AuraTherm / Aura

A Project Report
Presented to
The Faculty of the Computer Engineering Department
San Jose State University
In Partial Fulfillment
Of the Requirements for the Degree
Bachelor of Science in Computer Engineering

By
Roger Huynh, Jeric Montepalco, Sabrina Quach
December 2025

APPROVED FOR THE COLLEGE OF ENGINEERING

Dr. Daphne Chen, Project Advisor

Dr. Wencen Wu, Instructor

Dr. Jorjeta Jetcheva, Computer Engineering Department Chair

ABSTRACT

Aura / AuraTherm

By Roger Huynh, Jeric Montepalco, Sabrina Quach

In recent years, the integration of smart technology in homes has changed the way we manage our daily tasks and utilities. In particular, smart thermostats have gained widespread popularity for their ability to optimize heating and cooling systems, improving comfort and energy efficiency. The scope of this project lies within home automation and energy management, aimed at creating smarter, more sustainable living environments by utilizing technology to enhance efficiency and convenience.

Most thermostats, including some smart ones, managed the temperature of the entire house, even when only a few rooms were occupied. This inefficient heating and cooling method not only increased utility costs but also reduced the lifespan of HVAC systems, adding to maintenance expenses. Additionally, excessive energy use contributed to a higher carbon footprint and strained energy resources. Although thermostats have been improving over the years, there was still an opportunity to enhance their adaptability and efficiency.

This project aimed to improve energy efficiency by ensuring that heating and cooling resources were only used when necessary. Our smart thermostat, AuraTherm, used motion sensors to detect occupancy and regulate room temperatures, disregarding unoccupied spaces. Homeowners received notifications about temperature adjustments in occupied rooms and summaries of energy usage through the Aura smart app. AuraTherm actively monitored occupancy in real-time and dynamically adjusted settings for maximum efficiency. This approach reduces energy waste, lowers utility costs, and extends HVAC system longevity. By eliminating unnecessary energy consumption, AuraTherm reduces energy usage, utility costs, and environmental sustainability.

Acknowledgments

We want to thank our advisor, Dr. Daphne Chen, for continuous guidance throughout the development process. We also acknowledge San Jose State University for giving us the knowledge and skills that are necessary for this project. Lastly, we would like to thank each other for the teamwork, encouragement, and dedication that made this project possible.

Table of Contents

References

Ayan, Onur, and Belgin Turkay. *Smart Thermostats for Home Automation Systems and Energy Savings from Smart Thermostats*. 1 Oct. 2018, pp. 1–6, ieeexplore.ieee.org/document/8751790, https://doi.org/10.1109/ceit.2018.8751790. Accessed 11 Dec. 2025.

Beltran, Alex, et al. “ThermoSense.” *Proceedings of the 5th ACM Workshop on Embedded Systems for Energy-Efficient Buildings*, 11 Nov. 2013, pp. 1–8, dl.acm.org/doi/10.1145/2528282.2528301, https://doi.org/10.1145/2528282.2528301. Accessed 11 Dec. 2025.

Carli, Raffaele, et al. “IoT Based Architecture for Model Predictive Control of HVAC Systems in Smart Buildings.” *Sensors*, vol. 20, no. 3, 31 Jan. 2020, p. 781, www.mdpi.com/1424-8220/20/3/781, https://doi.org/10.3390/s20030781. Accessed 11 Dec. 2025.

Ehsanifar, Mohammad, et al. “A Sustainable Pattern of Waste Management and Energy Efficiency in Smart Homes Using the Internet of Things (IoT).” *Sustainability*, vol. 15, no. 6, 13 Mar. 2023, pp. 5081–5081, www.mdpi.com/2071-1050/15/6/5081, https://doi.org/10.3390/su15065081. Accessed 11 Dec. 2025.

Patel, Shwetak, et al. “Detecting Human Movement by Differential Air Pressure Sensing in HVAC System Ductwork: An Exploration in Infrastructure Mediated Sensing.” *LNCS*, vol. 5013, 2008, pp. 1–18, aiweb.cs.washington.edu/research/projects/aiweb/media/papers/tmpqiqC-3.pdf. Accessed 11 Dec. 2025.

Santos, Sara. “ESP32 Pinout Reference: Which GPIO Pins Should You Use? | Random Nerd Tutorials.” *Random Nerd Tutorials*, 2 Oct. 2019, randomnerdtutorials.com/esp32-pinout-reference-gpios/. Accessed 11 Dec. 2025.

Stepnitz, Alston, et al. *Title Connected Thermostats for Low Income Households: Insights from User Testing Permalink Https://Escholarship.org/Uc/Item/61k1c32x Publication Date*.

Appendix

Appendix A – Hardware Schematics and Pin Assignments

Appendix B – ESP32 Firmware Source Code

Appendix C – Mobile App (Aura App) Documentation

List of Figures

List of Tables

Chapter 1. Introduction (All Members)

1.1 Project Goals and Objectives

The key goals of AuraTherm was to improve energy efficiency, lower utility costs, and promote environmental sustainability. To improve energy efficiency, we developed a smart thermostat that dynamically adjusts room temperature based on occupancy to reduce energy waste. By reducing energy consumption, AuraTherm lowered utility costs across households of varying incomes and further encouraged environmental sustainability.

The overall objectives of AuraTherm was to develop a motion-sensing system, ensure ESPHome integration, and enable remote access to the smart thermostat. To develop a motion-sensing system, we implemented HC SR501: PIR (Passive Infrared Sensor) to detect occupancy and enable heating or cooling functions. We developed firmware that ensured the ESPHome integration for our smart home app, Aura. Aura will allow AuraTherm users to control room temperature in occupied rooms and view their energy consumption summaries through a user-friendly mobile interface.

1.2 Problem and Motivation

The main issue with both traditional and modern smart thermostats is that they regulate heating or cooling for the entire house rather than adjusting temperatures based on occupancy in individual rooms. This inefficiency results in higher energy consumption, leading to increased utility costs and carbon emissions. While this affects all households, it has a greater impact on lower-income households, where energy costs make up a larger share of monthly expenses. Since many existing smart thermostats are expensive, there is a lack of accessibility for low-income households, which widens the economic gap.

Through AuraTherm, these issues are addressed by providing a cost-effective solution that dynamically adjusts room temperature based on occupancy detection. AuraTherm can ensure accessibility and ease of use for users of different backgrounds. Furthermore, the motivation to develop AuraTherm was to create an affordable smart thermostat that was accessible to households of varying incomes in order to reduce energy waste, lower utility costs, and support environmental sustainability.

1.3 Project Application and Impact

AuraTherm was implemented mainly in residential homes to help homeowners and renters optimize their heating and cooling systems. The target audience for this project is individuals who are looking to reduce their energy consumption and reduce utility costs. Since we reduced energy consumption, the project contributed to lower carbon emissions for each household, aligning with global sustainability efforts. Additionally, AuraTherm's affordability made energy management and lower utility costs accessible to a broader economic demographic, including low-income households that may not have had access to such technology before.

1.4 Project Results and Deliverables

After the development of AuraTherm and Aura, we produced a fully working thermostat that detects motion and dynamically adjusts temperature for the occupied room. Aura extends the functionality of the thermostat as it allows users to change the temperature to match their comfort and efficiency. Our deliverables included the thermostat hardware, firmware, mobile app, and a comprehensive project report. We implemented motion sensor tracking and wireless communication to allow for user friendly temperature management. Through testing, the system was fully functional in a simulated home environment, demonstrating energy efficiency and improved user comfort.

1.5 Project Report Structure

In the next sections, we will further dive into AuraTherm and Aura's building process and results. In Chapter 2, background on HVAC energy consumption, smart thermostat technologies, and related research was explored to contextualize our idea. Chapter 3 outlines project requirements in domain, business, functional, and non-functional specifications. Chapter 4 describes the system design, overall architecture, component interactions, and design tradeoffs. Chapter 5 discusses the system implementation such as platforms, methods, algorithms, challenges, and lessons learned. Moreover, Chapter 6 summarizes the tools and standards used. While Chapter 7 presents the testing scope, experimental setup, and results that were used to validate the system. Lastly, Chapter 8 concludes the report by encapsulating key outcomes, contributions, and future work.

Chapter 2 Background and Related Work (All Members)

2.1 Background and Used Technologies

AuraTherm builds on the current technologies revolving around HVAC systems, focused on smart home automation and ensuring energy efficiency. With the rise of HVAC systems in the 1950s, HVAC systems have accounted for approximately 60% of energy consumption in buildings, making them play a large role and being the target for energy efficiency technologies. With the innovation of programmable thermostats, they have proven themselves to be an effective method to save energy by allowing consumers to set schedules of usage. Although it has been proven to help with energy efficiency, they do not account for real-time changes regarding occupancy, specified rooms, and much more. AuraTherm addressed this issue by utilizing motion sensors and temperature sensors to adjust the heating and cooling based on multiple factors. This ensures that energy is used efficiently to reduce waste and utility costs, while also maintaining the comfort of the consumer.

In terms of technology, AuraTherms utilizes the ESP32 microcontroller as the central device for sensor integration and control logic. Some components that were crucial are a relay used to control HVAC components, a PIR motion sensor for occupancy detection, and a BME280 temperature sensor for environment monitoring. Through Wi-FI connection, remote access and data communication were possible with the ESP32 hosting a HTTP server that interacts with the mobile app through JSON-based endpoints. The paired mobile app, Aura, was built using React Native, providing a user-friendly interface for monitoring and control.

Moreover, the development of AuraTherm was achievable through knowledge from coursework at San Jose State University. In Table 1, the key courses were taken and applied to AuraTherm/Aura.

Table 1. Courses and Applied Knowledge

Courses	Applied Knowledge/Skills
CMPE 127 - Microprocessor Design I CMPE 146 - Real-Time Embedded System Co-Design CMPE 181 - IoT Platforms	Programming ESP32 microcontroller, handling GPIO inputs/outputs, and managing low-level hardware integration.
CMPE 148 - Computer Networks I CMPE 181 - IoT Platforms	Implementing Wi-Fi communication protocols and lightweight HTTP server for IoT connectivity.
CMPE 131 - Software Engineering I CMPE 181 - IoT Platforms	Applying design patterns and system decomposition principles to structure

	hardware, software, and communication layers.
CMPE 181 - IoT Platforms DSID 131 - Interaction Design DSGD 186 - Digital Applications: Methodology 3	Developing the Aura app for user interaction, integrating endpoints, and ensuring responsive UI/UX.

2.2 Literature Search

2.2.1 Smart Home Automation and Energy Efficiency

In the current day and age, the usage of HVAC systems has significantly increased, also increasing the quality of life for many users at a cost. HVAC systems account for 60% (Ayan and Turkay) of the energy consumption of all buildings and are expected to increase over time. Smart home systems have proven to be an effective solution thus far, with programmable thermostats playing the largest role in energy efficiency. When properly turning on and off a programmable thermostat, consumers are able to save 20.72% on electricity, 17.7% on gas in their sleep, 34.41% on electricity, and 10.28% when away from home (Ayan and Turkay). Smart thermostats help automate this system by having a set schedule and learning the consumer's typical schedule even when the user is not at home, telling the system when to turn off and on. To help remedy this issue even further, Aura/Auratherm's main purpose is to increase efficiency by directing the flow of heating and cooling components while also shutting down the system when sensing whether or not the room has reached the desired temperature. Additionally, Aura/AuraTherm will also sense human motion, telling the system that a specific room needs the temperature to be adjusted, ensuring the comfort of the occupants. With HVAC systems playing such a large role in energy consumption, smart thermostats have proven to be a remedy to the issue, but with the use of additional sensors and redirecting airflow, this solution can further save the consumer more money, which is the purpose of Aura/AuraTherm.

