

Design of a Security-Aware Smart Home Automation Architecture With Mobile Application Control: A Software-in-The-Loop Evaluation

Kariola Basira Ogaga¹, Michael Oghale Ighofiomoni², Omokaro Idama³, Friday Elohor Odoh⁴

^{1,2,3,4} department Of Computer Engineering, Southern Delta University, Ozoro, Delta State, -Nigeria

Abstract- Smart-home systems increasingly combine networked sensors, embedded controllers, software services, and mobile interfaces to make domestic devices observable and controllable. This paper presents the design of a smart home automation architecture with a mobile application and manual controls, emphasising low-cost architecture, responsive control, security-aware messaging, and graceful local operation. The proposed system adopts an ESP32-class controller at the device layer, Wi-Fi connectivity, sensor inputs for temperature/humidity, illumination, motion and gas condition, relay outputs for representative household loads, and a mobile interface for authentication, status monitoring, direct switching, automation rules, alerts, and event history. A layered Internet-of-Things and cyber-physical feedback perspective guided the architecture, while recent user-acceptance literature informed interaction choices such as state visibility, simple device grouping, and explicit manual override. Because physical deployment conditions vary with router quality, mains wiring, sensors, and actuators, the quantitative evaluation reported here is deliberately limited to a reproducible, host-local software-in-the-loop environment rather than field measurements: the command-processing API path and the deterministic automation-rule engine were implemented and exercised, while the ESP32 wiring, sensor mapping, and relay behaviour are specified as a build-ready design that has not yet been physically assembled or timed. Across 880 authenticated API requests covering one-, two-, four-, and eight-device command batches, mean response time ranged from 1.747 to 2.059 ms and the 95th-percentile response time ranged from 2.220 to 2.762 ms, with all requests completing successfully. Six negative API/security checks returned their intended error codes, and thirty deterministic automation-rule tests covering ordinary automation, safety conditions, and manual override all passed. The results demonstrate that the command-validation, authentication, and rule-evaluation logic underlying a mobile-controlled home platform is sound and computationally light, and they establish a reproducible software baseline against which a future hardware-in-the-loop study, connecting the specified ESP32 controller, sensors, and relays, can measure genuine end-to-end response time, energy behaviour, and field reliability.

Keywords- Internet of Things; smart home automation; mobile application control; ESP32; edge computing; home automation security; software-in-the-loop testing.

I. INTRODUCTION

Homes are becoming computational spaces. Lighting points, fans, sockets, security sensors, and climate-control devices that were once isolated electromechanical objects can now participate in a network where their state is measured, interpreted, and changed by software. This shift is largely enabled by the Internet of Things (IoT), which connects resource-constrained sensors and actuators to local or remote services. Sepasgozar et al. (2020) describe smart-home research as an increasingly integrated field in which IoT sensing and intelligent services

support energy, comfort, safety, and assisted-living functions, while El-Azab (2021) notes that the potential of smart homes lies in coordinated services, tempered by continuing obstacles of interoperability, cost, and security.

A contemporary smart-home system must solve several computing problems at once: embedded controllers must read sensors reliably and drive actuators without unsafe state transitions; messages must arrive quickly enough for interactive control while remaining meaningful over intermittent networks; and the application layer must distinguish

authorised commands from invalid or malicious traffic. Majeed et al. (2020) and Sowah et al. (2020) show that home automation can be designed around networked control while treating security as part of the architecture rather than an after thought a position this project adopts directly. Mobile applications have become the natural control surface for such systems (Shakir et al., 2021; Chen et al., 2021), yet convenience alone does not guarantee acceptance: Nascimento et al. (2022) and Reisinger et al. (2023) identify perceived usefulness, transparency, reliability, and fit with everyday routines as persistent requirements. Edge-oriented designs that keep time-sensitive decisions close to the device (Rahimi et al., 2020; Yar et al., 2021) and lightweight, explicit authentication (Oh et al., 2021; Yu et al., 2021) are likewise treated here as architectural requirements rather than optional refinements, since compromise of a home-automation system can carry physical as well as informational consequences (Bugeja et al., 2021; Edu et al., 2021).

