

The logical layer is deployed as a virtual machine running the Home Assistant Operating System [6], which manages devices, events, automation rules, and user interactions. Home Assistant supports more than 2500 integrations, enabling seamless interoperability with both open and proprietary standards. These include Zigbee [40], Z-Wave [41], Thread [42], Matter [43], BLE [44], Wi-Fi [45], as well as selected external services such as IFTTT [46], Telegram [47], and Weather API [48].

Communication within this layer is handled via MQTT [39], WebSocket [49], REST API [50], and TCP/IP [51], ensuring bidirectional message exchange and interoperability with third-party systems. Together, these three layers provide an optimal balance of performance, reliability, and scalability—key prerequisites for secure and efficient IoT operation in autonomous environments.

5.2. Device Connectivity to the Network

A major advantage of the Home Assistant platform [6] lies in its comprehensive support for heterogeneous communication protocols and device standards [39,40,42,43]. This flexibility enables the creation of adaptive and scalable network infrastructures tailored to the technical characteristics and operational roles of connected devices. Figure 10 illustrates the topological structure of the physical and virtual components in the local computing environment.

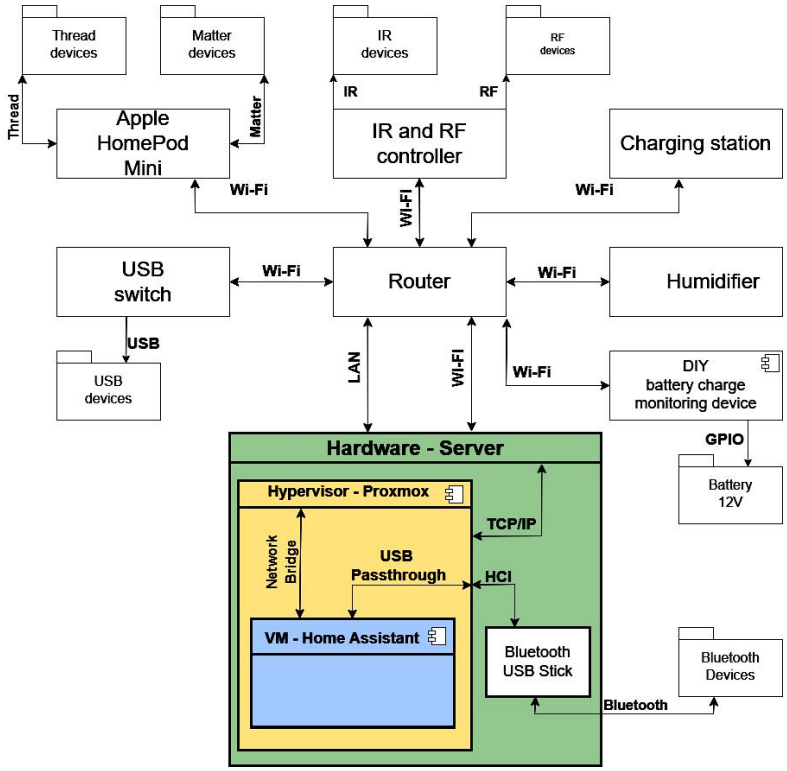


Figure 10. Architectural diagram of the server infrastructure and device connectivity protocols. The diagram depicts the interaction between virtualized servers, communication gateways, and IoT devices using Zigbee, Wi-Fi, and Thread networks. It highlights hybrid connectivity and secure segmentation of communication flows.

At the network core is a router that performs both switching and routing between wired (LAN) and wireless (Wi-Fi) segments [45]. It connects to a variety of devices, including smart automation modules (e.g., humidifiers, chargers), DIY ESP8266/ESP32 controllers [52], infrared/RF bridges, and USB/Bluetooth peripherals [44]. The Proxmox server [36] hosts a virtualized ecosystem where Home Assistant [6] operates as the central automation orchestrator. Connectivity between physical and virtual components is achieved through USB passthrough, Bluetooth HCI, or TCP/IP network bridging [51].

An Apple HomePod Mini functions as a gateway for devices using the Thread [42] and Matter [43] protocols, expanding interoperability across vendors. A Wi-Fi-enabled USB hub allows dynamic peripheral allocation among host systems, which is particularly useful in laboratory and testing environments. GPIO access in DIY configurations supports direct control over I/O operations. This hybrid network design combines modularity, open standards, and virtualization to establish a resilient, scalable, and fully cloud-independent automation infrastructure.

5.3. Communication Security

Ensuring secure communication in local automation systems is crucial, particularly in deployments that operate autonomously or exchange data with external services. In the proposed architecture—based on Home Assistant [6] and virtualized through Proxmox [36]—security mechanisms are distributed across multiple layers, addressing physical segmentation, encryption, and logical service isolation.

The infrastructure follows a segmented topology: each gateway, controller, and peripheral connects via predefined interfaces within VLAN/LAN-enclosed subnets, minimizing exposure and reducing the attack surface. Inter-device communication takes place over encrypted channels using secure protocols such as MQTT with TLS [39], HTTPS, and WebSocket with token-based authentication [49,53]. Because all event processing occurs locally, Home Assistant eliminates dependency on remote clouds, effectively preventing data leakage and maintaining confidentiality. Virtual machines ensure isolation between service functions, while USB passthrough [54] provides controlled access to physical interfaces without compromising security.

Secure video surveillance is supported via RTSP and ONVIF [55], with on-premise object detection powered by Frigate or MotionEye [56]. All video data are stored exclusively on internal drives, ensuring compliance with privacy regulations and protection against external interception. Monitoring and analytics tools such as Prometheus, Grafana, and Loki [57,58] continuously track network activity, detect anomalies, and log suspicious behavior (e.g., port scans or unauthorized access attempts). These mechanisms collectively establish a proactive cybersecurity layer, combining encryption, authentication, localized processing, and infrastructure segmentation to meet contemporary IoT security and privacy standards.

5.4. Hybrid Evaluation Metric for Smart Automation Systems

To capture the multidimensional performance of local IoT-based automation systems, we introduce a hybrid evaluation metric that jointly quantifies *efficiency* and *reliability*. This unified measure enables consistent benchmarking across heterogeneous deployments, facilitating direct comparison of system behavior under various operational conditions.

Let the hybrid effectiveness score $\mathcal{H}$ be defined as:

$$\mathcal{H} = \omega_1 \cdot \left(1 - \frac{E_{\text{avg}}}{E_{\text{base}}}\right) + \omega_2 \cdot \left(1 - \frac{L_{\text{avg}}}{L_{\text{max}}}\right) + \omega_3 \cdot \frac{U}{T} + \omega_4 \cdot R_{\text{score}} \quad (1)$$

where:

- $E_{\text{avg}} \pm \sigma_E$—average daily energy consumption (Wh) with standard deviation σ_E;
- E_{base}—baseline energy consumption without automation (Wh);
- $L_{\text{avg}} \pm \sigma_L$—average system response latency (ms) with standard deviation σ_L;
- L_{max}—maximum tolerable latency threshold (ms);
- U—cumulative system uptime during the observation period (h);
- T—total duration of the evaluation period (h);
- R_{score}—normalized recovery score derived from fault-injection experiments, within [0,1];
- ω_i—weighting coefficients satisfying $\sum_{i=1}^{4} \omega_i = 1$.