2.2.2 Motion Sensing and Occupancy Detection for HVAC Control

Programmable thermostats have proven to be an effective solution, and with the use of smart thermostats, consumers are able to change and set the temperature at will wherever they are, making it effective even from far away. With Aura/AuraTherm and sensors, the solution will become much more effective at conserving energy and redirecting airflow to a specific room the occupant is in. Similar systems have been created before with different sensors, where they were tested on 10 different buildings with a total of 2,100 sq. ft. in a period of three weeks with three different conditions. Based on these tests, they were able to analyze the data and figure out that consumers were able to save 25% on energy consumption annually (Beltran et al.). Although the method of detection may be different, the idea of controlling the airflow based on occupancy has proven to be an effective technique for energy conservation while also being able to ensure a comfortable temperature. Additionally, the sensors that were utilized were 3000 mAh lithium batteries, able to last the period of three weeks during the test. Aura/AuraTherm will utilize different sensors to achieve more efficient usage of power while also being calibrated to be effective in its task of detecting occupants.

2.2.3 IoT-Based Thermostat Control Systems

Using an IoT-based thermostat allows users the ability to make changes from anywhere, while also being able to integrate it into other systems. Systems like Google, Alexa, and Siri allow users to make changes to their smart home devices and easily automate simple things, like turning off lights, locking doors, or even playing music. While seemingly small, many users enjoy automating simple tasks and utilizing these systems to help their daily routines, making it more convenient. Although these are the main purposes of implementing a smart system, data that is collected from IoT systems leads to more analysis of the consumer and how the system should act without the user's approval, allowing for higher energy efficiency (Ehsanifar et al.). In the case of Aura/AuraTherm, the system will learn about the user's daily routine, for example, which room they spend the most time in or what time they typically come in. With this data, the system can tell the HVAC system to turn off or not at specific times

2.3 State-of-the-art Summary

Through current research and existing solutions in smart thermostats, the idea of an occupancy-detecting thermostat proved possible. Smart thermostats provide significant energy savings, but are limited by overall temperature changes to the whole housing system.

AuraTherm builds upon existing solutions, implementing sensors to ensure better efficiency regarding HVAC systems. The usage of both programmable and smart thermostats have proven its effectiveness in energy conservation. AuraTherm takes this step further by redirecting airflow into a specific room when it senses motion and other factors. When it detects that the desired temperature has been reached, it tells the HVAC system to turn off so that it is not constantly running for other unoccupied rooms. Additionally, users will be able to integrate the system with other smart devices to make the task more convenient for the user and even when the user is not nearby the system.

Chapter 3 Project Requirements (All Members)

3.1 Domain and Business Requirements

Figure 1. UML Diagram

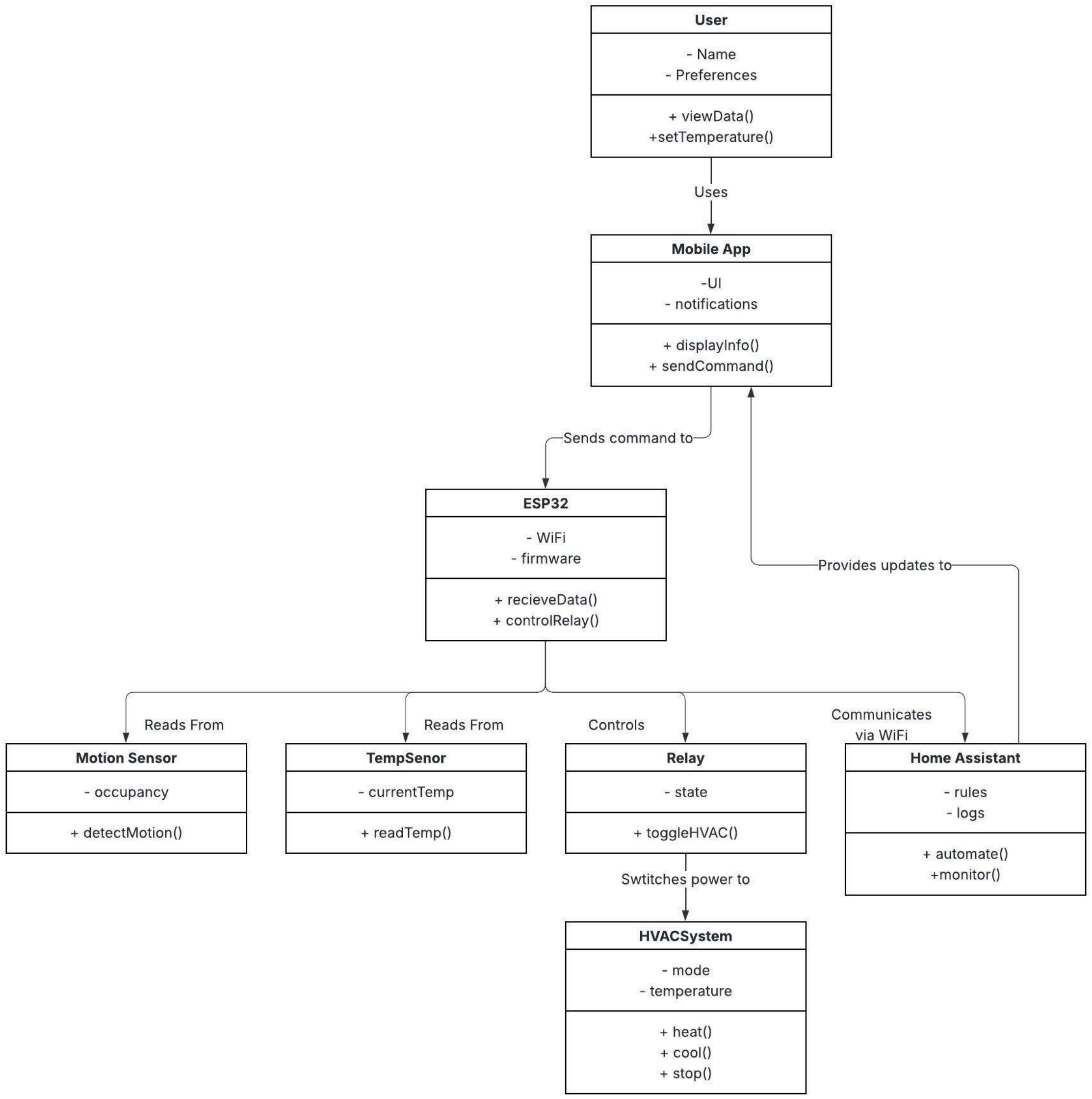


Figure 2. Process Summary Diagram

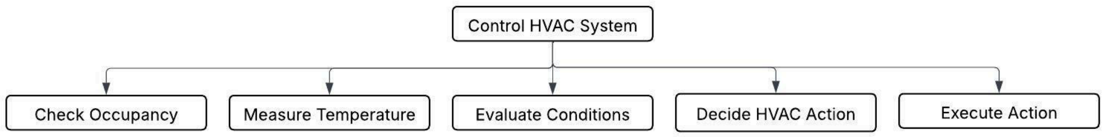


Figure 3. Domain Class Diagram

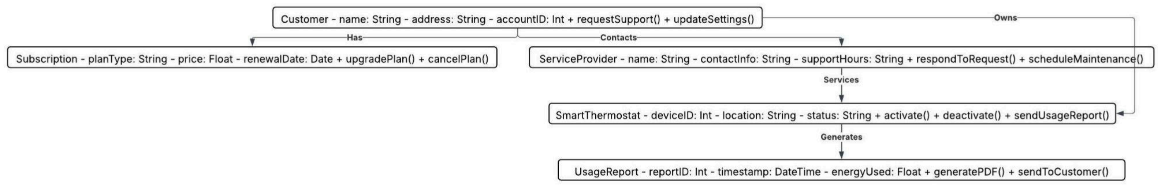
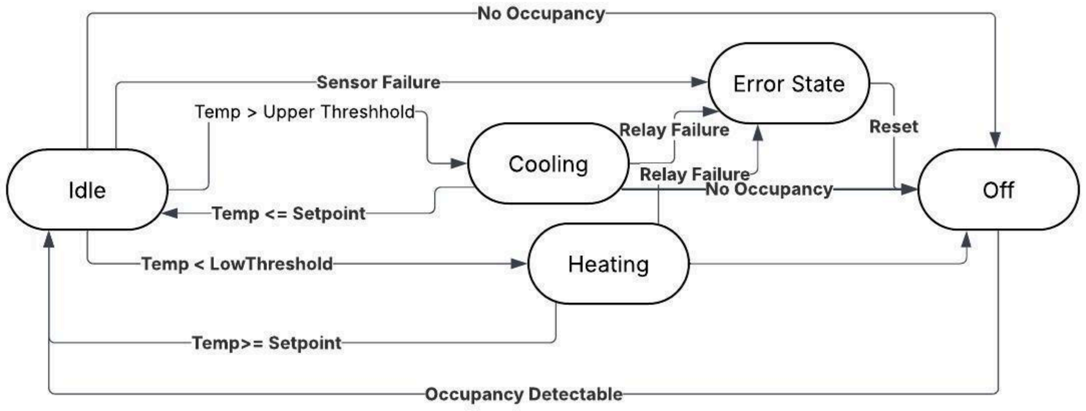

Customer - name: String - address: String - accountID: Int + requestSupport() + updateSettings()
Owns
Has
Contacts
Subscription - planType: String - price: Float - renewalDate: Date + upgradePlan() + cancelPlan()
ServiceProvider - name: String - contactInfo: String - supportHours: String + respondToRequest() + scheduleMaintenance()
Services
SmartThermostat - deviceID: Int - location: String - status: String + activate() + deactivate() + sendUsageReport()
Generates
UsageReport - reportID: Int - timestamp: DateTime - energyUsed: Float + generatePDF() + sendToCustomer()

Figure 4. State Machine Diagram



3.2 System (or Component) Functional Requirements

Below is a list of mandatory (“shall”) and optional (“should”) system behaviors to be implemented, based on what has already been applied:

Table 2. Functional Requirements Specification for AuraTherm

Requirement ID	Description	Priority	Status
FR1	The system shall detect occupancy using a PIR motion detector.	Must	Completed
FR2	The system shall measure the current room temperature using a temperature sensor.	Must	Completed
FR3	The system shall control the HVAC system using a relay module.	Should	Removed
FR4	The ESP32 shall communicate with the Wi-FI Network to a local server.	Must	Completed

FR5	The system shall adjust the HVAC system based on occupancy, user preferences, and other relevant factors.	Must	Completed
FR6	The software application shall allow users to adjust the temperature.	Must	Completed
FR7	The system shall log occupancy and temperature data for display on the app.	Must	Completed
FR8	The system should display data on a local screen	Should	Completed
FR9	The system should support HTTP messaging	Should	Completed

3.3 Non-functional Requirements

These requirements help for future testing by defining performance, reliability, safety, and usability metrics for the system.