Many domestic appliances are still operated as independent devices, and mixed commercial products often use incompatible protocols or closed ecosystems (Phan & Kim, 2020; Li et al., 2021). The practical problem is not the absence of remote switches but the lack of an integrated, understandable, and secure control environment that coordinates heterogeneous household functions while remaining usable when the network or the mobile application is unavailable. This paper addresses that gap by presenting a modular smart-home architecture device, communication, service, and mobile layers together with a reproducible software evaluation of its command-processing and automation-rule logic. The specific objectives are to

- design a modular IoT architecture linking sensors, an ESP32-class controller, relay-controlled loads, a communication service, a database, and a mobile application.
- specify authentication, device-state monitoring, direct control, automation rules, alerts, and event logging.
- retain manual/local control so core functions do not depend solely on the mobile interface.
- define security-aware command validation and error handling

- evaluate the implemented software control path with reproducible response-time, success-rate, and rule-engine tests
- state clearly which claims are supported by the present evidence and which require future hardware-in-the-loop validation.

II. RELATED WORK

Layered IoT architecture is the dominant structural model for smart-home systems, separating a device/sensing layer, a communication layer, and a service/application layer so that responsibility for sensing, transport, and decision-making can be reasoned about independently (Sepasgozar et al., 2020; Taiwo & Ezugwu, 2021). Embedded controllers such as the ESP32 are widely used because integrated Wi-Fi removes the need for a separate communication module, and a growing body of work demonstrates MQTT-based device messaging as a lightweight alternative to continuous HTTP polling (Kim & Ku, 2020).

Edge- and fog-computing research extends this model by arguing that latency-sensitive decisions should execute close to the physical environment rather than depend entirely on a distant cloud service: Rahimi et al. (2020), Hamdan et al. (2020), and Laroui et al. (2021) review the architectural motivations, and Yar et al. (2021) apply an edge-oriented approach directly to smart-home automation, a position this project's command-processing design follows.

Security and privacy research consistently treats constrained smart-home devices as requiring explicit, lightweight authentication rather than assumptions of a trusted local network. Oh et al. (2021) and Yu et al. (2021) propose lightweight authentication protocols for resource-constrained environments, Bugeja et al. (2021) provide a structured framework for smart-home privacy-risk analysis, and Al-Begain et al. (2022) show that denial-of-service defence requires dedicated detection mechanisms beyond ordinary input validation.

Karale (2021) and Williams et al. (2022) situate these concerns within the broader IoT security literature, while Edu et al. (2021) demonstrate that even consumer smart-home assistants create a substantial

privacy and attack surface. These findings motivate the fail-closed, explicitly validated command path adopted in the present design.

A separate strand of literature addresses why technically functional smart-home systems are not always adopted or trusted. Schomakers et al. (2021) connect privacy and trust directly to automation preferences, Reisinger et al. (2023) identify control, transparency, and reliability as user requirements, and Nascimento et al. (2022) confirm in a statistical meta-analysis that perceived usefulness remains a dominant predictor of acceptance.

This literature supports a design principle followed here: automation should remain visible and overridable rather than opaque, and state shown on the mobile interface should reflect confirmed device state rather than an assumed outcome of a sent command. Taken together, the reviewed literature justifies the architectural choices of this project — layered design, edge-oriented rule execution, explicit authentication, and visible manual override — while also indicating a gap: much of the published work addresses either the architecture or the security model in isolation, with comparatively few final-year-scale prototypes reporting an explicit, reproducible software evaluation of the command path before physical deployment. This paper addresses that gap directly.