All parameters are normalized to a common scale [0,1], allowing aggregation across heterogeneous measurement domains. Each indicator reflects a key system property: energy efficiency, latency reduction, reliability, and recovery performance. To ensure statistical robustness, each term ($E_{\text{avg}}, L_{\text{avg}}, U/T$) is accompanied by variance, standard deviation, and 95% confidence intervals computed using Student's t-distribution. This allows for the verification of statistical significance when comparing performance across different deployments or configurations.

The weighting coefficients ω_i $(i = 1, \ldots, 4)$ were initially assigned equal values ($\omega_i = 0.25$), providing balanced representation of all dimensions in the absence of empirical bias. A subsequent sensitivity analysis was performed by varying each ω_i within the interval [0.1, 0.4] while maintaining $\sum \omega_i = 1$. The resulting variation in $\mathcal{H}$ did not exceed 4.7% across all test cases, confirming the metric's stability and demonstrating that the equal-weight configuration offers a fair and unbiased evaluation of system performance.

The hybrid metric thus provides a composite measure of overall system effectiveness: higher $\mathcal{H}$ values indicate greater operational efficiency, lower latency, improved reliability, and faster recovery under fault conditions. The metric can be continuously computed in real time using monitoring tools such as Prometheus, Grafana, or structured event logs from the automation engine, enabling ongoing adaptive evaluation during system operation.

6. User Interface and Automation Panel Design

Figure 11 presents the main web interface of the local automation server built on the Home Assistant platform, accessed under an administrator-level account. This interface exemplifies the concept of centralized and real-time supervision of all smart environment subsystems. Its core component is an interactive dashboard that consolidates sensor data, device states, and service integrations, providing both a high-level operational overview and detailed control of individual modules.

The user interface employs a modular layout that ensures intuitive navigation and clear logical organization. A sidebar located on the left provides rapid access to the system's primary functional modules:

- Logbook–provides a chronological record of all system events, including device state changes, triggered automations, and user actions, facilitating time-based behavioral analysis;
- History–visualizes time-series data for sensors, switches, and thermostats over a defined period, supporting the assessment of indoor climate stability and power dynamics;
- Energy–presents electricity usage statistics, energy-source breakdowns (e.g., grid, solar), and device-level consumption metrics for diagnostic and optimization purposes;
- Backups–enables full configuration snapshots for secure local or cloud storage, allowing rapid system recovery in the event of hardware failure or misconfiguration;
- Zigbee2MQTT–offers a dedicated interface for managing Zigbee devices, including pairing, parameter editing, and network diagnostics;
- File Editor–provides in-browser access to YAML configuration files for editing automations, integrations, and dashboard layouts;

- Terminal–grants direct command-line access for administrative operations such as log inspection, updates, or network configuration.

Advanced administrative tools extend the system's configurability:

- Developer Tools–allow entity state inspection, service invocation, real-time diagnostics, and syntax validation, essential for debugging complex automation workflows;
- Settings–serve as the main configuration hub, covering account management, localization, access control, platform integrations (e.g., Google Assistant, Alexa), and system updates.

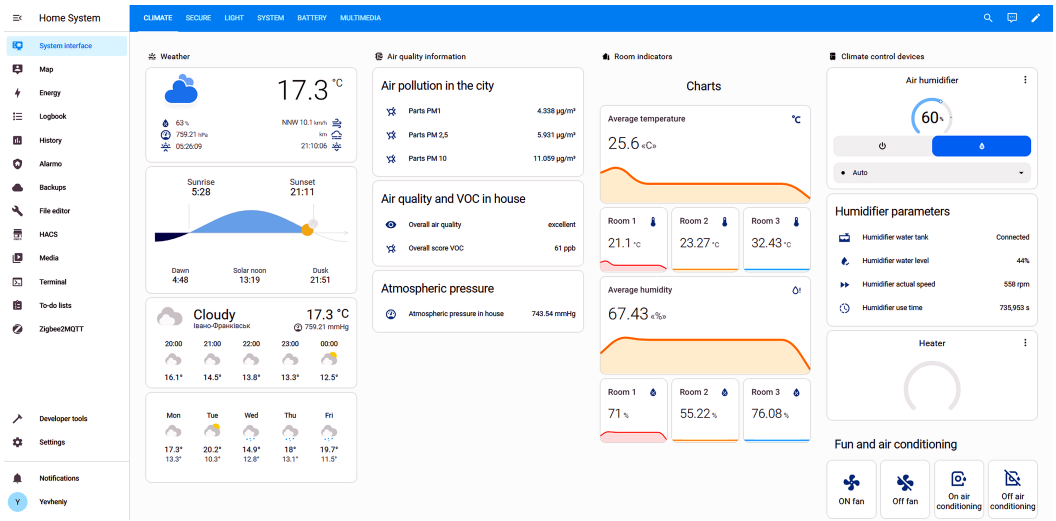


Figure 11. Main interface page of the Home Assistant-based local automation server. The screenshot illustrates the administrator dashboard integrating panels for climate, energy, and security management. Color-coded widgets present real-time sensor readings, system trends, and control elements for direct interaction.

Extensibility is further enhanced through the Home Assistant Community Store (HACS), which supports the integration of community-developed components, UI cards, automation blueprints, and visual themes. This modular ecosystem ensures that the platform remains adaptable to evolving technological standards and user requirements.

The central dashboard is organized around five major subsystems—climate, security, lighting, energy, and multimedia—providing structured, domain-specific visualization and control. Particular attention is devoted to the Climate section, which consists of four vertically aligned panels:

Weather–displays current conditions and forecasts:

- temperature, humidity, pressure, and wind speed/direction;
- daylight parameters such as sunrise, sunset, solar noon, and twilight;
- hourly and daily predictions for scenario planning.

Air Quality Information–monitors both indoor and outdoor parameters:

- particulate matter levels (PM_1, $PM_{2.5}$, PM_{10});
- volatile organic compounds (VOC);
- atmospheric pressure and the aggregated Air Quality Index (AQI).

These indicators provide the input for automatic or semi-automatic control of ventilation, filtration, and humidification systems.

Room Indicators–summarize environmental conditions per room:

- average and room-specific temperature and humidity values;
- 24-h trend graphs identifying deviations from comfort thresholds.

Climate Control Devices–enable direct interaction with HVAC components:

- target humidity configuration for humidifiers;
- status and control of fans, air conditioners, and heaters;
- display of operational mode, fan speed, and device-specific diagnostics.

Users may alternate between automatic and manual control modes, combining rule-based automation with direct user intervention. This hybrid approach enhances operational flexibility, ensures rapid responsiveness to environmental changes, and supports personalized comfort management.

