The Smart Home Safety & Auto Lighting System

Pavan Kalam
Masters in Computer Science
University of Missouri – Kansas City

Muhammad Samran Ali
Masters in Computer Science
University of Missouri – Kansas City

Lakshmi Sai Deep Bomidi
Masters in Computer Science
University of Missouri – Kansas City

Nabil Kana
Masters in Electrical Engineering
University of Missouri – Kansas City

***Abstract*—This project presents the design and implementation of smart home system that integrates safety monitoring, automated lighting, and security alerting using IoT devices. The system employs two ESP32 microcontrollers, seven environmental and security sensors, and a Python Flask web server with a PostgreSQL backend to provide real-time monitoring, automated actuation, and remote control through a web dashboard. Sensor data are transmitted wirelessly using the ESP-NOW protocol, encrypted using AES-128-CBC, and processed by server-side automation logic that controls lighting, buzzer alerts, and notification mechanisms. Experimental results demonstrate sub-second end-to-end latency, high alert reliability, and significant potential energy savings through intelligent lighting control and timeout management.**

Keywords— Automated lighting control, ESP32 microcontroller, ESP-NOW, AES-128-CBC encryption, Flask web server, Real-time monitoring and control, Multi-sensor safety monitoring.

I. Introduction

Smart home technologies are increasingly deployed to enhance safety, comfort, and energy efficiency in residential and commercial environments, yet many available solutions remain fragmented, proprietary, and expensive. Typical systems separate security features from lighting automation and often lack strong data security or unified monitoring interfaces. This creates gaps in both user experience and safety, particularly for applications that require timely detection of hazardous conditions such as fire or gas leaks.

In this project we aim to address these issues by developing a comprehensive IoT smart home safety and auto lighting system that combines multi sensor monitoring, automated control, and secure data transmission in a single architecture. The system integrates seven different sensors PIR motion, flame, air quality (MQ135), door/window (reed switch), sound, light (LDR), and temperature/humidity (DHT11), with two ESP32 development boards and a cloud connected Python server. Real-time data are visualized and controlled through a browser based dashboard, enabling users to monitor environmental conditions, configure system behaviour, and manually override actuators when needed.

II. System Architecture

A. Three Tier Architecture

The system follows a three tier architecture with a sensor layer, control layer, and application layer. The sensor layer is implemented on an ESP32 board that periodically acquires readings from seven sensors and transmits the data to the control layer via ESP-NOW. The control layer is implemented on a second ESP32 board that receives sensor packets, optionally encrypts them, forwards them to the server over HTTP, and drives actuators based on commands polled from the application layer. The application layer is a Python Flask web server backed by a PostgreSQL database that performs automation logic, persists data, manages user sessions, and serves a web dashboard.

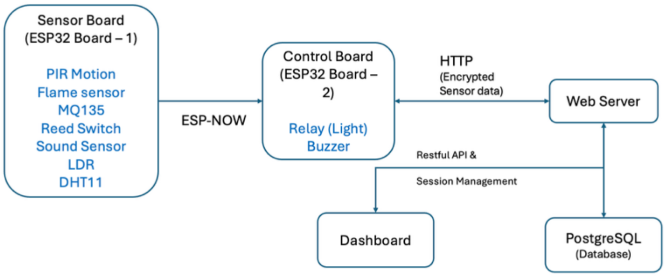


Fig. I. System Communication Flow

B. System Communication

Communication between layers uses multiple protocols tailored to different requirements. ESP-NOW is used for sensor board to control board communication, providing low-latency, peer-to-peer wireless links without requiring Wi-Fi association or access points. The control board communicates with the server over HTTP using JSON for encrypted payloads and form-encoded data for unencrypted modes. The server exposes RESTful APIs that are consumed both by the control board and by the JavaScript front-end running in the user's browser.

The dashboard communicates with the Flask backend using periodic HTTP requests to fetch system state, sensor data, event logs, and configuration parameters, while user actions such as toggling modes or per-sensor options are sent as POST or PUT requests to corresponding API endpoints.

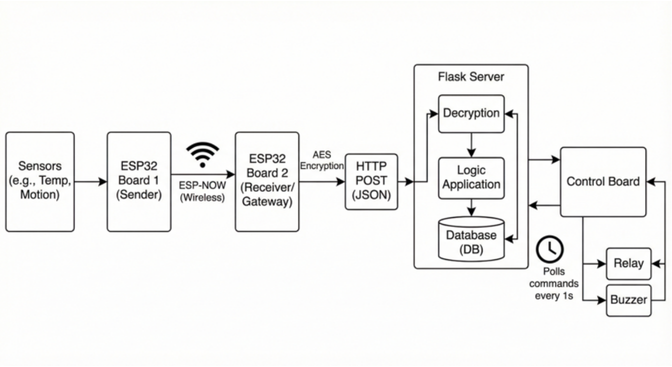


Fig. II. System Communication Flow

C. *Data Flow*

The data flow pipeline is structured as follow, firstly, the sensor board samples all sensor channels at a predefined interval (typically 2 seconds), applies debouncing or averaging as needed, and forwards a structured payload including sensor readings and a timestamp via ESP-NOW to the control board. Second, the control board optionally encrypts the payload using AES-128-CBC, base64-encodes the result, and sends it to the server through an HTTP POST request with appropriate metadata indicating whether the content is encrypted. Third, the server validates and decrypts the data, stores raw readings in PostgreSQL, and runs automation logic to determine alerts and actuator commands. Fourth, the control board periodically polls the server for the latest control state -- light status, buzzer status, brightness level, operating mode -- and applies the commands to the hardware. Finally, the dashboard queries the server to visualize sensor history, current status, and recent events for the user.

D. *Design Principle*

The system architecture adheres to modularity, scalability, reliability, and security as core design principles. Modularity is achieved by separating sensor acquisition, actuation, and business logic across three layers, enabling independent development and testing. Scalability is supported by an extensible database schema and REST interface that can be expanded to additional sensors. Reliability is improved through error handling, timeout management, and reconnection mechanisms at the firmware and server levels. Security is enforced with cryptographic protection of sensor data, authentication for dashboard access, and safe handling of database operations.