III. SYSTEM DESIGN AND METHODOLOGY

1. Development Methodology And Scope

An iterative prototyping methodology was adopted because home automation combines software and hardware interfaces whose usability and timing properties are easier to refine through repeated builds than a single final integration. The first iteration defined a minimal vertical slice — authenticate a user, identify a device, submit a command, update state, record an event — and later iterations added sensor telemetry, automation rules, alerts, manual input, and negative validation.

The design deliberately separates the full production-oriented architecture, which includes a mobile application, service/gateway, message broker, ESP32 controller, sensors, and actuators, from the

quantitative evaluation that could actually be performed within the project's scope. The benchmark reported in Section 4 exercises the implemented service-side command path on a single host; it validates authentication, payload parsing, target resolution, and state updates, but does not substitute for a future Wi-Fi, broker, microcontroller, or relay-timing test. This boundary is stated explicitly because a measurement is valid only for the path that was actually executed.

IV. SYSTEM ARCHITECTURE

The architecture contains four principal computational zones. The mobile application handles user interaction and communicates with the service layer over an authenticated API. The application server or local edge gateway owns user sessions, device metadata, rule management and persistence, and translates high-level actions into device messages. An MQTT-oriented message layer decouples the service from the ESP32 node. The embedded controller owns GPIO state, sensor sampling and time-sensitive local rules. The data store retains users, devices, readings, rules, alerts and event history.

Manual inputs connect directly to the embedded controller so that a person inside the home can still operate designated loads. When a local input changes a state, the controller publishes an event so that the service and mobile display can converge on the new confirmed value. When a mobile command arrives, the reverse occurs: the service validates identity and payload, publishes the logical command, and the controller applies it before returning state. This bidirectional synchronisation is central to the design because it avoids treating the phone as the only source of truth.

The architecture also distinguishes control plane from data plane. Control messages are comparatively infrequent and require strong validation because they can change physical outputs. Telemetry messages may be more frequent, but they should be bounded by sampling policy so unnecessary traffic is not generated. Event logs form a third category: they provide an auditable narrative of state changes and exceptions. This separation makes retention, access

control and performance decisions easier than treating every message as an undifferentiated JSON object.

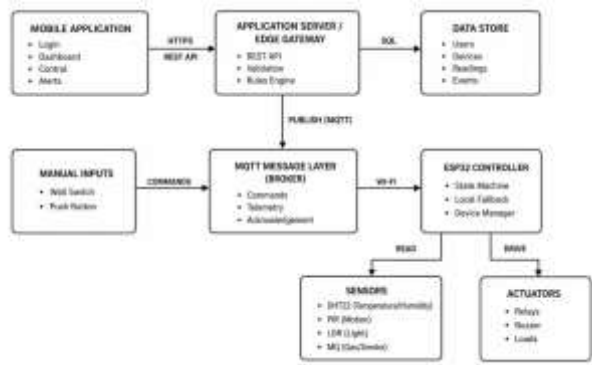


Figure 3.1: Proposed smart-home system architecture

V. HARDWARE DESIGN

The reference hardware platform uses an ESP32 development board because its Wi-Fi capability reduces the number of separate communication components required. Sensor inputs include temperature/humidity, light level, PIR motion and a demonstrative gas/smoke channel. Relay outputs represent loads. A buzzer provides local warning, and push-button or wall-switch inputs represent manual control. The system can be assembled initially using low-voltage lamps or LEDs so logic can be validated without exposing a student prototype to household mains voltage.

The relay interface is treated as an isolation boundary. GPIO pins must never drive a mains load directly; they drive a properly designed relay module or certified switching interface. The controller firmware should define a fail-safe boot state before network connection, and a safety-related event should not depend on the mobile application being open. Sensor calibration parameters belong in configuration rather than being scattered as numeric constants in interface code.