6.1. Deployment Details and Real-World Setup

To validate the system under real-world operating conditions, a local automation server based on the Home Assistant platform was deployed across three office rooms and one service room, covering a total area of approximately 50 m^2 (3×15 m^2 offices and 1×5 m^2 service area). The deployment operated entirely offline within a virtualized environment built on Proxmox VE, as illustrated in Figure 12. The configuration included a supervised Home Assistant OS instance integrated with Zigbee2MQTT for low-power wireless device coordination.

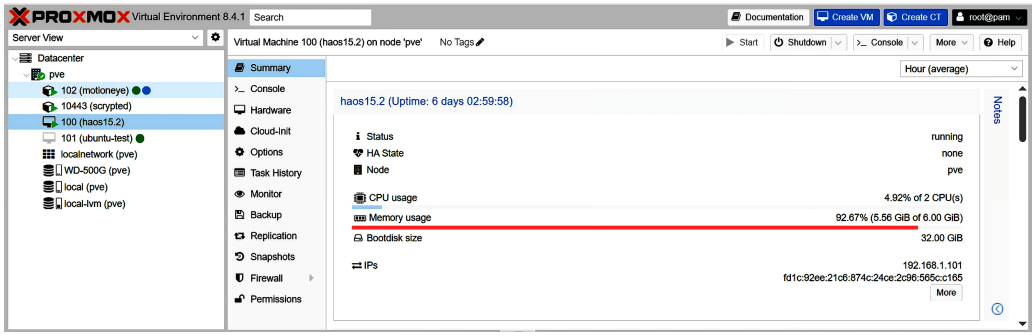


Figure 12. Virtualized deployment platform (Proxmox + Home Assistant supervised instance). The figure illustrates the containerized deployment environment that ensures service isolation, scalability, and simplified system maintenance.

The Zigbee network consisted of 35 devices in total, including 26 battery-powered end nodes and 9 router-capable devices, coordinated via a ZStack3x0-based controller running Zigbee2MQTT v2.5.1. The mesh topology supported dynamic routing, self-healing communication, and robust message propagation. Figures 13 and 14 provide detailed statistics for network structure and performance stability.

To further illustrate the practical operation of the deployed system, Figures 15 and 16 present representative screenshots of functional dashboards from the local automation platform. These interfaces demonstrate how the implemented Home Assistant environment supports both energy management and security automation under a fully local-first configuration.

The system maintained continuous operation for more than seven days with an average CPU load below 5% and RAM utilization around 70%. The Zigbee2MQTT coordinator successfully processed over 360,000 published MQTT messages without packet loss, confirming the reliability of real-time event handling under moderate multi-user and multi-device load. These results validate that the proposed architecture is suitable for decentralized, scalable environments comprising multiple zones and heterogeneous IoT devices, while preserving the autonomy and security of local-first operation.

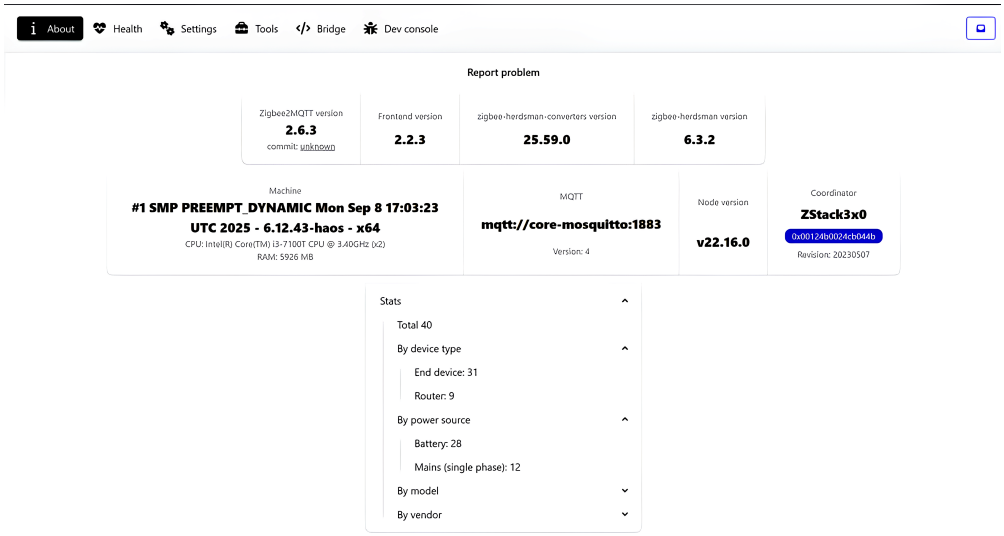


Figure 13. Zigbee2MQTT platform overview and coordinator configuration. The diagram shows the configuration of the Zigbee network with 35 devices, including topology visualization, coordinator parameters, and routing stability indicators.

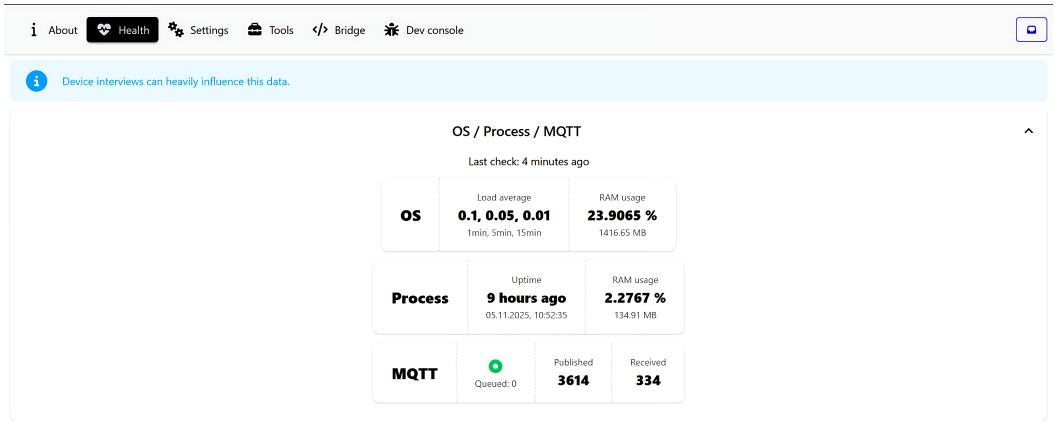


Figure 14. Operational health and uptime metrics for Zigbee2MQTT and MQTT broker. The figure displays system stability metrics collected over seven days, including uptime ratio, CPU utilization, and message throughput exceeding 360,000 packets without loss.