III. Hardware Design

A. *ESP32 Boards*

The system uses two ESP32 boards based on the ESP-WROOM-32 module, each featuring a dual-core 32-bit microprocessor running up to 240 MHz, integrated Wi-Fi (802.11 b/g/n), Bluetooth/BLE, and multiple GPIOs with 12-bit ADC channels. One board is dedicated to sensor acquisition, while the other is responsible for encryption, network communication with the server, and actuator control. Both boards are powered by 5 V via USB, with onboard regulators providing 3.3 V for logic and sensor interfaces.

B. *Sensors*

Seven sensors are connected to the sensor board to monitor motion, fire, gas concentration, door/window state, sound level, ambient light, and environmental conditions. A PIR motion sensor on a digital GPIO pin detects movement within a 3–7 m range and approximately 120° field of view. A flame sensor on a digital pin detects infrared radiation from flames within about 1 m. An MQ135 sensor on an analog ADC channel measures air quality in terms of multiple gases such as CO_2, CO, NH_3, and smoke using a 0–4095 ADC range. A reed switch on a digital input tracks door or window state via magnet proximity. A sound sensor on an analog channel measures noise intensity, allowing detection of events such as glass breaking. An LDR connected to an analog channel provides a measure of ambient light level, where higher ADC values correspond to darker environments. A DHT11 digital sensor reports temperature and relative humidity with modest accuracy and 1 Hz sampling frequency.

Table 1. Sensor & Pin Configuration

Sensor	***Type***	***Pin***
PIR Motion	Digital	GPIO 12
Flame	Digital	GPIO 14
MQ135 Air Quality Sensor	Analog	GPIO 34
Reed Switch	Digital	GPIO 13
Sound Sensor	Analog	GPIO 33
LDR Light	Analog	GPIO 32
DHT11	Digital	GPIO 27

C. *Actuators*

Two actuators are used by the control board. A 5 V relay module on a digital output pin controls a lighting circuit, supporting typical AC lighting loads with optocoupler isolation between low-voltage control and mains voltage. An active buzzer on another GPIO pin provides audio alerts at approximately 85 dB when activated. This combination enables both visual and auditory feedback for safety and security events

IV. Software And Communication

A. *ESP32 Firmware*

Both ESP32 boards are programmed using Arduino C++ with the ESP32 board support package. The sensor board firmware initializes each sensor, periodically reads their values, performs basic filtering (such as averaging for analog sensors and debouncing for digital transitions), and assembles a JSON-like payload that includes readings and a timestamp. This payload is transmitted via ESP-NOW to the control board, using the MAC address of the receiving board and automatic retry handling.

The control board firmware is responsible for receiving sensor packets, optionally encrypting them, and forwarding the result to the server via HTTP. When encryption is enabled, the firmware uses mbedTLS to perform AES-128-CBC encryption with a 16-byte key and randomly generated initialization vector (IV) for each transmission, then concatenates IV and ciphertext and encodes them using Base64 before sending as part of a JSON body. The control board also polls the server at a configurable interval (e.g., 2 s) to retrieve updated control parameters such as light state, buzzer state, brightness level, and system modes.

B. *Python Web Server and RESTful APIs*

The application layer is a Python Flask web server backed by PostgreSQL that exposes a comprehensive set of RESTful APIs (Table I). Authentication is session-based: /login verifies SHA-256 hashed credentials and issues a secure cookie; '/register' creates new accounts; /logout clears the session; '/forgot-password' enables password reset. All protected endpoints are decorated with @login_required.

Sensor data ingestion occurs via POST /api/sensor-data, accepting both unencrypted form data and AES-128-CBC encrypted Base64 payloads (with an is_encrypted flag). Control endpoints (PUT /api/control/*) allow authenticated clients to toggle light, buzzer, brightness (0–100 %), Auto/Manual mode, and Home/Away mode. Visualization endpoints (GET /api/system-state, GET /api/sensor-events) return real-time JSON used by the dashboard. Configuration endpoints update per-sensor flags, upload intervals, and Wi-Fi credentials dynamically.

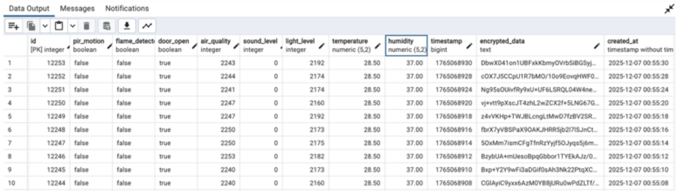
TABLE II. Key RESTful API Endpoints

Method	*Endpoint*	*Description*	*Auth Required*
POST	/login	User login	No
POST	/register	Create account	No
POST	/forgot-password	Reset password	No
POST	/api/sensor-data	Submit encrypted/unencrypted data	No
GET	/api/system-state	Live sensor + actuator state	Yes
GET	/api/sensor-events	Latest event per sensor	Yes
PUT	/api/control/light	Light on/off	Yes
PUT	/api/control/buzzer	Buzzer on/off	Yes
PUT	/api/control/mode	Auto ↔ Manual	Yes
PUT	/api/control/home-mode	Home ↔ Away	Yes
PUT	/api/control/brightness	0–100 %	Yes
POST	/api/sensor-control/toggle	Per-sensor light/buzzer enable	Yes

C. *Database Schema*

The PostgreSQL database schema includes tables for sensor data, per-sensor controls, system-wide control state, user accounts, event logs, and notifications. The 'sensor_data' table stores each reading with fields including booleans for motion, flame, and door status; integers for air quality, sound level, and light level; floating-point values for temperature and humidity; raw timestamp; optional encrypted payload; and creation time. The 'system_control' table maintains the canonical light and buzzer state, brightness level, manual or automatic mode, home or away mode, and timestamps for when actuators were last activated, supporting timeout logic. The 'sensor_controls' table stores per-sensor flags indicating whether each sensor is allowed to trigger the light or buzzer and is initialized with sensible defaults, such as disabling buzzer activation for the LDR and treating the DHT11 as monitoring-only.