Power design also matters. The ESP32 and sensors operate at logic-level voltages, while relay coils may require a different current budget. An undersized supply can cause brownout resets precisely when multiple relays change state. The prototype should therefore use a stable regulated supply with adequate current margin and proper common-ground design

where the selected relay module requires it. Any mains-side circuit must remain physically segregated from low-voltage logic and enclosed appropriately.

Table 1. Proposed hardware bill of materials

Component	Qty.	Function
ESP32 DevKit V1 (or equivalent)	1	Main controller and Wi-Fi interface
DHT22 (or equivalent)	1	Temperature and humidity sensing
BH1750 or LDR module	1	Ambient light input
PIR sensor	1	Motion detection
MQ-2 (or equivalent)	1	Demonstrative gas/smoke proxy
4-channel isolated relay module	1	Representative load switching
Push buttons	4	Local control / manual override
Buzzer / status LEDs	As required	Local alert and diagnostics
Regulated power supply	1	Safe low-voltage supply

Table 2. GPIO allocation and device mapping

GPIO	Interface	Direction	Logical function
GPIO 4	DHT22 data	Input	Temperature / humidity
GPIO 34	Light sensor	Input	Illumination proxy
GPIO 27	PIR	Input	Motion
GPIO 35	Gas sensor	Input	Gas-condition proxy
GPIO 16–19	Relay 1–4	Output	Living-room light, fan, socket, auxiliary load
GPIO 23	Buzzer	Output	Local warning
GPIO 25/26/32/33	Manual switches	Input	Local override

VI. SOFTWARE DESIGN

The service/API module is the policy boundary between user requests and device messages, responsible for authentication, authorization, request validation, device lookup, event persistence, and message publication. The reference API uses JSON payloads and predictable HTTP status codes: a valid command returns 200 or 202, missing credentials return 401, insufficient privilege returns 403, unknown devices return 404, and malformed payloads return 400 (Table 3). A signed-request option computes a Hash-based Message Authentication Code (HMAC) over a canonical request body; production systems should additionally use per-device credentials and TLS transport rather than relying on HMAC alone.

Table 3. API endpoint specification

Endpoint	Input	Output	Expected status
POST /auth/login	email, password	Token/session metadata	200; 401
GET /devices	Authorization header	Device list / latest states	200; 401/403
POST /devices/{id}/command	state, request_id, token	Accepted/confirmed state	200/202 ; 400/401 /404
GET /sensors/latest	Room/device filters	Latest telemetry with timestamps	200; 401
POST /rules	condition, action, enabled	Created rule	201; 400/401
GET /events	Filters / pagination	Event history	200; 401

On the device side, the ESP32 firmware maintains a small state machine per controlled output, and direct user commands, automation actions, and manual inputs all resolve through one common state-update function that records the source of the change,

preventing uncoordinated writes to the same relay from multiple code paths. The command-processing algorithm fails closed: a request first passes authentication and payload validation, only then is the target device resolved, and an authorisation check determines whether the user may control that target before a canonical command is published. Idempotent commands (“set light = ON”) are preferred over toggle commands because a repeated message then produces the same desired state rather than reversing it. The relational database schema separates users, devices, sensor readings, automation rules, event logs, and alerts, with passwords stored only as salted hashes and event records retaining user, device, source, action, result, and timestamp so the history view can distinguish a user command from an automation action or a manual wall-switch change.

VII. DATA-FLOW DESIGN

The context-level data-flow model treats the smart-home control process as the central transformation. Users provide credentials, commands and preferences; the system returns status, alerts and history. Administrators provide account and device configuration. ESP32/device nodes provide telemetry and acknowledgements and receive validated commands. The data store retains records that support later retrieval and state reconstruction. Figure 3.1 intentionally shows logical flows rather than network packets so each flow maps to a system responsibility.

