TANTA UNIVERSITY
FACULTY OF COMPUTERS
INFORMATION

22 JUN 2025

Tanta Univesity

Faculty of Computers and Information

Information Technology Department

“ SMART HOME WITH IOT ”

Graduation Project 2025

ACADEMIC YEAR

2024 / 2025

“Smart Home With IOT USING ESP32”

SUBMITTED BY

1 Adham Mohamed Youssef Nemr

2. Ibrahem Hereez Raslan Taima

3 Ahmed Nasr Salem Albhwar

4 Abdulrahman Adel Mostafa Deraz

5 Ahmed Mostafa Abo Almkarm Mady

6 Ahmed Abd - ElFatah

7 Saja shebl Mohamed Ibrahim

8 Shams mohamed abd el-gwad elbaz

SUPERVISOR

Dr. Hanaa Abdel Hady Eissa

ACKNOWLEDGMENT

We would like to express our sincere gratitude to our supervisor, Dr. Hanaa, for her invaluable guidance, continuous support, and encouragement throughout our graduation project journey.

We are also thankful to the Faculty of Computers and Artificial Intelligence at Tanta University for providing us with the tools and knowledge that helped bring this project to life.

Special thanks to our teammates for their efforts, cooperation, and dedication in every phase of this work.

Without the combined support of all these individuals, this project would not have been possible.

ABSTRACT

This project introduces a smart home system developed using IoT technology and the ESP32 microcontroller. The goal is to improve safety, comfort, and energy efficiency within residential environments by allowing users to monitor and control their homes remotely in real time.

The system integrates several sensors, including DHT22 (temperature and humidity), PIR (motion), MQ-2 (gas), flame, LDR (light) and reed switch. These sensors collect environmental data and send it wirelessly via Wi-Fi using the ESP32 board.

A web-based dashboard visualizes the real-time readings and allows users to respond to detected conditions, such as gas leakage, fire, motion, or unwanted motion. The interface is designed to be responsive and accessible from any device, ensuring seamless user experience.

By combining embedded systems with cloud communication, the project demonstrates a practical and low-cost approach to home automation. This solution can be scaled and customized to suit different home layouts and user needs.

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

INTRODUCTION

FIGURE 1-1

The Internet of Things (IoT) is transforming the way we live by enabling devices to communicate, collect data, and respond to real-world conditions. Smart home systems are among the most impactful applications of IoT, offering increased safety, comfort, and energy efficiency.

In this project, we designed and implemented a smart home system using an ESP32 microcontroller and multiple sensors to monitor various environmental factors such as temperature, gas levels, motion, fire, and more. The system allows users to remotely access real-time data and receive alerts through a user-friendly dashboard. This project bridges hardware and software to create an accessible and scalable IoT-based home automation solution.

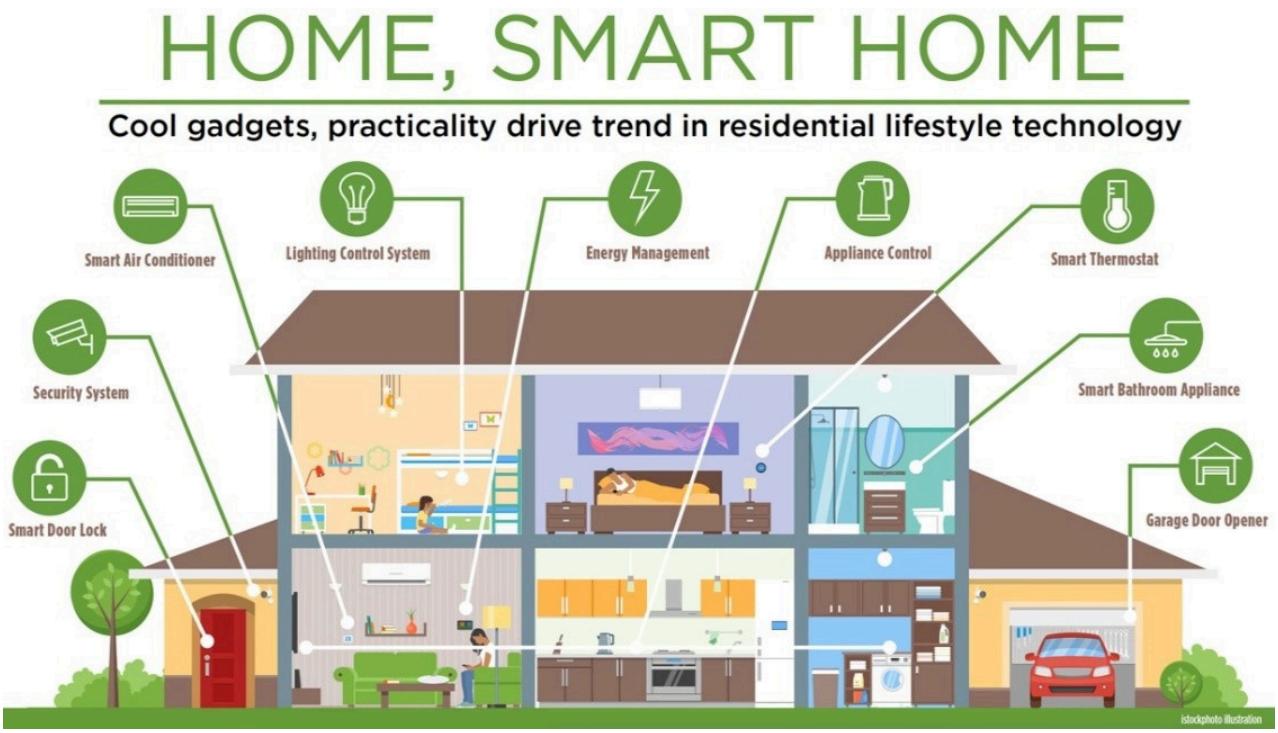


FIGURE 1-2

The smart system has a function use to help users to make decisions incorporating functions of sensing, actuation, and controls in order to describe and analyses a situation and make decisions based on the available data in a predictive or adaptive manner, thereby performing smart actions. There are different shapes of smart system residential, commercial and hospitality. Smart system has solution for all categories light, HVAC, voice control, security of building and temperature.

Through smart homes we can make different scenarios that will be fully remotely for the user for example in hotels the welcoming scenario is developed. and in the homes and the scenario of sleeping and waking up as much as choosing any scenario with one pressure.

The smart system is very important in building for the entire important thing which can get from it.

1. High safety and security
2. Power saving
3. Choosing constant lux
4. Easy to use
5. Operation efficiency
6. Comfortable and healthy

HOW DOES THE SMART BUILDING WORK?

The main concept of how a Smart Building works is simple, an input device (sensor, pushbutton) sensing a signal (could be temperature, amount of lux in a given area, a simple press on the button by the user) this signal is then transferred to the devices that have the pre-programmed instructions by the user, then an actuating signal is produced that activates a certain contacts or does a certain task in a brief and oversimplified manner that represents a closed-loop control system which is energy-efficient.

WHAT ARE THE ADVANTAGES OF SMART BUILDINGS?

1. **Comfort for occupants** due to controlling lighting, temperature, humidity, and other parameters and allowing for personalized comfort settings.
2. **Automated control** of a building's HVAC, electrical, lighting, shading, access, and security systems based on collecting and analyzing data on environmental conditions, occupant behavior, and more.
3. **Cost optimization** due to analyzing building usage patterns and making adjustments to improve a building's upkeep, optimize HVAC operation, match occupancy patterns to energy use, enhance space utilization efficiency, and more.
4. **Reduced environmental impact** due to analyzing indoor and outdoor environment conditions, occupants' behavior, and other data to optimize energy and water consumption patterns and reduce emissions.
5. **Integration capabilities** due to which there are no need to construct or move to a new building to benefit from the smart technology. Modern smart building solutions can be embedded into older structures.

CHAPTER 2

SYSTEM ANALYSIS

SYSTEM ANALYSIS

Problem Statement

In traditional home environments, there is often a lack of real-time monitoring and automated response to hazardous conditions such as gas leaks, fire, motion intrusion, or sudden changes in temperature and humidity.

This limitation not only compromises the safety and security of the residents, but also leads to inefficiencies in energy usage and inconvenience in daily life. Without smart integration, users are unable to react to environmental events promptly, which increases the risk of damage, accidents, and energy waste.

Proposed Solution

To solve these issues, this project presents a smart home system that combines various environmental sensors with an ESP32 microcontroller.

The system continuously monitors temperature, gas levels, motion, fire, and rain. Data is processed locally on the ESP32 and transmitted via Wi-Fi to a web-based dashboard. This allows users to track sensor data, receive alerts, and interact with their home in real time from any device.

The solution aims to provide a low-cost, scalable, and easy-to-use system that enhances safety, comfort, and energy efficiency through the power of IoT.

HARDWARE REQUIREMENTS

The hardware components of the Smart Home system play a critical role in collecting data from the environment and interacting with the real world. These components are connected to the ESP32 microcontroller, which acts as the brain of the system.

ESP32 microcontroller

FIGURE 2-1

The ESP32 is the central microcontroller unit of the system. It is responsible for reading all sensor data, processing it, and sending it wirelessly to the web interface. One of the key advantages of the ESP32 is its built-in Wi-Fi and Bluetooth capabilities, making it ideal for IoT applications. Its dual-core processor and large memory space allow for smooth multitasking and responsive real-time communication with external devices.

DHT22 Sensor

DHT22 pins	
1	VCC
2	DATA
3	NC
4	GND

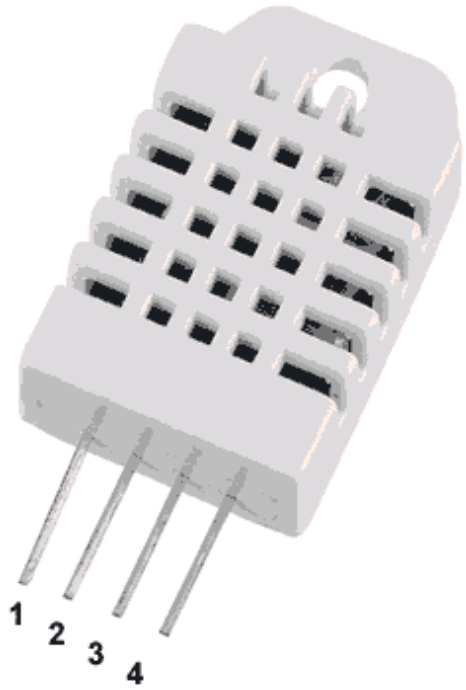


FIGURE 2-2

The DHT22 is a digital sensor used to measure both temperature and humidity with high accuracy and stability. Unlike its simpler version, the DHT11, this sensor offers a wider range and more precise readings, making it suitable for applications that require real-time environmental monitoring. In this smart home system, it helps in identifying temperature fluctuations or abnormal humidity conditions that could affect comfort or indicate potential dangers like overheating or mold formation. The data it provides is constantly updated and reflected on the dashboard for live tracking.