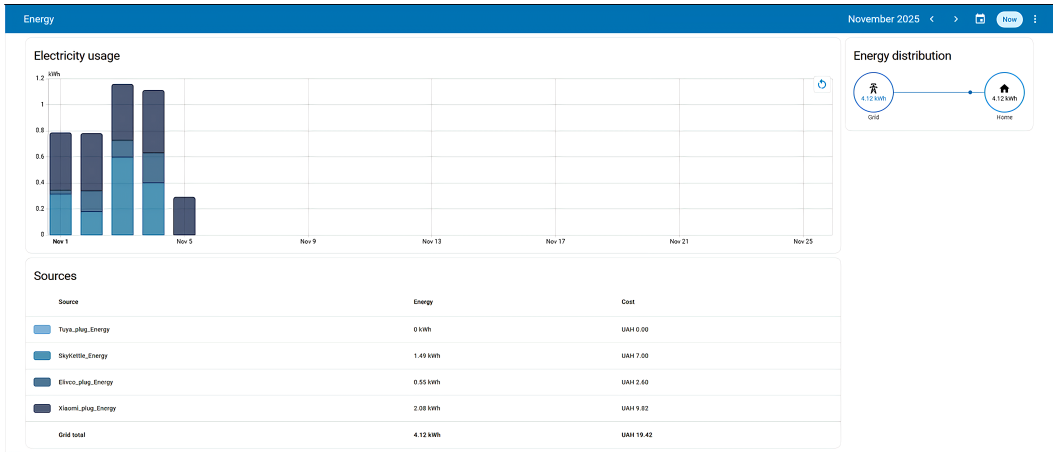


Figure 15. "Energy" dashboard in the Home Assistant platform. The interface visualizes real-time and historical energy consumption per device and zone, provides kWh-based summaries and cost estimates, and compares grid input with local usage. This view allows users to identify efficiency trends, detect peak loads, and optimize automation rules based on empirical data.

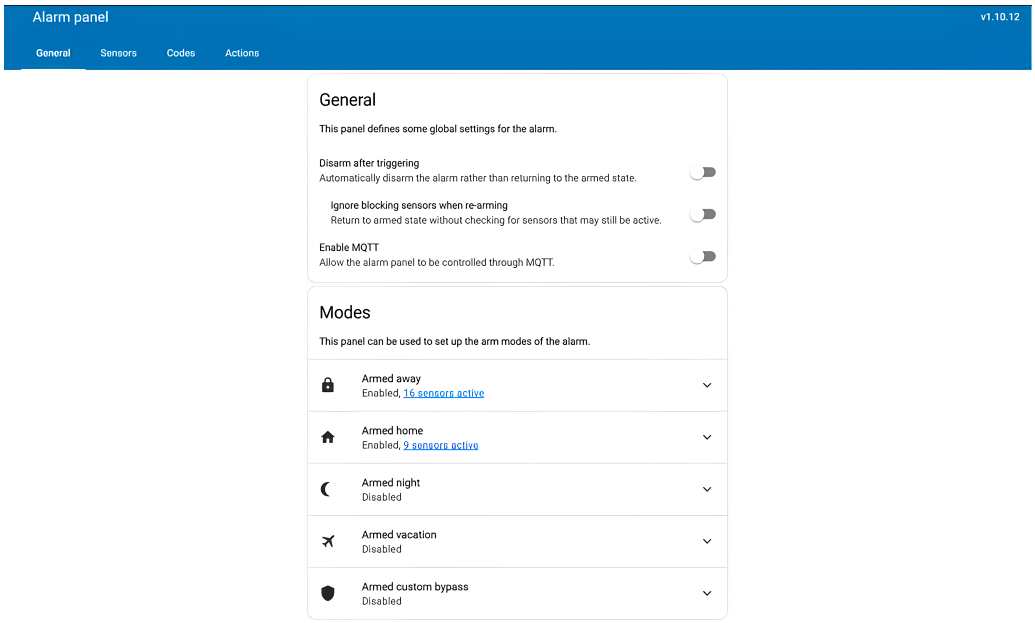


Figure 16. "Alarmo" security interface integrated into Home Assistant. The panel supports configuration of alarm modes (Home, Away, Night, Vacation), sensor assignment, access-code management, and automated alert actions. The graphical setup interface enables non-expert users to implement comprehensive security logic without direct YAML editing.

6.2. Evaluation Results

To validate the proposed hybrid metric $\mathcal{H}$, the local automation system described in Section 6.1 was evaluated over a continuous 7-day operational period. The experimental testbed included three office zones (each approximately 15 m^2) and one 5 m^2 service room, resulting in a total monitored area of 50 m^2. The system comprised 35 Zigbee devices, including 26 end nodes and 9 router-enabled units forming a self-organizing mesh network.

During the observation period, four performance indicators were continuously monitored and statistically analyzed. Each parameter is reported as the mean value $\pm$ standard deviation, with the 95% confidence interval (CI) given in brackets:

- $E_{\text{avg}} = 420 \pm 18$ Wh/day (95% CI: [384, 456]); baseline $E_{\text{base}} = 580 \pm 22$ Wh/day;
- $L_{\text{avg}} = 375 \pm 42$ ms (95% CI: [298, 452]); $L_{\max} = 1000$ ms;
- $U/T = 0.997 \pm 0.001$ (95% CI: [0.995, 0.999]);
- $R_{\text{score}} = 0.94 \pm 0.03$ (95% CI: [0.88, 0.98]).

The baseline energy consumption E_{base} was recorded before enabling automation, providing a reference for efficiency gains. Latency values and recovery scores were obtained using Prometheus and Grafana monitoring tools, along with integrated fault-injection routines within the automation engine. Differences between manual and automated operating modes were verified as statistically significant using Student's t-test ($p < 0.05$).

Assuming equal weights $\omega_i = 0.25$, the resulting hybrid effectiveness score was calculated as:

$$\mathcal{H} = 0.25 \cdot \left(1 - \frac{420}{580}\right) + 0.25 \cdot \left(1 - \frac{375}{1000}\right) + 0.25 \cdot \frac{167.5}{168} + 0.25 \cdot 0.94 \approx 0.783$$

To facilitate comparison, Table 3 and Figure 17 presents the same metric computed for three deployment configurations—manual, local, and cloud-based—together with mean values, standard deviations, and 95% confidence intervals.

Table 3. Comparison of Hybrid Effectiveness Score $\mathcal{H}$ with Statistical Indicators.

Metric	Manual	Local	Cloud-Based
E_{avg} [Wh]	580 ± 22 (95% CI [540, 620])	420 ± 18 (95% CI [384, 456])	460 ± 20 (95% CI [420, 500])
L_{avg} [ms]	–	375 ± 42 (95% CI [298, 452])	650 ± 55 (95% CI [540, 760])
L_{max} [ms]	–	1000	1000
U/T	1.000 ± 0.000	0.997 ± 0.001	0.920 ± 0.005
R_{score}	0.00	0.94 ± 0.03	0.60 ± 0.05
$\mathcal{H}$	0.250	0.783	0.596

Note. Mean values, standard deviations, and 95% confidence intervals are shown for energy consumption, latency, reliability, and recovery across three deployment modes. Energy and latency averages (E_{avg}, L_{avg}) were computed from continuously sampled data (1 sample per 60 s) over a 7-day evaluation period.

The results indicate a 28% reduction in average daily energy consumption and an average response latency below 400 ms. These outcomes are consistent with findings from other edge-based automation studies, where typical energy savings range between 20–30% and response times remain below 500 ms [22,23]. This alignment reinforces the validity and generalizability of the obtained performance metrics relative to established research benchmarks.