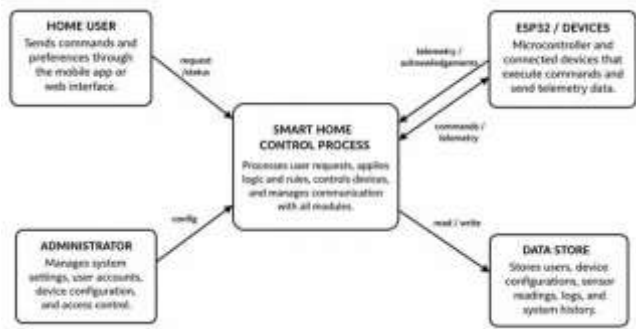


Figure 3.1: Context-level data-flow diagram

VIII. TESTING AND EVALUATION PROCEDURE

Testing was organised into performance and rule-correctness categories, both executed as deterministic, reproducible software tests. The performance

benchmark is host-local and software-in-the-loop: a lightweight HTTP server and HMAC-validated JSON requests run on the same machine, deliberately avoiding claims about wireless or microcontroller timing. For each of four command-batch sizes (one, two, four, and eight target devices), 220 requests were issued, and payload size, mean, median, and 95th-percentile latency, and success rate were recorded. A second, independent deterministic harness evaluated the automation-rule engine: representative temperature, illuminance, motion, and gas values were combined with manual light and fan overrides, with expected Boolean outputs written before observed output was compared, including values immediately below and exactly at defined thresholds to check boundary ($\geq$) semantics explicitly. Six representative positive and negative API checks (valid signed command, missing token, invalid token, unknown device, empty command, unknown endpoint) were also executed against the local service to verify that invalid input is rejected with the intended status code rather than falling through to a state change.

IX. RESULTS

1. Software-In-The-Loop Api Timing

Across 880 benchmarked requests (220 per batch size), mean host-local API processing time ranged from 1.747 ms to 2.059 ms, and the 95th-percentile value ranged from 2.220 ms to 2.762 ms; every request completed successfully (Table 4). Increasing the number of devices represented in a single command increased the mean JSON payload from 64.5 bytes (one device) to 194.0 bytes (eight devices), but measured processing time did not increase monotonically with payload size — expected at sub-millisecond scale, where operating-system scheduling and interpreter overhead are comparable to the marginal cost of parsing a few extra identifiers.

These figures describe only request authentication, validation, target lookup, and state update on a single host; they are not Wi-Fi, MQTT, ESP32, relay, or public-internet latency measurements, and should not be read as a prediction of real household switching time.

Table 4. Software-in-the-loop API timing results

Devices/co mmand	Requ ests	Mea n payl oad (B)	Me an (ms)	Med ian (ms)	P9 5 (m s)	Succ ess
1	220	64.5	2.0 59	1.74 2	2.7 62	100. 0%
2	220	83	1.7 47	1.63 7	2.2 2	100. 0%
4	220	120	1.8 51	1.70 4	2.4 47	100. 0%
8	220	194	1.8 57	1.65 6	2.3 06	100. 0%

X. NEGATIVE API AND SECURITY CHECKS

All six negative and positive API checks produced the intended status code (Table 5): a valid signed command returned 200; a missing token and an invalid token each returned 401; an unknown device and an unknown endpoint each returned 404; and an empty command returned 400.

Table 5. Negative API and security acceptance checks

Test	Observed status	Expected status	Outcome
Valid signed command	200	200	PASS
Missing token rejected	401	401	PASS
Invalid token rejected	401	401	PASS
Unknown device rejected	404	404	PASS
Empty command rejected	400	400	PASS
Unknown endpoint rejected	404	404	PASS

The result that matters is not the 100% match rate itself but the absence of unintended state-changing

fall-through — missing or invalid identity, unknown targets, and empty command bodies were all rejected before reaching the device-state path.

XI. AUTOMATION-RULE TESTING

Thirty deterministic rule tests were executed: twelve ordinary automation combinations, six safety-threshold cases, and twelve manual-override cases, including values immediately below and exactly at defined boundaries (Table 6). All expected outputs matched observed outputs. Manual-override tests confirmed that a resident can force the light or fan state even when the ordinary comfort rule would select the opposite state, establishing that “automatic” does not imply uncontrollable within the tested rule set.