Below screenshot shows the 'sensor_data' information in the database,



	id [PK] integer	pir_motion boolean	flame_detect boolean	door_open boolean	air_quality integer	sound_level integer	light_level integer	temperature numeric (5,2)	humidity numeric (5,2)	timestamp bigint	encrypted_data text	created_at timestamp without tim
1	12253	false	false	true	2243	0	2192	28.50	37.00	1765068930	DbwX041on1UBFxkKbmyOVrbSiBGSyj…	2025-12-07 00:55:30
2	12252	false	false	true	2244	0	2174	28.50	37.00	1765068928	cOX7J5CCpU1R7bMO/10o9EovqHWF0…	2025-12-07 00:55:28
3	12251	false	false	true	2241	0	2174	28.50	37.00	1765068924	Ng95sOUivfRy9xU+UF6LSRQL04W4ne…	2025-12-07 00:55:24
4	12250	false	false	true	2247	0	2160	28.50	37.00	1765068920	vj+vtt9pXscJT4zhL2wZCX2f+5LNG67G…	2025-12-07 00:55:20
5	12249	false	false	true	2247	0	2192	28.50	37.00	1765068918	z4vVKHp+TWJBLcngLtMwD7fzBV2SR…	2025-12-07 00:55:18
6	12248	false	false	true	2250	0	2173	28.50	37.00	1765068916	fbrX7yVBSPaX9OAKJHRR5jb2l7lSJnCt…	2025-12-07 00:55:16
7	12247	false	false	true	2250	0	2175	28.50	37.00	1765068914	5OxMm7ismCFgTfnRzYyjf5OJyqs5j6m…	2025-12-07 00:55:14
8	12246	false	false	true	2253	0	2182	28.50	37.00	1765068912	BzybUA+mUesoBpqGbbor1TYEkAJz/0…	2025-12-07 00:55:12
9	12245	false	false	true	2240	0	2173	28.50	37.00	1765068910	Bxp+Y2Y9wFi3aDGif0sAh3Nk22PtqXC…	2025-12-07 00:55:10
10	12244	false	false	true	2240	0	2160	28.50	37.00	1765068908	CGlAyiC9yxx6AzM0YB8jURu0wPdZLTf/…	2025-12-07 00:55:08

Fig III. Sensor Data Information

Below table shows all the tables used in the projects and usefulness of tables,

TABLE III. Database Tables and their Purpose

Table	***Key Columns***	***Purpose***
sensor_data	pir_motion, flame_detected, air_quality, timestamp, encrypted_data, sound sensor, light_level, temp, humidity	Raw sensor history
system_control	light_on, buzzer_on, manual_mode, home_mode, brightness_level	Current actuator state
sensor_controls	sensor_name, light_enabled, buzzer_enabled, updated_time	Per-sensor customization
event_log	event_type, event_message, timestamp	Human-readable alerts
notifications	title, message, read, notification_type created_at	Dashboard + email alerts
users	username, password_hash, role, timestamp	Admin/user authentication

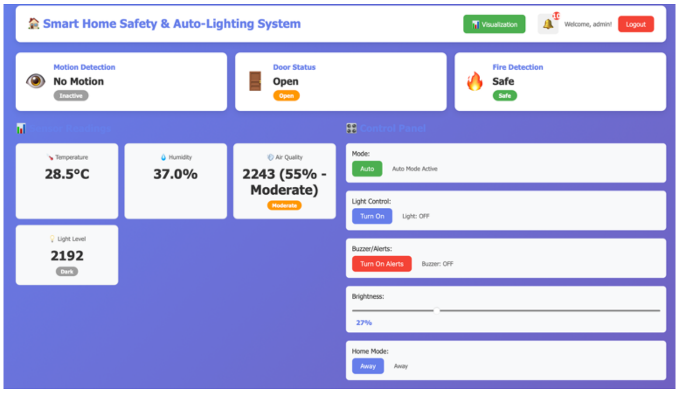
sensor_events	Sensor_name, sensor_information, action_taken, timestamp	Realtime action taken by each sensor (light/ buzzer on/off)
sensor_board_control	Monitoring, encryption_enabled, upload_interval, wifi_ssid, wofo_password, server_url, updated_time	To check sensor_board information

Indexing on timestamp and other frequently queried columns improves query performance for real-time dashboards and historical analytics. Timezone aware timestamps or explicit UTC conversion are used to prevent ambiguity when correlating events across layers.

D. *Web Dashboard*

The front-end dashboard is implemented using HTML, CSS, and JavaScript. It periodically polls server APIs (approximately every 2 s) to display live sensor readings, actuator status, and system modes, with color-coded status indicators that differentiate normal, warning, and critical states. The dashboard includes a control panel for manually toggling the light and buzzer, switching between Auto and Manual control modes, and selecting Home or Away occupancy modes. It also presents an event log with scrolling history of sensor events, mode changes, and automated actions, along with a dedicated table summarizing the most recent event per sensor type.

Per-sensor configuration options are exposed through toggles for enabling or disabling light and buzzer activation per sensor, and changes are sent via asynchronous requests to update the database immediately. The interface is responsive, adapting to desktop and mobile devices using flexible layouts and media queries.



