An-Najah National University

Faculty of Engineering and Information Technology

Department of Computer Engineering

Graduation Project II

Smart Home Automation System Using ESP and IoT

SmartNest

Supervisor: Dr. Hanal Abu-Zant

Presented By: Saba Majdi Shoqo

Submitted in partial fulfillment of the requirements for a bachelor's degree in Computer Engineering

September 24, 2025

Dedication

This project is dedicated to our beloved families, whose unwavering support, encouragement, and love have been the foundation of our success. To our mentors, whose guidance and wisdom have shaped our journey and inspired us to strive for excellence.

We also dedicate this work to our friends and colleagues, who stood by us through every challenge and celebrated every achievement. Lastly, to all future innovators and developers striving to make everyday life more efficient and intelligent, this project is a step towards a smarter, more connected world.

Acknowledgment

As we reach the pinnacle of our academic journey, we are profoundly grateful to those who have been instrumental in our success. First and foremost, we extend our deepest appreciation to our esteemed supervisor, whose unwavering support, insightful guidance, and invaluable expertise have been pivotal in shaping this project. Their mentorship has refined our technical skills and inspired us to push the boundaries of innovation.

We are also immensely thankful to our families and friends, whose encouragement, patience, and belief in our potential have been a constant source of strength. Their unwavering support has fueled our determination and resilience throughout this journey.

Finally, we express our gratitude to everyone who has contributed, directly or indirectly, to the success of this project. This experience has been a journey of growth, learning, and perseverance that we will cherish as we step into the future.

Table of Contents (TOC)

List of Figures (LOF)

Abstract

Abstract

This project presents a wireless smart home automation system that utilizes ESP-based microcontrollers to connect and control various sensors and devices remotely via Wi-Fi. By integrating real-time monitoring and automation, the system enhances security, energy efficiency, and convenience, allowing users to manage their home environment seamlessly through a mobile application. The system includes LED lighting control, enabling users to adjust indoor lights remotely for energy efficiency and convenience. A temperature and humidity sensor continuously monitors environmental conditions, and an automatic cooling fan is activated if the temperature exceeds or drops below a predefined threshold to maintain a comfortable indoor atmosphere. To ensure safety, a fire detection sensor promptly alerts homeowners in case of potential fire hazards. For enhanced security and automation, a PIR motion sensor detects movement, automatically controlling lighting and notifying users of unexpected activity. In addition to security and comfort, the system also optimizes water resource management. A soil moisture sensor is linked to a water pump, enabling automated garden irrigation by supplying water only when necessary, preventing waste and ensuring plant health.

Chapter One

Chapter One: Introductory

1.1: Introduction.

Smart technologies are becoming increasingly common in everyday life, especially inside our homes. Home automation - one of the most practical applications of the Internet of Things (IoT) - allows users to control lighting, temperature, appliances, and security systems remotely. What was once considered a luxury is now a key feature of modern living.

This project aims to build a smart home system using the ESP32 microcontroller, chosen for its low cost, built-in Wi-Fi, and compatibility with IoT applications. The system monitors environmental conditions in real time and automates responses based on sensor data. It also allows remote control through web interface.

The project addresses five key areas within a smart home:

1. Environmental Monitoring:
   A DHT11 sensor tracks temperature and humidity. A fan turns on or off automatically to maintain indoor comfort.

2. Energy-Efficient Lighting:
   Lights can be controlled manually via an app or triggered automatically when motion is detected by a PIR sensor.

3. Fire Safety Alerts:
   A flame sensor detects fire hazards and triggers audio and visual alerts to notify the user.

4. Automated Irrigation:
   A soil moisture sensor activates a water pump only when needed, helping to save water and maintain healthy plants.

The system uses HTML and CSS for the website dashboard. Communication between the ESP32 and the application is handled through HTTP APIs using JSON. This allows smooth data flow between hardware and software and ensures reliable real-time control.

Designed to be modular and expandable, the system supports adding more sensors or actuators without changing its structure. By relying on open-source tools and affordable components, the project remains low-cost, flexible, and user-friendly.

In summary, this project demonstrates how smart home technology can be made accessible and practical using basic hardware, efficient software, and thoughtful design. It offers a simple yet powerful system that makes everyday living more efficient, secure, and convenient.

1.2: Problems.

Although smart technologies are advancing quickly, many traditional homes still lack basic automation and real-time monitoring. This results in wasted resources, limited safety features, and more manual effort in daily tasks. Several key problems highlight the need for smarter and more accessible home systems.

1. Lack of Intelligent Climate Control:
   Most homes rely on manually operated fans or air conditioners, which can lead to energy waste and discomfort due to inconsistent temperatures.

2. Inefficient Irrigation Systems:
   Gardens are often watered based on guesswork, leading to overwatering or neglect. This wastes water and may damage plants.

3. Limited Fire and Motion Detection:
   Without sensors, homes lack alerts for fire or unusual movement. This reduces safety and delays response in emergencies.

4. High Cost of Commercial Solutions:
   Many smart home products are expensive, require professional installation, and use closed systems that limit customization.

5. Lack of Integration and Ease of Use:
   Existing systems are often isolated or difficult for non-technical users to manage or expand.