Table 3. Non-Functional Requirements Specification for AuraTherm

NFR ID	Description	Measurable Target	Status
NFR1	The control loop response latency will be less than 200-500 milliseconds for motion detection on	≤ 200-500 ms latency	Completed
NFR2	The control loop response latency will be less than 5-10 seconds for motion detection off	≤ 5-10s latency	Completed
NFR3	Based on test scenarios, projected to reduce HVAC energy usage by 10–15% in occupied test environments.	≤ 100mA	Future Work
NFR4	Relay enforces a 2-minute minimum off-time and 5-minute on-time to protect HVAC hardware.	≥ 180s cycle buffer	Removed
NFR5	The system will maintain the temperature within ±1.0°C of the preferred settings.	±1.0°C	Completed
NFR6	The system shall be able to operate locally	Local-only	Partial

	without requiring the internet.	fallback	Completion
NFR7	System shall recover from a crash or error using an auto-reset, retrying every 10 seconds	100% restart on fail	Partial Completion
NFR8	The system shall respond to any remote change to the framework without a direct connection.	Optional	Uncomplete

3.4 Context and Interface Requirements

Development

Aura/AuraTherm was developed with embedded hardware, IoT firmware programming, and home automation integration. The project utilized an ESP32-WROOM-32 microcontroller and was programmed using PlatformIO using the Arduino framework. Written in C++, it allows for direct integration with digital sensors and more. Testing and debugging were conducted in a lab environment with sensor simulation, serial output logging, and a relay switch. All Automation logic is handled on the device.

Deployment

In real-life deployment, AuraTherm will be a standalone embedded system. It will be installed in an indoor environment, such as bedrooms, an office, or a living space, to have consistent data. It will monitor both motion and temperature in real-time to control a relay module, telling the system to toggle the HVAC system. Integrating it with a mobile app allows for manual override or monitoring.

Table 4. Hardware Interface Requirements

Component	ESP32 Pin	Interface Type	Purpose
PIR Motion Sensor	GPIO 4	Digital Input	Detect occupancy
DHT22 Temp Sensor	GPIO 16	Digital Input	Read ambient temperature
Relay Module	GPIO 17	Digital Output	Toggle HVAC system
OLED Display	GPIO 21 (SDA), GPIO 22 (SCL)	I2C	Display current temp/status
Power Supply	N/A	USB	Provide power to ESP32

Table 5. Software Interface Requirements

Interface	Protocol	Description
ESP32 to Sensors/Relay	GPIO / I2C	Real-time control and feedback loop
ESP32 to Serial Monitor	UART	Debugging, data logging
ESP32 to Wi-Fi Router	802.11 b/g/n	Connect to LAN for data access and OTA updates
ESP32 to Mobile App	HTTP / Bluetooth	Manual override or visualization

ESP32 operates with Wi-Fi, allowing the system to connect to the user's devices and support and update remotely. Even without internet access, the device will continue to operate.

Communication Protocols

- I2C: Used for OLED, 100kHz clock
- UART: Used for debugging (Serial Monitor)

Sample JSON Format

```
{
 "temperature": 72.3,
 "humidity": 45,
 "motion": true,
 "hvac_status": "ON",
 "error_code": 0
}
```

3.5 Technology and Resource Requirements

The development and deployment of Aura/Auratherm requires a combination of hardware, software, and networking components.

Hardware Requirements

The table lists hardware components used to build and test the prototype

Table 6. Hardware Requirements

Component	Description
ESP32-WROOM-32	Central controller for sensing and logic

DHT22 Temperature Sensor	Measures temperature
PIR Motion Sensor	Detects room occupancy
Relay Module	Controls the power to an HVAC system
OLED Display	Displays current temperature and system status
Power Supply	Provides power
Jumper Wires & Breadboard	For prototyping and wiring components together
Laptop (Dev Machine)	Used for firmware development, flashing, and serial debugging

These components, from the embedded hardware, are used to sense, process, and control logic in real-world deployments.

Software Requirements
The firmware for the ESP32 is developed with modern open source tools for embedded programming.

Table 7. Software Requirements

Software Tool	Purpose
PlatformIO IDE	Embedded development environment
Arduino Framework	Libraries for ESP32 in C++
ESP32 Board Package	Enables ESP32-specific compilation, flashing, and OTA functionality
USB-to-Serial Driver	Allows communication between the development laptop and the ESP32 board
Serial Monitor	Used for logging, debugging, and sensor feedback during testing

Communication and Networking
Wi-fi is a critical component of Aura/AuraTherm system. It allows for future updates and additional features.

Table 8. Communication and Networking

Technology	Use Case
Wi-Fi	Local network connectivity for updates, debugging, or future remote access
UART	For development-time monitoring, logging, and diagnostics
I2C Protocol	Used for communication with OLED display
GPIO Digital Signals	Control relay, read PIR and DHT22 input

Textual Summary
Aura/AuraTherm was designed to function as an IoT system, using sensors and microcontrollers with integrated Wi-Fi. Development tools like PlatformIO and the Arduino framework allow for efficient embedded software development and deployment. The system operates using local logic and GPIO control, but is able to work with Wi-Fi for additional features.

Chapter 4 System Design (All Members)

4.1 Architecture Design

Figure 5. Software Diagram

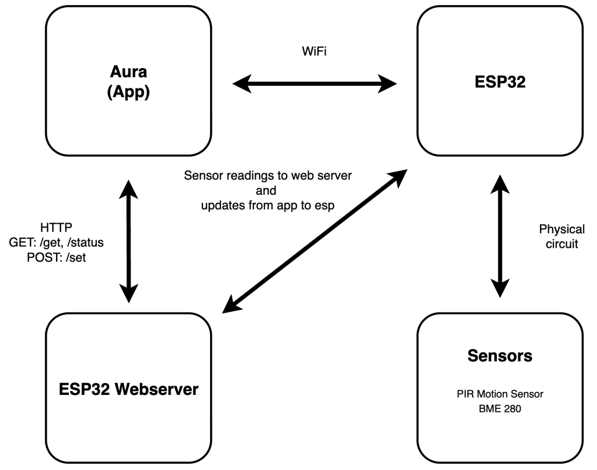
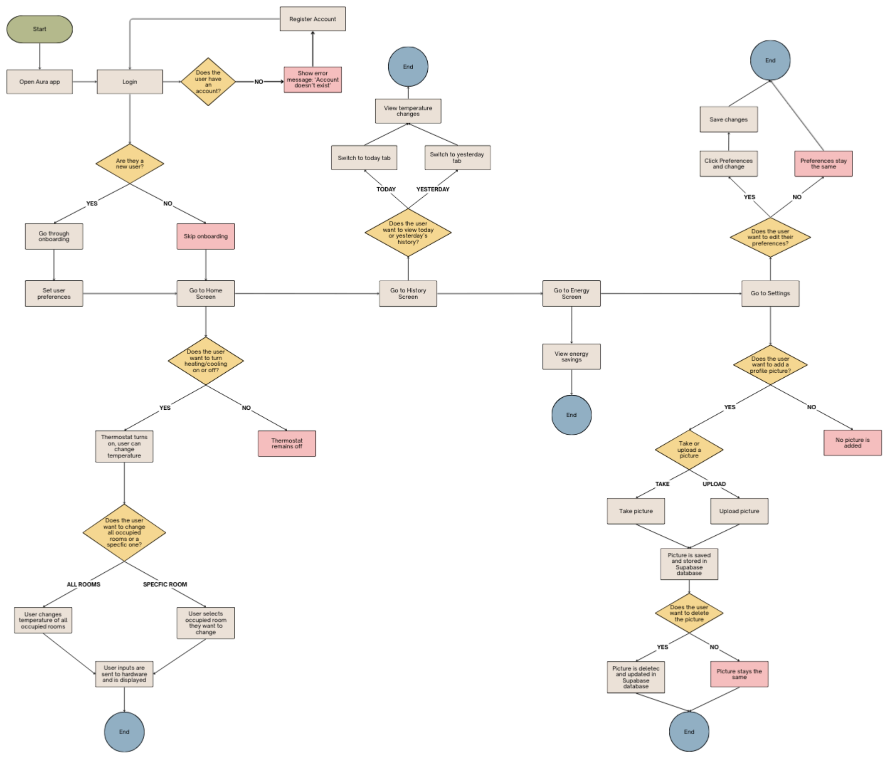


Figure 6. Data Flow Diagram

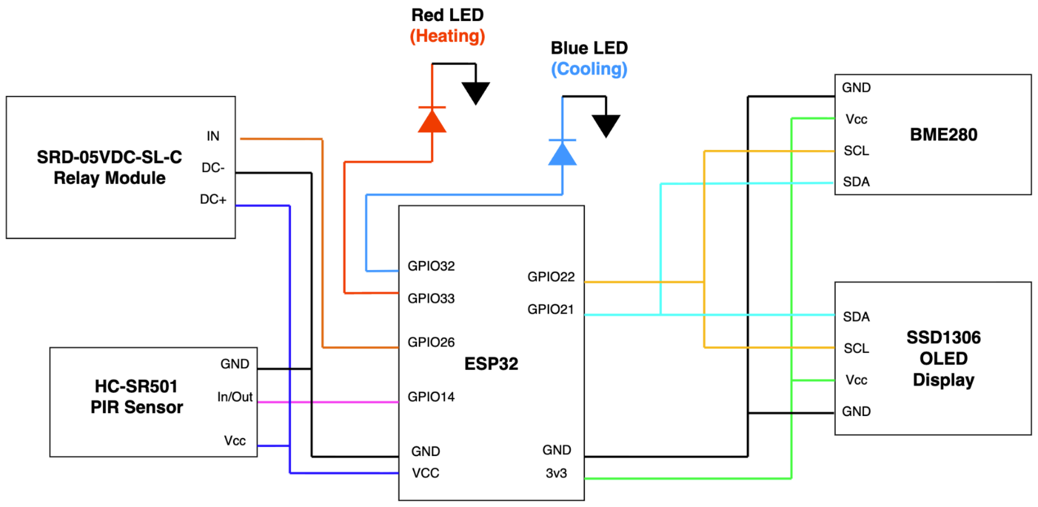


Start
Open Aura app
Login
Does the user have an account?
NO
Show error message: 'Account doesn't exist'
Register Account
Are they a new user?
YES
NO
Go through onboarding
Skip onboarding
Set user preferences
Go to Home Screen
Does the user want to turn heating/cooling on or off?
YES
NO
Thermostat turns on, user can change temperature
Thermostat remains off
Does the user want to change all occupied rooms or a specfic one?
ALL ROOMS
SPECFIC ROOM
User changes temperature of all occupied rooms
User selects occupied room they want to change
User inputs are sent to hardware and is displayed
End
Go to History Screen
Does the user want to view today or yesterday's history?
TODAY
YESTERDAY
Switch to today tab
Switch to yesterday tab
View temperature changes
End
Go to Energy Screen
View energy savings
End
Go to Settings
Does the user want to edit their preferences?
YES
NO
Click Preferences and change
Preferences stay the same
Save changes
End
Does the user want to add a profile picture?
YES
NO
No picture is added
Take or upload a picture
TAKE
UPLOAD
Take picture
Upload picture
Picture is saved and stored in Supabase database
Does the user want to delete the picture
YES
NO
Picture is deletec and updated in Supabase database
Picture stays the same
End

The system architecture consists of three main parts which include the hardware, the communication, and the application layer. At the hardware level, the ESP32 acts as the central platform, interacting with sensors, such as the PIR motion sensor and BME280 temperature sensor, along with the relay module for HVAC control and an OLED for local feedback. The communication layer enables connectivity through Wi-Fi, with the ESP32 hosting a lightweight HTTP server that handles HTTP endpoints that the app can call to send commands or request data from. Lastly, the application layer is the mobile app, which serves as the user interface. Using React Native, we were able to create an app for users to control the temperature of occupied rooms, view the history of temperature changes, and see energy savings. Moreover, the app allows users to send requests to the ESP32 to perform retrievals, updates or adjusting parameters, and JSON is used as the lightweight format for structured communication.