PIR Sensor

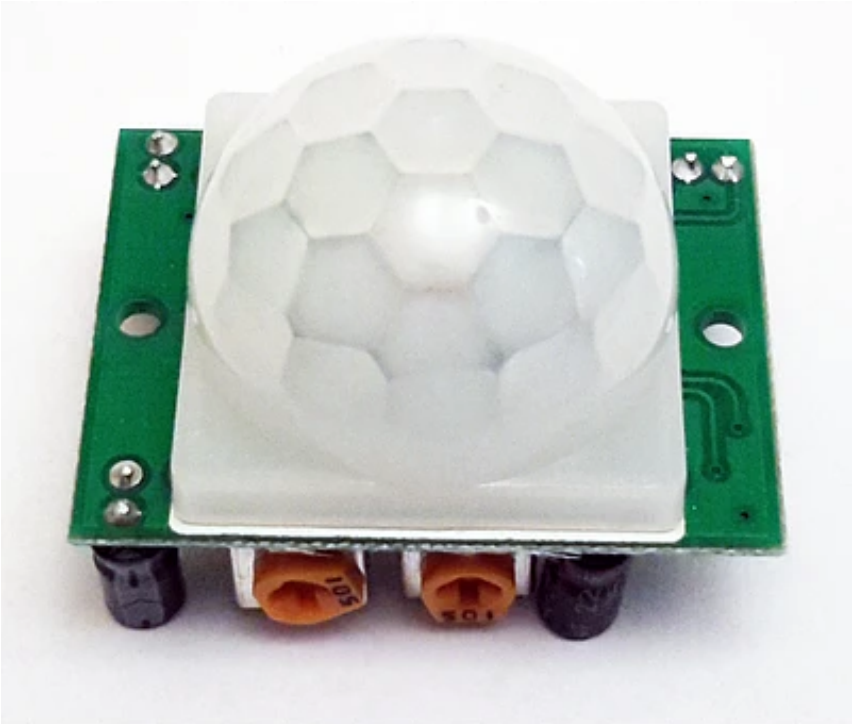


FIGURE 2-3

The Passive Infrared (PIR) sensor is designed to detect motion by measuring changes in infrared radiation from surrounding objects. It is highly effective in detecting human movement, making it ideal for security systems or motion-activated automation. In this smart home system, the PIR sensor acts as a first line of defense against unauthorized entry or unexpected presence in restricted areas. When motion is detected, the ESP32 triggers an alert, logs the event, and can also be configured to activate other actions such as lights or alarms.

MQ-2 Sensor

FIGURE 2-4

The MQ-2 sensor is a versatile module capable of detecting a range of flammable gases including methane, LPG, smoke, and hydrogen. It serves as a critical safety feature in the smart home setup. The sensor continuously monitors the air quality, and when the gas concentration exceeds a safe threshold, it sends a signal to the ESP32 to generate an alert. This allows the user to take immediate action, preventing potential accidents or fires caused by gas leaks. It is particularly useful in kitchens, storage rooms, and near gas-operated appliances.

Flame Sensor

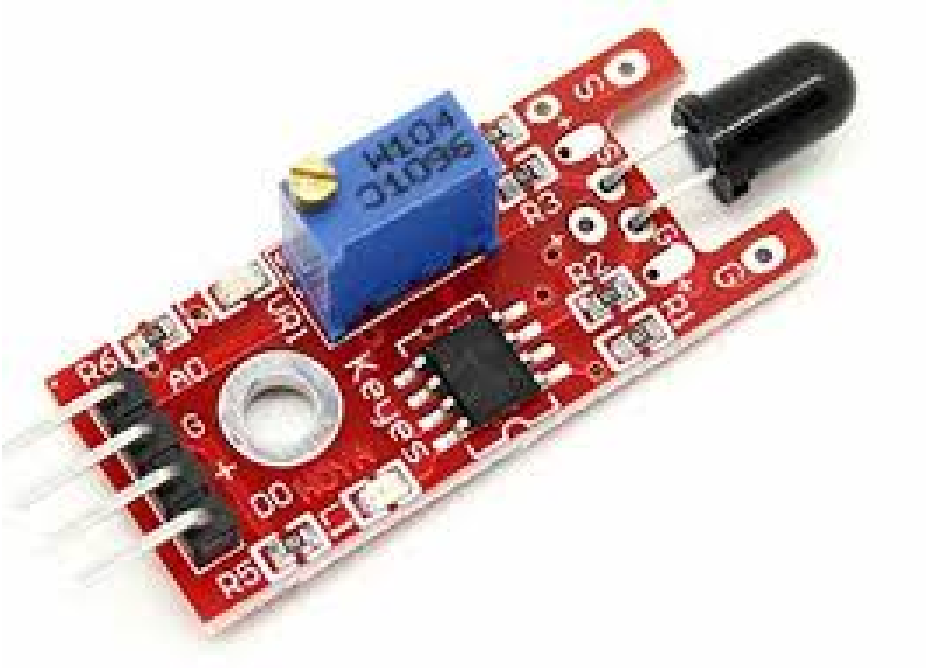


FIGURE 2-5

The flame sensor is an infrared receiver that is sensitive to light wavelengths typically emitted by fire. It adds an essential safety layer to the smart home system by detecting the presence of fire at an early stage. In the event of a flame being detected, the system can instantly notify the user via the dashboard, trigger alarms, and even send data to the cloud. This rapid response capability is crucial in minimizing fire-related damage and improving home safety.

FIGURE 2-6

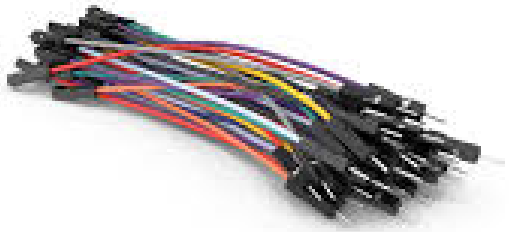
The LDR, or photoresistor, is a sensor that detects the level of light in its surroundings. It is used to automate systems based on ambient lighting conditions. For instance, when the environment becomes dark, the system can automatically switch on the lights or notify the user. The LDR contributes to energy efficiency and user comfort, as well as enhancing the intelligence of lighting systems by enabling them to adapt to real-time environmental changes.

Reed Switch Sensor

FIGURE 2-7

The magnetic door sensor consists of two parts: a switch and a magnet. When the door is closed, the magnet keeps the switch circuit intact. When the door is opened, the circuit breaks, sending a signal to the ESP32. This sensor is widely used in security systems to detect unauthorized access or to track whether doors and windows are properly closed. In this smart home system, it helps enhance home security and enables status monitoring of entry points.

Jumper Wires



Breadboard

FIGURE 2-8

These components serve as the backbone of the system's hardware integration. The breadboard allows temporary circuit construction without soldering, which is essential during the development and testing phases. Jumper wires are used to create electrical connections between components on the breadboard and the ESP32. The power supply ensures all components receive stable and reliable voltage, which is necessary for consistent performance. These basic components form the physical infrastructure that brings the smart home system to life.

SOFTWARE REQUIREMENTS

FIGURE 3-1

The software components of the smart home system provide the backbone for data processing, communication, and user interaction. These tools and platforms are used to write the code, interface with the ESP32, and build the web-based dashboard that the user interacts with. Together, they form the digital infrastructure that allows the hardware components to function as an intelligent, connected system.

ARDUINO IDE

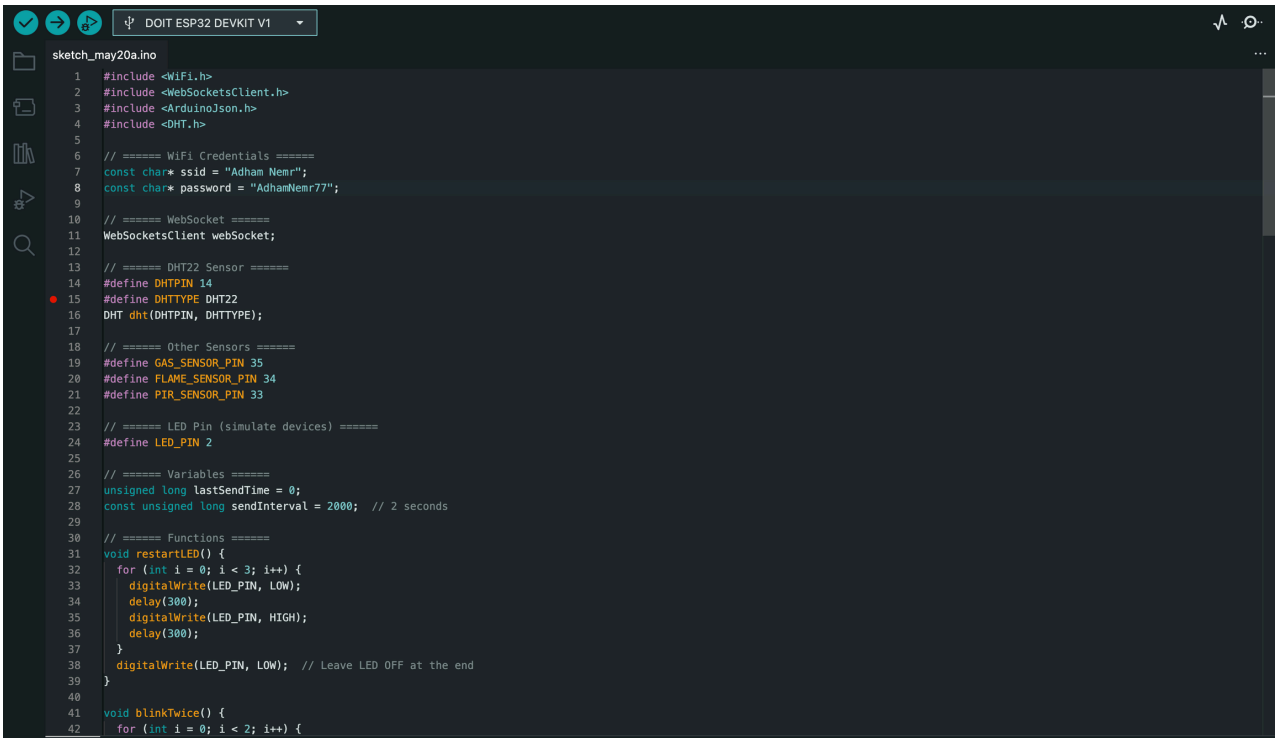


FIGURE 3-2

The Arduino IDE is the primary development environment used to write and upload code to the ESP32 microcontroller. It provides an easy-to-use interface for programming in C/C++, along with a large library ecosystem and community support. In this project, Arduino IDE was used to write the firmware responsible for initializing sensors, reading sensor data, and communicating with the dashboard through Wi-Fi. It played a critical role in testing and debugging the hardware integration process.