These issues create a gap between available technology and practical, user-friendly solutions. This project was developed to close that gap by offering a low-cost, open-source smart home system that focuses on automation, safety, and ease of use - without the complexity or high cost of commercial alternatives.

1.3: Objectives and work significance.

This project was developed with several key objectives in mind, aiming to deliver a practical and effective solution for smart home automation. Instead of relying on high-cost or overly complex

commercial systems, this project focuses on building a solution that is affordable, easy to install, flexible and scalable, and built using open-source and widely available tools.

1.3.1: Main Objectives:

1. Enable Real-Time Monitoring and Control
   The system continuously collects data from temperature, humidity, fire, motion, and soil moisture sensors. Based on this data, it can trigger automated actions such as turning on a fan or activating a water pump.

2. Reduce Energy and Water Waste
   By automatically controlling lights and fans based on sensor input, energy is used only when needed. The system prevents unnecessary irrigation by watering plants only when the soil moisture drops below a certain threshold.

3. Improve Home Safety
   A flame sensor detects fire hazards and can alert the user immediately. A motion sensor identifies unexpected movement, which could indicate intrusion, and can turn on lights or trigger alerts.

4. Allow Remote Access and Control
   The system is connected via Wi-Fi, allowing users to check sensor readings or send control commands through a simple web built using HTML Dashboard.

5. Create a Modular and Expandable Architecture
   New sensors or devices can be added with minimal changes to the existing system.

1.3.2: Significance of the Project:

1. Practical: Focused on solving real-world issues like energy waste and manual monitoring.
2. Approachable: It doesn't require special skills or professional installation.
3. Customizable: The design can be adapted to different home environments or user needs.
4. Educational: To understand real-world applications of microcontrollers, sensors, and IoT.

In short, this smart home system demonstrates that advanced home control and automation don't have to be expensive or complicated. With just an ESP32 and a few basic components, users can make their homes smarter, safer, and more efficient.

1.4: Scope of the work

The scope of this project is centered around designing and implementing a small-scale yet functional smart home automation system. It covers both the hardware and software aspects of the

system, aiming to provide a real-time, wireless control and monitoring solution using affordable and accessible tools. This project includes:

1. Environmental Monitoring: Use of temperature and humidity sensors to track indoor climate, with automatic fan control to maintain comfort.

2. Lighting Control: Manual and automatic control of an LED light using a webinterface or based on motion detection.

3. Fire Safety Detection: Integration of a flame sensor to detect potential fire hazards and alert the user immediately.

4. Security Automation: Use of a PIR motion sensor to detect movement and trigger alerts or lighting, enhancing home security.

5. Smart Irrigation: Soil moisture sensor controls a water pump to automate garden watering based on real-time readings.

6. Wi-Fi Connectivity: Wireless communication between ESP32 and the backend website, enabling remote access and control.

7. User Interface: A simple front-end built using HTML and CSS, allowing users to monitor sensor readings and control devices in real time.

1.5: Report Organization

This report is structured into five main chapters, each reflecting a specific stage of the project's development. Chapter One introduces the project's background, scope, and objectives. Chapter Two outlines the technical and practical constraints encountered, along with applicable standards and references to prior academic work. Chapter Three details the methodology, including the hardware used, software tools, system design, and implementation process. Chapter Four presents the results and discusses system performance and limitations. Finally, Chapter Five concludes the work and offers recommendations for future improvements. This project was completed individually during the Spring semester, from March 10, 2025 to June 10, 2025.

Chapter Two

Chapter Two: Constraints and Standards

2.1: Constraints and work limitations.

While building the smart home system, several limitations affected how the system was designed and what features could be included. These limitations were both technical and practical, and they helped shape the final version of the project.

1. Limited Budget:
   The project was self-funded and completed individually, so the choice of components was based on affordability. This limited the use of multiple ESP32 units and reduced the overall modularity of the system.

2. Short Time Frame:
   The project was completed within one academic semester (March 10 – June 10, 2025), which restricted how much could be tested or added. Features like cloud integration or mobile notifications were postponed to focus on core functionality.

3. Component Availability:
   Some hardware parts, such as flame sensors or relays, were either delayed or unavailable. This required substitutions that sometimes-affected performance or stability.

4. Wi-Fi Dependency:
   The system relies entirely on a stable Wi-Fi connection. Any connection issues can break communication between the ESP32 and the app, which is a limitation in areas with weak networks.

5. Hardware Limitations of ESP32:
   Although powerful, the ESP32 has limited memory and processing capacity. Running multiple tasks at the same time—like reading sensors and serving web requests—pushed the device near its limits.

6. Solo Development:
   As this was an individual project, all stages (design, coding, testing, and documentation) were done by one person. This limited the ability to test the system under multiple scenarios or scale the work further.

Despite these constraints, the system achieved its main goals: it monitors the environment in real time, controls devices efficiently, and provides a user-friendly interface. The challenges also offered valuable learning opportunities and created a clear path for future improvements.

2.2: Standards / Codes.

Although this smart home project was not built under formal industry regulations, it followed key academic and open-source development standards to ensure the system is reliable, secure, and maintainable. The focus was on writing clean, modular code and building a system that is both functional and user-friendly.