🏠 Smart Home Safety & Auto-Lighting System

Visualization | 🔔 10 | Welcome, admin! | Logout

Motion Detection: No Motion (Inactive)

Door Status: Open (Open)

Fire Detection: Safe (Safe)

Sensor Readings

Temperature: 28.5°C

Humidity: 37.0%

Air Quality: 2243 (55% - Moderate) (Moderate)

Light Level: 2192 (Dark)

Control Panel

Mode: Auto — Auto Mode Active

Light Control: Turn On — Light: OFF

Buzzer/Alerts: Turn On Alerts — Buzzer: OFF

Brightness: 27%

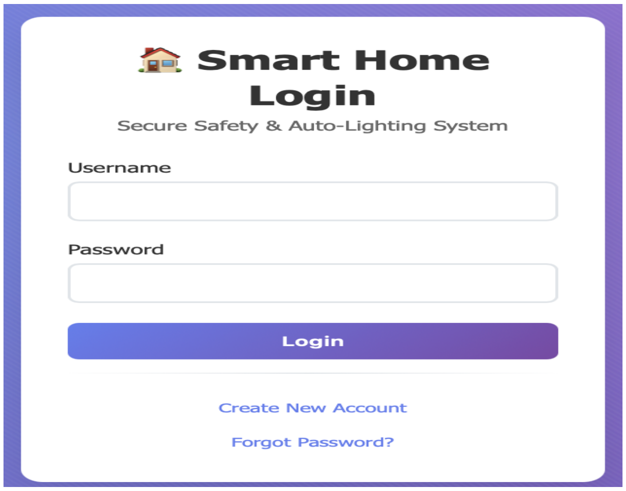
Home Mode: Away — Away

Fig. IV. Web Dashboard

E. *User Authentication and Account Management*

User authentication is implemented using traditional web forms and session cookies, with all credentials processed on the server. The '/register' endpoint allows new users to create accounts by submitting a username, password, and confirmation, the server validates basic constraints (such as minimum length and uniqueness) and stores only a SHA-256 hash of the password in the users table. The '/login' endpoint verifies credentials, increments an active-sessions counter, and populates the Flask session with user identifier and role information, which is then checked by a '@login_required' decorator on protected routes.

A '/forgot-password' workflow permits users to reset their password by supplying their username and a new password, which is again hashed before being persisted. The /logout endpoint clears the session, decrements the active-sessions count, and redirects users to the login page, ensuring clean termination of authenticated sessions. All RESTful control and visualization endpoints used by the dashboard, including those returning JSON system state and those mutating control settings, are guarded by session checks so that only logged-in users can access sensitive data or change system behaviour.



🏠 Smart Home Login

Secure Safety & Auto-Lighting System

Username

Password

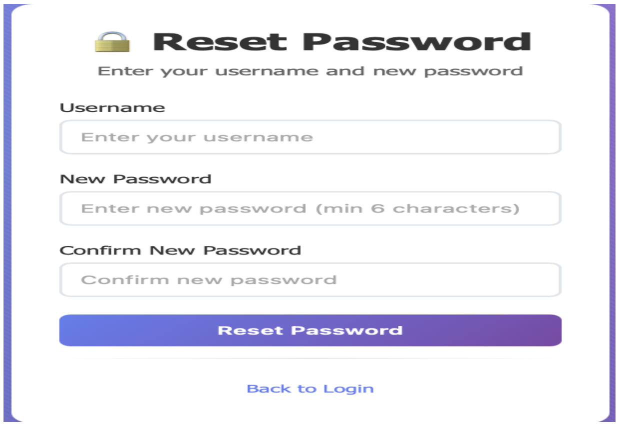
Login

Create New Account

Forgot Password?

Fig. V. User Authentication (Login)

Below both the screenshots show the changing user password and creating new user and every change directly hit the database,



🔒 Reset Password

Enter your username and new password

Username: Enter your username

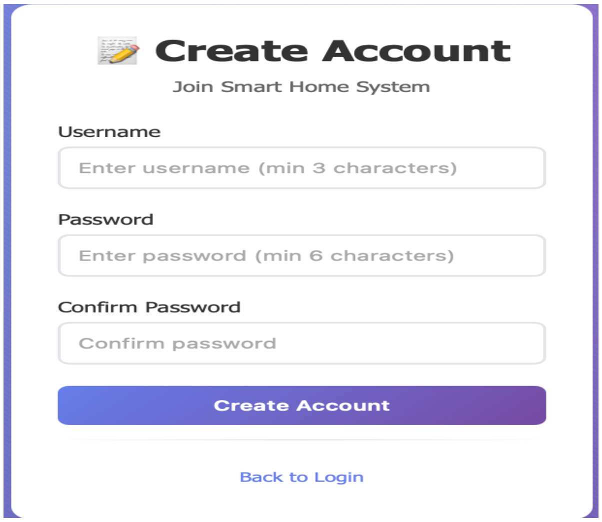
New Password: Enter new password (min 6 characters)

Confirm New Password: Confirm new password

Reset Password

Back to Login

Fig. VI. Change Password



Create Account

Join Smart Home System

Username

Enter username (min 3 characters)

Password

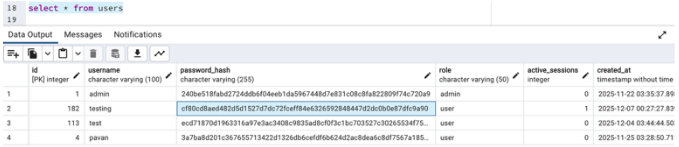
Enter password (min 6 characters)

Confirm Password

Confirm password

Fig. VII. Creating New User Account

Below is the screenshot of every user encrypted password stored in the PostgreSQL database,



```
select * from users
```

	id [PK] integer	username character varying (100)	password_hash character varying (255)	role character varying (50)	active_sessions integer	created_at timestamp without time
1	1	admin	240be518fabd2724ddb6f04eeb1da5967448d7e831c08c8fa822809f74c720a9	admin	0	2025-11-22 03:35:37.89
2	182	testing	cf80cd8aed482d5d1527d7dc72fceff84e6326592848447d2dc0b0e87dfc9a90	user	1	2025-12-07 00:27:27.83
3	113	test	ecd71870d1963316a97e3ac3408c9835ad8cf0f3c1bc703527c30265534f75...	user	0	2025-12-04 03:44:44.50
4	4	pavan	3a7ba8d201c367655713422d1326db6cefdf6b624d2ac8dea6c8df7567a185...	user	0	2025-11-25 03:28:50.71

Fig. VIII. Encrypted User Passwords

V. System Features and Automation

A. Multi Sensor Integration

The system continuously monitors all seven sensors to provide comprehensive situational awareness. Motion sensors detect occupant movement, flame and air quality sensors detect safety hazards, door sensors monitor entry points, sound sensors capture loud or abnormal noises, LDR sensors estimate ambient brightness, and DHT11 sensors report environmental conditions. Automation logic uses these readings in combination to trigger appropriate actions and classify events as normal, warning, or critical.

