# Smart Home Automation

Nick Tool, Tyler Mayou, Matthew McLean

## Final Report

28 April 2025

# Smart Home Automation

Nick Tool, Tyler Mayou, Matthew McLean

# CONCEPT OF OPERATIONS

REVISION –3
28 April 2025

# Concept of Operations
for
# Smart Home Automation


Prepared by:

Team 61

Nick Tool: Server and Database Subsystem
Tyler Mayou: Hardware Subsystem
Matthew McLean: Phone App and Web Interface


Approved by:

_____
Project Leader Date

_____
Prof. Kalafatis Date

_____
T/A Date

## Change Record

| Rev. | Date | Originator | Approvals | Description |
|---|---|---|---|---|
| **1** | 9/15/2024 | Nick Tool | | Draft Release |
| **2** | 12/5/2024 | Matthew McLean | | 403 Final Report |
| **3** | 4/28/2025 | Tyler Mayou, Nick Tool, Matthew McLean | | 404 Final Report |

# Table of Contents

# List of Tables

**No List of Tables entries.**

# List of Figures

# 1. Executive Summary

Our sponsor is looking for us to create a smart home automation system to function in a residential space. This system should be able to handle multiple different types of smart devices, with the ability to control the devices either through a mobile app or a website. The entire system will be hosted locally, with a local server and database to handle all of the interactions between the devices and the user interface, as well as collect data to provide further optimization, all while being accessible through a public IP. The desired outcome is a system that allows users to control the devices in their home from anywhere in the world.

# 2. Introduction

In the last few years, automated smart home systems have seen a tremendous increase in both use and development, with many major technology companies creating both software and hardware devices and systems in order to capitalize on this increasing market. While there are many innovators in the field, each company tends to specialize in one specific area of smart home development. This causes issues downstream for the end users, who have to try to combine different software and hardware platforms into one smart home system that works flawlessly, which often leads to more failure than success for the user. Our team hopes to solve this problem by creating a single, local smart home system with full functionality, able to integrate different products into the system and still operate seamlessly.

## *2.1. Background*

This project will enhance already existing smart home systems, by giving the user a single access point to each of the different devices through the app and website that we will be creating. It also allows the user to take advantage of having access to data that will be collected in order to make the entire system more efficient. This system will replace already existing products such as a Google Home or Amazon Alexa due to the local host of the system and the creation of our own UI. The system also decreases the need to buy individual smart devices, as the device we are creating will allow normal devices to be controlled by the user in the smart home system.

## *2.2. Overview*

The goal of this project is to create an integrated smart home system with full automation capabilities using different devices such as light bulbs and speakers. The devices will have digital controls from an Android application and website that will give control to the user. The mobile app and the website will provide the user with precise controls, as well as real time updates of each of the devices. The entire system will run on a local server hosted on a Raspberry Pi, and data will be stored on a local database in order to provide data trends to the user through the website. The devices will communicate with the host server through MQTT communication protocols, and the mobile app and website will communicate with the host and database through WiFi protocols, and there will be basic security measures implemented to protect the system from any unauthorized access.

## *2.3. Referenced Documents*

The following documents, of the exact issue and revision shown, form a part of this specification to the extent specified herein:

**Version 3.4**
**ESP32-WROOM-32 Datasheet**
2023

**PHP Language Reference**
2025

**HTML: The Markup Language (an HTML language reference)**
2025

**CSS Reference**
2025

**JavaScript Reference**
2025

**Kotlin Docs**
2025

# 3. Operating Concept

## *3.1. Scope*

The system we are tasked with creating is designed to create a full smart home environment, with full user control from any device with internet access. This system will replace already existing smart home systems, and give the user added control and convenience, with every device being controlled through a single app or website. All of the data will be available to the user from the local host and database that serve as the backbone for the smart home system.

## *3.2. Operational Description and Constraints*

The user must first have any devices configured and ready for use before the system is ready to use. These devices are paired with the Raspberry Pi using MQTT protocols, and the devices send constant updates to the database. The host collects and processes this data, and then pushes it to both the database and the user interface. Status changes to any of the devices can be controlled by both the user through the UI, as well as sensors integrated into the hardware. When these status changes are detected, the Raspberry Pi receives these commands and executes the desired action. Automation is also possible, and if the user sets up a schedule for certain devices, the database stores these automated tasks and executes them when desired by the user.

The associated constraints with this subsystem deal largely with size and dependency on other systems. Both the device hardware and the Raspberry Pi have physical size constraints that limit the amount of processing power and memory of each device. This may limit the number of devices that the system can handle, and may cause issues if there are large amounts of data being transferred and stored on the physical memory device, such as a Micro SD card. Another large constraint is the dependency on Wi-fi connection and constant power consumption. In order for the Raspberry Pi to function properly, it must communicate with the devices constantly, meaning it must be powered constantly. Any disconnection of power could result in the system being unable to function as a whole. On top of this, in order for the user to interact with the system, they must have a stable connection to the local Wi-Fi network. If the connection is unstable, it might cause delays in execution of commands. While these constraints do affect this system, these are constraints that all smart home systems must manage, and they should not affect the creation of our specific system.

## *3.3. System Description*

The smart home system consists of 3 subsystems: The device hardware, the mobile app and website acting as the User Interface, and a centralized server hosted on a Raspberry Pi. The hardware consists of devices such as lights, speaker, and temperature sensor equipped with MQTT capabilities to communicate with the host. The devices are controlled from the mobile application or the website, which gives the user the ability to manage the smart devices and view real-time data. The app and website communicate with the host and database using WiFi capabilities. Each of these subsystems and their interactions are described in greater detail below:

**Hardware:** This subsystem acts as the switching and controlling hardware for devices around a home. The main hardware component will consist of a microcontroller (ESP32) to be the processor and controller for the devices. The MCU will be able to communicate to the host server and send data to be stored in the database. The actuators are the switches and variable control of certain devices in a home. The sensors are to collect data and send the data to the MCU.

**User Interface:** This subsystem is the user interface with two modes of usage: an android application and a website. Both the application and the website will allow the user to control the devices in the same way one would manually as well as receive valuable information about their home. The user will be able to turn on and off lights, adjust volume of speakers, and adjust the temperature of their thermostat. They also will receive information covering usage, energy consumption, favorite settings, and more. Both the app and the website will be made using kotlin in the android design studio.

**Host & Database:** This subsystem acts as the hub for the entire smart home system, processing user requests and storing user data in the database. The database uses SQL as the backbone, creating a lightweight database structure that will be easy to manage. The host implements APIs to communicate with the devices through HTTPS protocol, as well as security features such as email verification to protect against any unauthorized access to the system. The management of data will make it possible for the user to see data trends through the user interface.