Table 6. Automation-rule acceptance summary

Category	Passed	Total	Pass rate
Automation	12	12	100.0%
Safety	6	6	100.0%
Override	12	12	100.0%

The result establishes logical correctness for the tested rule set under the stated conditions, not sensor calibration or field safety; boundary thresholds (for example, 30.0 °C for the fan rule and 120 lux for the lighting rule) are prototype values chosen to expose comparison semantics rather than calibrated comfort standards.

XII. DISCUSSION

The measured API path is small relative to the delays likely to arise from wireless transport and physical actuation, reinforcing the value of treating end-to-end latency as a chain rather than a single server-side number. This is consistent with the edge-oriented position of Yar et al. (2021): the present design keeps time-sensitive automation-rule evaluation logically close to the controller while using the service layer for authentication and management, though the benchmark reported here validates only the service component of that chain, not the complete edge claim. The negative-test results similarly align with

lightweight-authentication research (Oh et al., 2021; Yu et al., 2021): missing credentials, invalid credentials, unknown targets, and malformed commands were not accepted silently, which should be read as functional evidence of a fail-closed command path rather than cryptographic assurance — a production implementation still requires a stronger key lifecycle, TLS transport, and device-identity management.

The rule-engine results support a cyber-physical feedback view of automation, in which decisions are explicit, inspectable transformations from observed conditions and policy to actuator state, consistent with sensor-feedback control work such as Figueiredo et al. (2020) and energy-management extensions such as Alhasnawi et al. (2021) and Rochd et al. (2021); the present rules are intentionally simpler, but their transparency and the explicit precedence given to manual override are consistent with the trust and control considerations identified by Schomakers et al. (2021) and Reisinger et al. (2023).

The most important limitation of this study, stated directly rather than implied, is that no ESP32 controller, sensor, or relay has yet been physically connected and timed: Wi-Fi range, relay-switching delay, sensor accuracy, mains-load endurance, and any energy saving remain unmeasured, and the 100% success rate describes reliability inside one finite, controlled, host-local run rather than a guarantee under real network and power conditions.

Reporting the software-in-the-loop benchmark for exactly what it measures, rather than extrapolating it into a field result, preserves the credibility of the baseline and defines the next experimental stage precisely: connect the specified ESP32 controller and relay hardware, repeat the same request identifiers, and measure the additional wireless and electromechanical delay directly

XIII. CONCLUSION AND FUTURE WORK

This paper presented the design of a layered, security-aware smart-home automation architecture with mobile application control, built around an ESP32-class device layer, an authenticated and validated command/API service layer, a deterministic

automation-rule engine, and a mobile dashboard that privileges confirmed device state and explicit manual override. Within the software boundary that was actually tested, the implementation met its principal software objectives: across 880 requests the mean host-local API response time remained below 2.1 ms (95th percentile below 2.8 ms) with a 100% success rate, six representative negative/security checks returned their intended status codes, and thirty deterministic automation-rule tests — spanning ordinary, safety, and manual-override cases — produced correct outputs. No claim is made that these results demonstrate residential electrical safety, field-measured Wi-Fi range, verified sensor accuracy, or real household switching latency; those outcomes depend on hardware that has been specified (Section 3.3) but not yet assembled and timed.