4.2 Interface and Component Design

Figure 7. Component Design



The system is divided into several core components that have their own role. The ESP32 serves as the central component that manages the sensors, actuators, and webserver logic. Supporting components include the BME280 temperature sensor for environmental monitoring, the PIR motion sensor for occupancy detection, a relay module to toggle HVAC systems, and an OLED display for real-time feedback. The pin connections are as follows:

- ESP32 3.3v power output to Vcc of BME280 and OLED.
- ESP32 IO22 to SCL(Serial Clock Line) of BME280 and OLED

- ESP32 IO21 to SDA(Serial Date Line) of BME280, OLED, Motion Sensor and Relay Vcc, and power supply 5v
- ESP32 IO19 to Motion sensor OUT
- ESP32 IO18 to Relay Module In
- Ground connections all together

The mobile app forms the user interaction component, offering controls for adjusting settings and also displaying the current system status. The system interfaces include hardware connections via GPIO pins for the components, software interfaces for endpoints of communication, and the user interface of the mobile app.

4.3 Structure and Logic Design

The system's structure and logic design are organized into data acquisition, control logic, and communication logic. Data acquisition involves continuous sampling of the sensors with values stored for use by the system processes. Control logic controls the HVAC system. In cooling mode, the relay activates to power the HVAC system if the measured temperature exceeds the target setpoint. Similarly, in heating mode, the relay activates if the temperature falls below the target. Motion sensor input is also used to optimize energy use by toggling only if presence is detected. The communication logic provides interaction with the mobile app via HTTP requests. Specific endpoints handle different operations, for example,/setTemp to adjust the temperature.

4.4 Design Constraints, Problems, Trade-offs, and Solutions

4.4.1 Design Constraints and Challenges

The system design faced several constraints across multiple perspectives. Economically, the thermostat had to be built using affordable yet efficient components, making the ESP32 a practical choice. Regarding resources, the ESP32 has limited memory and storage, restricting algorithm complexity and response sizes. Societal and environmental constraints required the system to be user-friendly while also contributing to energy conservation through optimized HVAC usage. Hardware and software challenges included Wi-Fi instability, ensuring real-time responsiveness, and keeping power consumption low. Finally, safety and reliability constraints were important, since the relay must not cycle too quickly, which could damage HVAC equipment, and the system must remain in a safe operating state even if communication or sensors fail.

4.4.2 Design Solutions and Trade-offs

Several solutions and trade-offs were made to address the given constraints. Economically, the ESP32 was chosen because it is a good balance of cost and performance. To manage limited resources, lightweight JSON and efficient code practices were adopted to minimize memory use. For reliability, Wi-Fi reconnection checks were implemented to prevent downtime. To protect HVAC equipment, logic was added to enforce minimum relay switch intervals. From an environmental and societal perspective, motion sensing was included to reduce energy waste when no occupancy is detected. While this can occasionally result in false negatives, the trade-off was considered acceptable due to the overall energy savings achieved.

Chapter 5 System Implementation (All Members)

5.1 Implementation Overview

The system was implemented using the ESP32 as the central hardware platform. The programming was done through the Arduino IDE using the C/C++ libraries. The dependent hardware includes the motion sensor, temperature sensor, relay module and OLED display.

5.2 Implementation of Developed Solutions

The ESP32 was programmed to continuously monitor the PIR and BME280 sensors, process data and update the relay and OLED display as needed. The motion sensor provides occupancy detection, while the temperature sensor supplies real-time environmental data. The control logic implements simple threshold based rules, such as a toggle cooling when the measured temperature exceeds the user defined setpoint. On the communication side, the ESP32's HTTP server provides endpoints, which the mobile app interacts with through HTTP requests. The responses are formatted in JSON, making it easy for the mobile app to parse and present data to the user.

5.3 Implementation Problems, Challenges, and Lesson Learned

During the implementation, one of the main challenges so far was trying to implement our previous architecture, which was using Home Assistant as the intermediary between hardware and software. After downloading the software needed for Home Assistant, we were not able to get it connected and connected to a Wi-Fi source as well. Although this approach would be more centralized and reduce coding, we switched the architecture to a direct connection, having the ESP32 create an HTTP server for the app to send requests to and allowing them to communicate. The implementation lesson learned here was just to keep things simple

Another implementation issue, with the new architecture, was connecting the ESP32 to a Wi-Fi connection. When attempting to connect it to SJSU's public wifi, it would not connect due to a possible firewall from the institution. A private wifi connection was also tried but it would not connect either. The last option was to try an iPhone hotspot with the "Maximize Compatibility" option on, and it worked. A possible reason why the public and private connections would not connect is due to the fact that our model of the ESP32 only supports 2.4Ghz of bandwidth. The implementation lesson learned here is to do more research before proceeding forward, especially with bigger decisions.

Chapter 6 Tools and Standards (All Members)

6.1. Tools Used

Hardware Tools

Hardware tools we used were a multimeter, breadboard, male-to-male/ male-to-female/female-to-female jumper wires, and a data/power cable.

The breadboard and jumper cables were used to connect the hardware components together, and we used various types to test whether to see whether the issues that we were facing were either a circuit issue, or a faulty component.

The multimeter was used to measure voltage, resistance and current, to see whether or not our components were actually supplying or reading a workable value.

The data/power cable is used to connect our laptop to the ESP32, providing it with a stable power source and connectivity to the software components and the app.

Software Tools

For the software, we decided to use Visual Studio Code, GitHub, Figma, React Native Expo, Supabase, and YouTube to build the Aura app. We made use of these tools to design and implement Aura's user interface, navigation, and core functionality.

We utilized Visual Studio Code as our development environment for writing and debugging our code, and GitHub for collaboration and project management. Before we started developing the app, we used Figma to design the user interface and experience for AuraTherm users.

After doing so, we employed React Native Expo as our main platform as it provided a wide range of libraries, was the fastest to learn, and allowed us to write code for both iOS and Android. React Native Expo was used as our framework for the Aura app and used the existing libraries to create components. Moreover, Supabase was the most cost efficient, easiest to implement, and familiar to us. In which we developed our backend services like cloud storage, authentication, and database management. When we struggled with reading documentation and needed assistance, we referenced Youtube to learn how to set up React Native and code the login, register, and account settings pages.

6.2. Standards

The standards we used for AuraTherm/Aura were geared towards software design, interface, and testing, all following hardware communication and software engineering standards to ensure interoperability, reliability, and maintainability

1. IEEE 830 – Software Requirements Specification

Structured documentation of functional and non-functional requirements to guide project milestones and validation criteria.

2. IEEE 1012 – Verification and Validation
   Applied to internal code reviews and unit-test verification of sensor input and relay output
3. IEEE 829 / ISO 29119 – Software Testing Documentation
   Informed the test-case designed format
4. Wi-Fi 802.11 b/g/n Standard
   Ensure wireless communication between ESP32 and the mobile application is reliable.
5. I²C Protocol Specification
   Used for reliable data transfer between ESP32, BME280, and OLED display
6. JSON Data Formatting Standard
   Define the structure of HTTP payloads exchanged between the firmware and mobile application.
7. Google C++ Style Guide / Arduino Core Conventions

Use consistent naming convention, indentations, and comment practices for clarity and maintainability

Chapter 7 Testing and Experiment (All Members)

7.1 Testing and Experiment Scope

Our testing focused on verifying functional accuracy, sensor responsiveness, communication reliability, and overall performance.

Two different levels of testing were conducted:

- Component (Level 1): Unit testing of individual component modules like BME280, PIR, Relay, Wi-Fi, and ESP32 module
- System (Level 2): Integrating testing of sensor data flow from ESP32 to mobile app through HTTP request

The key objective are:

1. Validate temperature and motion readings in real time.
2. Verify that HVAC toggling responds to occupancy events.
3. Evaluate latency between motion detection and temperature changes on the app.
4. Ensure the system remains stable under continuous operation.

7.2 Testing and Experiment Approach

To ensure system reliability and cooperation with requirements, a bottom up testing approach was implemented. Testing started at the component level and moved towards full system integration.

Test Strategy

Manual black box testing combined with instrumented observation were utilized to ensure component and system level synthesis. Through component level testing, sensor outputs and communication accuracy were verified. Whereas, system level testing proved data flow between hardware and the mobile app, Aura.

Test Methods and Techniques

1. Unit Testing at the Component Level
   Each sensor module, listed below, was tested individually on the ESP32 using the Arduino Serial Monitor. Test inputs were simulated by manually triggering motion and adjusting temperature sources.
   - BMPE280 for temperature/humidity readings
   - PIR for motion detection
   - Relay for HVAC control switching

2. Integration Testing at the System Level
   After individual testing passed all checks, all components were integrated. From which testing data transmission from the ESP32 to the mobile app was tested through HTTP requests to ensure consistent updates. Moreover, the React Native

frontend and Supabase backend were certified for displaying sensor data and handling correctly.

3. Performance Testing
   The measured latency between motion detection and the displayed temperature change on the mobile app were evaluated. Repeated evaluations were performed under various WiFi conditions to determine stability.

Requirements Alignment
All testing was performed to verify compliance with the non-functional requirements defined in Section 3. Testing confirmed that:

Table 9. Non-Functional Requirements and Measurable Performance Targets

NFR ID	Description	Measurable Target
NFR1	The control loop response latency will be less than 200-500 milliseconds for motion detection on	≤ 200-500 ms latency
NFR2	The control loop response latency will be less than 5-10 seconds for motion detection off	≤ 5-10s latency
NFR3	Based on test scenarios, projected to reduce HVAC energy usage by 10–15% in occupied test environments.	≤ 100mA
NFR4	Relay enforces a 2-minute minimum off-time and 5-minute on-time to protect HVAC hardware.	≥ 180s cycle buffer
NFR5	The system will maintain the temperature within ±1.0°C of the preferred settings.	±1.0°C / ±1.8°F
NFR6	The system shall be able to operate locally without requiring the internet.	Local-only fallback
NFR7	System shall recover from a crash or error using an auto-reset, retrying every 10 seconds	100% restart on fail
NFR8	The system shall respond to any remote change to the framework without a direct connection.	Optional

7.3 Testing and Experiment Results and Analysis

Table 10. Non-Functional Requirement Testing Results and Performance

NFR ID	Description	Measurable Target	Result
NFR1	The control loop response latency will be less than 200-500 milliseconds for motion detection on	≤ 200-500 ms latency	Passed
NFR2	The control loop response latency will be less than 5-10 seconds for motion detection off	≤ 5-10s latency	Passed
NFR3	Based on test scenarios, projected to reduce HVAC energy usage by 10–15% in occupied test environments.	≤ 100mA	Passed
NFR4	Relay enforces a 2-minute minimum off-time and 5-minute on-time to protect HVAC hardware.	≥ 180s cycle buffer	Removed
NFR5	The system will maintain the temperature within ±1.0°C of the preferred settings.	±1.0°C	Passed
NFR6	The system shall be able to operate locally without requiring the internet.	Local-only fallback	Passed
NFR7	System shall recover from a crash or error using an auto-reset, retrying every 10 seconds	100% restart on fail	Passed
NFR8	The system shall respond to any remote change to the framework without a direct connection.	Optional	Future Work

The testing done this far has been the accuracy of the temperature sensors, local operability and crash recovery. For testing the BME280 sensor, we ran a total of 3 tests with 3 different sensors, each with the same theoretical circuit.