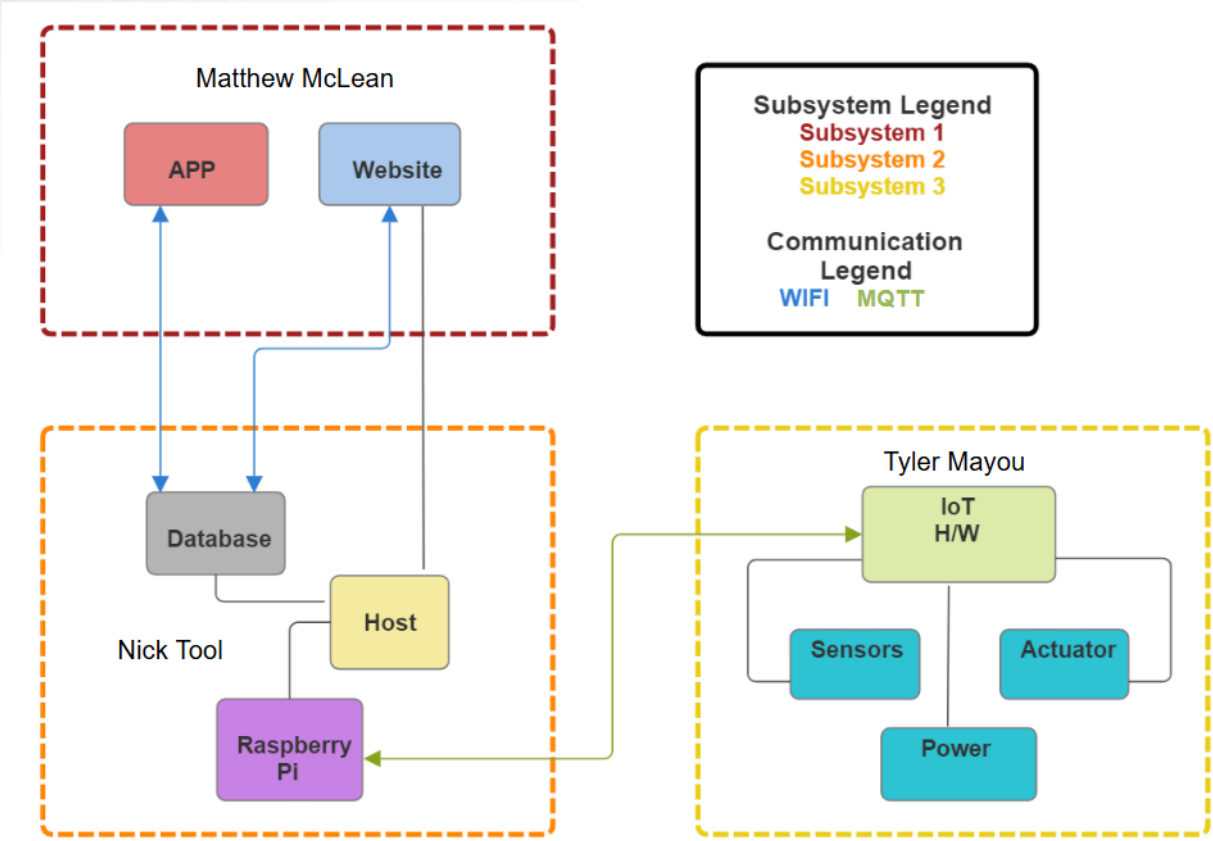



*Figure 1. Block Diagram of System with Subsystems*

### 3.4. Modes of Operations

The system has 2 main modes of operation, active and passive mode. Active mode occurs when the user makes any changes to the status of the system and its devices, views data from the UI, or interacts with the system in any other way. Passive mode is the default mode, and occurs when there are no changes to be made to the system. While the system is in passive mode, the host may communicate with the user if there is any unusual activity in the system, and the host stays running and monitors the devices in both passive and active mode.

### 3.5. Users

The system will be used by anyone who wishes to add smart capabilities to their house or apartment. The user would simply need to plug in the host device, as well as the hardware device, and then either access the system through the mobile application or the website. Anyone who would like to be able to manage all of their electronics through their phone or computer would be able to benefit from this system that combines all smart devices into one simple and clean user interface. The user should be somewhat familiar with running command prompts; common errors will be addressed in the user manual, but in the event that there is a more complicated issue, the user should be able to run commands in order to fix the issue at hand.

### *3.6. Support*

Many of the common errors will be addressed, and the solutions will be provided to the user in a small user manual. These solutions would often be simple commands that the user would have to run on the system in order to address and fix the issue at hand. If the issue is more complex than these simple fixes provided, the user would have to use online resources in order to troubleshoot the issue.

# 4. Scenario(s)

## *4.1. Basic Device Control*

When the smart home system is implemented, a user may want to turn a lightbulb off using the mobile app. To do this, the user logs into the android app using their personalized username and password. When the user logs in, the app requests the status of all of the smart home devices connected to the system, and displays the status on the app for the user to see. The user then selects the lightbulb that they wish to turn off, and hits the “Turn Off” button on the app. The app sends this command to the host, which relays the command to the hardware, turning off the lightbulb. The host then notes this change of status in the database, and the status of the lightbulb changes from on to off in the mobile application.

# 5. Analysis

## *5.1. Summary of Proposed Improvements*

Our system proposes a single smart home system using a common hardware and software to give the user control of their smart devices. Most people who create smart home systems today use devices from many different companies, with differing hardware and software which can make full integration difficult. Our system aims to remove this problem from the user, with easy installation, access, and a simple user interface to run the smart home system.

## *5.2. Disadvantages and Limitations*

The main disadvantage of this system is that the user must do most of the error handling if there are any problems with the system. Other smart home systems are created with millions of dollars in funding behind them, as well as the necessary support team to keep the overarching system running smoothly, and as such, if a user experiences an error, there is trained support staff that are able to help deal with any issue that may arise. If there are any errors that arise that are out of the scope of the user manual, the user needs to troubleshoot and solve the issue on their own.

The main limitations on the proposed system have to do with either size limitations or other system dependencies. All of the hardware must be limited in size due to the constraints on power consumption and the size of the underlying device. The Raspberry Pi also has a limited size, which limits both the internal processing power and memory capabilities. This puts limitations on the overall number of devices in the system, the amount of data that the system can handle, as well as the capabilities of the device hardware. This system is also dependent on already existing communication protocols such as Wi-Fi and MQTT, and if these communication channels are not working properly, it could cause loss of functionality to the system.

## *5.3. Alternatives*

Many major companies either create smart home hubs, mimicking the user interface and host of our system, or create IoT devices, and these alternatives are well known. On the hub side of the business, there are devices like Amazon Alexa that give the user a host system and a mobile interface to be able to interact with each of the smart devices on the system. On the IoT device side of things, there are smart thermostats like Nest that allow remote control of the thermostat from a mobile app or website. While these devices do have much larger existing support for errors in the system, when creating a smart system using many different brands, there are issues that arise with communication between devices that were designed to be used independently; with the Nest thermostat, the best way to control the device is through the Nest app. If a user decides to choose this alternative, they may be using multiple applications in order to control different devices on the smart home system. Our system combines everything into one system, controlled by one app, with the ability to add or remove devices from the system with ease.

### *5.4. Impact*

This home automation system will make simple mundane tasks around the home easier and more engaging. The information given about the device usage in the home will empower users to make smart choices concerning energy consumption and safety which in turn will help users financially and, if implemented on a wide scale, improve the environment. The main ethical concern is safety. Users will be placing their trust in our ability to protect their homes from online attackers. It is our obligation to ensure that the usage of our product not only does not cause harm, but effectively increases the safety of the home. It is likewise our responsibility to ensure that the data given to the users is accurate and beneficial. These concerns are going to be given the focus necessary to provide a system that meets the users needs.