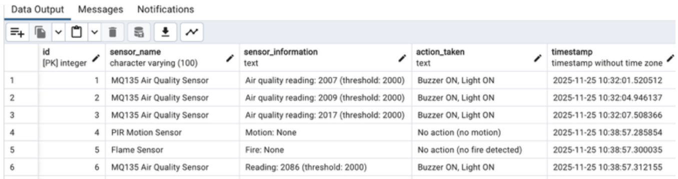
B. Automated Lighting

Lighting automation is a central feature of the system. In typical operation, motion detected by the PIR sensor in dark conditions -- as measured by the LDR -- causes the light to turn on for approximately 60 seconds, after which it is automatically switched off if no further motion is observed. Door opening can also trigger the light to turn on for a shorter duration (e.g., 10 s), providing entry lighting when someone enters in a dark environment. The brightness level is dynamically computed based on the LDR reading, mapping darker conditions to higher brightness within a predefined range such as 20–100%. These behaviors are conditioned on the current control mode (Auto or Manual) and occupancy mode (Home or Away).

C. Security Alerts

The system implements multiple security and safety alert mechanisms that use both the light and buzzer as actuators. When the flame sensor detects fire or the MQ135 sensor reports air quality above a configured hazard threshold, both light and buzzer are activated immediately for a short but noticeable duration, and the event is logged as critical. In Away mode, motion detection or door opening also triggers security alerts, causing both actuators to activate and generating prominent notifications in the dashboard and, optionally, via email. Sound sensor thresholds can be configured so that loud events such as glass breaking led to security alerts in both Home and Away modes, depending on user preferences.

Below screenshot shows the sensor events and provides the sensor information and type of action taken,

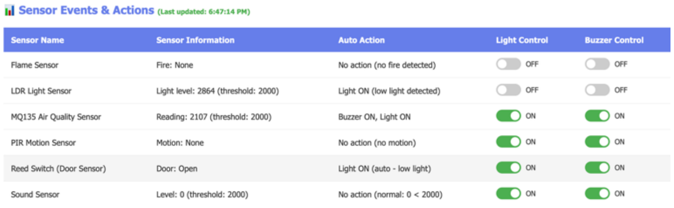


	id [PK] integer	sensor_name character varying (100)	sensor_information text	action_taken text	timestamp timestamp without time zone
1	1	MQ135 Air Quality Sensor	Air quality reading: 2007 (threshold: 2000)	Buzzer ON, Light ON	2025-11-25 10:32:01.520512
2	2	MQ135 Air Quality Sensor	Air quality reading: 2009 (threshold: 2000)	Buzzer ON, Light ON	2025-11-25 10:32:04.946137
3	3	MQ135 Air Quality Sensor	Air quality reading: 2017 (threshold: 2000)	Buzzer ON, Light ON	2025-11-25 10:32:07.508366
4	4	PIR Motion Sensor	Motion: None	No action (no motion)	2025-11-25 10:38:57.285854
5	5	Flame Sensor	Fire: None	No action (no fire detected)	2025-11-25 10:38:57.300035
6	6	MQ135 Air Quality Sensor	Reading: 2086 (threshold: 2000)	Buzzer ON, Light ON	2025-11-25 10:38:57.312155

Fig. IX. sensor Events Table Info

D. Per-Sensor Customization

A notable feature is per-sensor customization of responses. For each sensor type, separate flags indicate whether that sensor is allowed to trigger the light and whether it is allowed to trigger the buzzer. This enables behaviors such as allowing motion to control lighting but not alarms in Home mode or disabling buzzer responses for the LDR while still using its readings to compute brightness. These settings are stored persistently in the database and take effect without restarting the system; automation logic consults them on each sensor update to decide which actions are permitted.



Sensor Events & Actions (Last updated: 6:47:14 PM)

Sensor Name	Sensor Information	Auto Action	Light Control	Buzzer Control
Flame Sensor	Fire: None	No action (no fire detected)	OFF	OFF
LDR Light Sensor	Light level: 2864 (threshold: 2000)	Light ON (low light detected)	OFF	OFF
MQ135 Air Quality Sensor	Reading: 2107 (threshold: 2000)	Buzzer ON, Light ON	ON	ON
PIR Motion Sensor	Motion: None	No action (no motion)	ON	ON
Reed Switch (Door Sensor)	Door: Open	Light ON (auto - low light)	ON	ON
Sound Sensor	Level: 0 (threshold: 2000)	No action (normal: 0 < 2000)	ON	ON

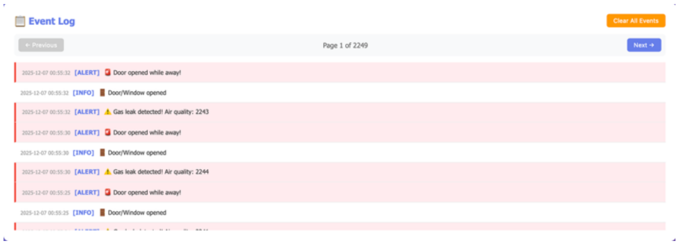
Fig. X. Per-Sensor Customization

E. Modes of Operation

The system defines both control modes (Auto vs. Manual) and occupancy modes (Home vs. Away), (refer Fig III). In Auto mode, automation logic controls the light and buzzer based on sensor inputs and configured policies, whereas in Manual mode, automatic activations are mostly suppressed and the user is expected to control actuators directly from the dashboard, except for critical safety alerts which may override manual settings. In Home mode, sensor events such as motion or door opening are treated primarily as convenience triggers for lighting, while in Away mode, the same events are interpreted as potential intrusions and result in security alerts. A combined mode table can describe the behavior for the four combinations of Auto/Home, Auto/Away, Manual/Home, and Manual/Away, including which sensors trigger which actuators under each configuration.