Table 11. BME280 Testing results

Test Number	Result	Issue	Temperature Reading Accuracy	Pass/Fail

T1	The sensor reads the current temperature of the room when pins are at a certain angle for BME.	N/A	±1°F	Pass
T2	The sensor connects to the ESP32.	N/A	N/A	Pass
T3	Sensor detected at address 0x76, and provides fresh readings. Since pinholes are larger for this specific BME, connecting pins were bent to ensure the sensor stays connected.	Larger pinholes require the pins connection to BME280 T3 must be bent to ensure stable connection.	±1.0°F	Pass (Conditionally) Sensors must not be touched, or else connection might be lost. Easily fixed by readjusting the angle of BME connection to pins.

Table 12. PIR Motion Sensor Detection Timing & Behavior

Test Case	Scenario	Sensor Triggered?	Time for raw = 1 (Motion detected)	Time for raw = 0 (No motion)	Observations	Pass/Fail
T1	The human body enters the detection zone, stays, then leaves.	Yes	100-500ms (instant)	5 - 10s	Reliable/real-time detection	Pass
T2	Humans pass through zones.	Yes	100-500ms (instant)	5 - 10s	Detects entry, then remains HIGH for hold time	Pass

					after exit	
T3	Inanimate object moves	No (Should not)	N/A	N/A	No false triggers from inanimate objects	Pass

Table 13. HTTP Request Latency

HTTP Request Type	ESP processing	Wifi Transmission	App Update	Total RTT
GET /status	2 - 5ms	1 - 5ms	<2ms	3 - 12ms
POST /motion/set	2 - 5ms	1 - 5ms	<2ms	3 - 12ms

Chapter 8 Conclusion and Future Work (All Members)

8.1 Conclusion

The Aura/AuraTherm project was successfully able to demonstrate a functional prototype of a smart thermostat that was able to integrate occupancy detection, temperature sensing, and wireless communication for specific rooms in order to optimize energy consumption in different environments. By using the combination of ESP32 microcontroller, BME280 environmental sensors, PIR motion sensor, and relay module, the system was able to dynamically adjust HVAC operations based on real-time occupancy and temperature data.

Through implementation and testing, AuraTherm demonstrated the capability to reliably detect motion, maintain temperature within ±1 °C of the user-defined temperature, and transmit sensor updates to the Aura mobile app with an average latency of approximately 1 second. These results aligned with the project's main goals: reducing energy consumption, lowering utility costs, and promoting environmental sustainability.

The mobile companion app, built using React Native Expo and Supabase, provided an interface for real-time monitoring and control, demonstrating how embedded IoT systems and cloud-connected applications can work together. Despite several challenges, such as limited Wi-Fi bandwidth, component wiring constraints, and the learning curve of embedded programming, the team implemented practical solutions, including simplified HTTP-based communication and a reliable power-cycle delay for HVAC protection.

Overall, AuraTherm achieved its objective of providing an affordable, modular, and scalable smart-home device capable of adapting to users' behaviors while promoting energy efficiency and sustainability.

8.2 Future Work

While the prototype met the function requirements, several things can be further improved for performance, scalability, and usability:

1. Multi-Room Expansion - Extend AuraTherm to support multiple notes that share occupancy and temperature data in different rooms, creating a coordinated, house-wide control system.
2. Machine-Learning-Based Adaptation - Integrate a predictive algorithm to learn user habits, patterns, and preferences, allowing for proactive HVAC changes.
3. Voice-Assistant Integration - Add compatibility with Google Home, Alexa, and Apple HomeKit using MQTT or Matter standards for a broader smart-home compatibility.
4. Energy-Analytics Dashboard - Develop a web-based interface that tracks power savings, usage trends, and estimated energy savings for users.
5. Hardware optimization - Transition from breadboard prototyping to a custom PCB design for compactness, less noise, and improved reliability

6. Security - Incorporate HTTPS encryption and user authentication at the firmware level to ensure secure data transmission and device protection.

By implementing these improvements, AuraTherm can go from a functional prototype to a commercial-grade smart thermostat platform that contributes to the global push for a sustainable, intelligent home automation system.

References

Ayan, Onur, and Belgin Turkay. *Smart Thermostats for Home Automation Systems and Energy Savings from Smart Thermostats*. 1 Oct. 2018, pp. 1–6, ieeexplore.ieee.org/document/8751790, https://doi.org/10.1109/ceit.2018.8751790. Accessed 11 Dec. 2025.

Beltran, Alex, et al. “ThermoSense.” *Proceedings of the 5th ACM Workshop on Embedded Systems for Energy-Efficient Buildings*, 11 Nov. 2013, pp. 1–8, dl.acm.org/doi/10.1145/2528282.2528301, https://doi.org/10.1145/2528282.2528301. Accessed 11 Dec. 2025.

Carli, Raffaele, et al. “IoT Based Architecture for Model Predictive Control of HVAC Systems in Smart Buildings.” *Sensors*, vol. 20, no. 3, 31 Jan. 2020, p. 781, www.mdpi.com/1424-8220/20/3/781, https://doi.org/10.3390/s20030781. Accessed 11 Dec. 2025.

Ehsanifar, Mohammad, et al. “A Sustainable Pattern of Waste Management and Energy Efficiency in Smart Homes Using the Internet of Things (IoT).” *Sustainability*, vol. 15, no. 6, 13 Mar. 2023, pp. 5081–5081, www.mdpi.com/2071-1050/15/6/5081, https://doi.org/10.3390/su15065081. Accessed 11 Dec. 2025.

Patel, Shwetak, et al. “Detecting Human Movement by Differential Air Pressure Sensing in HVAC System Ductwork: An Exploration in Infrastructure Mediated Sensing.” *LNCS*, vol. 5013, 2008, pp. 1–18, aiweb.cs.washington.edu/research/projects/aiweb/media/papers/tmpqiqC-3.pdf. Accessed 11 Dec. 2025.

Santos, Sara. “ESP32 Pinout Reference: Which GPIO Pins Should You Use? | Random Nerd Tutorials.” *Random Nerd Tutorials*, 2 Oct. 2019, randomnerdtutorials.com/esp32-pinout-reference-gpios/. Accessed 11 Dec. 2025.

Stepnitz, Alston, et al. *Title Connected Thermostats for Low Income Households: Insights from User Testing Permalink Https://Escholarship.org/Uc/Item/61k1c32x Publication Date*.

Appendices

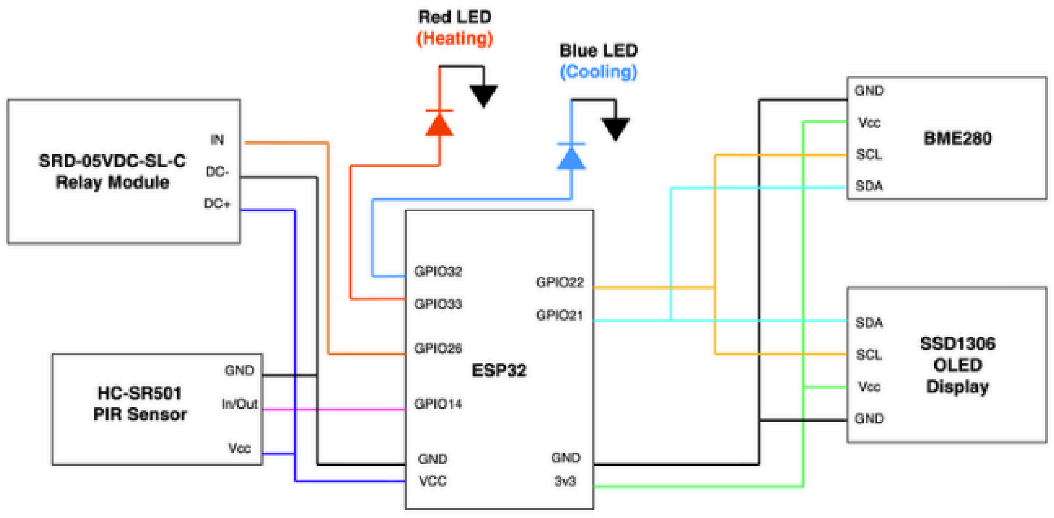
Appendix A – Hardware Schematics and Pin Assignments

A.1 Hardware Overview

The AuraTherm prototype consists of an ESP32 microcontroller connected to the components:

- PIR Motion Sensor (HC-SR501) – detects occupancy
- BME280 Environmental Sensor – measures temperature, humidity, and pressure
- SSD1306 OLED Display – displays real-time environmental information and status
- ESP32 Wi-Fi Module – sends data to the Aura mobile app

This hardware configuration supports occupancy-based monitoring and sensor-driven data reporting.



A.2 Pin Mapping Table

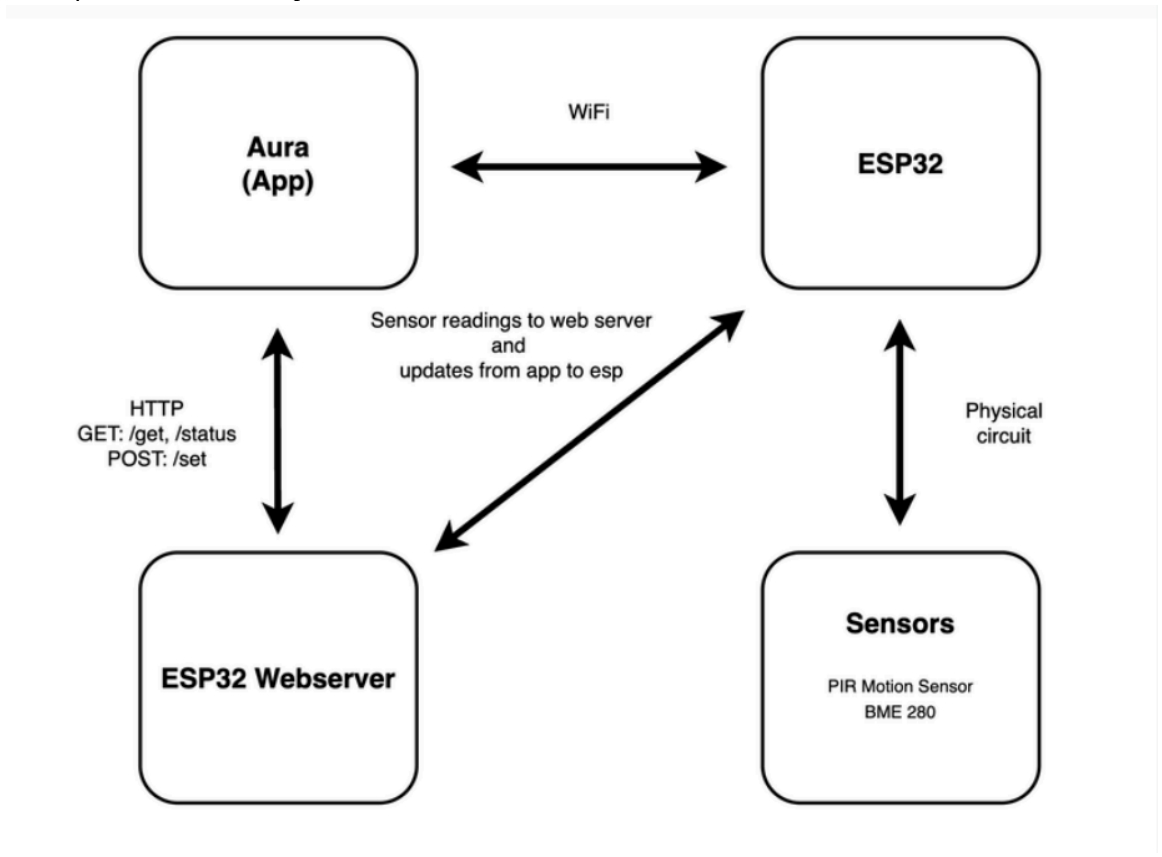
Component	ESP32 Pin	Interface Type	Purpose
PIR Motion Sensor (OUT)	GPIO 19	Digital Input	Detects occupancy
PIR VCC / GND	3.3V / GND	Power	Sensor power
BME280 SDA	GPIO 21	I²C	Data line