There are some standards and codes we applied and used in our project and source code.

1. IEEE Standards Bluetooth connections for controlling.

2. Protues Standard Design.

3. Modularity: Each part of the system (hardware) was designed separately to simplify updates and debugging.

4. Code Clarity: Consistent naming, proper documentation, and reusable functions were used to make the code readable and easy to maintain.

5. Safe Hardware Integration: Care was taken to safely connect sensors, relays, and other components to avoid electrical issues or system failures.

6. User Interface Standards: The frontend was designed to be simple and clear, following modern UI/UX practices for smooth navigation and accessibility.

7. Secure Data Handling: APIs used HTTP and JSON to send and receive data in a structured, safe way between the ESP32 and the application.

2.3: Earlier coursework

The development of this smart home project was based on skills learned in several university courses. Programming and web development courses provided the foundation for writing clean code and building a responsive user interface. Courses in embedded systems and microcontrollers taught how to connect and control hardware components using GPIOs. These pins allowed the ESP32 to read sensor input and control devices like lights, fans, and pumps. Networking and software engineering courses also played a key role by introducing HTTP communication, system organization, and structured development methods—all of which were essential to completing the project successfully.

Chapter Three

Chapter Three: Methodology

3.1: Development Tools and Languages

The development of the smart home automation system involved two main areas: assembling and simulating the hardware circuit, and developing the software required to control and monitor the system. Both hardware and software tools were carefully selected based on compatibility, accessibility, and ease of integration.

3.1.1: Hardware Tools

1. ESP32 Dev Module:
   The core microcontroller used in the system. It reads sensor data, controls actuators, and communicates via Wi-Fi with the backend system.

2. Proteus Design Suite 8.13: (Labcenter Electronics)
   Used for basic simulation of circuits and sensor behavior before physical assembly.

   Proteus Libraries Used:
   - ESP32 simulation library (community-supported)
   - DHT11 sensor library
   - Flame sensor and PIR sensor modules
   - Soil moisture analog sensor simulation
   - Relay module library

3. Breadboard, Jumper Wires, and Multimeter:
   Used during the prototyping phase to test connections, verify signals, and safely assemble the hardware layout before final installation.

3.1.2: Software Tools

1. Arduino IDE: (v1.8.19)
   Used to program the ESP32 in C++. It handled the reading of sensor inputs, managing device outputs, and enabling communication over HTTP.

Libraries Used in Arduino IDE:

- DHT.h: For reading temperature and humidity data
- WiFi.h & WebServer.h: For setting up the HTTP server
- ArduinoJson.h: For encoding and decoding JSON requests/responses

3.1.3: Programming Languages and Frameworks

1. C++ (Arduino Core):
   Used to develop the firmware running on the ESP32.

2. HTML / CSS / JavaScript:
   Formed the foundation of the user interface, ensuring it was interactive and responsive.

3. HTTP API routes / requests, system logic, and communication with the ESP32.

3.2: Project Hardware Components

The hardware setup for this smart home system includes two ESP32 microcontrollers and a range of sensors and output devices, all mounted inside a custom 3D-printed house model. One ESP32 handles data collection from sensors, while the second ESP32 acts as the main controller that receives user commands and controls the entire system. Below is a list of the main components used and how each one works in the system.

Main Components and Functions:

1. ESP32 (Main Controller)
   Receives commands from the web app or a physical button and sends control signals to the second ESP32 over Wi-Fi.

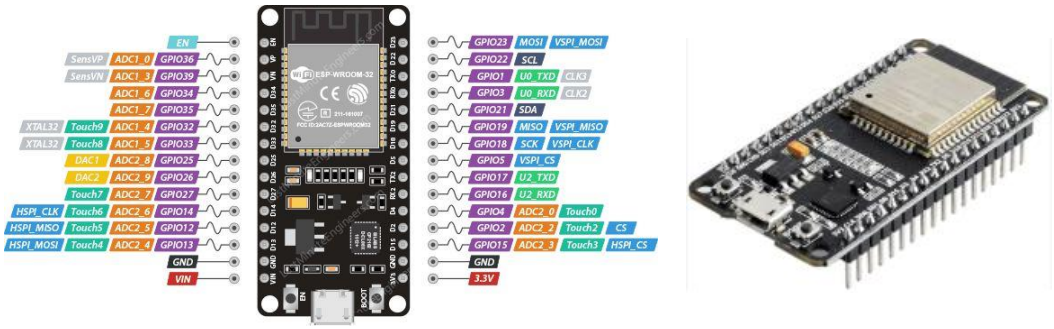


Figure 1: ESP32 (Main Controller)

2. ESP32 (Sensor Node)
   Reads data from sensors (temperature, fire, motion, soil) and controls outputs (fan, lights, pump) based on commands or automatic conditions.