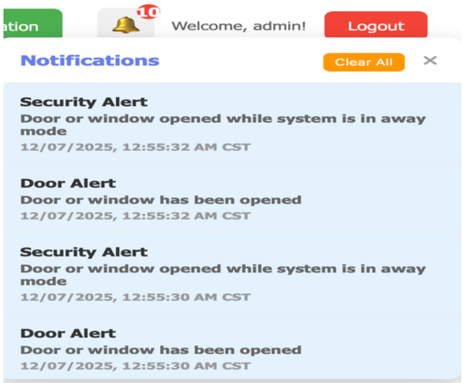
F. Event Logging, Notifications and Visualization

The system logs all significant activity, including sensor events, alerts, configuration changes, and user actions in a dedicated 'event_log' table with timestamps and descriptive messages, which can be queried via history-oriented endpoints. For high-level user feedback, a notifications table stores structured messages with severity types (such as info, warning, and critical), a read/unread flag, and creation time, and corresponding notification endpoints expose these entries to the dashboard or external clients. Critical conditions such as fire or gas leaks can also trigger optional email notifications using SMTP, with throttling logic for air-quality alerts to prevent excessive repeated messages during prolonged hazardous periods.



```
2025-12-07 00:55:32 [ALERT] 🚨 Door opened while away!
2025-12-07 00:55:32 [INFO] 🚪 Door/Window opened
2025-12-07 00:55:32 [ALERT] ⚠ Gas leak detected! Air quality: 2243
2025-12-07 00:55:30 [ALERT] 🚨 Door opened while away!
2025-12-07 00:55:30 [INFO] 🚪 Door/Window opened
2025-12-07 00:55:30 [ALERT] ⚠ Gas leak detected! Air quality: 2244
2025-12-07 00:55:25 [ALERT] 🚨 Door opened while away!
2025-12-07 00:55:25 [INFO] 🚪 Door/Window opened
```

Fig. XI. Event Logs



Security Alert
Door or window opened while system is in away mode
12/07/2025, 12:55:32 AM CST

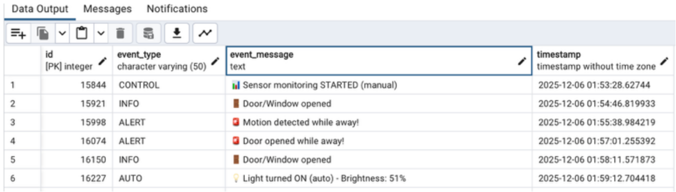
Door Alert
Door or window has been opened
12/07/2025, 12:55:32 AM CST

Security Alert
Door or window opened while system is in away mode
12/07/2025, 12:55:30 AM CST

Door Alert
Door or window has been opened
12/07/2025, 12:55:30 AM CST

Fig. XII. Alert Notifications

Below screenshot shows event logs stored in the database,



	id [PK] integer	event_type character varying (50)	event_message text	timestamp timestamp without time zone
1	15844	CONTROL	📊 Sensor monitoring STARTED (manual)	2025-12-06 01:53:28.62744
2	15921	INFO	🚪 Door/Window opened	2025-12-06 01:54:46.819933
3	15998	ALERT	🚨 Motion detected while away!	2025-12-06 01:55:38.984219
4	16074	ALERT	🚨 Door opened while away!	2025-12-06 01:57:01.255392
5	16150	INFO	🚪 Door/Window opened	2025-12-06 01:58:11.571873
6	16227	AUTO	💡 Light turned ON (auto) - Brightness: 51%	2025-12-06 01:59:12.704418

Fig. XIII. Event Logs stored in DB

Visualization is provided through the real-time web dashboard, which uses periodic calls to GET /api/system-state to retrieve the latest sensor readings, actuator states, mode flags, and derived metrics such as air quality percentage and qualitative status labels. A complementary call to GET /api/sensor-events returns a compact list of the most recent event per sensor, including textual descriptions and the actions taken by the automation logic, which is rendered as a sensor-events table in the UI. The dashboard presents these data in a combination of numeric readouts, color-coded badges, and structured tables, giving users an at-a-glance view of environmental conditions, system health, and recent behaviour.

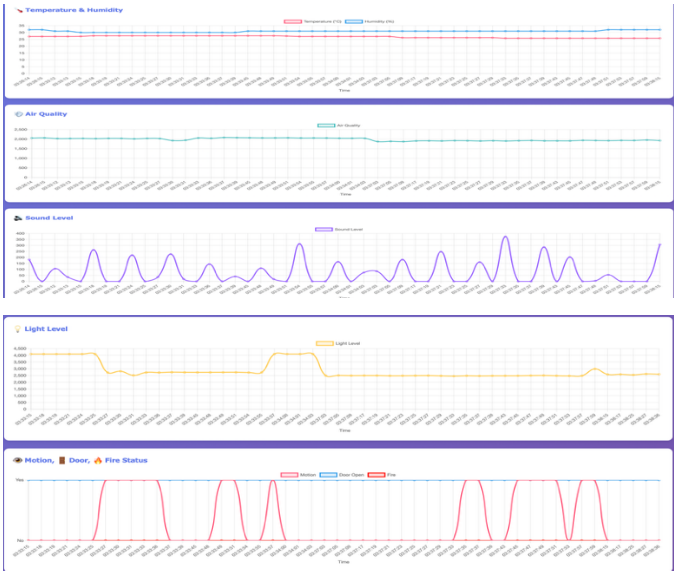
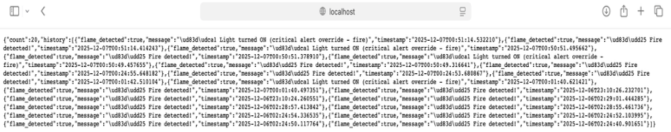


Fig. XIV. Graphical Visualization (Sensor Readings)

G. RESTful APIs

The system uses RESTful APIs for dashboard interactions with the server and all the interactions will perform in JSON format, before fetching the information the system checks the user authentication, once it is successful the system fetch the information in JSON format, below are some examples of URLs,

- http://localhost:8888/api/history/fire?limit=20 (it will show the last 20 alerts of fire detection.