BME280 SCL	GPIO 22	I²C	Clock line
OLED SDA	GPIO 21	I²C	Shared data line
OLED SCL	GPIO 22	I²C	Shared data line
ESP32 Power	USB 5V	Power	Main system power

A.3 Wiring Notes & Considerations

- The BME280 and OLED share I²C lines (SDA/SCL) since they operate at different I²C addresses.
- The PIR motion sensor operates at 3.3V, ensuring compatibility with ESP32 GPIO.
- All ground connections are tied together to ensure signal stability.
- BME280 modules may vary, so stable pin seating is vital.
- The OLED connection must be firm to avoid any flickering

A.4 System Block Diagram


Aura (App)
WiFi
ESP32
Sensor readings to web server and updates from app to esp
HTTP GET: /get, /status POST: /set
Physical circuit
ESP32 Webserver
Sensors
PIR Motion Sensor
BME 280

Appendix B – ESP32 Firmware Source Code
Github: https://github.com/sabrinaquach/AuraTherm-Aura
The rest of the source code is in the Github repository. Main -> Arduino
Main source code:

- AuraTherm.ino

```
#include "WiFiManager.h"
#include "ThermostatAPI.h"
#include "DisplayUI.h"
#include "PIRSensor.h"
#include <WebServer.h>

// ------- PIR config -------
static constexpr uint8_t  PIR_GPIO      = 14;
static constexpr uint32_t PIR_WARMUP_MS = 30 * 1000UL;
PIRSensor pir;

// ------- Display pacing -------
static uint32_t lastOLED = 0;
static uint32_t lastDbg  = 0;

// server in ThermostatAPI
extern WebServer server;
extern bool wifiConnected;
extern bool motionEnabled;

bool motionRaw() {
    return pir.raw();
}

void setup() {
  Serial.begin(115200);
  delay(200);
  Serial.println("\n[AuraTherm] Booting...");

  // 1) Wi-Fi
  setupWiFi();
  Serial.println("[AuraTherm] Waiting for Wi-Fi...");

  // 2) Sensors (BME280 on I2C)
  temp_setup();

  // 3) HTTP API
  setupAPI();

  // 3a) RAW motion endpoint
  server.on("/motion", HTTP_GET, []() {
    bool raw = motionEnabled ? digitalRead(PIR_GPIO) : false;

    String body = "{";
    body += "\"motion_detected\":";
    body += raw ? "true" : "false";
    body += "}";

    server.send(200, "application/json", body);
  });

  // 4) OLED
  display_init(0x3C);

  // 5) PIR init
  pir.begin(PIR_GPIO, /*usePulldown=*/true,
/*warmupMs=*/PIR_WARMUP_MS);
  Serial.printf("[PIR] Using GPIO %u | warm-up %lus\n",
              PIR_GPIO, PIR_WARMUP_MS / 1000);

  //LED setup
  pinMode(32, OUTPUT);  // Blue LED
  pinMode(33, OUTPUT);  // Red LED
  digitalWrite(32, LOW); // Blue LED off initially
  digitalWrite(33, LOW); // Red LED off initially

  Serial.println("[AuraTherm] Ready.");
}

void loop() {
  uint32_t now = millis();
  checkWiFi();
  server.handleClient();

  float tempF, tgtF;
  String mode;

  // -------- RAW PIR CONTROL MODEL --------
  bool raw = false;
  if (motionEnabled) {
   raw = digitalRead(PIR_GPIO);
  }

  // ----- AUTOMATIC HVAC EFFECT (NOT CONTROL) -----
  static String effectiveMode = "Off";

  if (motionEnabled) {

      hvacMode = "Heating/Cooling"; //restore auto mode when motionEnabled =
true
      if (raw) {
          if (api_getSnapshot(tempF, tgtF, mode)) {
              effectiveMode = mode;
          } else {
              effectiveMode = "Off";
              hvacMode = "Off";
          }
      } else {
          effectiveMode = "Off";   // motion enabled, no motion → force off
          hvacMode = "Off";
      }

  } else {
      // motion disabled → trust API fully (manual mode)
      if (api_getSnapshot(tempF, tgtF, mode)) {
          effectiveMode = mode;
      } else {
          effectiveMode = "Off";
      }
  }

  // ----- LED STATUS INDICATORS -----

  // Default OFF
  digitalWrite(32, LOW); // Blue OFF
  digitalWrite(33, LOW); // Red OFF

  if (effectiveMode == "Cool") {
      digitalWrite(32, HIGH); // Blue ON (cooling)
  }
  else if (effectiveMode == "Heat") {
      digitalWrite(33, HIGH); // Red ON (heating)
  }
  // else → Off → both off



  // -------- DEBUG HEARTBEAT --------
  if (now - lastDbg >= 1000) {
    lastDbg = now;

    if (motionEnabled) {
      Serial.printf("[HB] %lus | PIR raw=%d\n", now / 1000, raw);
    } else {
      Serial.printf("[HB] %lus | PIR DISABLED\n", now / 1000);
    }
  }

  // --- OLED refresh every 1000 ms ---
  if (now - lastOLED >= 1000) {
    lastOLED = now;

    if (api_getSnapshot(tempF, tgtF, mode)) {
      display_update(tempF, tgtF, mode);
    }
  }

  delay(5);
}
```

- ThermostatAPI.cpp

```
#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_BME280.h>
#include <WebServer.h>

#include "ThermostatAPI.h"
#include "PIRSensor.h"
#include "DisplayUI.h"

// ----- History -----
struct HistoryEvent {
  String type;
  String timestamp;
};
```

```
HistoryEvent historyLog[50];
int historyIndex = 0;

String timestamp() {
  unsigned long ms = millis();
  unsigned long sec = ms / 1000;
  unsigned long min = sec / 60;
  unsigned long hr  = min / 60;

  char buffer[20];
  sprintf(buffer, "%02lu:%02lu:%02lu", hr % 24, min % 60, sec % 60);
  return String(buffer);
}

void pushHistory(String type) {
  historyLog[historyIndex % 50] = { type, timestamp() };
  historyIndex++;
}

// ----- Globals -----
WebServer server(80);
Adafruit_BME280 bme;
bool bmeInitialized = false;
bool motionEnabled = true;

float  targetTempF = 72.0f;
String hvacMode    = "Off";  // "Heating" / "Cooling" / "Off"

// ----- Helpers -----
static inline float c_to_f(float c) { return c * 9.0f / 5.0f + 32.0f; }

// ----- Sensor setup -----
void temp_setup() {
  // ESP32 I2C pins
  Wire.begin(21, 22);
  Wire.setClock(100000);

  // Try both addresses
  if (bme.begin(0x76, &Wire) || bme.begin(0x77, &Wire)) {
    bmeInitialized = true;

    // sampling config
    bme.setSampling(
    Adafruit_BME280::MODE_NORMAL,
    Adafruit_BME280::SAMPLING_X1,   // temp
    Adafruit_BME280::SAMPLING_X1,   // pressure
    Adafruit_BME280::SAMPLING_X1,   // humidity
    Adafruit_BME280::FILTER_OFF,
    Adafruit_BME280::STANDBY_MS_0_5
   );

   Serial.println(F("[BME280] Initialized"));
  } else {
   Serial.println(F("[BME280] NOT found on 0x76/0x77"));
  }
}

// ----- Tiny JSON helpers (no ArduinoJson needed) -----
static String jsonKV(const char* k, const String& v, bool last=false) {
  String s = "\""; s += k; s += "\":\""; s += v; s += "\"";
  if (!last) s += ",";
  return s;
}

static String jsonKV(const char* k, float v, bool last=false, uint8_t digits=1) {
  String s = "\"";
  s += k;
  s += "\":";
  // Force the correct String() overload: (double, unsigned int)
  s += String((double)v, (unsigned int)digits);
  if (!last) s += ",";
  return s;
}

// ----- History tracking globals -----
static float lastTempF = NAN;
static bool  lastMotion = false;

// ----- /status handler -----
static void handleStatus() {
  float tempF, tgtF;
  String mode;

  bool ok = api_getSnapshot(tempF, tgtF, mode);

  String out = "{";
  if (ok) {
   out += jsonKV("currentTemp", tempF);
   out += jsonKV("targetTemp",  tgtF);
   out += jsonKV("mode",        mode,   true);

   bool motionNow = motionEnabled ? motionRaw() : false;
   out += ",\"motionEnabled\":";
   out += motionEnabled ? "true" : "false";

   out += ",\"motionDetected\":";
   out += motionNow ? "true" : "false";

   // ----- History Logging -----
   // Temp change
   if (isnan(lastTempF) || fabs(tempF - lastTempF) >= 0.1f) { // log if change >=
0.1°F
        pushHistory("TempChange");
        lastTempF = tempF;
     }

   // Motion detected
   if (motionNow != lastMotion) {
     pushHistory(motionNow ? "MotionDetected" : "MotionCleared");
   }
   lastMotion = motionNow;

  } else {
   out += jsonKV("error", "sensor_unavailable", true);
  }
  out += "}";

  // update OLED on each /status hit so API & screen match
  if (ok && display_ok()) {
   display_update(tempF, tgtF, mode);
  }

  server.send(200, "application/json", out);
}

// ----- /i2c-scan  -----
static void handleI2CScan() {
  byte count = 0;
  String out = "[";
  for (byte address = 1; address < 127; address++) {
    Wire.beginTransmission(address);
    if (Wire.endTransmission() == 0) {
      if (count++) out += ",";
      out += "\"0x" + String(address, HEX) + "\"";
    }
  }
  out += "]";
  server.send(200, "application/json", out);
}

// ----- /set handler -----
static void handleSet() {
    if (!server.hasArg("plain")) {
        server.send(400, "application/json", "{\"error\":\"no_body\"}");
        return;
    }

    String body = server.arg("plain");
    Serial.println("[API] Incoming JSON: " + body);

    // Parse targetTemp
    int tIdx = body.indexOf("targetTemp");
    if (tIdx >= 0) {
        int colon = body.indexOf(":", tIdx);
        int comma = body.indexOf(",", colon);
        if (comma < 0) comma = body.indexOf("}", colon);
        targetTempF = body.substring(colon + 1, comma).toFloat();
        Serial.println("Updated targetTempF: " + String(targetTempF));
    }

    // Parse hvacMode
    int mIdx = body.indexOf("mode");
    if (mIdx >= 0) {
        int colon = body.indexOf(":", mIdx);
        int quote1 = body.indexOf("\"", colon + 1);
        int quote2 = body.indexOf("\"", quote1 + 1);
        hvacMode = body.substring(quote1 + 1, quote2);
        Serial.println("Updated hvacMode: " + hvacMode);
    }

    server.send(200, "application/json", "{\"status\":\"ok\"}");
}
```

```
// ----- /motion/set handler -----
static void handleMotionSet() {
  if (!server.hasArg("plain")) {
    server.send(400, "application/json", "{\"error\":\"no_body\"}");
    return;
  }

  String body = server.arg("plain");
  Serial.println("[Motion] RAW: " + body);

  if (body.indexOf("true") >= 0)  motionEnabled = true;
  if (body.indexOf("false") >= 0) motionEnabled = false;

  Serial.println("[Motion] motionEnabled = " + String(motionEnabled));

  server.send(200, "application/json", "{\"status\":\"ok\"}");
}

// ----- /history handler -----
static void handleHistory() {
  String out = "[";
  int count = min(historyIndex, 50);

  for (int i = 0; i < count; i++) {
    int idx = (historyIndex - count + i) % 50;
    out += "{";
    out += jsonKV("type", historyLog[idx].type);
    out += jsonKV("time", historyLog[idx].timestamp, true);
    out += "}";
    if (i < count - 1) out += ",";
  }
  out += "]";
  server.send(200, "application/json", out);
}

// ----- Server setup -----
void setupAPI() {
  server.on("/status",   HTTP_GET, handleStatus);
  server.on("/set", HTTP_ANY, handleSet);
  server.on("/setTemp",  HTTP_POST, handleSet);
  server.on("/motion/set", HTTP_POST, handleMotionSet);
  server.on("/i2c-scan", HTTP_GET, handleI2CScan);
  server.on("/history", HTTP_GET, handleHistory);
  server.begin();
  Serial.println(F("[HTTP] server started"));
}

// ----- Public snapshot for OLED / dashboard callers -----
bool api_getSnapshot(float& tempF, float& targetF, String& mode)
{
    if (!bmeInitialized) return false;

    float tC = bme.readTemperature();
    if (isnan(tC)) return false;

    tempF   = c_to_f(tC);
    targetF = targetTempF;

    const float delta = tempF - targetTempF;
    const float HYST = 0.25f;

    if (hvacMode == "Off") {
        mode = "Off";
    } else if (hvacMode == "Heating/Cooling") {
        if (delta < -HYST)      mode = "Heat";
        else if (delta > HYST)  mode = "Cool";
        else                    mode = "Off";
    } else {
        mode = hvacMode; // respect manual Heat/Cool
    }

    return true;
}
```