The principal direction for future work is hardware-in-the-loop validation: connecting the specified ESP32 controller, sensors, and relay module to the tested service, tracing the same request identifiers used in this software benchmark through command publication, controller receipt, actuator change, and acknowledgement, and thereby obtaining genuine end-to-end response time. Mains switching should be implemented only with appropriately rated, isolated hardware reviewed by a competent electrical practitioner; transport security should be strengthened with TLS and per-device credentials before any residential deployment; local operation, including manual switches and safety rules, should continue to function during internet or broker failure; and sensor thresholds should be calibrated in the target environment rather than left as prototype constants. Longer-duration reliability testing (repeated restarts, packet loss, power interruption) and representative-user interface evaluation are recommended before the system is considered ready for residential use. Taken together, this work provides a complete, testable software architecture and a reproducible evaluation baseline whose logical next step is disciplined physical validation rather than further software iteration.

REFERENCES

1. Al-Begain, K., Khan, M., Alothman, B., Joumaa, C., & Alrashed, E. (2022). A DDoS detection and prevention

- system for IoT devices and its application to smart home environment. *Applied Sciences*, 12(22), Article 11853. <https://doi.org/10.3390/app122211853>
2. Alhasnawi, B. N., Jasim, B. H., Rahman, Z.-A. S. A., & Siano, P. (2021). A novel robust smart energy management and demand reduction for smart homes based on Internet of Energy. *Sensors*, 21(14), Article 4756. <https://doi.org/10.3390/s21144756>
3. Aliero, M. S., Qureshi, K. N., Pasha, M. F., & Jeon, G. (2021). Smart home energy management systems in Internet of Things networks for green cities demands and services. *Environmental Technology & Innovation*, 22, Article 101443. <https://doi.org/10.1016/j.eti.2021.101443>
4. Bugeja, J., Jacobsson, A., & Davidsson, P. (2021). PRASH: A framework for privacy risk analysis of smart homes. *Sensors*, 21(19), Article 6399. <https://doi.org/10.3390/s21196399>
5. Cerino, L., FakhrHosseini, S., Lee, C., Lee, S.-H., Son, H., & Coughlin, J. (2022). Towards a more connected home: User attitudes and perceptions after an integrated home technology exposure. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 66(1), 1917–1921. <https://doi.org/10.1177/1071181322661229>
6. Chen, Y.-Y., Chen, M.-H., Chang, C.-M., Chang, F.-S., & Lin, Y.-H. (2021). A smart home energy management system using two-stage non-intrusive appliance load monitoring over fog-cloud analytics based on Tridium's Niagara framework for residential demand-side management. *Sensors*, 21(8), Article 2883. <https://doi.org/10.3390/s21082883>
7. Edu, J. S., Such, J. M., & Suarez-Tangil, G. (2021). Smart home personal assistants: A security and privacy review. *ACM Computing Surveys*, 53(6), Article 116. <https://doi.org/10.1145/3412383>
8. El-Azab, R. (2021). Smart homes: Potentials and challenges. *Clean Energy*, 5(2), 302–315. <https://doi.org/10.1093/ce/zkab010>
9. Fahim, M., Kazmi, S. M. A., & Khattak, A. M. (2022). ApplianceNet: A neural network based framework to recognize daily life activities and behavior in smart home using smart plugs. *Neural Computing and Applications*, 34, 12749–12763. <https://doi.org/10.1007/s00521-022-07144-1>
10. Fazeldehkhordi, E., & Grønli, T.-M. (2022). A survey of security architectures for edge computing-based IoT. *IoT*, 3(3), 332–365. <https://doi.org/10.3390/iot3030019>
11. Figueiredo, R. E., Alves, A. A., Monteiro, V., Pinto, J. G., Afonso, J. L., & Afonso, J. A. (2020). Development and evaluation of smart home IoT systems applied to HVAC monitoring and control. *EAI Endorsed Transactions on*