WEB APP

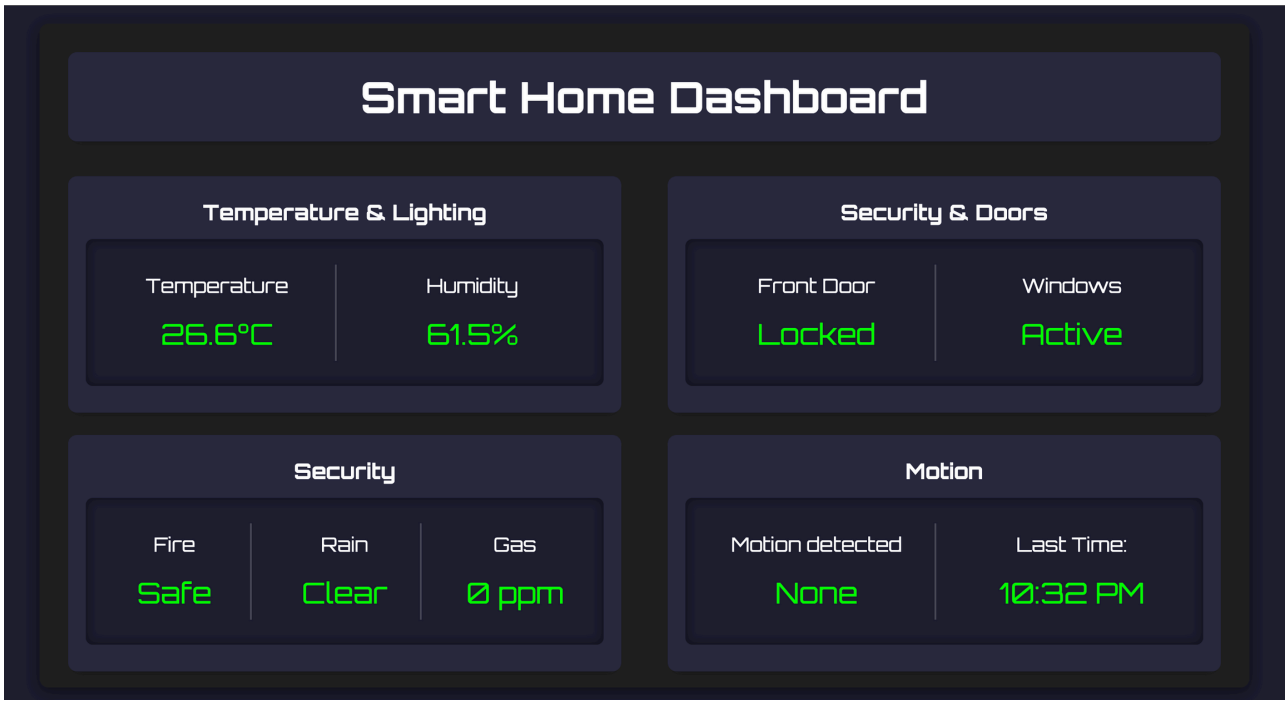


FIGURE 3-3

These are the core technologies used to create the front-end dashboard for the smart home system. HTML structures the content of the web interface, CSS provides styling and visual layout, and JavaScript adds interactivity and real-time data visualization. The dashboard allows users to monitor sensor data, receive alerts, and interact with the system remotely. Using these web technologies ensures the interface is lightweight, responsive, and accessible from any device with a browser.

WEBSOCKET

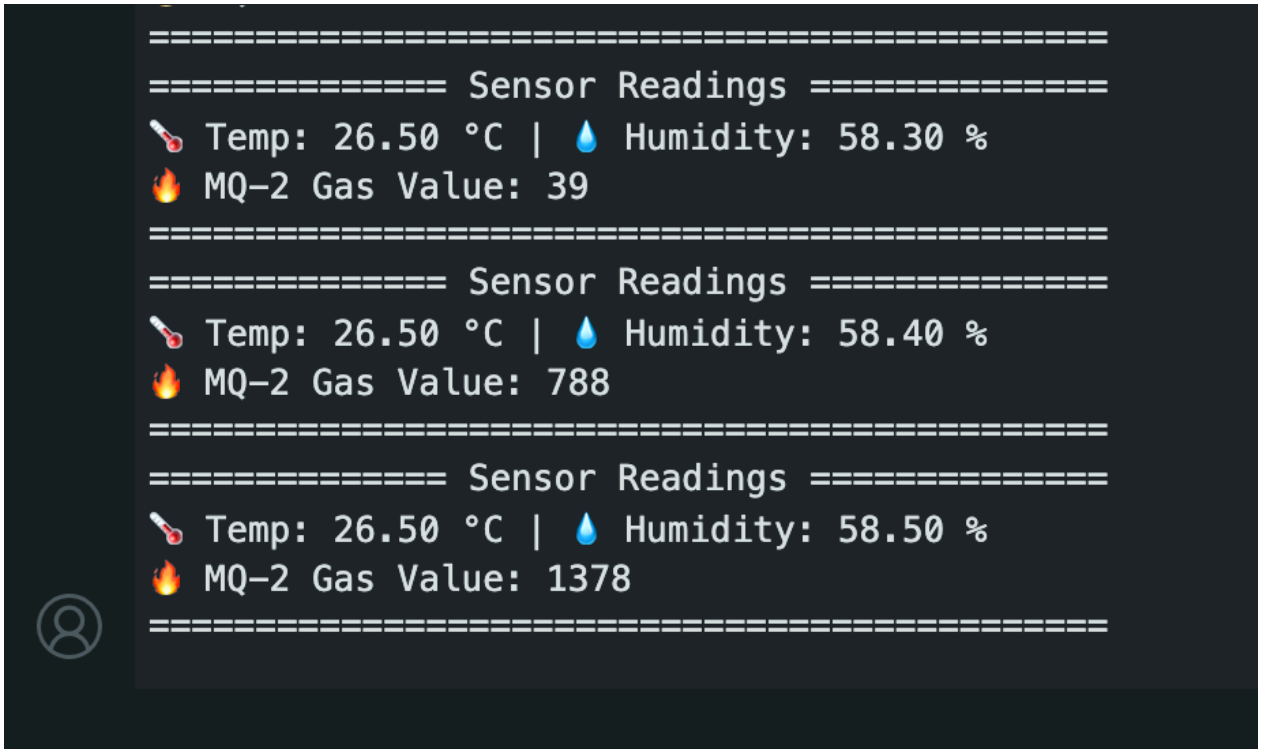


FIGURE 3-3

WebSocket and HTTP protocols are used for data communication between the ESP32 and the web dashboard. WebSocket enables real-time, two-way communication with low latency, making it ideal for instant updates from sensors. HTTP is used for simple request-response operations such as sending commands or logging data. Together, these protocols allow seamless interaction between hardware and software components, ensuring that users receive timely and accurate feedback from the system.

VISUAL STUDIO CODE

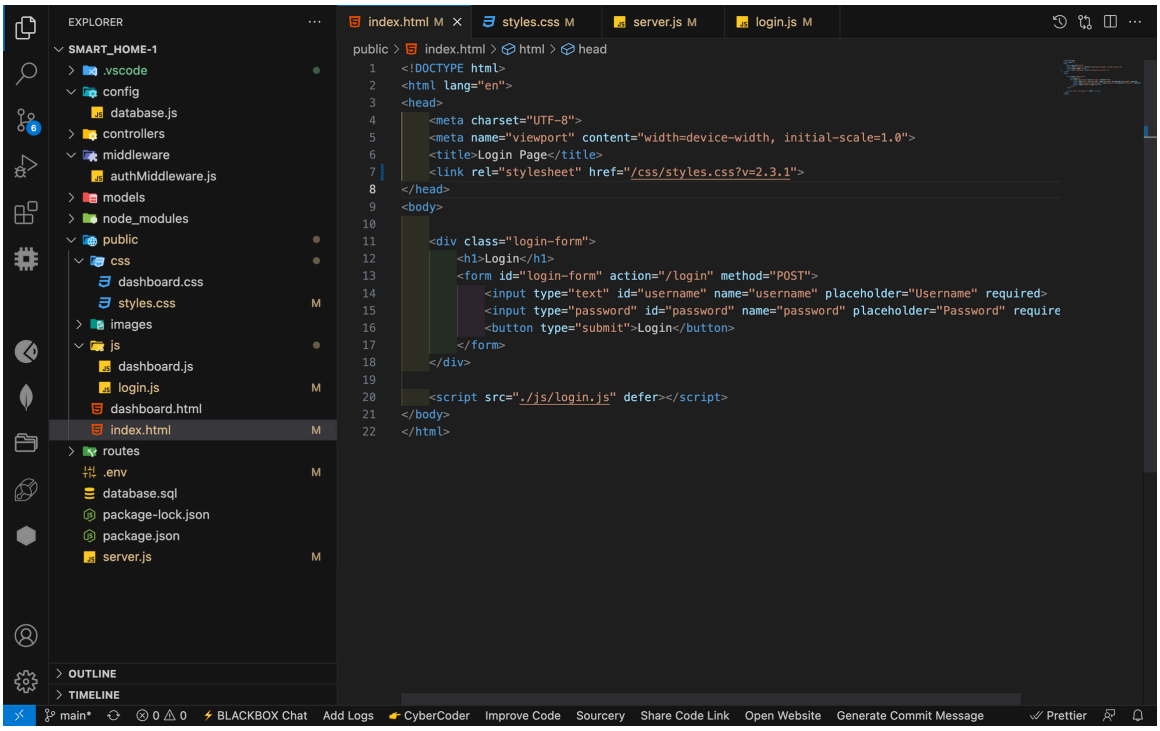


FIGURE 3-3

Visual Studio Code (VS Code) and Notepad++ are code editors used for developing the HTML/CSS/JS part of the project. These editors offer syntax highlighting, extension support, and integrated version control, which help streamline the development process. They are lightweight and highly customizable, which makes them excellent choices for both front-end and embedded development.