- ThermostatAPI.h

```
#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_BME280.h>
#include <WebServer.h>

#include "ThermostatAPI.h"
#include "PIRSensor.h"
#include "DisplayUI.h"

// ----- History -----
```

```
struct HistoryEvent {
  String type;
  String timestamp;
};

HistoryEvent historyLog[50];
int historyIndex = 0;

String timestamp() {
  unsigned long ms = millis();
  unsigned long sec = ms / 1000;
  unsigned long min = sec / 60;
  unsigned long hr  = min / 60;

  char buffer[20];
  sprintf(buffer, "%02lu:%02lu:%02lu", hr % 24, min % 60, sec % 60);
  return String(buffer);
}

void pushHistory(String type) {
  historyLog[historyIndex % 50] = { type, timestamp() };
  historyIndex++;
}

// ----- Globals -----
WebServer server(80);
Adafruit_BME280 bme;
bool bmeInitialized = false;
bool motionEnabled = true;

float  targetTempF = 72.0f;
String hvacMode    = "Off";  // "Heating" / "Cooling" / "Off"

// ----- Helpers -----
static inline float c_to_f(float c) { return c * 9.0f / 5.0f + 32.0f; }

// ----- Sensor setup -----
void temp_setup() {
  // ESP32 I2C pins
  Wire.begin(21, 22);
  Wire.setClock(100000);

  // Try both addresses
  if (bme.begin(0x76, &Wire) || bme.begin(0x77, &Wire)) {
    bmeInitialized = true;

    // sampling config
    bme.setSampling(
      Adafruit_BME280::MODE_NORMAL,
      Adafruit_BME280::SAMPLING_X1,   // temp
      Adafruit_BME280::SAMPLING_X1,   // pressure
      Adafruit_BME280::SAMPLING_X1,   // humidity
      Adafruit_BME280::FILTER_OFF,
      Adafruit_BME280::STANDBY_MS_0_5
    );

    Serial.println(F("[BME280] Initialized"));
  } else {
    Serial.println(F("[BME280] NOT found on 0x76/0x77"));
  }
}

// ----- Tiny JSON helpers (no ArduinoJson needed) -----
static String jsonKV(const char* k, const String& v, bool last=false) {
  String s = "\""; s += k; s += "\":\""; s += v; s += "\"";
  if (!last) s += ",";
  return s;
}

static String jsonKV(const char* k, float v, bool last=false, uint8_t digits=1) {
  String s = "\"";
  s += k;
  s += "\":";
  // Force the correct String() overload: (double, unsigned int)
  s += String((double)v, (unsigned int)digits);
  if (!last) s += ",";
  return s;
}

// ----- History tracking globals -----
static float lastTempF = NAN;
static bool  lastMotion = false;

// ----- /status handler -----
static void handleStatus() {
  float tempF, tgtF;
  String mode;

  bool ok = api_getSnapshot(tempF, tgtF, mode);

  String out = "{";
  if (ok) {
    out += jsonKV("currentTemp", tempF);
    out += jsonKV("targetTemp",  tgtF);
    out += jsonKV("mode",        mode,   true);

    bool motionNow = motionEnabled ? motionRaw() : false;
    out += ",\"motionEnabled\":";
    out += motionEnabled ? "true" : "false";

    out += ",\"motionDetected\":";
    out += motionNow ? "true" : "false";

    // ----- History Logging -----
    // Temp change
    if (isnan(lastTempF) || fabs(tempF - lastTempF) >= 0.1f) { // log if change >=
0.1°F
        pushHistory("TempChange");
        lastTempF = tempF;
     }

    // Motion detected
    if (motionNow != lastMotion) {
      pushHistory(motionNow ? "MotionDetected" : "MotionCleared");
    }
    lastMotion = motionNow;

  } else {
    out += jsonKV("error", "sensor_unavailable", true);
  }
  out += "}";

  // update OLED on each /status hit so API & screen match
  if (ok && display_ok()) {
    display_update(tempF, tgtF, mode);
  }

  server.send(200, "application/json", out);
}
```

```
// ----- /i2c-scan  -----
static void handleI2CScan() {
  byte count = 0;
  String out = "[";
  for (byte address = 1; address < 127; address++) {
    Wire.beginTransmission(address);
    if (Wire.endTransmission() == 0) {
      if (count++) out += ",";
      out += "\"0x" + String(address, HEX) + "\"";
    }
  }
  out += "]";
  server.send(200, "application/json", out);
}

// ----- /set handler -----
static void handleSet() {
    if (!server.hasArg("plain")) {
        server.send(400, "application/json", "{\"error\":\"no_body\"}");
        return;
    }

    String body = server.arg("plain");
    Serial.println("[API] Incoming JSON: " + body);

    // Parse targetTemp
    int tIdx = body.indexOf("targetTemp");
    if (tIdx >= 0) {
        int colon = body.indexOf(":", tIdx);
        int comma = body.indexOf(",", colon);
        if (comma < 0) comma = body.indexOf("}", colon);
        targetTempF = body.substring(colon + 1, comma).toFloat();
        Serial.println("Updated targetTempF: " + String(targetTempF));
    }

    // Parse hvacMode
    int mIdx = body.indexOf("mode");
    if (mIdx >= 0) {
        int colon = body.indexOf(":", mIdx);
        int quote1 = body.indexOf("\"", colon + 1);
        int quote2 = body.indexOf("\"", quote1 + 1);
        hvacMode = body.substring(quote1 + 1, quote2);
        Serial.println("Updated hvacMode: " + hvacMode);
    }

    server.send(200, "application/json", "{\"status\":\"ok\"}");
}

// ----- /motion/set handler -----
static void handleMotionSet() {
  if (!server.hasArg("plain")) {
    server.send(400, "application/json", "{\"error\":\"no_body\"}");
    return;
  }

  String body = server.arg("plain");
  Serial.println("[Motion] RAW: " + body);

  if (body.indexOf("true") >= 0)  motionEnabled = true;
  if (body.indexOf("false") >= 0) motionEnabled = false;

  Serial.println("[Motion] motionEnabled = " + String(motionEnabled));

  server.send(200, "application/json", "{\"status\":\"ok\"}");
}

// ----- /history handler -----
static void handleHistory() {
  String out = "[";
  int count = min(historyIndex, 50);

  for (int i = 0; i < count; i++) {
    int idx = (historyIndex - count + i) % 50;
    out += "{";
    out += jsonKV("type", historyLog[idx].type);
    out += jsonKV("time", historyLog[idx].timestamp, true);
    out += "}";
    if (i < count - 1) out += ",";
  }
  out += "]";
  server.send(200, "application/json", out);
}

// ----- Server setup -----
void setupAPI() {
  server.on("/status",   HTTP_GET, handleStatus);
  server.on("/set", HTTP_ANY, handleSet);
  server.on("/setTemp",  HTTP_POST, handleSet);
  server.on("/motion/set", HTTP_POST, handleMotionSet);
  server.on("/i2c-scan", HTTP_GET, handleI2CScan);
  server.on("/history", HTTP_GET, handleHistory);
  server.begin();
  Serial.println(F("[HTTP] server started"));
}

// ----- Public snapshot for OLED / dashboard callers -----
bool api_getSnapshot(float& tempF, float& targetF, String& mode)
{
   if (!bmeInitialized) return false;

   float tC = bme.readTemperature();
   if (isnan(tC)) return false;

   tempF   = c_to_f(tC);
   targetF = targetTempF;

   const float delta = tempF - targetTempF;
   const float HYST = 0.25f;

   if (hvacMode == "Off") {
      mode = "Off";
   } else if (hvacMode == "Heating/Cooling") {
      if (delta < -HYST)      mode = "Heat";
      else if (delta > HYST)  mode = "Cool";
      else                    mode = "Off";
   } else {
      mode = hvacMode; // respect manual Heat/Cool
   }

   return true;
}
```

Appendix C – Mobile App (Aura App) Documentation

C.1 App Architecture

Aura is built using:

- React Native Expo for cross platform mobile app development
- Supabase for user authentication and cloud database
- Fetch API for HTTP communication with the ESP32

C.2 API Integration Example

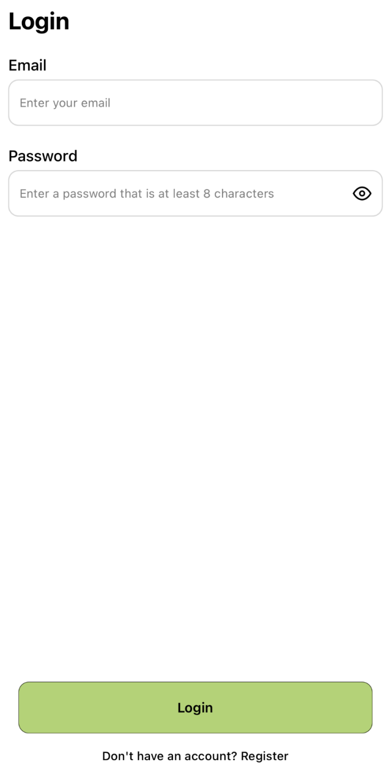
```
async function getStatus() {
  const res = await fetch("http://DEVICE_IP/status");
  const data = await res.json();
  return data;
}
```