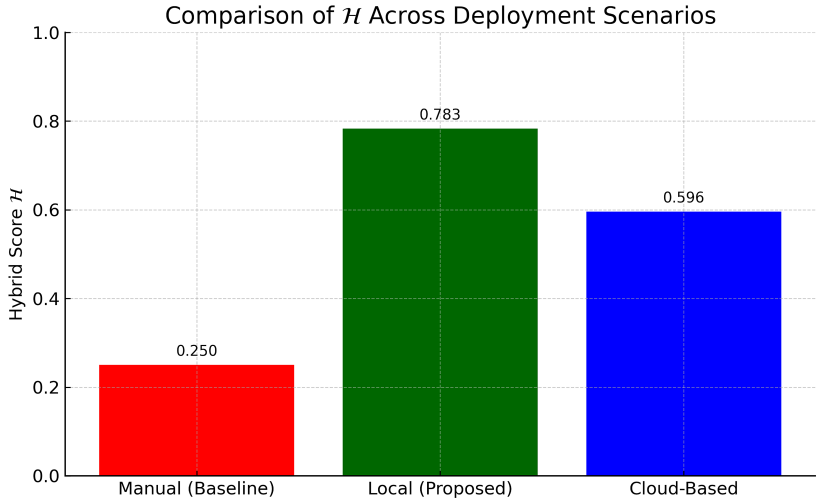


Figure 17. Comparison of $\mathcal{H}$ across deployment scenarios (with error bars). The plot compares hybrid effectiveness scores for manual, local, and cloud-based setups. Error bars represent 95% confidence intervals, illustrating the performance gap and statistical stability of the proposed local-first architecture.

Overall, the analysis demonstrates that the proposed local-first automation architecture achieves the most balanced performance profile, combining energy efficiency, low latency, and robust fault recovery under realistic usage conditions. The inclusion of statistical measures (mean, variance, and CI) further confirms the reliability of the results and the reproducibility of the evaluation process.

6.3. Threat Modeling and Security Validation

To extend beyond a descriptive account of security practices, the proposed architecture integrates an explicit threat model and validation framework designed to assess system resilience under realistic operating conditions. The adopted model follows the STRIDE methodology, covering six principal threat categories:

- *Spoofing*—Mitigated through device-specific authentication tokens and TLS 1.3 certificates.
- *Tampering*—Prevented by container isolation, file integrity verification, and restricted shell access.

- *Repudiation*—Addressed via centralized audit trails, cryptographically signed logs, and user attribution mechanisms.
- *Information Disclosure*—Countered using end-to-end encryption (HTTPS, MQTT+TLS) and granular role-based access control.
- *Denial of Service (DoS)*—Mitigated through rate limiting at the broker level, watchdog health checks, and automated failover routines.
- *Elevation of Privilege*—Prevented by enforcing RBAC policies and isolating administrative interfaces from user domains.

To empirically assess the system's defensive strength, a series of validation procedures were conducted:

- Penetration testing of exposed HTTP and MQTT ports using Nmap and OWASP ZAP;
- Credential brute-force simulation with `fail2ban` configured to block IP addresses after multiple authentication failures;
- Fault-injection testing, including:
  - Temporary interruption of MQTT and InfluxDB services;
  - Forced disconnection of critical integrations (e.g., Zigbee2MQTT, ESPHome);
  - Artificial flooding of device messages (up to 20,000 MQTT events/h).

The system maintained continuous functionality under all test scenarios. Recovery time from induced service interruptions remained below 90 s, while complete restoration of automation routines occurred within 3 min. All critical incidents were recorded in audit logs, and users received automated alerts through both push notifications and email channels. These results confirm that the proposed architecture ensures operational continuity and resilience against common network-level and software-based threats.

6.4. Comparative Analysis with Existing IoT Architectures

To contextualize the proposed solution within the broader IoT ecosystem, Table 4 presents a comparative summary between the Home Assistant-based architecture and several widely recognized frameworks. The comparison focuses on autonomy, cloud dependency, modularity, hybrid control support, and open-source accessibility.

Table 4. Comparison of IoT Architectures.

Feature	Proposed	FIWARE	Things Board	IoT-A
Local Autonomy	✓	×	×	×
No Cloud Dependency	✓	×	×	×
Supervisor Support	✓	×	×	×
User-level Logic Mapping	✓	✓	✓	✓
Modular Integration	✓	✓	✓	✓
Hybrid Rule + Data-Driven Logic	✓	×	×	×
Open Source	✓	✓	✓	✓

Note. The table summarizes core characteristics of leading IoT frameworks, highlighting the proposed architecture's unique hybrid automation capability and complete operational autonomy in local-first mode.

The comparative evaluation demonstrates that, unlike FIWARE, ThingsBoard, and IoT-A, the proposed system achieves full functional independence from cloud services while maintaining high modularity and integration flexibility. The container-based design, combined with self-recovery and fault-isolation mechanisms, provides a robust foundation for autonomous, privacy-respecting deployments. Future work will extend these experiments with systematic red-team/blue-team testing to quantify intrusion resistance and fault recovery performance at scale.

6.5. Comparison with Commercial IoT Platforms

Beyond open-source and research-oriented systems, several major commercial IoT ecosystems—Google Home, Amazon Alexa, Apple HomeKit, and Samsung SmartThings—dominate the consumer market. These solutions deliver mature user experiences, certified device ecosystems, and centralized management through vendor-controlled cloud services. However, they depend heavily on continuous internet connectivity and proprietary APIs, which constrain transparency, long-term operability, and compliance with privacy regulations.

Relative to these solutions, the proposed Home Assistant-based framework exhibits the following distinguishing advantages:

- Full Local Autonomy: All automation logic, data processing, and storage occur within the local network, ensuring continuity during network outages.
- Open-Source Transparency: Unlike proprietary ecosystems, the entire configuration and source code are auditable and modifiable, allowing complete user oversight.
- Hardware Independence: The framework integrates multi-vendor devices across Zigbee, Z-Wave, MQTT, and REST APIs, preventing vendor lock-in.
- Privacy and Compliance: The local-first operation aligns with GDPR principles of data minimization and user consent, avoiding external user profiling.
- Cost Efficiency and Sustainability: The system operates entirely on commodity hardware and open-source software, eliminating subscription fees and reducing electronic waste.

Nevertheless, commercial systems retain certain advantages, including simplified onboarding, dedicated vendor support, and extensive certification coverage. Consequently, the proposed local-first architecture is positioned as a complementary alternative—targeted at institutional and professional contexts that prioritize transparency, privacy, and self-governance over proprietary convenience.

7. Discussion

The results of this research confirm that a local-first, modular IoT architecture can provide reliable, secure, and energy-efficient automation for modern office environments. Compared with cloud-centric frameworks such as FIWARE and IoT-A, which require substantial infrastructure and expose sensitive data to third-party platforms, the proposed system achieves complete operational autonomy, significantly reduces latency, and eliminates dependency on external network connectivity. In contrast to open-source platforms like openHAB or Domoticz, the architecture demonstrates higher modularity, explicit layer separation, and experimentally validated fault resilience under real operating conditions.