TARGET USERS

The Smart Home system is primarily designed for homeowners who seek to enhance the security, comfort, and energy efficiency of their living environment.

It is suitable for individuals living in urban or suburban areas who are interested in adopting modern IoT-based automation within their homes. The system is particularly beneficial for:

- Families who want real-time alerts for safety risks like gas leaks, fire, or intrusions.
- Elderly or disabled individuals who require remote control and monitoring features for convenience.
- Users who want to reduce energy consumption by automating lights and devices based on sensor data.
- Technology enthusiasts and DIY makers looking for a scalable and affordable smart home platform.

In short, this system is aimed at anyone looking for a low-cost, user-friendly, and flexible smart home solution that provides real-time feedback and control from any device.

CHAPTER 3

SYSTEM DESIGN

SYSTEM DESIGN

The system design of the Smart Home project is based on a layered architecture that connects various sensors to an ESP32 microcontroller, which communicates with a web dashboard over Wi-Fi. The system ensures real-time data collection, processing, and user interaction through a structured data flow.

This design allows scalability, reliability, and separation of responsibilities between components such as sensing, control, communication, and visualization.

01 Sensor Layer

Includes all physical sensors that collect environmental data such as temperature, gas levels, and motion.

02 Controller Layer (ESP32)

Processes the data from sensors and manages logic and communication.

03 Communication Layer

Uses the ESP32's built-in Wi-Fi to transmit data to the cloud and receive commands.

04 Dashboard Layer

A web-based user interface that visualizes sensor readings and allows control from any device.

01 Sensor Layer

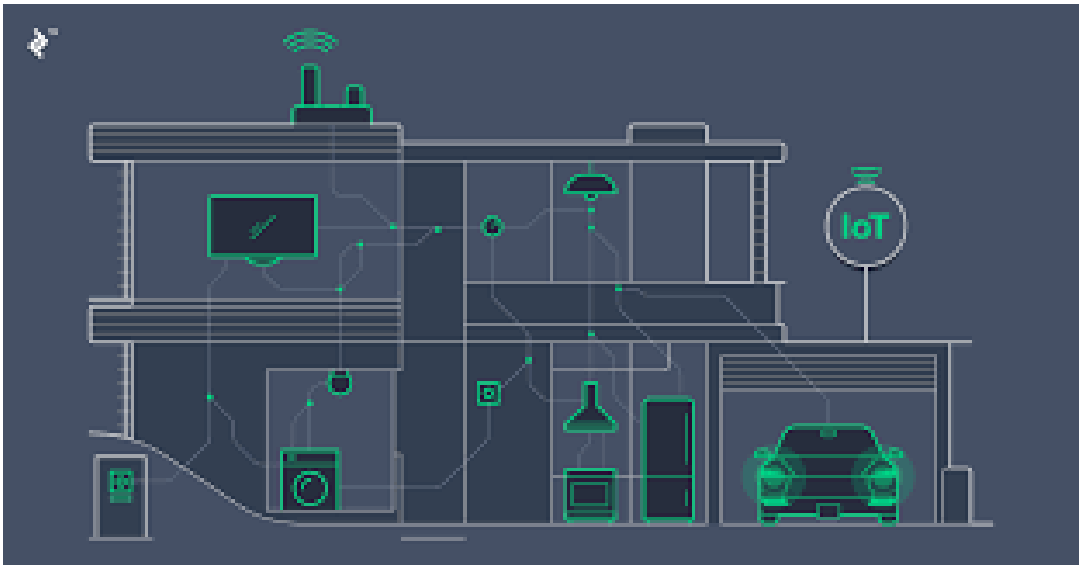


FIGURE 4-1

The sensor layer serves as the system's first point of contact with the real world. It is composed of several types of sensors, each designed to measure specific environmental factors. These include:

- DHT22, which measures both temperature and humidity, ensuring that indoor climate can be monitored and managed.
- MQ-2, used for detecting gas leaks such as smoke, methane, or LPG.
- PIR Motion Sensor, which identifies movement in the surrounding space to detect intrusions or activity.
- Flame Sensor, which detects the presence of fire or strong infrared sources as a safety precaution.
- LDR (Light-Dependent Resistor), which measures ambient light to enable automation of lighting based on brightness levels.
- Magnetic Door Sensor, which monitors the open/closed state of doors, contributing to security and awareness.

02 Controller Layer (ESP32)

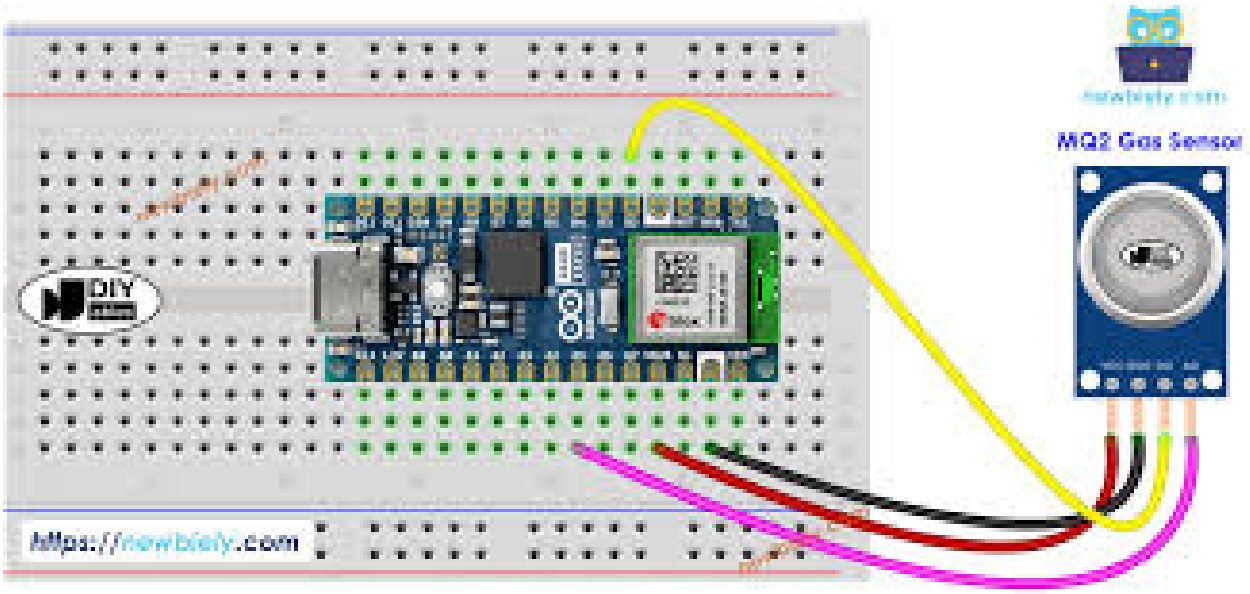


FIGURE 4-2

The controller layer is centered around the ESP32 microcontroller, which acts as the brain of the system. This module is tasked with collecting sensor data, processing it, and making logical decisions based on pre-defined conditions. For example, if the MQ-2 detects gas beyond a safe limit, the ESP32 can trigger an alert.

What makes the ESP32 particularly valuable is its built-in Wi-Fi and dual-core processor, which enables efficient multitasking. It handles real-time data collection, condition checking, and communication with the web interface simultaneously. Additionally, it controls the timing, event responses, and ensures stable system behavior.

03 Communication Layer (Wi-Fi)

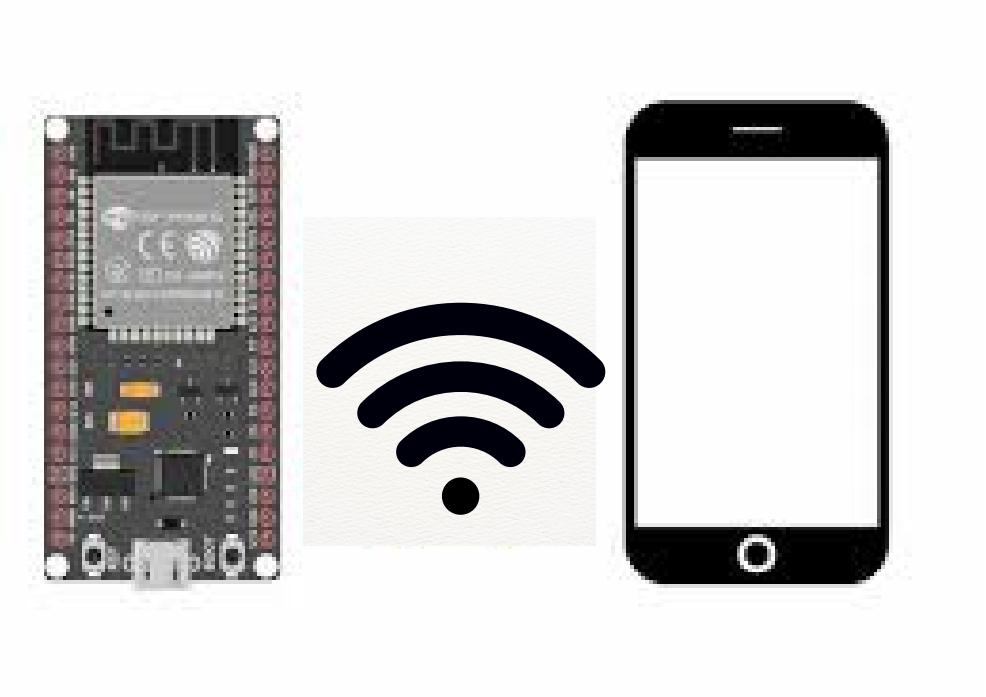


FIGURE 4-3

This layer handles the transmission of data between the ESP32 and the web-based dashboard. It operates over standard wireless protocols such as HTTP (for one-way communication) and WebSocket (for real-time, two-way updates).

Thanks to the ESP32's integrated Wi-Fi module, no external hardware is required for this purpose. The system sends sensor data to the cloud (e.g., Firebase), and also listens for control commands from the user interface. This layer ensures seamless connectivity and real-time responsiveness across devices.