# Smart Home Automation

Nick Tool, Tyler Mayou, Matthew McLean

# FUNCTIONAL SYSTEM REQUIREMENTS

REVISION – 3
28 April 2025

# Functional System Requirements
FOR
# Smart Home Automation


Prepared by:

Team 61

Nick Tool: Server and Database Subsystem
Tyler Mayou: Hardware Subsystem
Matthew McLean: Phone App and Web Interface


Approved by:

_____
Project Leader Date

_____
John Lusher, P.E. Date

_____
T/A Date

## Change Record

| Rev. | Date | Originator | Approvals | Description |
|---|---|---|---|---|
| **1** | 9/26/2024 | Tyler Mayou | | Draft Release |
| **2** | 12/5/2024 | Matthew McLean | | 403 Final Report |
| **3** | 4/28/2025 | Tyler Mayou, Nick Tool, Matthew McLean | | 404 Final Report |

# Table of Contents

# List of Tables

# List of Figures

# 1. Introduction

## *1.1. Purpose and Scope*

The smart home automation system is an efficient way to control the devices within a home. These devices may be lights, speakers, or sensors. A phone app and web interface will be the primary way to control the connected devices. It is designed to be used and installed by the average person/household. Figure 1 shows a conceptual graphic of a smart home.

**Figure 1. Smart Home Automation Conceptual Image**

The system will be powered by home wall outlets, and use WiFi to connect the phone app/web interface to the local server and MQTT to connect the local server to the device hardware for data transmission. The device hardware shall be a box in one room of the home where multiple devices shall be connected by wire. The status of connected devices shall be transmitted to the local server, which shall upload it to a database that shall be displayed and controllable by the phone app/web interface.

## *1.2. Responsibility and Change Authority*

Each team member is responsible for their own subsystem and making sure the requirements are met. The team as a whole is responsible for integration of each subsystem. The sponsor for this project has the authority to make changes to the requirements of the project.

# 2. Applicable and Reference Documents

## *2.1. Applicable Documents*

The following documents, of the exact issue and revision shown, form a part of this specification to the extent specified herein:

| Document Number | Revision/Release Date | Document Title |
|---|---|---|
| IEEE 802.11 | 2022 | IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements |
| IEEE 802.15.1 | 2005 | Wireless MAC and PHY Specifications for Wireless Personal Area Networks (WPANs) |

**Table 1. Applicable Documents**

## *2.2. Reference Documents*

The following documents are reference documents utilized in the development of this specification. These documents do not form a part of this specification and are not controlled by their reference herein.

| Document Number | Revision/Release Date | Document Title |
|---|---|---|
| Version 3.4 | 2023 | ESP32-WROOM-32 Datasheet |
| Version 5.3.1 | 2023 | Espressif ESP-IDF API Reference |
| Version 2.4 | 2025 | Apache HTTP Server Documentation |
| Version 8.4 | 2025 | PHP Manual |
| Living Standard | 2025 | HTML Manuel |
| Snapshot 2025 | 2025 | CSS Manuel |
| ECMA Edition 16 | 2025 | JS Manuel |
| Version 2.1.20 | 2025 | Kotlin Manuel |

**Table 2. Reference Documents**

## *2.3. Order of Precedence*

In the event of a conflict between the text of this specification and an applicable document cited herein, the text of this specification takes precedence without any exceptions.

All specifications, standards, exhibits, drawings or other documents that are invoked as “applicable” in this specification are incorporated as cited. All documents that are referred to within an applicable report are considered to be for guidance and information only, except ICDs that have their relevant documents considered to be incorporated as cited.

# 3. Requirements

## *3.1. System Definition*

The Smart Home Automation System will consist of 3 subsystems, which are shown in Figure 2. The first subsystem is the phone app and the web interface. Subsystem 1 will use WiFi for data transmission to connect to Subsystem 2, which will comprise the database, host, and a Raspberry PI. The information Subsystem 2 will store is from Subsystem 3. Subsystem 3 will contain a temperature sensor, power supply, relays, desired devices, and a microcontroller with MQTT capabilities to connect to the Raspberry in Subsystem 2.

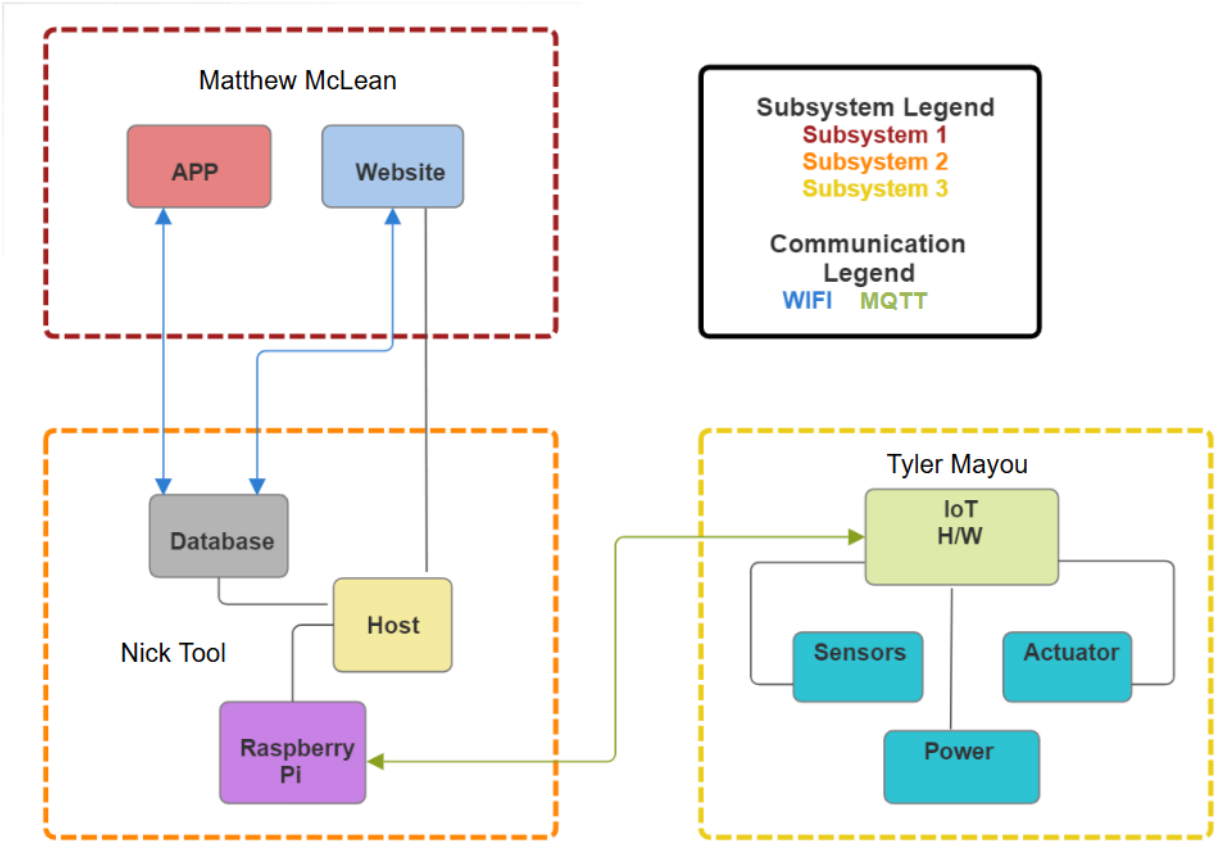



**Figure 2. Block Diagram of System**

## *3.2. Characteristics*

### 3.2.1. Functional / Performance Requirements

#### 3.2.1.1. Device Hardware

The hardware will consist of a microcontroller as the main component which is an ESP32. The ESP32 has MQTT capabilities that allows for communication to the Raspberry PI of the server. The ESP32 will be programmed using ESP-IDF in Visual Studio Code. Figure 3 shows the hardware system with the ESP32. For a light or speaker, the website/mobile app will be able to control the brightness/volume via pulse width modulation through the ESP32 and turn the device on or off. The temperature sensor will be the DHT11 which can measure the temperature of the home with temperature accuracy of ±2°. The relays will be for light bulbs which are higher voltage (AC 120V). The power supply for the system will be a 5V 6A DC power supply to power the device hardware and the Raspberry PI. This hardware system design has been developed on a PCB.