{"count":20,"history":[{"flame_detected":true,"message":"\ud83d\udca1 Light turned ON (critical alert override - fire)","timestamp":"2025-12-07T00:51:14.532210"},{"flame_detected":true,"message":"\ud83d\udd25 Fire detected!","timestamp":"2025-12-07T00:51:14.414243"},{"flame_detected":true,"message":"\ud83d\udca1 Light turned ON (critical alert override - fire)","timestamp":"2025-12-07T00:50:51.495662"},
{"flame_detected":true,"message":"\ud83d\udd25 Fire detected!","timestamp":"2025-12-07T00:50:51.378910"},{"flame_detected":true,"message":"\ud83d\udca1 Light turned ON (critical alert override - fire)","timestamp":"2025-12-07T00:50:49.457655"},{"flame_detected":true,"message":"\ud83d\udd25 Fire detected!","timestamp":"2025-12-07T00:50:49.316641"},{"flame_detected":true,"message":"\ud83d\udd25 Fire detected!","timestamp":"2025-12-07T00:24:55.648182"},{"flame_detected":true,"message":"\ud83d\udd25 Fire detected!","timestamp":"2025-12-07T00:24:53.680867"},{"flame_detected":true,"message":"\ud83d\udd25 Fire detected!","timestamp":"2025-12-07T00:01:42.510104"},{"flame_detected":true,"message":"\ud83d\udca1 Light turned ON (critical alert override - fire)","timestamp":"2025-12-07T00:01:40.621421"},
{"flame_detected":true,"message":"\ud83d\udd25 Fire detected!","timestamp":"2025-12-07T00:01:40.497351"},{"flame_detected":true,"message":"\ud83d\udd25 Fire detected!","timestamp":"2025-12-06T23:10:26.232701"},
{"flame_detected":true,"message":"\ud83d\udd25 Fire detected!","timestamp":"2025-12-06T23:10:24.260551"},{"flame_detected":true,"message":"\ud83d\udd25 Fire detected!","timestamp":"2025-12-06T02:29:01.444285"},
{"flame_detected":true,"message":"\ud83d\udd25 Fire detected!","timestamp":"2025-12-06T02:28:57.413842"},{"flame_detected":true,"message":"\ud83d\udd25 Fire detected!","timestamp":"2025-12-06T02:28:55.461736"},
{"flame_detected":true,"message":"\ud83d\udd25 Fire detected!","timestamp":"2025-12-06T02:24:54.336535"},{"flame_detected":true,"message":"\ud83d\udd25 Fire detected!","timestamp":"2025-12-06T02:24:52.103995"},
{"flame_detected":true,"message":"\ud83d\udd25 Fire detected!","timestamp":"2025-12-06T02:24:50.117764"},{"flame_detected":true,"message":"\ud83d\udd25 Fire detected!","timestamp":"2025-12-06T02:24:40.901651"}]}

Fig. XV. Last 20 Fire Alerts in JSON Format

- http://localhost:8888/api/sensor-board/info (it is to get the sensor board information)
- curl -X PUT http://localhost:8888/api/sensor-board/encryption \

  -b cookies.txt \

  -H "Content-Type: application/json" \

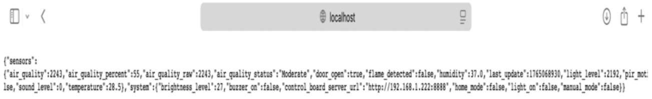
```
-d '{"encryption_enabled": true}'
```

This the command to encrypt the sensor data.

- Below is the command to start monitoring,

```
curl -X PUT http://localhost:8888/api/sensor-board/monitoring \
-b cookies.txt \
-H "Content-Type: application/json" \
-d '{"monitoring": true}'
```

- http://localhost:8888/api/system-state (this will show all the system information)



```
{"sensors":
{"air_quality":2243,"air_quality_percent":55,"air_quality_raw":2243,"air_quality_status":"Moderate","door_open":true,"flame_detected":false,"humidity":37.0,"last_update":1765068930,"light_level":2192,"pir_moti
lse,"sound_level":0,"temperature":28.5},"system":{"brightness_level":27,"buzzer_on":false,"control_board_server_url":"http://192.168.1.222:8888","home_mode":false,"light_on":false,"manual_mode":false}}
```

Fig XVI. Comple system Info (JSON)

VI. Security Implementations

The system security is addressed across transport, application, and data-storage layers through a combination of cryptography, authentication. Through this the sensor data is protected in transit, restrict access to control endpoints, and prevent common web-application attacks.

A. *End-to-End Encryption*

Communication between the ESP32 control board and the Flask server supports end-to-end encryption of sensor payloads using the 'AES-128-CBC' block cipher. On the control board, sensor readings received from the sensor board via ESP-NOW are serialized to JSON, padded using 'PKCS7' to a multiple of the 16-byte block size, and encrypted with a 16-byte shared secret key and a freshly generated 16-byte random initialization vector (IV) per message. The IV is prepended to the ciphertext, and the combined byte array is Base64-encoded and URL-safe encoded before being included in the 'encrypted_data' field of a 'POST /api/sensor-data' request, along with a flag 'is_encrypted=true.

On the server, the '/api/sensor-data' handler detects encrypted payloads, cleans and decodes the Base64 string, splits out the IV and ciphertext, and uses the same 'AES-128-CBC' key to decrypt the message. PKCS7 padding is removed to recover the original JSON, which is parsed into structured sensor values; if decryption or JSON parsing fails, the server falls back to rejecting the payload or using unencrypted fields, and logs the error for diagnosis. This design ensures confidentiality and integrity of sensor readings over HTTP while remaining implementable on low-power microcontrollers.

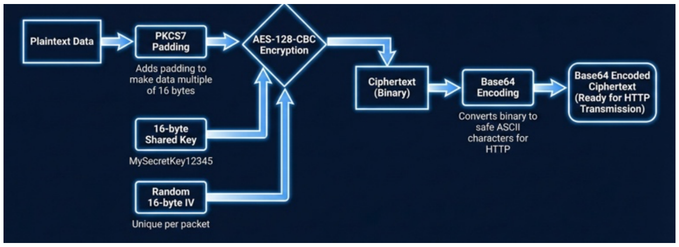


Fig. XVII. Security Implementation Flow

B. *Session Based Authentication*

All user-facing control and visualization functions are protected by session-based authentication implemented with Flask sessions and guarded routes. The '/register' endpoint creates user accounts by validating input, hashing passwords with SHA-256, and storing the hashes and roles in a 'users' table, the system enforces basic password length constraints and uniqueness of usernames. The '/login' endpoint verifies credentials by recomputing the password hash and comparing it to the stored value, and on success, populates session variables such as user identifier, username, and role, and increments an active-session counter for that user.