04 Dashboard Layer

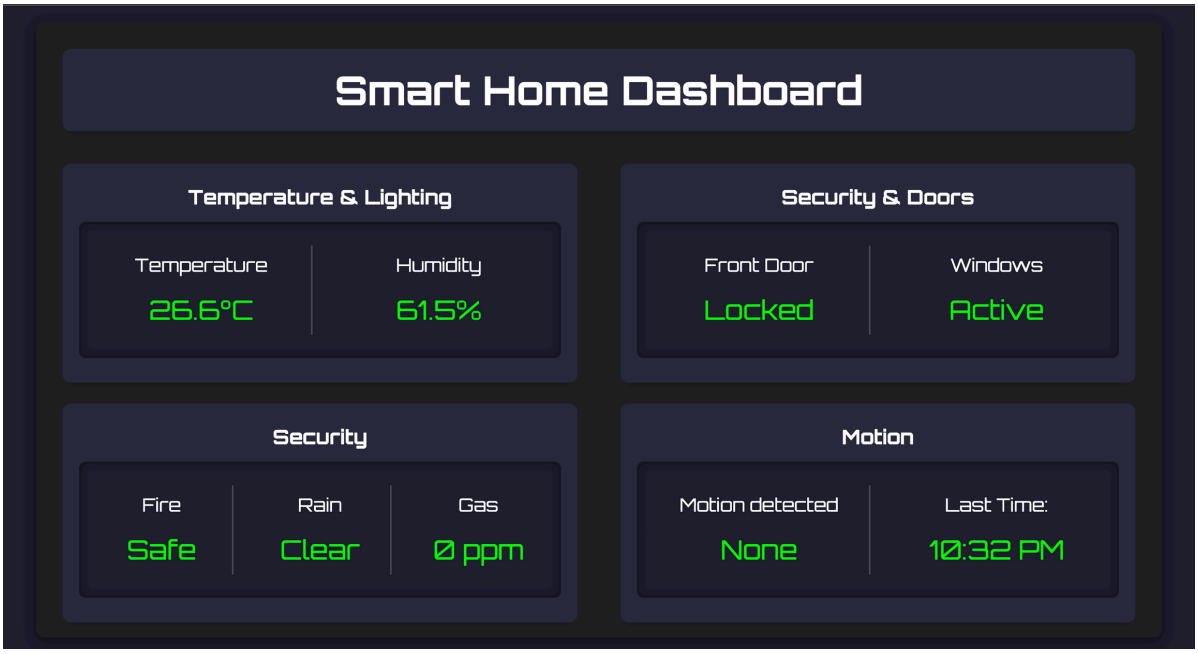


FIGURE 4-4

The dashboard layer is the main point of interaction for the user. It is developed using web technologies including HTML, CSS, and JavaScript. The dashboard provides a graphical user interface (GUI) to view live sensor data, receive alerts (such as motion detection or gas leaks), and control outputs manually.

The dashboard is designed to be responsive and accessible from any device with internet access. It enables users to remotely manage and monitor their smart home environment, whether they are at home or away. The interface is clean, intuitive, and updates in real-time thanks to WebSocket integration.

•

TELEGRAM INTEGRATION

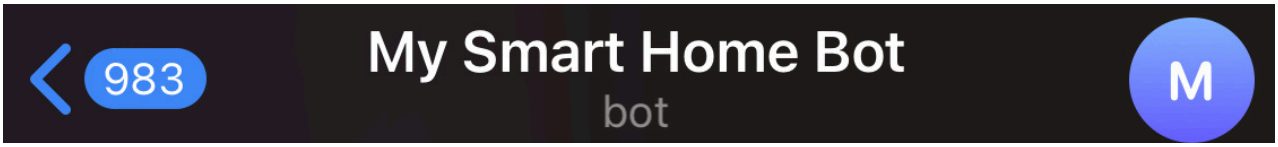


FIGURE 4-5

To enhance the communication capabilities of the Smart Home system, a Telegram Bot was integrated as an additional notification channel. This allows the system to send real-time alerts directly to the user's smartphone via the Telegram messaging platform.

Telegram Bots are automated programs that can send messages, photos, and notifications. In this project, the bot is used to deliver instant alerts in case of events such as gas leaks, fire detection, or unauthorized motion.

This integration ensures that even if the user is not logged into the web dashboard, they can still receive critical notifications immediately through their phone.

01 Telegram alerts

FIGURE 4-6

- 🚨 Sends alerts for gas leaks, motion detection, fire, etc.
- 📡 Triggered directly by the ESP32 or server when a sensor is activated
- 📲 Messages are delivered in real-time to a configured user or group
- 🔐 Uses Bot Token + Chat ID to control where messages are sent

02 Code Snippet

```
// Telegram Alert Logic
const sendTelegramAlert = async (message) => {
  const token = "**********"; // Hidden for security
  const chatId = "********";
  const url = `https://api.telegram.org/bot${token}/sendMessage`;

  try {
    await axios.post(url, {
      chat_id: chatId,
      text: message,
    });
    console.log("✅ Telegram alert sent.");
  } catch (err) {
    console.error("❌ Failed to send Telegram alert:", err.message);
  }
};
```

FIGURE 4-7

[SCREENSHOT OF TELEGRAM BOT FUNCTION]

```
// API Endpoint Handling Alerts
router.post("/api/alerts", async (req, res) => {
  const { type, value } = req.body;
  // ... validation and logic ...
  await sendTelegramAlert(message);
});
```

FIGURE 4-8

[SCREENSHOT OF BACKEND ROUTE WITH LOGIC]

This part of the system handles sending real-time alerts to the user via Telegram. When a sensor reports a dangerous value (like high gas or fire), the frontend sends the data to the server. The server checks if the value exceeds a safe threshold. If it does, a warning message is generated and sent to the Telegram bot, which instantly notifies the user.

CHAPTER 4

IMPLEMENTATION

IMPLEMENTATION

The implementation phase focused on transforming the system design into a fully working solution. Our team followed a structured development plan that covered all technical layers of the system. We divided the implementation into seven key areas: file structure, frontend development, backend logic, embedded firmware, real-time communication, user interface screenshots, and overall system summary. Each part was carefully developed, tested, and documented to ensure a reliable and scalable smart home solution.

We began by organizing the project into a clear file structure that separates frontend, backend, and firmware logic. This modular setup improved readability and made the system easier to maintain.

01 File Structure

The user interface was developed using HTML, CSS, and JavaScript to provide real-time sensor updates and alerts. It is designed to be clean, responsive, and accessible from any device.

02 Frontend Overview

The backend was built using Node.js and Express, managing data flow, routing, and alert logic. It also connects with Telegram to send notifications when sensor thresholds are exceeded.

03 Backend Logic

The ESP32 firmware was written using Arduino C/C++. It reads sensor values and sends the data to the server using WebSocket, enabling real-time monitoring of environmental conditions.

04 ESP32

WebSocket technology was used to establish a live connection between the ESP32 and the frontend. This ensures that users receive instant feedback and alerts as sensor data changes.

Visual documentation was captured during development to highlight the working dashboard, live data updates, Telegram alerts, and system wiring. These screenshots verify the successful implementation.

We concluded this phase by reviewing how each layer integrates to form a complete system. The final result is a reliable, real-time smart home solution that combines hardware, software, and cloud connectivity.

05 Real-Time Communication

06 Key Screenshots

07 Implementation Summary

FIGURE 5-1

Each of these areas was carefully implemented and tested during the development process. In the following sections, we will explore each part in detail, including its role in the system and how it was executed.

01 File Structure

The project follows a modular and well-structured file organization that separates the concerns of frontend, backend, and firmware code. This structure allows for easier development, debugging, and future scalability. Each folder in the project has a specific role, whether it's for handling API routes, serving the user interface, storing firmware code, or managing utility functions like Telegram alerts.

```
smartHome/
├── backend/              # Contains server code, routes, models, and utilities
│   ├── server.js         # Main Express server file (with WebSocket integration)
│   ├── routes/           # All REST API routes (users, devices, logs, etc.)
│   │   ├── user.routes.js
│   │   ├── device.routes.js
│   │   ├── sensorsData.routes.js
│   │   ├── room.routes.js
│   │   ├── log.routes.js
│   │   ├── command.routes.js
│   │   └── telegram.routes.js
│   ├── models/           # Sequelize models for the database
│   │   ├── user.model.js
│   │   ├── sensorLog.model.js
│   ├── config/
│   │   └── database.js   # Database configuration file
│   └── utils/
│       └── telegram.js   # Utility function to send Telegram alerts
├── public/               # Frontend static files (HTML, JS, CSS)
│   ├── index.html
│   ├── dashboard.html
│   ├── style.css
│   └── app.js
├── firmware/             # Arduino code for ESP32
│   └── esp32_code.ino
├── .env                  # Environment variables (e.g., Bot Token, DB URL)
├── README.md             # Project instructions and documentation
└── package.json          # Node.js dependencies and project metadata
```

FIGURE 5-2

This structure clearly separates frontend logic (public/), backend logic (backend/), and firmware (firmware/).
Each route, model, and configuration file is placed in its proper context,

02 Frontend Overview

The frontend of the system was designed with simplicity and clarity in mind. It focuses on providing a visual interface for the user to view sensor data through structured elements like cards, headings, and status indicators. The layout is responsive and adapts well.
Basic UI components were implemented using HTML and styled using CSS to give each sensor its own card with distinguishable colors and icons. The JavaScript layer handles the rendering of values inside these elements but avoids heavy logic, leaving the processing for the backend.

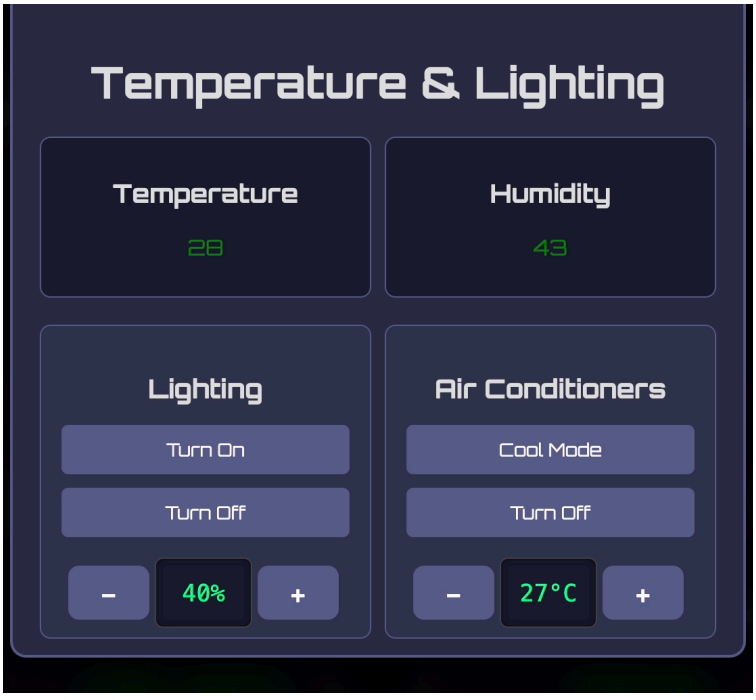


FIGURE 5-3

The goal of the frontend was to offer a clean and distraction-free user interface. By focusing only on display and layout, we ensured separation of concerns between logic and presentation.