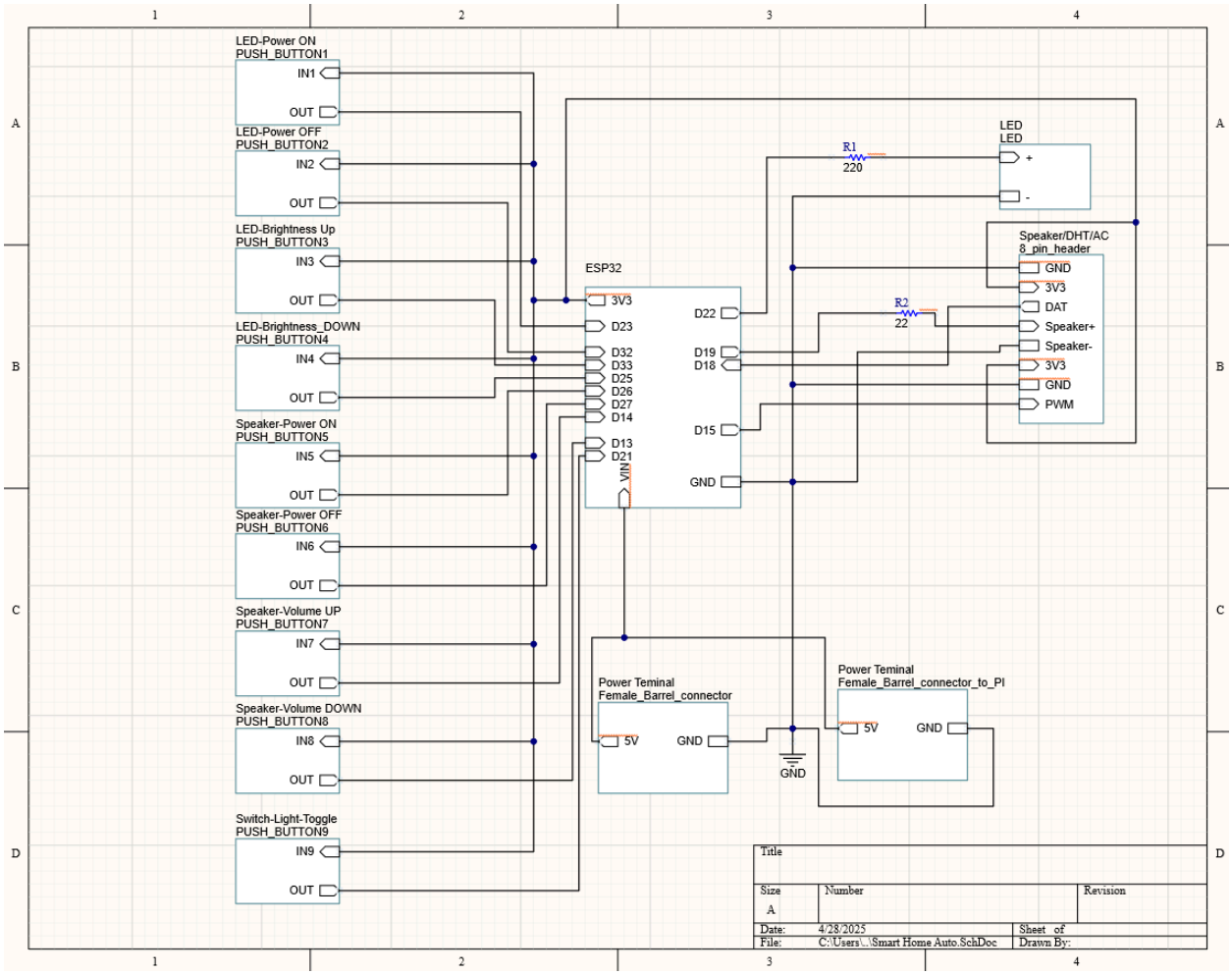



**Figure 3. Hardware System**

#### 3.2.1.2. Mobile App

This application will run on an android phone and allows the user to control the devices in real time as well as receive data that informs them of their usage, preferences, and security alerts. The application will be designed using Kotlin in Android Studio. The user interface will be created using Android Studio's built in layout inspector. It will allow the user in depth control of the devices including: ability to adjust volume with sliders, turn on and off lights, and adjust their thermostat.

#### 3.2.1.3. Web Interface

This Web Interface will be implemented to mimic the user interface built in the mobile app with CSS, JS, and HTML. It will allow users the same control as in their app, but through a web environment.The same data will also be available.

#### 3.2.1.4. Raspberry Pi System

This subsystem is the entire backbone of the project. This system is a LAMP model for a server, and consists of a host and a database. Linux will be used for the operating system of the Raspberry Pi, which is the standard for Raspberry Pi OS Lite. Apache will be used to handle requests from the mobile app and website, and is based on a Debian environment, which is the most common for home usage for server development. The database will be run on MariaDB, a faster and more recent version of MySQL, and both run using SQL as the programming language, which is standard practice for database creation. PHP will be used for the back-end development of the APIs to allow connectivity and communication between the Raspberry Pi system and the other subsystems.

#### 3.2.1.5. Database

The database will handle multiple types of data from both the device hardware and the UI, including user credentials, device status and information, and automation rules. The database will be able to communicate with the UI to provide visualization for data such as power consumption and usage trends, and it will also handle data validation and retreival. The database will keep logs of any server activity, and store these logs for a period where the user may access them in the future.

#### 3.2.1.6. Server Software

MariaDB is the choice of management system for this database, which is an updated version of MySQL, the common choice for database management in a LAMP server. Both MariaDB and MySQL were created by the same developers using SQL, with the key difference being MariaDB has improved speed and increased functionality over MySQL.

Apache is the choice of software for web server management. Apache runs on a Debian environment, and Apache is the most widely used architecture for a Debian environment. Apache works very well in conjunction with MariaDB and PHP, and can make system integration much easier. Because Apache is open-source and widely popular, there is a large amount of support available if any issues arise.

PHP is the choice for back-end development. While there are other choices for back-end development that use a variety of programming languages, PHP is standard for web development due to its speed and ease of integration with other aspects of the LAMP

server. PHP has built in functions for interacting with MariaDB, and also can be configured to interact with Apache easily..