- Energy Web, 8(34). <https://doi.org/10.4108/eai.19-11-2020.167205>
12. George, J. F., Chen, R., & Yuan, L. (2021). Intent to purchase IoT home security devices: Fear vs privacy. *PLOS ONE*, 16(9), Article e0257601. <https://doi.org/10.1371/journal.pone.0257601>
13. Hamdan, S., Ayyash, M., & Almajali, S. (2020). Edge-computing architectures for Internet of Things applications: A survey. *Sensors*, 20(22), Article 6441. <https://doi.org/10.3390/s20226441>
14. Iten, R., Wagner, J., & Zeier Röschmann, A. (2021). On the identification, evaluation and treatment of risks in smart homes: A systematic literature review. *Risks*, 9(6), Article 113. <https://doi.org/10.3390/risks9060113>
15. Karale, A. (2021). The challenges of IoT addressing security, ethics, privacy, and laws. *Internet of Things*, 15, Article 100420. <https://doi.org/10.1016/j.iot.2021.100420>
16. Laroui, M., Nour, B., Moun gla, H., Cherif, M. A., Afifi, H., & Guizani, M. (2021). Edge and fog computing for IoT: A survey on current research activities & future directions. *Computer Communications*, 180, 210–231. <https://doi.org/10.1016/j.comcom.2021.09.003>
17. Lee, D.-H., & Lee, I.-Y. (2020). A lightweight authentication and key agreement schemes for IoT environments. *Sensors*, 20(18), Article 5350. <https://doi.org/10.3390/s20185350>
18. Li, W. D., Yigitcanlar, T., Erol, I., & Liu, A. R. (2021). Motivations, barriers and risks of smart home adoption: From systematic literature review to conceptual framework. *Energy Research & Social Science*, 80, Article 102211. <https://doi.org/10.1016/j.erss.2021.102211>
19. Ma, Y., Chen, X., Wang, L., & Yang, J. (2021). Study on smart home energy management system based on artificial intelligence. *Journal of Sensors*, 2021, Article 9101453. <https://doi.org/10.1155/2021/9101453>
20. Majeed, R., Abdullah, N. A., Ashraf, I., Zikria, Y. B., Mushtaq, M. F., & Umer, M. (2020). An intelligent, secure, and smart home automation system. *Scientific Programming*, 2020, Article 4579291. <https://doi.org/10.1155/2020/4579291>
21. Mekruksavanich, S., & Jitpattanakul, A. (2021). LSTM networks using smartphone data for sensor-based human activity recognition in smart homes. *Sensors*, 21(5), Article 1636. <https://doi.org/10.3390/s21051636>
22. Oh, J., Yu, S., Lee, J., Son, S., Kim, M., & Park, Y. (2021). A secure and lightweight authentication protocol for IoT-based smart homes. *Sensors*, 21(4), Article 1488. <https://doi.org/10.3390/s21041488>
23. Rahimi, M., Songhorabadi, M., & Haghi Kashani, M. (2020). Fog-based smart homes: A systematic review. *Journal of Network and Computer Applications*, 153, Article 102531. <https://doi.org/10.1016/j.jnca.2020.102531>
24. Schomakers, E.-M., Biermann, H., & Ziefle, M. (2021). Users' preferences for smart home automation—Investigating aspects of privacy and trust. *Telematics and Informatics*, 64, Article 101689. <https://doi.org/10.1016/j.tele.2021.101689>
25. Sepasgozar, S., Karimi, R., Farahzadi, L., Moezzi, F., Shirowzhan, S., Ebrahimzadeh, S. M., Hui, F., & Aye, L. (2020). A systematic content review of artificial intelligence and the Internet of Things applications in smart home. *Applied Sciences*, 10(9), Article 3074. <https://doi.org/10.3390/app10093074>
26. Taiwo, O., & Ezugwu, A. E. (2021). Internet of Things-based intelligent smart home control system. *Security and Communication Networks*, 2021, Article 9928254. <https://doi.org/10.1155/2021/9928254>
27. Yar, H., Imran, A. S., Khan, Z. A., Sajjad, M., & Kastrati, Z. (2021). Towards smart home automation using IoT-enabled edge-computing paradigm. *Sensors*, 21(14), Article 4932. <https://doi.org/10.3390/s21144932>