03 Backend Logic

The backend of the system was built using Node.js with the Express framework. Its main responsibility is to handle HTTP requests, process incoming sensor data, manage user authentication, and organize API endpoints that connect the system's different components. Instead of handling logic in the frontend or ESP32, most of the processing was centralized in the backend to ensure consistency, security, and scalability.

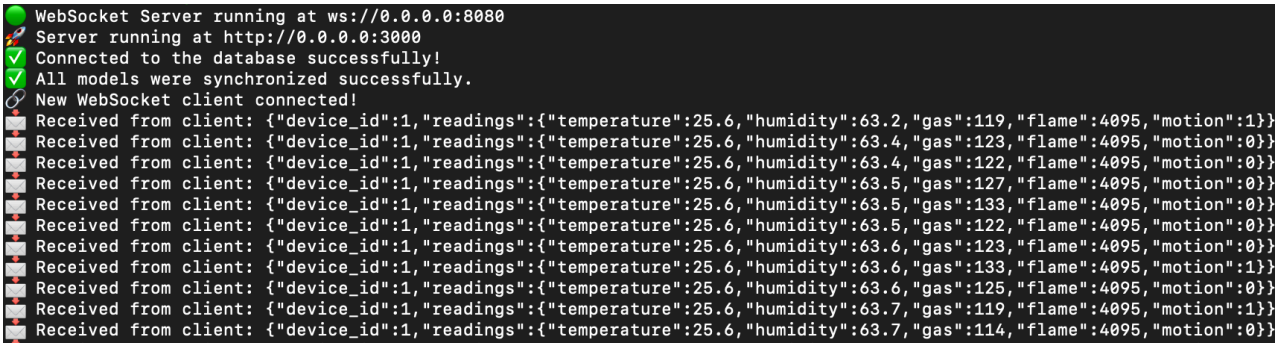


FIGURE 5-4

This terminal output confirms that data was successfully received from the ESP32 microcontroller. The server processed the incoming sensor readings, saved them to the database, and triggered alerts when certain thresholds were exceeded. This demonstrates the real-time backend functionality of the system.

04 ESP32 Firmware

The firmware running on the ESP32 microcontroller was developed using the Arduino IDE. It handles sensor initialization, periodic reading of sensor values, and sending the data in JSON format over WebSocket to the backend server. The firmware is optimized to ensure accurate and fast communication with minimal delay, forming the core bridge between the physical environment and the digital system.

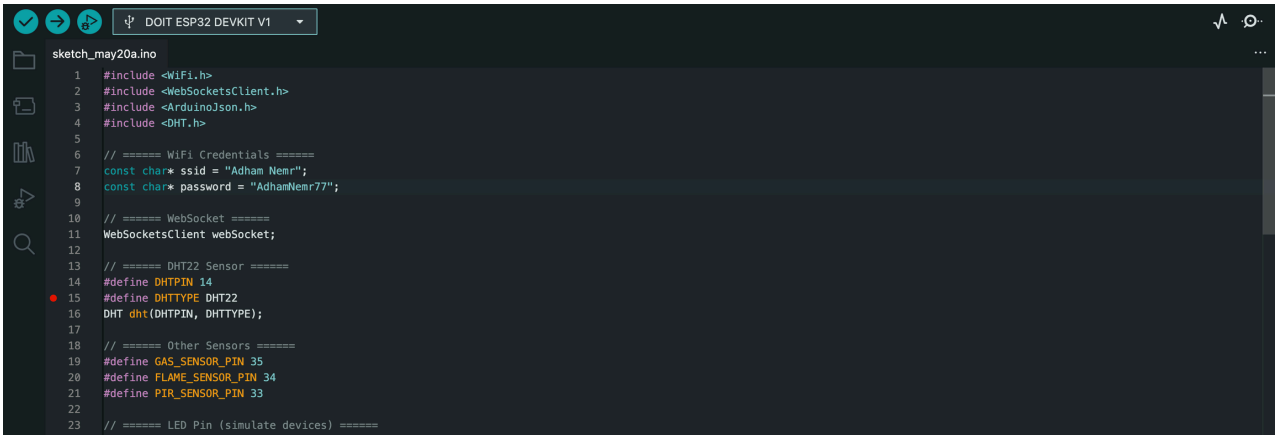


FIGURE 5-5

The firmware plays a critical role in collecting real-world data and making it digitally accessible. By using WebSocket, the ESP32 can transmit live sensor values to the backend, enabling real-time processing and feedback across the system.

05 Real-Time Communication

Real-time communication is one of the most critical features of the Smart Home system. To achieve immediate data transfer between the ESP32 and the web dashboard, the system uses WebSocket — a full-duplex communication protocol that allows both the server and the client to send and receive data instantly without refreshing the page or making repeated requests.

Overview

Real-time communication is a key part of the Smart Home system. It allows sensor data collected by the ESP32 to be sent instantly to the frontend dashboard without delays. This was achieved using WebSocket, a protocol that enables two-way, persistent communication between the microcontroller and the web interface.

Why WebSocket Instead of HTTP?

- HTTP is request-response, meaning the dashboard must ask for updates.
- WebSocket is always connected, so the ESP32 can send data immediately when it changes.
- This improves speed, efficiency, and creates a better user experience.

How It Works:

- ESP32 reads sensor values (gas, flame, temperature, etc.)
- It creates a JSON object from the readings
- It sends the data using WebSocket to the backend
- The backend pushes it to the frontend in real time
- The dashboard updates the cards instantly

Example from ESP32 Code

```
StaticJsonDocument<200> doc;
doc["gas"] = gasValue;
doc["temperature"] = tempValue;

String output;
serializeJson(doc, output);
webSocket.sendTXT(output);
```

FIGURE 5-6

This snippet shows how the ESP32 formats sensor readings into a JSON string and sends them over WebSocket to the server for real-time processing.

Example from Frontend (JavaScript)

```
socket.onmessage = function (event) {
  const data = JSON.parse(event.data);
  updateCard("gas", data.gas);
  updateCard("temperature", data.temperature);
};
```

FIGURE 5-7

This JavaScript code receives real-time data through WebSocket and updates the dashboard interface dynamically without the need to refresh the page.

Conclusion

The implementation of real-time communication using WebSocket played a critical role in achieving the responsiveness and reliability required for a smart home system. Unlike traditional request-response methods, WebSocket allowed the system to maintain a persistent connection between the ESP32 and the frontend dashboard. This made it possible to push sensor updates instantly, ensuring users receive immediate feedback about important changes such as gas leaks or fire detection. The seamless data flow between hardware and software layers added a strong sense of interactivity and modern usability to the overall solution.

06 Key Screenshots

This section presents key screenshots taken during the development and testing stages to demonstrate that the system functions correctly. These visuals help validate the real-time behavior of the Smart Home system, the integration between components, and the effectiveness of alerts and data display.

Live Dashboard Data View

Real-time sensor readings are displayed on the dashboard, showing live updates as data is received from the ESP32.

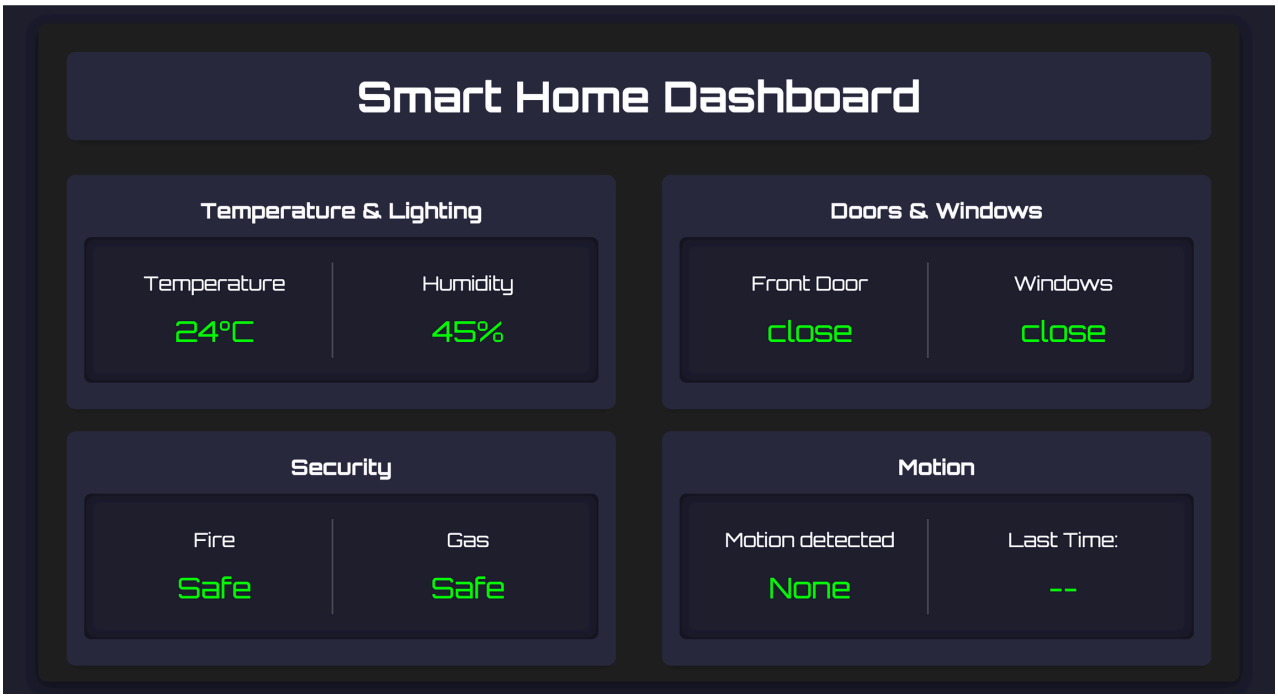


FIGURE 5-8

Sensor Alert on Dashboard

FIGURE 5-9

The interface highlights sensor alerts visually, changing the card's color (e.g., red) when a dangerous value is detected.