#### 3.2.1.7. Host

The host will handle constant status updates from multiple different devices, and process each of these status requests accordingly in the database. These status updates will consist of the current status of the device, as well as data pertaining to the usage of the device. The host shall also handle requests given by the user through the mobile app or the website, process any changes in status to any of the devices, and push commands to change the status of these devices. To handle these commands, MQTT topics will be subscribed to by the Raspberry Pi and the ESP32, and when there are status changes made, these commands will be published to both the database as well as the hardware devices. Most of the functional processing will be done through PHP, and this is where the majority of the security features and the error handling will be implemented. The host will handle both MQTT and Wi-Fi communication protocols; MQTT will be used to communicate with the hardware, and Wi-Fi will be used to communicate with the UI.

### 3.2.2. Physical Characteristics

**Figure 4. Final Product Showing Housing Compartment and Devices**

#### 3.2.2.1. Structure

The hardware system will be in a housing compartment with accessible connection terminals. The system will be mounted to the housing so there are no loose components inside. The Raspberry Pi will be enclosed in a housing compartment, with all of the connection ports being accessible from ports in the housing compartment.

*Rationale: The housing compartment will protect system hardware from being damaged and provide easy access to terminal connectors.*

### 3.2.3. Electrical Characteristics

#### 3.2.3.1. Inputs

The inputs the Smart Home Automation System is designed to take are user inputs, device status and temperature. The user input will be from the phone or web interface to adjust a device connected to the system. The device status input will be whether the device is on or off and the brightness/volume of the device. The temperature input will be from the sensor integrated into the system.

##### 3.2.3.1.1 Power Consumption

a. The maximum peak power of the system shall not exceed the power delivered from the power source. The power source supplies 30W (5V 6A).

*Rationale: The power supply shall be sufficient to power the microcontroller, Raspberry PI, temperature sensor, and the relay.*

##### 3.2.3.1.2 Input Voltage Level

The input voltage level for the Hardware System shall be +5 VDC. The input power level for the Raspberry Pi shall be 25W, which translates to 5 VDC at 5 A.

*Rationale: The microcontroller, temperature sensor, and relay will need +5VDC. The Raspberry Pi 5 operates at 25W, at a standard 5V 5A.*

#### 3.2.3.2. Outputs

##### 3.2.3.2.1 Data Output

The Smart Home Automation System shall include a phone app and web interface compatible with the data system.

*Rationale: The Smart Home and Automation information passes between the phone app/web interface and the hardware with the server as the medium.*

##### 3.2.3.2.2 Device Output

The Smart Home Automation System shall output based on how the user selects the desired status of the device.

### 3.2.4. Environmental Requirements

The Smart Home Automation System shall be designed to withstand and operate in environments similar to the inside of a home.

#### 3.2.4.1. Pressure (Altitude)

The Smart Home Automation System shall be designed to withstand and operate in average living elevation.

#### 3.2.4.2. Thermal

The Smart Home Automation System shall be designed to withstand and operate in the average home temperature.

#### 3.2.4.3. External Contamination

The Smart Home Automation System shall be designed to withstand and operate in natural home contamination. Since the structure contains external ports, spillage around the device may damage it.

### 3.2.5. Failure Propagation

#### 3.2.5.1.1 Wireless Transmission Errors

The Smart Home Automation System shall have error messages that display on the phone app/web interface signaling a problem such as “Device Not Connected.”

# 4. Support Requirements

### 4.1.1. Android Phone and Web Accsess

The user should have an Android Phone with a Android 7.0 operating system or any more recent android operating system. The user also needs the ability to access the web through a phone,tablet, laptop, or PC.

### 4.1.2. Devices

The user should consider which devices around their home to connect to the Smart Home Automation System.

### 4.1.3. Host Server & Database

The user should have a stable home Wi-Fi network that can support constant real-time updates of the system to the database.

# Appendix A: Acronyms and Abbreviations

| | |
|---|---|
| A | Ampere |
| API | Application Programming Interface |
| LAMP | Linux, Apache, MySQL, PHP |
| MQTT | Message Queuing Telemetry Transport |
| PHP | Hypertext Preprocessor |
| SQL | Standard Query Language |
| UI | User Interface |
| VDC | Volts Direct Current |
| W | Watts |
| Wi-Fi | Wireless Fidelity |

# Appendix B: Definition of Terms

Back-End: The systems that are not accessible by the user.
Front-End: The systems that are accessible by the user.

# Smart Home Automation

Nick Tool, Tyler Mayou, Matthew McLean

# INTERFACE CONTROL DOCUMENT

REVISION – 3

28 April 2025

# Interface Control Document
FOR
# Smart Home Automation

Prepared by:

Team 61

Nick Tool: Server and Database Subsystem
Tyler Mayou: Hardware Subsystem
Matthew McLean: Phone App and Web Interface

Approved by:

_____
Project Leader Date

_____
John Lusher II, P.E. Date

_____
T/A Date

## Change Record

| Rev. | Date | Originator | Approvals | Description |
|---|---|---|---|---|
| **1** | 9/26/24 | Tyler Mayou | | Draft Release |
| **2** | 12/5/24 | Matthew McLean | | 403 Final Report |
| **3** | 4/28/2025 | Tyler Mayou, Nick Tool, Matthew McLean | | 404 Final Report |

# Table of Contents

# List of Tables

## List of Figures

No table of figures entries found.

# 1. Overview

The Interface Control Document (ICD) for the Smart Home Automation System will expand upon the details mentioned in the Concept of Operations and the Functional System Requirements on how each subsystem will be developed. The ICD provides physical details about the subsystems that have physical interfaces. The document will also explain the electrical interface, thermal interface, and device interface protocols. The device interface protocol explains how subsystems in the Smart Home Automation System will communicate with each other.

# 2. References and Definitions

## *2.1. References*

**IEEE 802.11**
**IEEE Standard for Information Technology**
2022 Revision

**IEEE 802.15.1**
**Wireless MAC and PHY Specifications for Wireless Personal Area Networks (WPANs)**

## *2.2. Definitions*

| | |
|---|---|
| VDC | Direct Current Voltage |
| MQTT | Message Queuing Telemetry Transport |
| PCB | Printable Circuit Board |
| cm | Centimeters |
| oz | Ounces |

# 3. Physical Interface

## *3.1. Weight*

The weight of the Smart Home Automation hardware system will be approximately 1.5 pounds. All of the components for the system are small and do not weigh much. The heaviest component is the housing compartment to protect the PCB hardware and the housing component for the Raspberry Pi. Table 1 shows the weight of each component.

| Component | Weight |
|---|---|
| ESP32 | 0.705 oz |
| Temperature Sensor | 0.1 oz |
| Light Bulb | 3.53 oz |
| LED | 1.58 oz |
| 2-Channel Relay | 3.87 oz |
| PCB | 0.2 oz |
| Mini Speaker | 0.3 oz |
| Housing | 16 oz |
| Raspberry Pi 5 Kit | 19.04 oz |
| **Total** | 45.325 oz |

**Table 1. Hardware System Weight Specifications**

## *3.2. Dimensions*

### 3.2.1. Dimension of Hardware System

The hardware system will be in an enclosed housing compartment which is approximately 20 cm in length, 20 cm in width, and 9.4 cm tall. The Raspberry Pi will also be in the housing with the device hardware. Table 2 shows the exact dimensions of each component in the hardware system.

| Component | Length | Width | Height |
|---|---|---|---|
| ESP32 | 5.2 cm | 2.8 cm | 1.4 cm |
| Temperature Sensor | 1.2 cm | 0.55 cm | 1.55 cm |
| Light Bulb | 6 cm | 6 cm | 10.4 cm |
| LED | 0.6 cm | 0.6 cm | 0.7 cm |
| 2-Channel Relay | 5.5 cm | 4 cm | 2 cm |
| PCB | 7 cm | 7 cm | 0.16 cm |
| Mini Speaker | 2.9 cm | 2.9 cm | 1.2 cm |
| Housing | 20 cm | 20 cm | 9.4 cm |
| Raspberry Pi 5 Kit | 8.5 cm | 5.6 cm | 2.4 cm |

**Table 2. Hardware System Dimension Specifications**

#### 3.2.2. Dimension of PCB

The PCB dimensions are listed in Table 2.