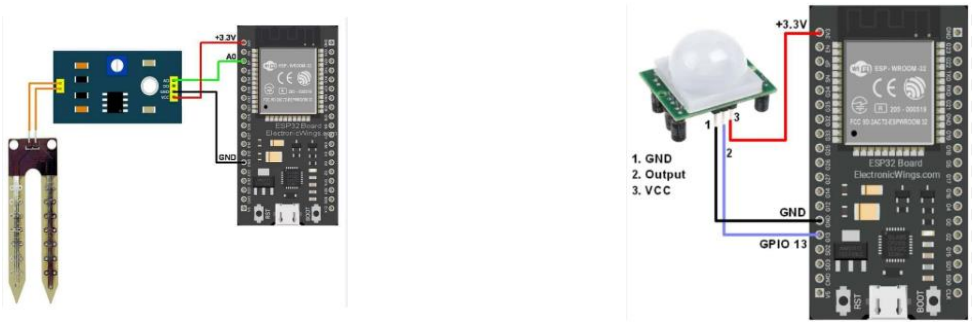


Figure 2: ESP32 (Sensor Node)

3. DHT11 Sensor
   Measures temperature and humidity. If the temperature is too high, it triggers the fan to maintain comfort.

Figure 3: Temperature Sensor

4. PIR Motion Sensor
   Detects movement inside the house. It automatically turns on the lights and alerts the system of activity.

Figure 4: PIR Motion Sensor

5. Flame Sensor
   Detects fire or high heat. When triggered, it activates the buzzer and warning lights for safety.

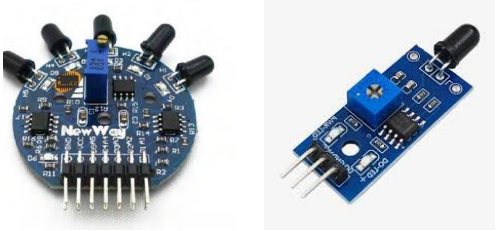


Figure 5: Flame Sensor

6. Soil Moisture Sensor
   Measures how dry the soil is. When dryness crosses a set threshold, the system turns on the water pump for irrigation.

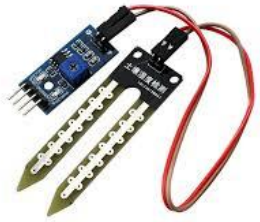


Figure 6: Soil Moisture Sensor

7. Fan (DC Motor)
   Used to cool the environment automatically when high temperature is detected.

Figure 7: Fan (DC Motor)

8. Mini Water Pump
   Irrigates plants based on readings from the soil moisture sensor. Helps save water by running only when needed.

Figure 8: Mini Water Pump

9. Relay Modules (x2)
   Allow the ESP32 to safely control the fan and pump, which use higher voltages than the microcontroller.

Figure 9: Relay Modules (x2)

10. LEDs Light (Interior & Exterior)
    Act as lights and indicators. Some LEDs turn on with motion, others signal alerts (like fire).

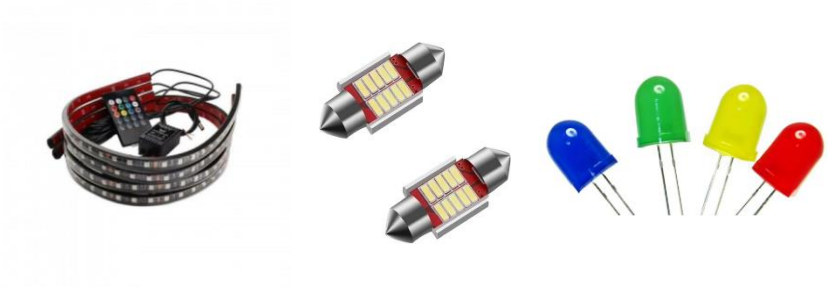


Figure 10: LEDs (Interior & Exterior)

11. Buzzer
    Used for audible alerts during fire detection .

Figure 11: Buzzer

12. System Switch
    Connected to the main ESP32 to manually turn the system on or off without using the app.

Figure 12: Push Button

13. Power Supply (Lithium Battery or USB Adapter)
    Provides power to both ESP32 units and all connected sensors and actuators.

Figure 13: Power Supply (Lithium Battery or USB Adapter)

3D-Printed House Model

Holds and organizes all components in a realistic, physical layout for demonstration and testing.

Figure 14: 3D-Printed House Model

This hardware setup was designed to be compact, affordable, and functional for real-time smart home automation, with flexibility for future upgrades.

.

3.3: Project Schematic and Design

This simulation step made it easier to catch early wiring issues, test signal flow, and ensure that all sensors and actuators worked well with the ESP32 microcontrollers.