ESP32 Serial Output

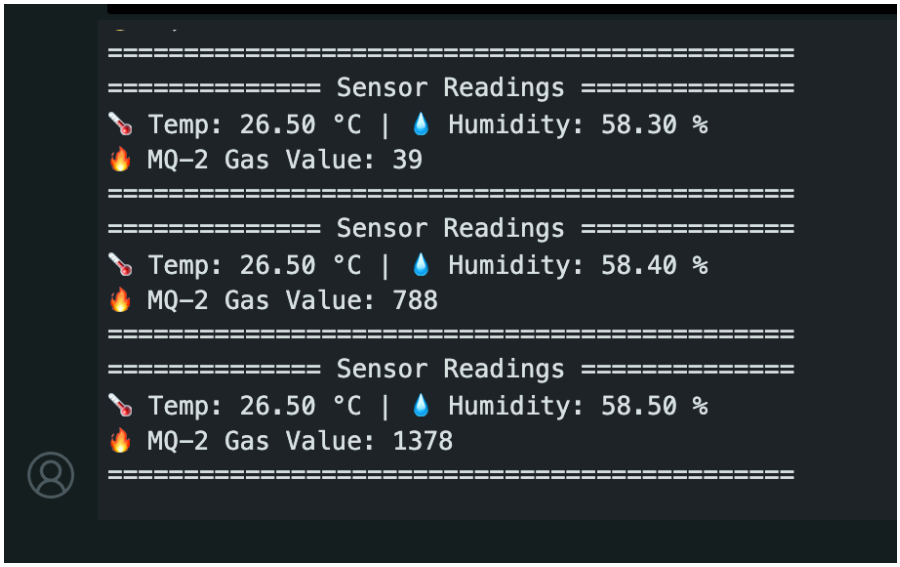


FIGURE 5-10

The ESP32's serial monitor logs sensor readings in real time before transmitting them to the backend.

Server Terminal Output

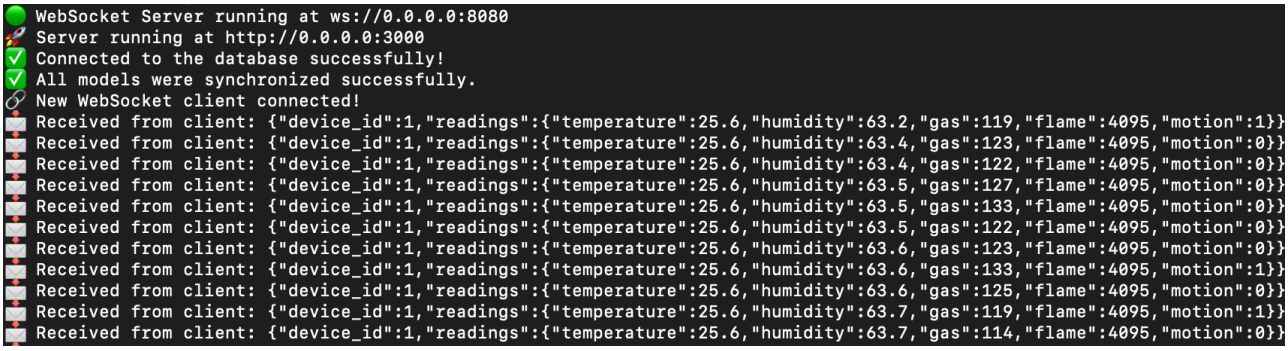


FIGURE 5-11

The ESP32's serial monitor logs sensor readings in real time before transmitting them to the backend.

Closing Note

These screenshots confirm that each part of the system—hardware, backend, and user interface—was successfully integrated and tested. They demonstrate that the Smart Home system is capable of detecting critical situations and responding immediately through visual and messaging alerts.

07 Implementation Summary

The implementation of the Smart Home system brought together multiple technologies and layers to create a fully functional, real-time monitoring platform. Starting with a well-organized file structure, each part of the system was developed to serve a clear purpose—frontend for visualization, backend for logic and data handling, ESP32 for sensing, and WebSocket for live communication.

Through testing and integration, the system proved capable of collecting real-time sensor data, displaying it clearly, triggering alerts both visually and remotely, and responding to dangerous conditions like gas leaks or fire. The use of Telegram further enhanced the system's responsiveness by delivering instant alerts directly to the user's phone.

Overall, the project successfully combined hardware and software to deliver a reliable, scalable, and practical smart home solution.

Conclusion

This implementation proves that even with low-cost components and open-source tools, it is possible to build an intelligent, real-time smart home system that adds real value to everyday safety and automation.

CHAPTER 5

TESTING

TESTING

This section documents how the Smart Home system was tested in various real-world scenarios. Each sensor was triggered under controlled conditions to verify that it correctly detects changes, sends data, triggers backend logic, and updates the user interface or Telegram notifications accordingly.

Test Table Format:

ID	Sensor	Input Condition	Expected Result
1	Gas Sensor	Gas value = 420	Telegram alert + turns red
2	Flame Sensor	Value = 2800	Telegram alert + turns red
3	Temperature	Value = 30	Turns red + Open AC
4	Door Sensor	Door Opend	UI updated to "Open"
5	Light Sensor	Light level = 850	log created

The table above summarizes the main test cases performed on the system. Each case was then verified individually to ensure proper response from the hardware, backend logic, and user interface.

1 - Gas Sensor Alert

The gas sensor was manually triggered by exposing it to a lighter's gas or a small leak. Once the value exceeded 400 (e.g., reached 420), the ESP32 immediately sent the reading to the backend via WebSocket.
The backend checked the value and recognized it as dangerous, triggering the Telegram bot. A message was sent:
"⚠️ Gas level too high!"
Simultaneously, the gas card on the dashboard turned red with a warning.
This confirmed the alert system works in both visual and remote forms.

2 - Flame Sensor Alert

To simulate a fire condition, a flame (like a lighter) was placed near the flame sensor. The reading dropped below 3000, which triggered the backend alert logic.
A Telegram message was sent:
"🔥 Fire detected!"
The flame card on the dashboard changed appearance to indicate danger. This test confirmed the system's fast response to fire risk.

3 - Normal Temperature Reading

The temperature sensor was allowed to operate normally in room temperature (e.g., 28–30°C).
The reading was below the set threshold (30°C), so no alert was triggered.
The value appeared normally on the dashboard and was logged to the database.
This proved that the system ignores non-critical readings while still displaying them live.

4 - Door Sensor Activity

The door sensor was tested by opening and closing a physical door or sliding the magnetic components apart.
When the door was opened, the dashboard instantly updated the status to “Open”, and when closed, it changed to “Locked”.
No Telegram alert was needed for this sensor, but the test confirmed the real-time UI update was functional.

5: Light Sensor (LDR) Reading

A flashlight or hand was used to adjust light exposure.
When the light level passed the threshold (e.g., >800), the backend recognized it and stored the value.
The card on the dashboard updated instantly.
No alert was required, but this proved data was received and recorded as expected.

BE SURE

All test cases were repeated more than once to ensure stability. The system responded correctly in each case, proving that both the logic and hardware are working reliably under different conditions.

CHAPTER 6

FUTURE WORK

CONCLUSION & FUTURE WORK

1 - What was achieve

The Smart Home system developed in this project aimed to create an efficient and accessible platform for monitoring and responding to environmental conditions in real time. By combining microcontroller technology with web development and cloud services, we built a fully functional solution capable of handling real-world scenarios such as gas leaks, fire detection, motion sensing, and more.

2 – Technical success

The system successfully integrated multiple components including the ESP32 microcontroller, a set of environmental sensors, a Node.js backend, a responsive web-based frontend, and Telegram Bot API for instant alerts. All modules communicated smoothly using WebSocket, and the system delivered data and notifications in real time with minimal latency.

3 – Reliability and usability

During testing, the system demonstrated reliable performance, responsive alerts, and consistent data handling. The dashboard was intuitive and responsive, allowing users to interact with the system easily. These outcomes confirmed that the system can be deployed in actual smart home environments and expanded in future updates.

Future Work

1. SMS Alert System Using Twilio

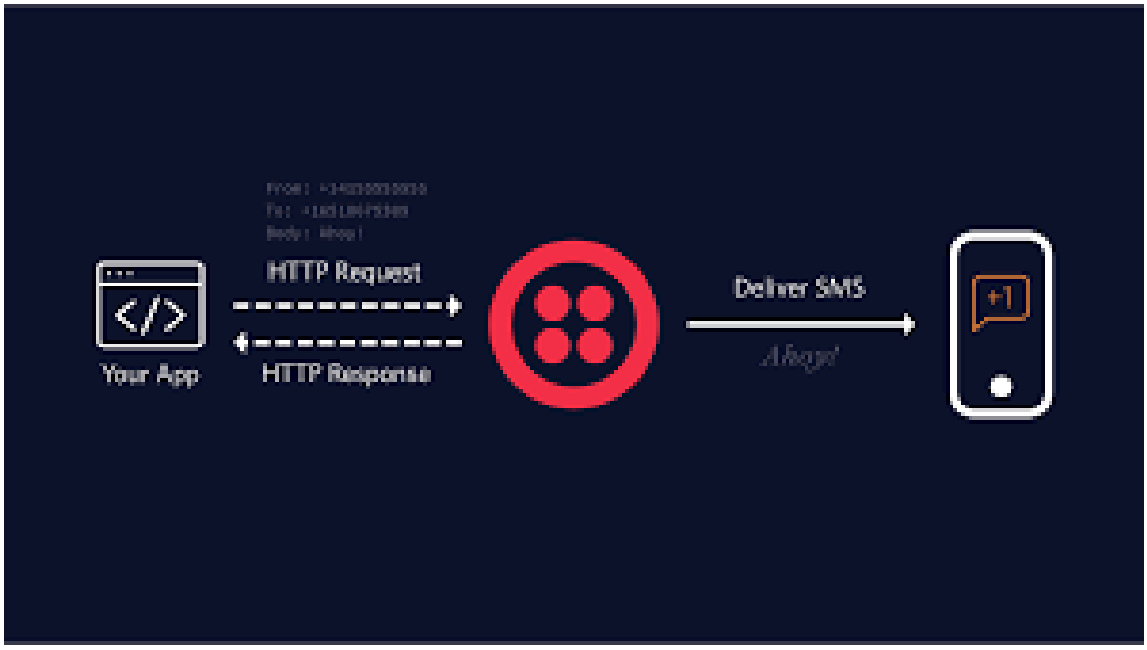


FIGURE 6-1

While the current system relies on Telegram for remote alerts, a future upgrade could include SMS notifications through services like Twilio. This would ensure that critical alerts such as gas leaks or fire are delivered directly to the user's mobile phone, even in the absence of internet connectivity. The backend can be extended with a simple API call to Twilio whenever a dangerous sensor reading is detected, making the alert system more robust and versatile.