## *3.3. Mounting Locations*

#### 3.3.1. Mounting PCB in Housing Compartment

The PCB is mounted inside of the housing compartment. This will avoid loose and rattling electronics inside of the compartment. Also the Raspberry Pi and the 2-channel relay module is mounted inside the housing compartment. The housing compartment has a mounting plate at the base inside.

# 4. Thermal Interface

## *4.1. Hardware System Thermal Interface*

The hardware system will require a small amount of power and will not produce enough heat to cause any problems. The Raspberry Pi is equipped with heat sinks and the housing compartment is large enough for heat dissipation.

# 5. Electrical Interface

## *5.1. Primary Input Power*

### 5.1.1. Primary Input Power for Hardware System

The hardware system will have a 5V 6A power supply. The power supply shall power all the components with necessary voltage regulations besides the AC light bulbs. The hardware for the devices requires a maximum power input of 5W (5V 1A). The Raspberry Pi will be powered by the same supply as the hardware system. The Raspberry Pi 5 typically runs at 25W and contains an onboard power modulation unit, so this power supply shall power the Pi with an appropriate amount of power and not expose it to power beyond its capabilities.

## *5.2. Signal Interfaces*

### 5.2.1. Microcontroller Signal Interface

The microcontroller will communicate to the Raspberry PI via MQTT. The microcontroller shall have a live status of a device connected to the microcontroller.

### 5.2.2. Temperature Sensor Signal Interface

The temperature sensor will communicate with the microcontroller by transmitting the data it is collecting.

## *5.3. User Control Interface*

### 5.3.1. Mobile App and Web Interface

The user will be able to control the full functionality of the devices through buttons, sliders, and scaled user inputs on the Mobile App and Web Interface.

# 6. Communications / Device Interface Protocols

## *6.1. Wireless Communications (WiFi/MQTT)*

Between the server subsystem and phone app/web interface, communication will be through WiFi. For the hardware system, communication to the server will be through MQTT. The use of communication via WiFi will operate in specification of IEEE standard 802.15.1 for communication between the mobile app, web app, host server, and hardware devices. The communication of MQTT relies on Wi-Fi protocols as well, so all communication will be done at the IEEE standard 802.15.1.

## *6.2. Microcontroller Input and Output*

The microcontroller, ESP32, will take in inputs and process output through wireless MQTT communication. The wireless inputs and outputs will be for the buttons (from web/app), temperature sensor, light, speaker, and relay.

## *6.3. Raspberry Pi Inputs and Outputs*

The Raspberry Pi will take in multiple types of HTTP inputs. In the communication between the mobile app and web application, the Raspberry Pi will both take in and send out payloads of the requested data, and this will be facilitated by the different HTTP protocols that the host system can handle. In communication with the ESP microcontroller, the Pi will receive commands through MQTT messages sent by the microcontroller, and process them using the same HTTP commands, based on the data being received. In the case of the microcontroller, the most used command is the POST command, and this command will be sending device updates to the microcontroller that were enacted by the user on the mobile and web application. For communication between the mobile app and web application, there will be a combination of GET and POST requests, with the GET requests being used to display device statuses to the user, and POST requests to make any changes to these statuses.

# Smart Home Automation

Nick Tool, Tyler Mayou, Matthew McLean

# EXECUTION PLAN

REVISION – 3
28 April 2025

| 403 Execution Plan | Start | End | Status | | Legend |
|---|---|---|---|---|---|
| **Matthew McLean** | | | | | Finished |
| Initial Project App | 24-Sep-2024 | 14-Oct-2024 | | | Working on it |
| Order phone | 24-Sep-2024 | 06-Oct-2024 | | | Haven't Started |
| Create SMA App | 14-Oct-2024 | 05-Nov-2024 | | | |
| Implement Web Interface | 14-Oct-2024 | 10-Nov-2024 | | | |
| Validation Tests for App | 05-Nov-2024 | 26-Nov-2024 | | | |
| Validations Test for Web Interface | 10-Nov-2024 | 26-Nov-2024 | | | |
| **Nick Tool** | | | | | |
| Raspberry Pi Functionality | 10-Sep-2024 | 21-Sep-2024 | | | |
| LAMP Installation | 10-Sep-2024 | 21-Sep-2024 | | | |
| Back-End API Development | 24-Sep-2024 | 05-Oct-2024 | | | |
| Host Validation | 24-Sep-2024 | 19-Oct-2024 | | | |
| Database Validation | 24-Sep-2024 | 19-Oct-2024 | | | |
| Security Implementation | 24-Sep-2024 | 26-Oct-2024 | | | |
| Connection to App Validation | 19-Oct-2024 | 16-Nov-2024 | | | |
| Connection to Devices Validation | 19-Oct-2024 | 16-Nov-2024 | | | |
| **Tyler Mayou** | | | | | |
| ESP32 Functionality | 20-Sep-2024 | 20-Oct-2024 | | | |
| Order Hardware Parts | 20-Sep-2024 | 20-Oct-2024 | | | |
| ESP32 Programming | 20-Sep-2024 | 29-Oct-2024 | | | |
| Circuit Validation Testing | 15-Oct-2024 | 01-Nov-2024 | | | |
| PCB Design and Order | 08-Oct-2024 | 01-Nov-2024 | | | |
| PCB Validation Testing | 29-Oct-2024 | 12-Nov-2024 | | | |
| **Team** | | | | | |
| ConOps | 26-Aug-2024 | 15-Sep-2024 | | | |
| FSR/ICD | 15-Sep-2024 | 26-Sep-2024 | | | |
| Midterm Presentation | 16-Sep-2024 | 23-Sep-2024 | | | |
| Introduction Project | 26-Sep-2024 | 15-Oct-2024 | | | |
| Status Update Presentation | 07-Oct-2024 | 14-Oct-2024 | | | |
| Final Presentation | 04-Nov-2024 | 11-Nov-2024 | | | |
| Final Demo | 12-Nov-2024 | 26-Nov-2024 | | | |
| Final Report | 21-Nov-2024 | 05-Dec-2024 | | | |