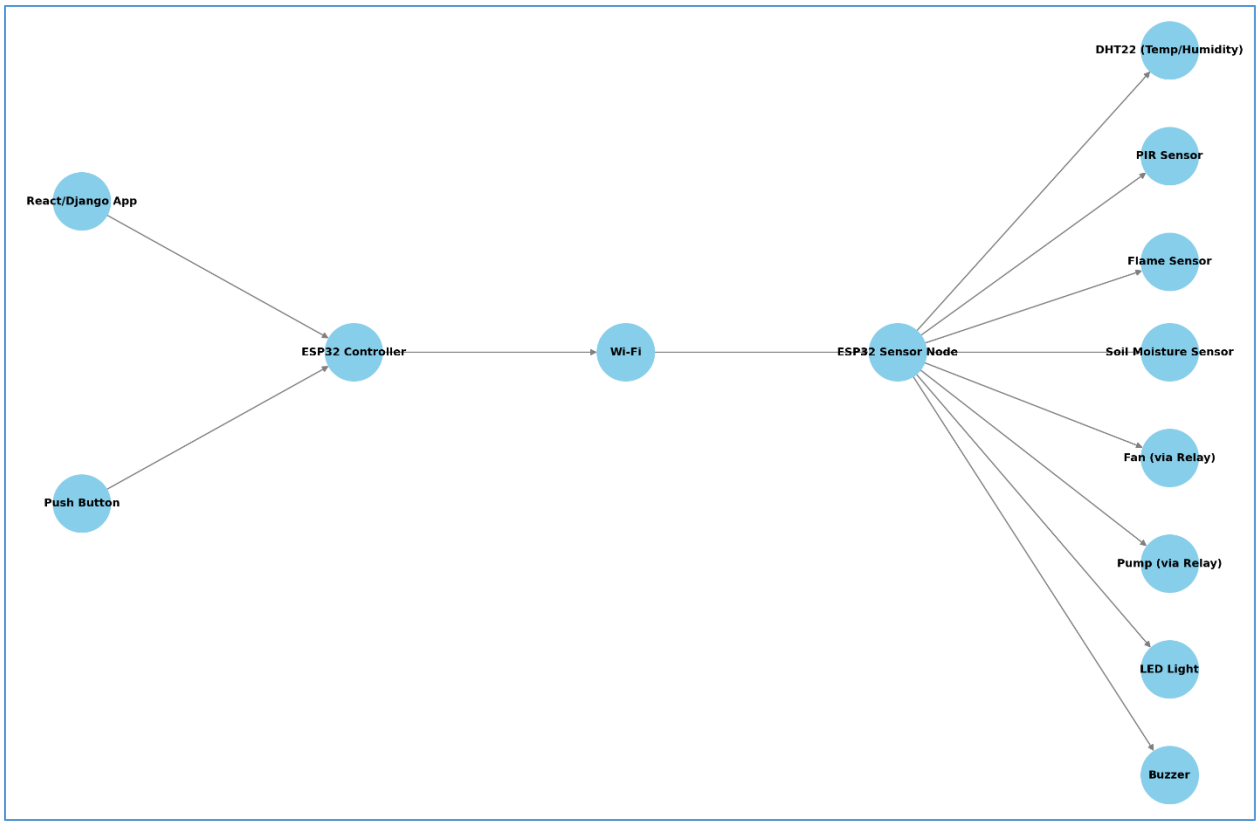


Figure 15 :Smart Home Automation System - Schematic Design

The system is made up of two ESP32 boards connected over Wi-Fi. One acts as the Main Controller, which handles input from the user (via a web app), while the second board is the Sensor Node, responsible for reading data from sensors and activating outputs like the fan, pump, or buzzer. This separation makes the system easier to scale or modify later.

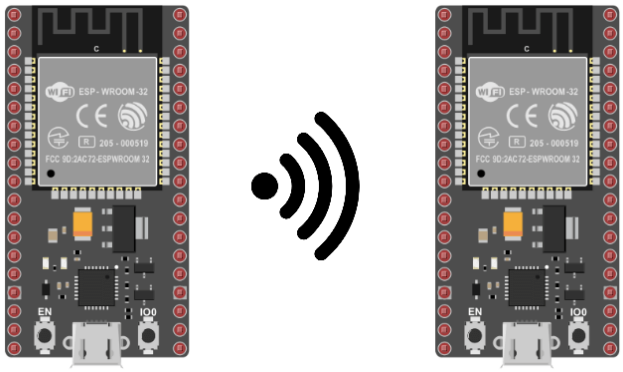


Figure 16: Two ESP32 boards connected over Wi-Fi

The Sensor Node connects to multiple sensors including:

1. DHT22 for temperature and humidity.
2. PIR motion sensor for detecting presence.
3. Flame sensor for fire alerts.
4. Soil moisture sensor for smart irrigation.

It also controls output devices:

1. Fan for cooling.
2. Pump for watering.
3. LED lights for status or indoor lighting.
4. Buzzer for audible alerts.

Component	ESP GPIO	Mode	Trigger / Function
DHT22	GPIO 4	Input	Measures temperature & humidity; triggers fan if > 35°C
PIR Sensor	GPIO 14	Input	Detects motion; turns on LED when movement is detected
Flame Sensor	GPIO 12	Input	Detects fire; triggers buzzer and water pump
Soil Sensor	GPIO 34	Analog	Triggers water pump if soil is too dry
DC Fan	GPIO 5	Output	Activated automatically if temperature > 35°C or manually via app/switch
Water Pump	GPIO 27	Output	Activated by low soil moisture, fire detection, or manually
LED Light	GPIO 2	Output	Turns on via app/switch or when motion is detected
Buzzer	GPIO 26	Output	Activated when flame is detected
Button	GPIO 15	Input	Toggles system on/off; fully synced with app state

Figure 17: ESP32 Sensor Node Pin Configuration

The Controller Node communicates with the Sensor Node using HTTP requests. It acts as the system's brain by sending control commands and receiving live sensor readings. The web app,interacts only with the Controller Node. All communication is done using JSON over HTTP.