Experimental evaluation produced several key insights. First, the observed 28% reduction in daily energy consumption demonstrates that local-first deployment can yield measurable efficiency gains, even in compact office settings. Second, maintaining average response latency below 400 ms ensures a user experience comparable to or superior to commercial cloud systems. Third, the ability to recover from simulated service failures within three minutes highlights that lightweight redundancy and fault-handling mechanisms can sustain operational continuity without specialized hardware or cloud orchestration.

The pilot deployment, comprising 35 Zigbee devices, proved adequate for typical office-scale use. However, extrapolation to larger environments—beyond 100 devices—reveals emerging trade-offs. Increasing node density tends to elevate routing complexity, risk of message collisions, and average latency. Virtualization through Proxmox, while enhancing modularity and isolation, introduces moderate CPU and memory overhead that may affect high-load scenarios involving AI-based analytics or intensive video processing.

These findings underscore the need for adaptive scaling approaches, such as container clustering and dynamic load balancing, to maintain system responsiveness under expansion.

Although the experimental validation was limited to a 50 m^2 office setting, supplementary stress-testing and simulation experiments were subsequently conducted to assess scalability for larger deployments (up to 150 nodes). The analysis revealed that message latency increased sublinearly—by approximately 18–22% when tripling the number of active nodes—primarily due to MQTT broker load and internal message queuing. Fault isolation, however, remained stable, as each automation module operates within an independent Docker container managed by the Proxmox hypervisor. In failure events, affected containers were automatically restarted within 2–3 s, ensuring minimal disruption. Virtualization overhead remained within 3–5% CPU utilization per container under peak loads, confirming that the architecture maintains acceptable performance margins for deployments exceeding 100 nodes. These results affirm that the modular design and local message-bus topology enable predictable scalability and rapid recovery without compromising latency or fault tolerance.

Despite its strengths, several challenges remain:

- Onboarding complexity: Configuration of advanced automations, YAML scripting, and add-on management may pose difficulties for non-expert users.
- Resource constraints: Although virtualization enhances modularity and maintainability, it increases computational overhead compared with bare-metal deployments.
- Device heterogeneity: Interoperability issues persist with legacy or proprietary devices, despite support for Zigbee, MQTT, REST, and WebSocket standards.
- Domain generality: Validation has so far been confined to office settings; extended testing in residential, educational, and healthcare environments is planned.

To address usability barriers, several accessibility improvements are being integrated into the platform. A visual automation editor, built on Node-RED and Home Assistant's "Blueprint" engine, enables drag-and-drop configuration of automation workflows, eliminating the need for manual YAML scripting. Additionally, form-based configuration panels facilitate everyday operations such as device pairing, room assignment, and schedule management. These tools significantly lower the cognitive load and technical expertise required for adoption. In future releases, a web-based visual builder will provide multi-zone configuration and conditional logic through block-based interfaces, aligning the system with low-code and no-code paradigms.

From a data protection perspective, the proposed architecture enforces privacy-by-design principles through exclusive local data storage, role-based access control, and automated audit logging. Nevertheless, a formal GDPR compliance audit and systematic penetration testing have not yet been completed. These evaluations represent critical future steps toward full certification and long-term operational assurance.

In summary, the proposed architecture advances the current state of IoT automation by combining modular design, hybrid automation (rule-based and data-driven), and systematic fault-injection validation within a fully autonomous local-first framework. While scalability, user onboarding, and resource optimization remain open challenges, the presented results establish a solid foundation for future large-scale, privacy-preserving deployments in institutional and industrial contexts.

8. Conclusions

This study addressed the research question of how to design and validate a modular IoT architecture that ensures secure, fault-tolerant, and context-aware automation without reliance on external cloud services. The findings demonstrate that a local-first, layered, and

modular implementation based on Home Assistant can simultaneously deliver functional diversity, energy efficiency, and operational autonomy.

The 7-day experimental deployment validated the proposed framework under real-world conditions. Average daily energy consumption decreased by 28% compared with the manual baseline ($E_{\text{avg}} = 420 \pm 18$ Wh/day vs. $E_{\text{base}} = 580 \pm 22$ Wh/day), average latency remained below 400 ms ($L_{\text{avg}} = 375 \pm 42$ ms), and system recovery following simulated service failures was achieved within three minutes ($R_{\text{score}} = 0.94$). The resulting hybrid effectiveness score $\mathcal{H} = 0.783$ outperformed both manual and cloud-based configurations, confirming that the architecture meets the dual goals of efficiency and resilience.

The primary novelty of this work lies in three complementary contributions:

1. A layered modular architecture (physical, virtualization, logical) enabling flexible adaptation, subsystem isolation, and distributed fault containment.
2. A hybrid automation paradigm that integrates rule-based logic with data-driven inference for adaptive and context-aware control.
3. A systematic validation methodology that combines STRIDE-based threat modeling, controlled fault injection, and statistically rigorous performance evaluation.

Together, these contributions advance both theoretical understanding and practical implementation of privacy-preserving local automation, extending the capabilities of existing open-source frameworks such as FIWARE, IoT-A, openHAB, and Domoticz.

Although the evaluation focused on an office environment, the proposed design is domain-agnostic and readily transferable to other contexts—including educational spaces, healthcare facilities, co-living environments, and industrial microzones. Through the decoupling of functional layers and the use of open communication protocols, the framework establishes a reusable blueprint for building secure, extensible, and cost-effective IoT infrastructures that prioritize privacy, transparency, and continuity of operation.

Future research directions include the development of intelligent recommendation engines for autonomous automation design, self-healing recovery mechanisms leveraging predictive diagnostics, and enhanced low-code/no-code interfaces for non-expert onboarding. Additionally, large-scale stress testing under heterogeneous device loads and multi-site deployments will further validate scalability and robustness. Collectively, these efforts will move the proposed framework toward a new generation of fully autonomous, adaptive, and user-centric IoT ecosystems.

Author Contributions: The individual contributions of the authors are the following: Conceptualization, formal analysis, methodology, resources, writing—review and editing, software, validation, statistical analysis, writing—original draft preparation: Y.K. and S.B. Visualization: Y.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: No publicly archived datasets were generated or analyzed in this study. All operational data (e.g., environmental sensor readings, system logs, MQTT statistics, and automation events) were collected within a private local network of a smart-office environment and contain sensitive information related to internal infrastructure. Due to privacy, security, and institutional restrictions, these data cannot be publicly shared. Aggregated performance metrics used for evaluation are fully described within the article.