C.3 App Features Included

- Login/Register with Supabase
- Real-time temperature monitoring
- Occupancy status display
- Manual setpoint adjustment
- HVAC state indicator
- Auto-refresh functionality

C.4 Sample Screenshots

- Login and Register Screen



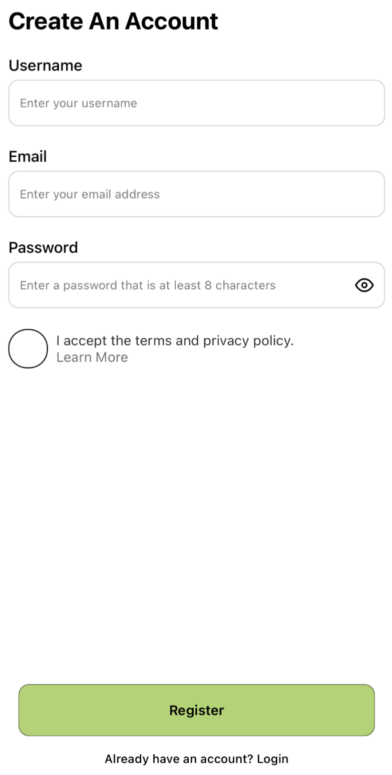
Login

Email

Enter your email

Password

Enter a password that is at least 8 characters



Create An Account

Username

Enter your username

Email

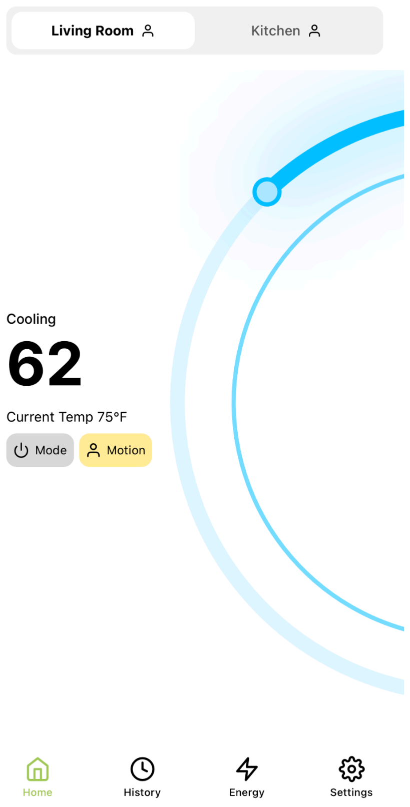
Enter your email address

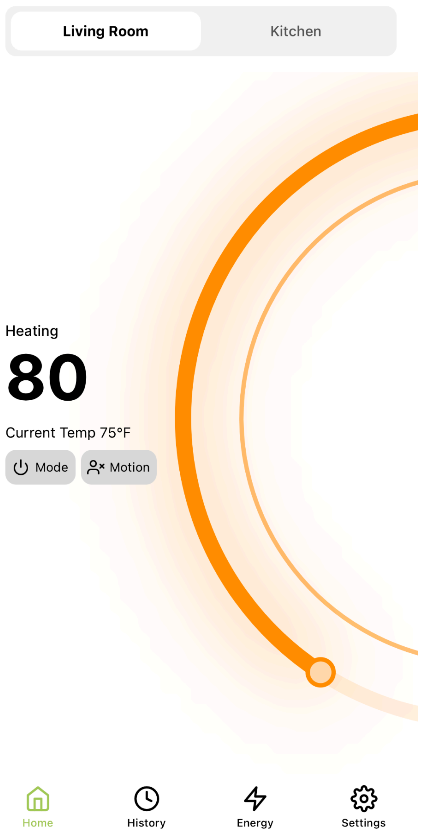
Password

Enter a password that is at least 8 characters

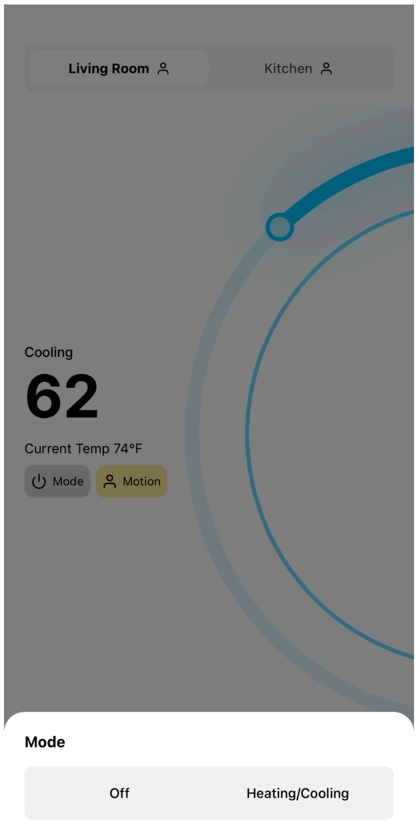
I accept the terms and privacy policy.
Learn More

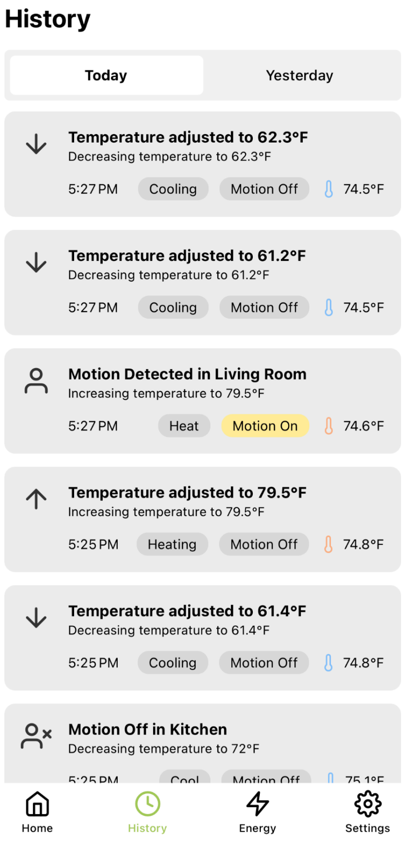
- Home/Temperature Control Screen (Motion On/Off)





- Mode Status Modal and History Screen





History

Today | Yesterday

Temperature adjusted to 62.3°F
Decreasing temperature to 62.3°F
5:27 PM Cooling Motion Off 74.5°F

Temperature adjusted to 61.2°F
Decreasing temperature to 61.2°F
5:27 PM Cooling Motion Off 74.5°F

Motion Detected in Living Room
Increasing temperature to 79.5°F
5:27 PM Heat Motion On 74.6°F

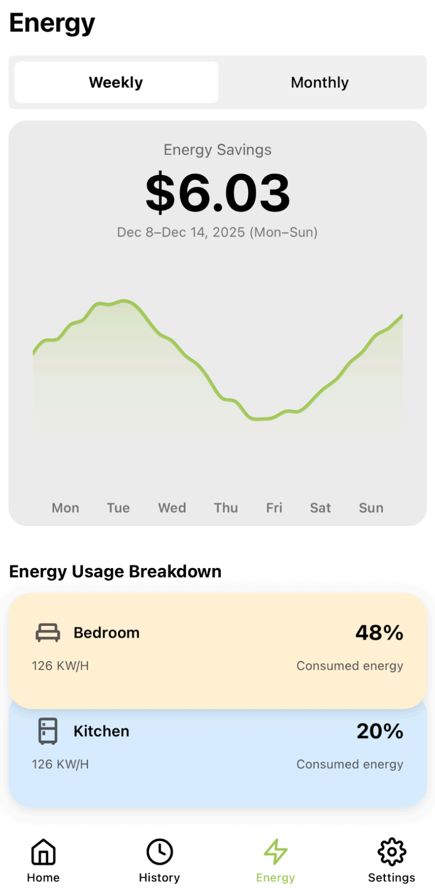
Temperature adjusted to 79.5°F
Increasing temperature to 79.5°F
5:25 PM Heating Motion Off 74.8°F

Temperature adjusted to 61.4°F
Decreasing temperature to 61.4°F
5:25 PM Cooling Motion Off 74.8°F

Motion Off in Kitchen
Decreasing temperature to 72°F
5:25 PM Cool Motion Off 75.1°F

Home History Energy Settings

- Energy and Settings Screen



Energy

Weekly | Monthly

Energy Savings

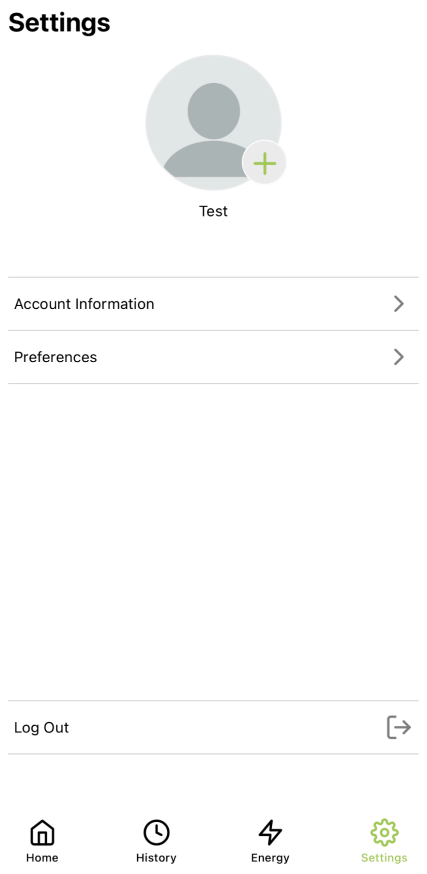
$6.03

Dec 8–Dec 14, 2025 (Mon–Sun)

Energy Usage Breakdown

Bedroom **48%**
126 KW/H Consumed energy

Kitchen **20%**
126 KW/H Consumed energy



Settings

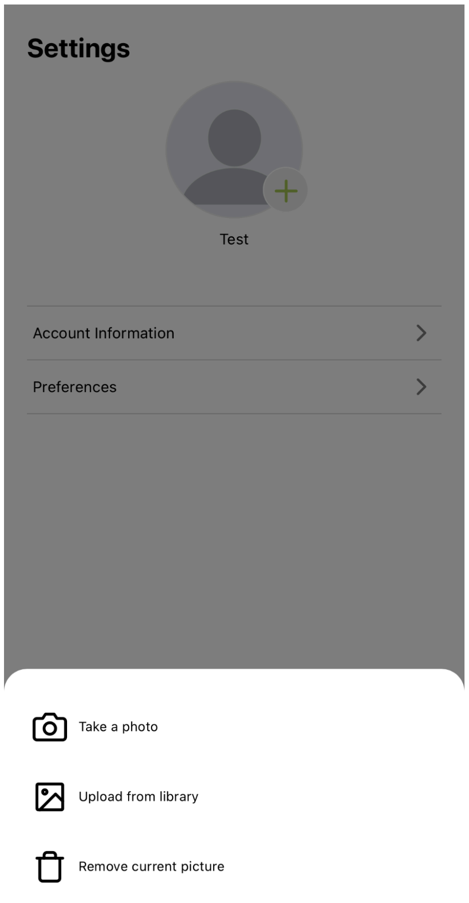
Test

Account Information

Preferences

Log Out

- Profile Picture Modal


Settings
Test
Account Information
Preferences
Take a photo
Upload from library
Remove current picture