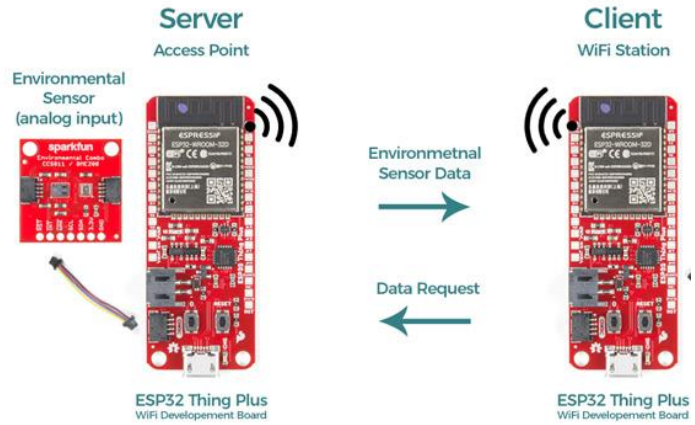


Figure 18: ESP32 Client–Server Communication Overview

After validating the system in Proteus, the full circuit was built using breadboards, jumper wires, and relays. Finally, the components were installed inside a 3D-printed house model, with LED lighting for visual feedback. The entire system runs on a 5V USB supply, and can optionally use a rechargeable battery.

This modular and wireless setup allows the system to grow easily by adding more devices in the future - without needing to rebuild everything.

Figure 19: Physical Circuit in 3D-Printed Smart Home

3.4: Software Implementation and Data analysis

The software implementation of this smart home system integrates multiple layers of hardware and software, working together wirelessly over Wi-Fi. The entire system is controlled via two ESP32 microcontrollers and a responsive web interface built with HTML and JS.

The design follows a client–Server model, where the ESP32 Sensor Node acts as a server providing sensor data and executing control commands, while the ESP32 Controller Node and the application interface act as clients sending HTTP requests and receiving updates. This setup allows the user to interact with the home system either through a physical button or a graphical user interface.

System Architecture Workflow

1. **System Start-up**
   The ESP32 Sensor Node connects to Wi-Fi and starts a local HTTP server.
   The Controller Node or the web application sends a fetch /control request to initialize the system.

2. **User Activation**
   From the physical button on the Controller Node.
   From the application UI using a toggle switch.

Upon activation:
The system LED lights up.
DHT11 sensor immediately reads temperature and humidity.
Data is sent to the app and displayed in real-time.

3. **Sensor Monitoring Loop**
   The ESP32 Sensor Node continuously monitors:
   - Environmental conditions.
   - Movement or fire.
   - Soil moisture level.

   And takes automatic action based on predefined rules.

4. **Data Communication**
   Status updates are requested via fetch /status.
   Control commands are received via fetch /control.

Component-Level Algorithmic Behavior
Here's a breakdown of how each sensor or component behaves within the system:

1. **DHT11 Sensor (Temperature & Humidity)**
   - Triggered automatically on system start.
   - If temperature > 30°C, the fan turns ON automatically.
   - Readings are sent to the app

2. **PIR Motion Sensor**
   - Continuously checks for motion (up to 7 meters, 120° angle).
   - If motion is detected:
   - Lights turn ON automatically.

3. **Flame Sensor**
   - Monitored in real-time.
   - If fire or flame is detected:
   - Buzzer activates for an audible alert.

4. **Soil Moisture Sensor**
   - Continuously monitors soil dryness.
   - If moisture drops below the threshold:
   - Water pump activates automatically.
   - Pump can also be controlled manually via app.

5. **Fan**

- Runs automatically when temperature exceeds 30°C.
- Can also be controlled manually through the app.

6. **Water Pump**
   - Activates in any of the following:
   - Soil moisture is too low (Less than 2500)
   - User manually toggles it from app .

7. **LED Light**
   - Turns ON:
   - When system is activated.
   - When motion is detected.
   - User manually toggles it from app or detects by the sensors.