## 404 Execution Plan

| | 1/20 | 1/27 | 2/3 | 2/10 | 2/17 | 2/24 | 3/3 | 3/10 | 3/17 | 3/24 | 3/31 | 4/7 | 4/14 | 4/21 | 4/28 | Legend |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Matthew McLean** | | | | | | | | | | | | | | | | Finished |
| Update Layout Of Website | | | | | | | | | | | | | | | | Working on it |
| Integrate System with Host Server | | | | | | | | | | | | | | | | Haven't Started |
| Integrate Entire System | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | |
| **Nick Tool** | | | | | | | | | | | | | | | | |
| Implement Public IP address | | | | | | | | | | | | | | | | |
| Implement Security Features | | | | | | | | | | | | | | | | |
| Integrate Host System with Mobile App and Website | | | | | | | | | | | | | | | | |
| Integrate Host System with IoT Hardware | | | | | | | | | | | | | | | | |
| Validate App & Website Integration | | | | | | | | | | | | | | | | |
| Validate IoT Hardware Integration | | | | | | | | | | | | | | | | |
| Test and Validate Basic Functions in Fully Integrated System | | | | | | | | | | | | | | | | |
| Test Edge Cases in Fully Integrated System | | | | | | | | | | | | | | | | |
| **Tyler Mayou** | | | | | | | | | | | | | | | | |
| Discuss 404 modifications | | | | | | | | | | | | | | | | |
| Order parts for modification | | | | | | | | | | | | | | | | |
| Order NEW PCB | | | | | | | | | | | | | | | | |
| Hardware integration | | | | | | | | | | | | | | | | |
| **Team** | | | | | | | | | | | | | | | | |
| Update #2 | | | | | | | | | | | | | | | | |
| Update #3 | | | | | | | | | | | | | | | | |
| Update #4 | | | | | | | | | | | | | | | | |
| Update #5 | | | | | | | | | | | | | | | | |
| Final Presentation | | | | | | | | | | | | | | | | |
| Final Demo | | | | | | | | | | | | | | | | |
| Final Report | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | |
| Integration | | | | | | | | | | | | | | | | |
| System Test | | | | | | | | | | | | | | | | |
| Validation | | | | | | | | | | | | | | | | |

# Smart Home Automation

Nick Tool, Tyler Mayou, Matthew McLean

# VALIDATION PLAN

REVISION – 3
28 April 2025

# 403 Validation Plan for Subsystems

| Test Name | Success Criteria | Methodology | Status | Owner |
|---|---|---|---|---|
| Power Supply | Power supply delivers 5V DC to MCU | Measure voltage the MCU using multimeter | TESTED | Tyler Mayou |
| Mini-speaker Power | Mini-speaker will receive ~8 mV at max duty cycle | Measure voltage/current using multimeter. | TESTED | Tyler Mayou |
| LED Voltage | LED will receive ~3.3V at max duty cycle | Measure voltage using multimeter. | TESTED | Tyler Mayou |
| ESP32 Programming/Functionality | Programs builds and flashes to ESP32. Program Works properly | Desired programs are compiled and executed correctly | TESTED | Tyler Mayou |
| MCU Receives Data from Temp Sensor | Data from sensors is correct and is received by the MCU and can be seen in the Device Monitor | Ensure the MCU receives data from sensor. | TESTED | Tyler Mayou |
| Light Control | Using push buttons as inputs for light dimming and PWM from MCU to light. Duty Cycle changes | Use multimeter to check change in voltage. Push buttons can control MCU output to light. | TESTED | Tyler Mayou |
| Volume Control | Using push buttons as inputs for volume control and PWM from MCU to speaker. | Use multimeter to check change in voltage. Push buttons can control MCU output to speaker. | TESTED | Tyler Mayou |
| | | | | |
| Navigation | Navigation around app and web layouts runs smoothly | Test that each navigation button properly navigates | TESTED | Matthew McLean |
| Input Checks | All user input text boxes properly update text | Check the functionality of each user intput text box | TESTED | Matthew McLean |
| Connectivity Mobile App | Mobile app can read and write data to the database (not directly) | Set up server and database and test connection and http communication | TESTED | Matthew McLean |
| Connectivity Web App | Web app can read and write data to the database (not directly) | Set up server and database and test connection and http communication | TESTED | Matthew McLean |
| Loading Data at Start | When device settings have been manually altered app should be able to load the correct settings | Change data in database when app is not running then run the app | TESTED | Matthew McLean |
| Runtime Manual Device Control | While the app is running, if device settings are manually adjusted, app should display changes in real time | Change data in database while app is running | TESTED | Matthew McLean |
| | | | | |
| Check all tables exist in database | Both users & devices tables exist in the database | Using SQL queries in validationTest.php | TESTED | Nick Tool |
| Insert New User and Device | New user & device inserted successfully into the database using SQL query | Using SQL queries in validationTest.php | TESTED | Nick Tool |
| Fetch All Users & Devices | All users and devices pulled from database and shown | Using SQL queries in validationTest.php | TESTED | Nick Tool |
| Update User & Device Status | Successfully updates user and device status | Using SQL queries in validationTest.php | TESTED | Nick Tool |
| Insert Duplicate | Attempts to insert duplicate user and fails | Using SQL queries in validationTest.php | TESTED | Nick |

| | | | | |
|---|---|---|---|---|
| User | | | | Tool |
| Delete User | Delete a user and make sure devices table is updated | Using SQL queries in validationTest.php | TESTED | Nick Tool |
| Create New User & Device using API | Use HTTP requests to create a new user & new device using API | Using cURL requests and SQL queries in validationTest.php | TESTED | Nick Tool |
| Fetch All Users & Devices via API | Use HTTP requests to fetch all users & devices using API | Using cURL requests and SQL queries in validationTest.php | TESTED | Nick Tool |
| Update User & Device Status via API | Use HTTP requests to update user & device status using API | Using cURL requests and SQL queries in validationTest.php | TESTED | Nick Tool |
| Handle Invalid API Request | Use HTTP requests to test invalid API request | Using cURL requests and SQL queries in validationTest.php | TESTED | Nick Tool |

## 404 Validation Plan for Integration

| Test Name | Success Criteria | Methodology | Status | Owner |
|---|---|---|---|---|
| Power to hardware and Raspberry PI | Hardware board/components and Raspberry PI are powered at rated limits (PI - 5V 5A, Hardware - 5V 1A). PI is hosting server. All components through ESP32 are functional. | Plug in all components, Check if PI is hosting website, check if device components are functional. Power source is 5V 6A. | TESTED | Tyler Mayou, Nick Tool |
| Relay/Light Bulb Control | Relay is powered from the ESP32 and the ESP32 can control the on/off capabilities of the Relay | Wire the light bulb through the relay and connect the VCC, GND, IN1 on the relay to the esp32. Use High/Low from the ESP32 GPIO pin to control the Relay | TESTED | Tyler Mayou |
| Communication between Raspberry PI and ESP32 | Both Raspberry PI and ESP32 can send and receive data through MQTT communication (Button Presses, Sensor Data). | First Establish connection between Raspberry PI and ESP32. Test if button presses are sent from backend. Test if sensor data can be sent/received. Lights/speaker are controllable from button press. | TESTED | Tyler Mayou, Nick Tool |
| Sign Up Functionality | User can create a new unique user in system and database reflects parameters. | Try to create a new user from the public website, check the database to make sure information matches sign up. | TESTED | Nick Tool, Matthew Mclean |
| Login Functionality | User can login to system, user stays logged in when navigating to different pages. | User will log in using created credentials, navigate to different pages, including settings, where the correct username should appear. | TESTED | Nick Tool, Matthew Mclean |
| Add and Delete Devices | User can add new devices to the system and delete devices from the system, while database reflects these changes. | On the settings page, add a new device while being logged in, check database to make sure that new device was added. | TESTED | Nick Tool, Matthew Mclean |

| | | | | |
|---|---|---|---|---|
| Toggle Specific Device Status | User can select a specific device, and update its status or a specific parameter (Brightness, Volume, Temperature) | On any of the device pages, user will select a specific device, make a change to its status, and changes show up in the database and hardware status will change | TESTED | Nick Tool, Matthew Mclean, Tyler Mayou |
| Web App and Mobile App are in sync | If a button is pressed on either the web app or mobile app the other will update with the status of the device | With any device change the status (ex. turn on LED) from either the web app or mobile app and the user source will be updated with the current status of the device | TESTED | Nick Tool, Matthew McLean |