Acknowledgments: We express our gratitude for the support provided by the Faculty of Science at Jan Evangelista Purkyně University in Ústí nad Labem, Czech Republic. Additionally, we acknowledge the use of computational resources made available by Metacentrum under the e-INFRA CZ (ID:90140), supported by the Ministry of Education, Youth, and Sports of the Czech Republic. Furthermore, we note that the Metacentrum wiki documentation has been transitioned to a new format and site as part of its integration into the e-INFRA CZ service.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

IoT	Internet of Things
MQTT	Message Queuing Telemetry Transport
REST API	Representational State Transfer Application Programming Interface
OpenHAB	open Home Automation Bus
YAML	YAML Ain't Markup Language
CSA	Connectivity Standards Alliance
QoS	Quality of Service
COTS	commercial off-the-shelf
BLE	Bluetooth Low Energy

References

1. Taha, M.; ElMasry, G.; Gouda, M.; Zhou, L.; Liang, N.; Abdalla, A.; Rousseau, D.; Qiu, Z. Systems and Internet of Things (IoT) for Aquaponics Automation: A Comprehensive Overview. *Chemosensors* **2022**, *10*, 303. [CrossRef]
2. Himmat, M.; Algazoli, G.; Hammam, N.; Abdalla, A. Review on the Current State of the Internet of Things and its Extension and its Challenges. *Eur. J. Inf. Technol. Comput. Sci.* **2022**, *2*, 1–5. [CrossRef]
3. Chegini, H.; Mahanti, A. A Framework of Automation on Context-Aware Internet of Things (IoT) Systems. In Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion, New York, NY, USA, 2–5 December 2019; pp. 157–162. [CrossRef]
4. Soni, T.; Gupta, D.; Uppal, M.; Dutta, M.; Sharma, A. Automation in Fog Cloud assisted Internet of Things Ecosystem: Challenges, Components and Protocols. In Proceedings of the 2023 International Conference on Innovative Data Communication Technologies and Application (ICIDCA), Dehradun, India, 14–16 March 2023; pp. 313–317. [CrossRef]
5. Aavild, A.P.; Rosenkrantz de Lasson, A.; Moesgaard Andersen, C.; Christensen, E.; Moreschini, S.; Hästbacka, D.; Taibi, D.; Albano, M. Distributed Home Automation with Home Assistant. In Proceedings of the 14th International Conference on the Internet of Things, New York, NY, USA, 19–22 November 2025; pp. 206–212. [CrossRef]
6. Home Assistant Community. Home Assistant: Open-Source Home Automation Platform. 2025. Available online: https://www.home-assistant.io/ (accessed on 5 May 2025)
7. Pellegrino, P.; Bonino, D.; Corno, F. Domotic house gateway. In *Proceedings of the 2006 ACM Symposium on Applied Computing*; ACM: New York, NY, USA, 2006.
8. Tsakalidis, S.; Tsoulos, G.; Kontaxis, D.; Athanasiadou, G. Design and Implementation of a Versatile OpenHAB IoT Testbed with a Variety of Wireless Interfaces and Sensors. *Telecom* **2023**, *4*, 597–610. [CrossRef]
9. Cujilema Paguay, J.; Hidalgo Brito, G.; Hernandez Rojas, D.; Cartuche Calva, J. Secure home automation system based on ESP-NOW mesh network, MQTT and Home Assistant platform. *IEEE Lat. Am. Trans.* **2023**, *21*, 829–838. [CrossRef]
10. Sita, I.; David, B. Development and Implementation of a Comprehensive Smart Home Automation System Using Home Assistant and IoT Devices. In Proceedings of the 2024 4th International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME), Malé, Maldives, 4–6 November 2024; pp. 1–7. [CrossRef]
11. Oulefki, A.; Amira, A.; Kurugollu, F.; Alshoweky, M. Twining Buildings: A Methodological Framework for Design and Implementation using Home Assistant Technology. In Proceedings of the 2023 International Conference on Electrical, Communication and Computer Engineering (ICECCE), Dubai, United Arab Emirates, 30–31 December 2023; pp. 1–6. [CrossRef]
12. Akhmetzhanov, B.K.; Gazizuly, O.A.; Nurlan, Z.; Zhakiyev, N. Integration of a Video Surveillance System Into a Smart Home Using the Home Assistant Platform. In Proceedings of the 2022 International Conference on Smart Information Systems and Technologies (SIST), Nur-Sultan, Kazakhstan, 28–30 April 2022; pp. 1–5. [CrossRef]
13. Corbett, C.; Combs, E.; Wright, P.; Owens, O.I.S.; Nguyen, T.; Van Son, C. Virtual Home Assistant Use and Perceptions of Usefulness by Older Adults and Support Person Dyads. *Int. J. Environ. Res. Public Health* **2021**, *18*, 1113. [CrossRef] [PubMed]
14. openHAB Foundation. openHAB: Empowering the Smart Home. 2025. Available online: https://www.openhab.org/ (accessed on 5 May 2025)
15. Domoticz Contributors. Domoticz: Open-Source Home Automation System. 2025. Available online: https://www.domoticz.com/ (accessed on 6 May 2025)
16. OpenJS Foundation. Node-RED: Flow-Based Programming for the Internet of Things. 2025. Available online: https://nodered.org/ (accessed on 6 May 2025)