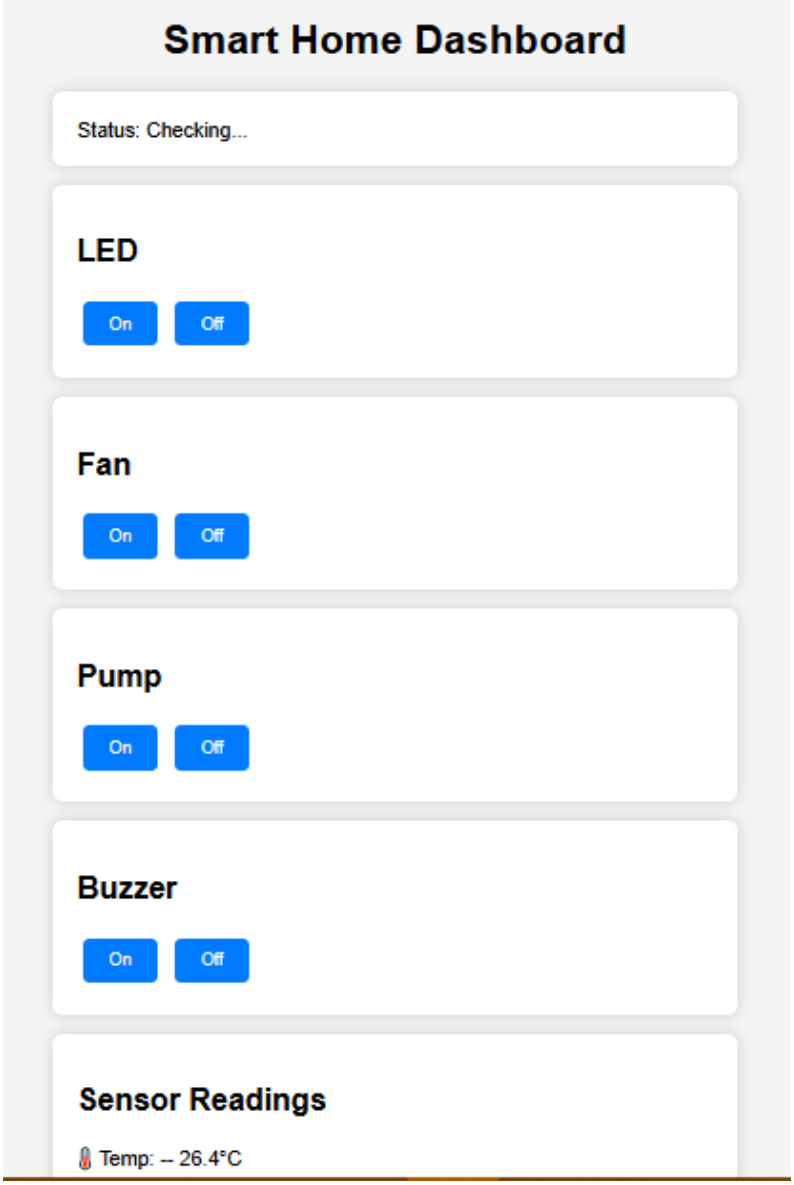
8. **Buzzer -** Sounds only in case of fire detection.

9. **Main Button**
   - Toggles the system ON or OFF.
   - Triggers LED reading on press.

Web Interface Logic and Control

The smart home control interface relies solely on frontend technologies, using HTML for the page structure, CSS for styling, and JavaScript to enable interaction between the user and the system. There is no traditional backend in this project; instead, the ESP32 acts as a web server, receiving and processing requests directly over the local network.

Control commands (such as turning the LED or fan on/off) are sent using HTTP requests via fetch, for example: http://192.168.1.54/led_on. Sensor readings are automatically fetched through a request to http://192.168.1.54/status, which returns a JSON response containing temperature, humidity, and soil moisture values. For successful communication, both the ESP32 and the device running the interface must be on the same Wi-Fi network, and the ESP32 must have a web server configured to handle these routes.



In future work, the site will be developed to become a web-application like the one we started working on, as shown in the pictures below, so that we can better connect the system and the site in terms of sending, receiving, and updating data in a better and faster way.

The new web application was developed to allow remote interaction with the home system via a simple and responsive interface. The app consists of:

- A dashboard showing live sensor data (temperature, humidity, motion, fire, soil moisture).
- Interactive controls (switches and buttons) to control devices like the LED, fan, and pump.
- Visual indicators to show system status and connectivity.

When the user enters the app:

- They are prompted to input the IP address of the ESP32 Sensor Node.
- Once connected, they can control the system and receive live data.
- All commands and responses are handled in real time using RESTful HTTP requests.