2. Voice Alerts via Buzzer

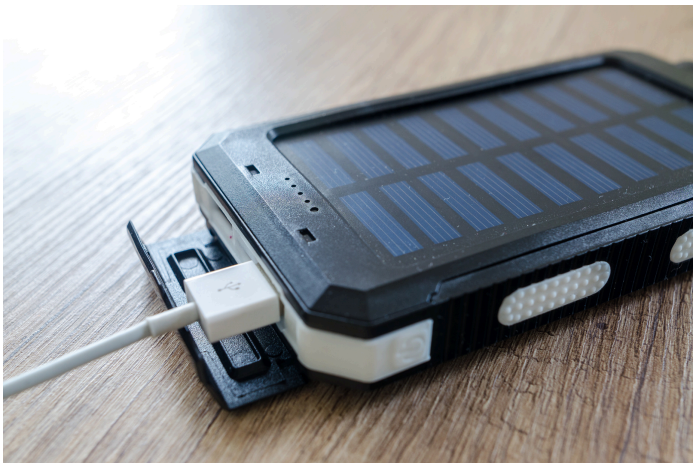
Adding audio feedback through a buzzer or a speaker module such as DFPlayer Mini could enhance the system's in-house warning capabilities. When a fire or gas leak is detected, the ESP32 could play a pre-recorded warning message or activate a loud buzzer. This would make the system more effective for users who are at home but may not be looking at the dashboard or receiving phone notifications.

3 - Mobile App for Real-Time Monitoring

To provide users with a smoother experience, a dedicated mobile application could be developed using Flutter or React Native. The app would replicate the dashboard functionality, allow remote control, and push notifications directly to the user's phone. This would eliminate the need to open the system through a browser and bring the smart home experience closer to commercial-grade IoT systems.

4 - Offline Power Backup (UPS Integration)

One practical improvement would be to add an uninterruptible power supply (UPS) system or battery backup to the ESP32 and sensors. In the case of a power outage, the system could continue operating for a limited time, or at least send a final alert to the user before shutdown. This feature is especially useful in critical environments where sudden power loss could hide dangerous situations.



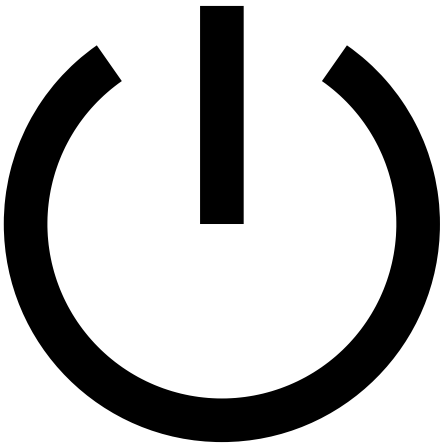


FIGURE 6-2

5 - Control of Physical Devices Using Relay

Moving from just monitoring to active control, the system can be expanded to trigger real-world actions. For example, when gas is detected, the ESP32 can use a relay module to automatically shut off the gas supply or activate a ventilation fan. Similarly, fire detection could trigger sprinklers or emergency lights. This level of automation significantly increases the value and practicality of the system.

6 - AI-Based Automation and Behavior Prediction

Integrating AI and machine learning into the system could allow it to learn user behavior over time. For instance, it could identify patterns in temperature, motion, or light usage, and suggest or automatically apply optimizations. Abnormal patterns could also be flagged as potential threats (e.g., motion when the house is empty), adding another layer of intelligence to the home.

7 - Data Logging and Historical Analysis

A valuable enhancement would be to implement a data logging feature that stores historical sensor data in Firebase or another cloud database. This data could then be visualized through charts on a separate “Reports” page, helping users understand long-term trends in temperature, gas levels, or system usage. This would not only improve user awareness but also aid in diagnostics and energy optimization.

8 - Integration with Smart Assistants (Google/Alexa)

Another advanced feature would be to integrate the system with smart voice assistants like Google Home or Amazon Alexa. Through services like IFTTT or MQTT, users could control their smart home setup using voice commands. For example, they could ask “Is the gas sensor reading safe?” or say “Activate night mode,” and the system would respond accordingly.

Conclusion

These proposed enhancements are technically feasible and would further improve the system’s usability, intelligence, and responsiveness. They can be implemented gradually to transform the current project into a fully-featured smart home platform.

CHAPTER 7

COMPARISON SMART HOME AND COMPETING SOLUTIONS

Comparison Smart Home and Competing Solutions

To better understand the strengths and limitations of our Smart Home system, we compared it with existing competing solutions in the smart home market. The comparison focuses on key features including cost, customization, alert systems, data handling, and scalability. The chosen competitors for this comparison are well-known commercial solutions such as Google Nest and Ring, as well as open-source platforms like Home Assistant.

Cost

Our Smart Home project is designed to be highly cost-effective. By using affordable components such as the ESP32 microcontroller and readily available sensors, we built a system that delivers core smart home functionality at a fraction of the price of commercial products. In contrast, commercial solutions like Google Nest or Ring involve significant upfront hardware costs, and often require ongoing subscription fees for premium features like video storage or extended alerts. Open-source platforms like Home Assistant may fall somewhere in between, as their cost depends heavily on the chosen hardware (e.g., Raspberry Pi, Zigbee hubs).

Customizability

A major strength of our project is full hardware and software customizability. Every part of the system — from firmware to frontend interface — can be modified to meet specific user needs. This level of control is often absent in commercial systems, where users are limited by vendor-defined features and locked ecosystems. Open-source platforms like Home Assistant also offer high customizability but often come with a steeper learning curve and complex setup requirements compared to our streamlined solution.

Alert System

Our system features a flexible alerting mechanism including Telegram notifications, on-dashboard warnings, and a roadmap for SMS integration. This ensures that users receive alerts both locally and remotely, tailored to their preferences. By comparison, commercial systems provide notifications mainly through proprietary apps, with limited options for customization. Open-source platforms like Home Assistant can match or exceed this flexibility but typically require manual configuration of services like MQTT, Node-RED, or third-party APIs.

Real-time Updates

The Smart Home project uses WebSocket technology to push real-time sensor data directly to the user dashboard, ensuring immediate feedback without needing to refresh or poll the server. This creates a seamless and interactive experience. In contrast, many commercial solutions rely on periodic cloud sync, which can introduce slight delays. Open-source solutions like Home Assistant can achieve real-time updates as well, typically through MQTT or WebSocket configurations, but again at the cost of additional setup effort.

Local Processing

Our system emphasizes local data processing on both the ESP32 and the backend server. This design keeps user data private and ensures functionality even without internet connectivity. Commercial solutions usually process and store data in vendor clouds, raising concerns over privacy and requiring constant internet access. Open-source platforms give users the choice, allowing both local and cloud processing, but typically need manual setup to achieve this.

Offline Operation

Our system is designed to operate fully on a local network (LAN) without requiring internet access. Users can continue to monitor their environment and receive alerts locally even if the internet is down. In contrast, commercial smart home systems often depend on cloud services for core functionality, making them vulnerable to connectivity issues. Open-source solutions can function offline but require more advanced configuration to set up LAN-based communication.

AI / Learning

While AI and behavior prediction are not yet part of the current implementation, they are included in our future development plan. This sets a clear roadmap for enhancing automation and intelligence. Commercial products like Google Nest already offer built-in AI features, such as adaptive temperature control. Open-source systems provide frameworks for adding AI manually through plugins or integrations with external AI services.

User Data Ownership

One of the key differentiators of our project is complete user data ownership. All data is processed and stored locally without reliance on third-party servers. This contrasts sharply with commercial solutions where user data is often stored in the vendor's cloud, potentially accessible by the provider. Open-source platforms also offer user-controlled data storage but typically require more technical knowledge to configure local-only setups.

Conclusion of the Comparison

In summary, our Smart Home project offers a highly customizable, cost-effective, and privacy-conscious alternative to commercial products, while delivering core smart home functionality. It also provides a simpler setup compared to open-source platforms, making it suitable for both technical and non-technical users. Planned future enhancements like AI integration and mobile apps will further strengthen its position as a competitive smart home solution.

CHAPTER 8

REFERENCES

References

[1] Espressif Systems, "ESP32 Technical Reference Manual", [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/. [Accessed: 2024].

[2] Arduino, "Getting Started with Arduino", [Online]. Available: https://www.arduino.cc/en/Guide. [Accessed: 2024].

[3] Firebase, "Firebase Realtime Database Documentation", [Online]. Available: https://firebase.google.com/docs/database. [Accessed: 2024].

[4] Mozilla Developer Network (MDN), "WebSocket API", [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API. [Accessed: 2024].

[5] Telegram, "Telegram Bot API", [Online]. Available: https://core.telegram.org/bots/api. [Accessed: 2024].

[6] Express.js, "Express Web Framework", [Online]. Available: https://expressjs.com/. [Accessed: 2024].

[7] Chart.js, "Chart.js Documentation", [Online]. Available: https://www.chartjs.org/docs/latest/. [Accessed: 2024].

[8] Techiesms YouTube Channel, "ESP32 IoT Tutorials", [Online]. Available: https://www.youtube.com/c/Techiesms. [Accessed: 2024].

[9] ProgrammingKnowledge, "ESP32 + Arduino Tutorials", [Online]. Available: https://www.youtube.com/c/ProgrammingKnowledge. [Accessed: 2024].

[10] Stack Overflow, n.d., "Various technical discussions", [Online]. Available: https://stackoverflow.com. [Accessed: 2024].

[11] Twilio, "Programmable SMS API Documentation", [Online]. Available: https://www.twilio.com/docs/sms. [Accessed: 2024].

[12] Flutter, "Flutter Official Documentation", [Online]. Available: https://flutter.dev/. [Accessed: 2024].

[13] Fritzing, "Fritzing Circuit Design Tool", [Online]. Available: https://fritzing.org/. [Accessed: 2024].

[14] GitHub, "Smart Home Project Source Code", [Private Repository]. Link available upon request.

[15] Coursera, "Internet of Things Specialization", [Online]. Available: https://www.coursera.org/specializations/iot. [Accessed: 2024].

Special Thanks

We would like to express our sincere gratitude and appreciation to the graduation project evaluation committee for their valuable time and effort in reviewing our work.

A special thank you goes to Dr. Hanaa Eissa, our respected supervisor, for her continuous guidance, support, and encouragement throughout the entire project. Her insights and feedback were instrumental to our progress.

We also extend our thanks and appreciation to Prof. Dr. Nancy El-Hefnawy, Dean of the Faculty of Computers and Information – Tanta University, for providing a supportive academic environment that enabled us to learn, innovate, and grow.

Thank you for your time, your knowledge, and your trust.