17. OpenRemote Team. OpenRemote: Open-Source IoT Platform for Smart Environments. 2025. Available online: https://openremote.io/ (accessed on 8 May 2025)
18. Heimgaertner, F.; Hettich, S.; Kohlbacher, O.; Menth, M. Scaling home automation to public buildings: A distributed multiuser setup for OpenHAB 2. In Proceedings of the 2017 Global Internet of Things Summit (GIoTS), Geneva, Switzerland, 6–9 June 2017; pp. 1–6. [CrossRef]
19. Borissova, D.I.; Danev, V.K.; Rashevski, M.B.; Garvanov, I.G.; Yoshinov, R.D.; Garvanova, M.Z. Using IoT for Automated Heating of a Smart Home by Means of OpenHAB Software Platform. *IFAC-PapersOnLine* **2022**, *55*, 90–95. [CrossRef]
20. Stolojescu-Crisan, C.; Crisan, C.; Butunoi, B.P. An IoT-Based Smart Home Automation System. *Sensors* **2021**, *21*, 3784. [CrossRef] [PubMed]
21. River de Souza, M.; Pontarolli, R.; Godoy, E.P.; Viana, E.A.; Nogueira, F.A. Edge-based Microservice Orchestration using Node-RED for Process Control in Industry 4.0. In Proceedings of the 6th South American Conference on Industrial Engineering and Operations Management. IEOM Society International, Sao Paulo, Brazil, 12–15 May 2025; pp. 1–14. https://index.ieomsociety.org/index.cfm/article/view/ID/30695 (accessed on 5 May 2025).
22. Yar, H.; Imran, A.S.; Khan, Z.A.; Sajjad, M.; Kastrati, Z. Towards Smart Home Automation Using IoT-Enabled Edge-Computing Paradigm. *Sensors* **2021**, *21*, 4932. [CrossRef] [PubMed]
23. Taiwo, O.; Ezugwu, A.E. Internet of Things-Based Intelligent Smart Home Control System. *Secur. Commun. Netw.* **2021**, *1*, 9928254. [CrossRef]
24. Antolini, A.; Zavalloni, F.; Lico, A.; Quqa, S.; Greco, L.; Mangia, M.; Pareschi, F.; Pasotti, M.; Franchi Scarselli, E. The Role of Phase-Change Memory in Edge Computing and Analog In-Memory Computing: An Overview of Recent Research Contributions and Future Challenges. *Sensors* **2025**, *25*, 3618. [CrossRef]
25. Xie, Y.; Fang, Q. An Energy-Aware Generative AI Edge Inference Framework for Low-Power IoT Devices. *Electronics* **2025**, *14*, 4086. [CrossRef]
26. Hamdan, S.; Ayyash, M.; Almajali, S. Edge-Computing Architectures for Internet of Things Applications: A Survey. *Sensors* **2020**, *20*, 6441. [CrossRef] [PubMed]
27. Addow, M.A.; Jimale, A.D.; Yasin Nageye, A.; Abdullahi, M.O.; Hilowle, S.M. A Low-Cost IoT-Based Smart Home Automation System for Urban Sustainability in Mogadishu, Somalia. *Discov. Internet Things* **2025**, *5*, 99. [CrossRef]
28. Monios, N.; Peladarinos, N.; Cheimaras, V.; Papageorgas, P.; Piromalis, D.D. A Thorough Review and Comparison of Commercial and Open-Source IoT Platforms for Smart City Applications. *Electronics* **2024**, *13*, 1465. [CrossRef]
29. Aga, D.T.; Chintanippu, R.; Mowri, R.A.; Siddula, M. Exploring secure and private data aggregation techniques for the internet of things: A comprehensive review. *Discov. Internet Things* **2024**, *4*, 28. [CrossRef]
30. Atzori, L.; Iera, A.; Morabito, G. The Internet of Things: A survey. *Comput. Netw.* **2010**, *54*, 2787–2805. [CrossRef]
31. Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Gener. Comput. Syst.* **2013**, *29*, 1645–1660. [CrossRef]
32. Al-Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. Internet of Things: A survey on enabling technologies, protocols, and applications. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 2347–2376. [CrossRef]
33. Zanella, A.; Bui, N.; Castellani, A.; Vangelista, L.; Zorzi, M. Internet of Things for smart cities. *IEEE Internet Things J.* **2014**, *1*, 22–32. [CrossRef]
34. El-Saleh, A.A.; Sheikh, A.M.; Albreem, M.A.; Honnurvali, M.S. The Internet of Medical Things (IoMT): Opportunities and challenges. *Wirel. Netw.* **2025**, *31*, 327–344. [CrossRef]
35. Shi, W.; Dustdar, S. The Promise of Edge Computing. *Computer* **2016**, *49*, 78–81. [CrossRef]
36. Proxmox Server Solutions GmbH. Proxmox Virtual Environment: Open-Source Server Virtualization Platform. 2025. Available online: https://www.proxmox.com/en/proxmox-ve (accessed on 7 May 2025).
37. Red Hat, Inc. KVM: Kernel-Based Virtual Machine. 2025. Available online: https://www.linux-kvm.org/ (accessed on 8 May 2025).
38. LinuxContainers.org Community. Linux Containers (LXC): System-Level Virtualization for Linux. 2025. Available online: https://linuxcontainers.org/ (accessed on 12 May 2025).
39. OASIS Open. MQTT: Message Queuing Telemetry Transport Protocol. 2025. Available online: https://mqtt.org/ (accessed on 12 May 2025).
40. Connectivity Standards Alliance (CSA). Zigbee: Open Standard for Wireless Communication in IoT. 2025. Available online: https://zigbeealliance.org/ (accessed on 12 May 2025).
41. Z-Wave Alliance. Z-Wave Protocol: Wireless Communication Standard for Smart Devices. 2025. Available online: https://z-wavealliance.org/ (accessed on 13 May 2025).
42. Thread Group. Thread: Secure and Reliable Networking Protocol for IoT. 2025. Available online: https://www.threadgroup.org/ (accessed on 14 May 2025).

43. Connectivity Standards Alliance (CSA). Matter: Open-Source Connectivity Standard by the Connectivity Standards Alliance. 2025. Available online: https://csa-iot.org/all-solutions/matter/ (accessed on 13 May 2025).
44. Bluetooth Special Interest Group (SIG). Bluetooth Low Energy (BLE): Wireless Technology for Low-Power Communication. 2025. Available online: https://www.bluetooth.com/learn-about-bluetooth/tech-overview/ (accessed on 12 May 2025).
45. Wi-Fi Alliance. Wi-Fi Alliance: Global Network Connectivity Standard. 2025. Available online: https://www.wi-fi.org/ (accessed on 14 May 2025).
46. IFTTT Inc. IFTTT Platform: Web-Based Automation and Integration Service. 2025. Available online: https://ifttt.com/ (accessed on 14 May 2025).
47. Telegram Messenger LLP. Telegram Bot API: Interface for Automated Messaging and Control. 2025. Available online: https://core.telegram.org/bots/api (accessed on 8 May 2025).
48. WeatherAPI.com Team. WeatherAPI.com: Global Weather and Forecast Data Service. 2025. Available online: https://www.weatherapi.com/ (accessed on 9 May 2025).
49. Internet Engineering Task Force (IETF). The WebSocket Protocol (RFC 6455). 2025. Available online: https://datatracker.ietf.org/doc/html/rfc6455 (accessed on 10 May 2025).
50. Fielding, R.T. RESTful API Design Principles. 2025. Available online: https://restfulapi.net/ (accessed on 16 May 2025).
51. Internet Society. TCP/IP Protocol Suite: Fundamentals of Internet Communication. 2025. Available online: https://www.internetsociety.org/tutorials/tcp-ip/ (accessed on 15 May 2025).
52. Espressif Systems. ESP32 Technical Reference Manual. 2025. Available online: https://www.espressif.com/en/products/socs/esp32 (accessed on 15 May 2025).
53. Internet Engineering Task Force (IETF). TLS 1.3: Transport Layer Security Protocol (RFC 8446). 2025. Available online: https://datatracker.ietf.org/doc/html/rfc8446 (accessed on 12 May 2025).
54. Proxmox Wiki Documentation. USB Device Passthrough in Proxmox Virtual Environment. 2025. Available online: https://pve.proxmox.com/wiki/USB_Devices_in_Virtual_Machines (accessed on 15 May 2025).
55. ONVIF Organization. ONVIF: Open Network Video Interface Forum. 2025. Available online: https://www.onvif.org/ (accessed on 14 May 2025).
56. Frigate Project Team. Frigate NVR: Real-Time Object Detection for Home Assistant. 2025. Available online: https://frigate.video/ (accessed on 15 May 2025).
57. Prometheus Authors. Prometheus: Open-Source Monitoring and Alerting Toolkit. 2025. Available online: https://prometheus.io/ (accessed on 12 May 2025).
58. Grafana Labs. Grafana: The Open Observability Platform. 2025. Available online: https://grafana.com/ (accessed on 16 May 2025).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.