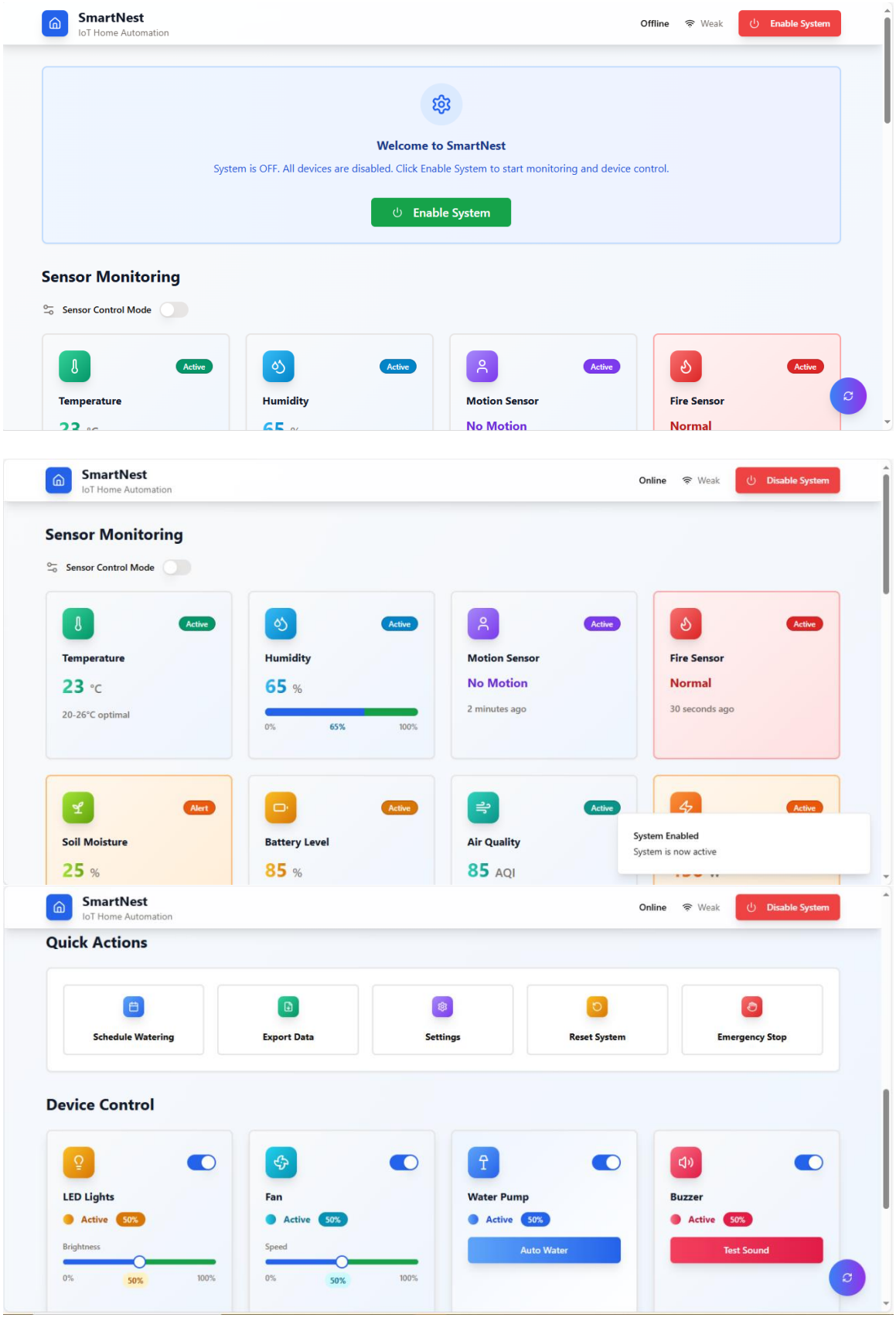


Figure 20: Smart Home Web Interface

Testing and Data Flow Validation

Before final deployment, each component was tested:

1. The system was booted using both manual (button) and digital (UI) triggers.
2. Sensor readings were validated under different conditions (heat, fire simulation, soil).
3. The app was used to confirm the correct flow of control data to the Sensor Node.

The entire flow from user action to sensor response and back to the user interface was verified to ensure minimal delay and correct synchronization.

Chapter Four

.

.

Chapter Four: Results and Discussions

The implementation of the smart home automation system was tested thoroughly after completing both the hardware setup and the software configuration. Throughout the testing phase, the ESP32 Sensor Node maintained a consistent Wi-Fi connection and successfully responded to commands from the application and the physical button. The system could be turned ON or OFF from either the web interface or the hardware switch.

During functional testing, all sensor-based triggers worked as expected. The fan was activated automatically when the temperature rose above 30°C, and the water pump turned on either when the soil became dry or in response to fire detection. The PIR sensor reliably detected motion and activated the LED light, while the buzzer provided immediate auditory feedback during flame detection. These automated reactions confirmed the stability and correctness of the logic implemented in the ESP32 Sensor Node. Manual overrides for the fan and pump were also successfully tested through the application, ensuring flexible control over the environment.

Overall, the system demonstrated fast response time, accurate sensor communication, and real-time data synchronization between the hardware and the web application. The dashboard interface allowed users to monitor the system and control devices remotely with ease. The interface provided clear feedback for each command and displayed warnings when the connection to the ESP32 Sensor Node was unavailable. These results suggest that the system is not only technically functional but also practical and user-friendly, making it suitable for small-scale smart home applications.

Chapter Five

.
.

Chapter Five: Conclusions and Recommendations

This project demonstrated the practical implementation of a smart home automation system using ESP32 microcontrollers, Wi-Fi communication, and a user-friendly web interface. By combining multiple environmental sensors, output devices, and wireless control through a mobile/web dashboard, the system was able to enhance comfort, safety, and energy efficiency within a residential setting. The architecture, based on a separation between sensor logic and control logic (via two ESP32 nodes), proved to be reliable and scalable.

The results showed that the system performed effectively across various scenarios. Automated responses to temperature, motion, fire, and soil moisture conditions were accurate and fast. The ability to control and monitor the system from both a physical push button and a remote application offered flexibility for users with different preferences. The real-time data updates, along with visual and auditory alerts, provided an engaging and informative experience.

In future iterations, several improvements can be made. First, the system could be extended to include more sensors or integrate with cloud platforms for long-term data logging and remote access beyond the local network. Adding support for mobile push notifications and voice assistants like Google Assistant or Alexa would further increase usability. Moreover, implementing secure authentication for the app would enhance safety and protect user data. Initially, there was an intention to implement an Ad hoc network to allow direct device-to-device communication without relying on a central router. However, due to the high cost and the fact that this project was carried out individually, this feature was not implemented. Overall, this project serves as a strong foundation for accessible, customizable smart home solutions, especially for low-budget or DIY environments.