The HTTP requests from the browser carry a secure session cookie, and all sensitive endpoints, including the dashboard, system-state APIs (/api/system-state, /api/sensor-events), and control APIs (/api/control/light, /api/control/buzzer, /api/control/mode, /api/control/home-mode, /api/ control/ brightness), are decorated with a '@login_required' wrapper that redirects unauthenticated clients to the login page. A '/forgot-password' endpoint allows password reset by updating the stored hash for a given username, and /logout clears the session and decrements the active-session counter, preventing stale sessions from lingering indefinitely. This model provides straightforward access control while keeping the client implementation simple.

C. *Secure Backend and Database*

On the backend, the system uses parameterized SQL queries for all database operations to mitigate SQL injection attacks and enforces type-safe handling of request parameters. Timezone handling is standardized by setting the database timezone to UTC and using timezone-aware timestamps when storing sensor readings, events, and control state, which prevents inconsistencies in log analysis and event correlation. The notifications table stores alert messages with severity levels and read flags, and access to notification and history endpoints is restricted to authenticated sessions, ensuring that sensitive historical data are not exposed to unauthorized users.

Error handling in the Flask application avoids exposing stack traces or configuration details in HTTP responses, instead, exceptions are logged server-side while clients receive generic error messages and status codes. Combined with encryption of sensor data, authentication on all control and visualization endpoints, and careful database access patterns, these measures provide a robust baseline security posture appropriate for an educational but realistic smart home deployment.

VII. Experimental Results

A. *Test Setup*

The system was deployed in an indoor environment resembling a small residential setting, with the sensor board placed near a doorway and the control board connected to a main light fixture and buzzer. The Flask server and PostgreSQL database were hosted on a local machine connected to the same network as the ESP32 control board. Various test scenarios were executed, including normal occupancy, simulated intrusions, simulated fire and gas events, and variations in ambient light and noise levels.

B. *Performance Metrics*

Performance was calculated in terms of latency, sensor update rate, system uptime, and alert reliability. The sensor reading and dashboard update interval was set to approximately 2 s, ensuring near real-time visualization of environmental conditions. The end-to-end latency from sensor event (e.g., motion) to actuator activation (e.g., light, buzzer) was observed to be below one second, often around a few hundred milliseconds, combining ESP-NOW transfer, encryption, HTTP transmission, server processing, and subsequent command polling. Internal server processing times, including decryption and database insertion, were on the order of milliseconds.

C. *Observations*

The dashboard shows 'sensor behaviour and system state', and making it straightforward to verify correct operation during testing. Color-coded indicators and event logs were helpful for quickly distinguishing normal events from alerts. Per-sensor configuration was found to be valuable in reducing nuisance alarms, for example by disabling buzzer activation for certain sensors while retaining visual notifications. Mode switching between Home and Away was intuitive and allowed quick reconfiguration between convenience-oriented and security oriented operation.

VIII.Challenges Faced

A. *Implementing AES-128-CBC on ESP32*

The first challenge was aligning mbedTLS based AES-128-CBC encryption on the ESP32 with PyCryptodome decryption on the server, particularly around PKCS7 padding, IV concatenation, and Base64 encoding. This was solved by standardizing the format to "IV ciphertext", enforcing 16-byte IVs, using consistent PKCS7 padding on the microcontroller, and adding detailed logging on both sides until every encrypted payload could be decrypted and parsed reliably.

B. *Automation Modes*

Combining Home/Away and Auto/Manual modes with seven sensors produced complex behaviour, and early versions either over-reacted (too many alerts) or under-reacted (missed events). The logic was iteratively refined by encoding clear rules for each mode combination, introducing per-sensor enable/disable flags, and testing realistic scenarios until the system behaved predictably—for example, treating motion as convenience lighting in Home mode but as a security alert in Away mode.

C. *ESP-NOW Communication*

The MAC address mismatch of both ESP32's leads to communication loss, and it was rectified by correctly providing interacting ESP32 MAC address.

D. *Authentication with RESTful APIs*

At first Integrating form-based login, session cookies, and protected REST endpoints but it caused issues like API calls were redirected to HTML login pages or left unprotected, and it got rectified by '@login_required' decorator, and it consistently use session cookies for authenticated API calls, and clearly separate public authentication routes (/login, /register, /forgot-password, /logout) from private JSON endpoints.

IX. Future Work

- We planned to extend this project by implementing mobile application.
- Implement biometric authentication (fingerprint/face ID) and remote control of all dashboard features.
- Implement server-side machine learning models (using TensorFlow Lite on ESP32 or scikit-learn on server) to detect abnormal patterns in temperature, air quality, sound, or motion sequences and generate predictive alerts.
- Extend the system to support multiple sensor/control board pairs (one per room), centralized dashboard with room-wise views, and hierarchical user roles with different permission levels (e.g., guests can only view, not change modes).
- Integration with Voice Assistants and Third-Party Smart Devices, like including compatibility with Amazon Alexa, Google Assistant, and Apple HomeKit.

X. Conclusion

The IoT Smart Home Safety and Auto-Lighting System successfully demonstrates that the ESP32-based architecture can integrate multi-sensor safety monitoring, automated lighting, and security alerting with secure, encrypted communication and an intuitive web dashboard for real-time visualization and control. By combining ESP-NOW for low-latency wireless links, AES-128-CBC for protecting sensor data in transit, and a Flask PostgreSQL backend for automation logic, logging, notifications, and user authentication, the system achieves sub-second response to critical events, high alert reliability, and meaningful reductions in lighting energy usage through context-aware control and timeout management.