# Smart Home Automation

Nick Tool, Tyler Mayou, Matthew McLean

## SUBSYSTEM REPORTS

REVISION – 2

28 April 2025

# Subsystem Report
FOR
# Smart Home Automation

Prepared by:

Team 61

Nick Tool: Server and Database Subsystem
Tyler Mayou: Hardware Subsystem
Matthew McLean: Phone App and Web Interface

Approved by:

_____
Project Leader Date

_____
John Lusher II, P.E. Date

_____
T/A Date

## Change Record

| Rev. | Date | Originator | Approvals | Description |
|---|---|---|---|---|
| **1** | 12/5/24 | Tyler Mayou | | Draft Release |
| **2** | 4/28/2025 | Tyler Mayou, Nick Tool, Matthew McLean | | 404 Final Report |

# Table of Contents

# List of Tables

# List of Figures

# 1. Overview

This document contains the final reports for each subsystem within this project completing the requirement for ECEN 403. Each subsystem has a section on introductions, details, validation, and conclusions. During ECEN 404, changes have been made to each subsystem and each report resembles how each system is now.

# 2. Hardware Subsystem Report

## *2.1. Subsystem Introduction*

The hardware subsystem is designed to control home devices (i.e. lights and speaker) and sensors that are connected to the hardware. The center component of the hardware is a ESP32 microcontroller with the ability to control such home devices. The subsystem has the ability to control some of the devices by using manual buttons. The push buttons used for testing the functionality of the ESP32 and the connected devices. The main control of the devices is through the mobile app and website. The ESP32 communicates with the server through MQTT after establishing WiFi connection, it will subscribe to topics on the server and will receive commands to change the status of the desired device. The push buttons and the commands from the server do the same operations. The push buttons are not used in the final product.

## *2.2. Subsystem Details*

### 2.2.1. Circuit Design

A digital circuit of the subsystem is shown below.

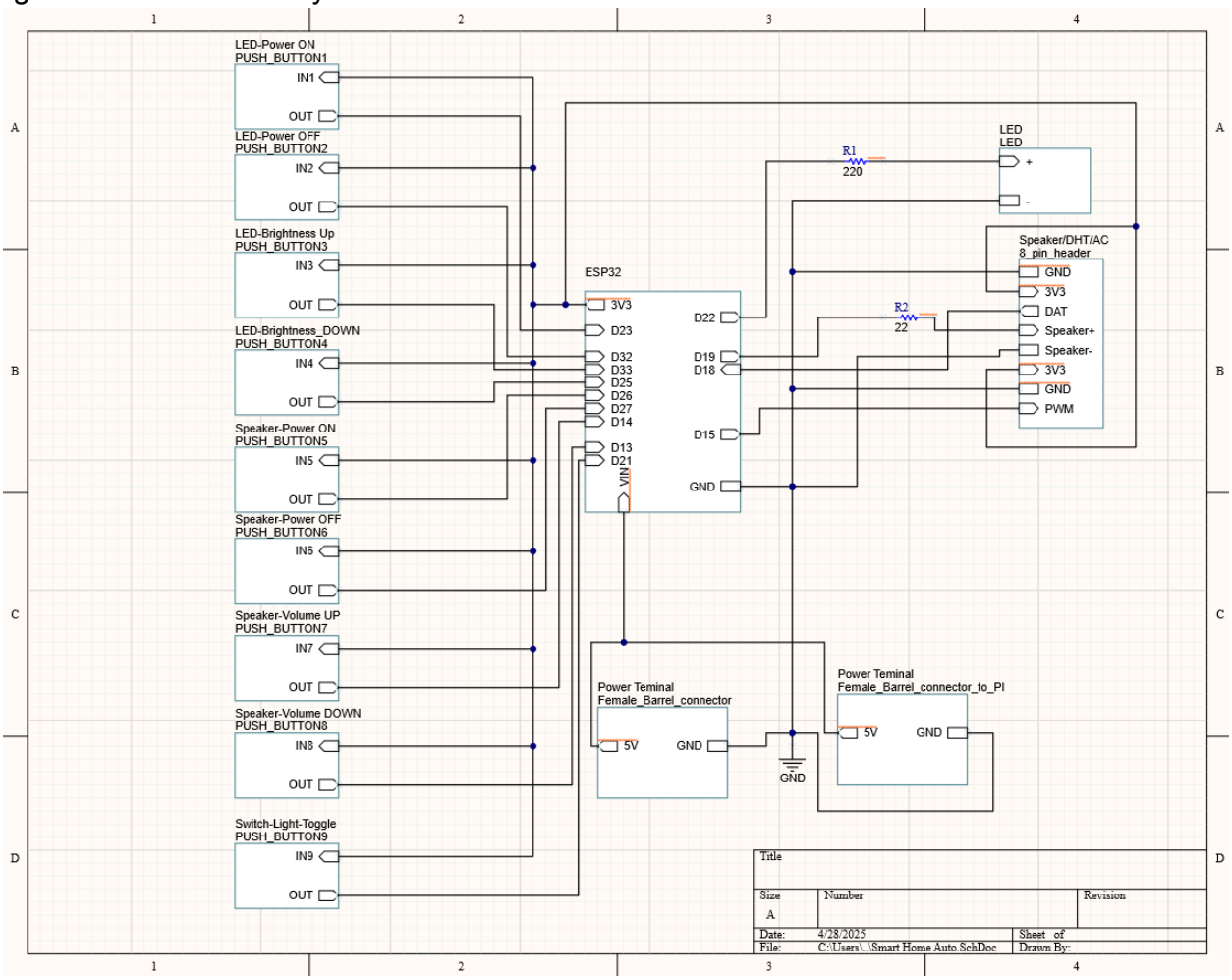



*Figure 1. Hardware Subsystem Circuit Design*

The push buttons in the circuit are all GPIO inputs to the ESP32 and are programmed as pulldown input GPIOs. The Vcc is a 5V 6A AC to DC converter that plugs into the wall that powers the PCB and Raspberry PI. The light and speaker are outputs of the ESP32 and the buttons control the on/off and the brightness/volume of the components. The light is connected to a 220 ohm resistor and the speaker is connected to a 22 ohm resistor. The temperature sensor is a DHT11 that is an input to the ESP32 where it reads the temperature of the room it is in.

| ESP32 Pin | Use |
|---|---|
| D23 | Light On |
| D32 | Light Off |
| D33 | Brightness Up |
| D25 | Brightness Down |
| D26 | Speaker On |
| D27 | Speaker Off |
| D14 | Volume Up |
| D13 | Volume Down |
| D15 | Relay Control |
| D18 | DHT11 |
| D19 | Speaker |
| D22 | Light |
| Vin | Input Voltage, 5V |
| GND | Ground |

*Table 1. Designated Pins for the ESP32*

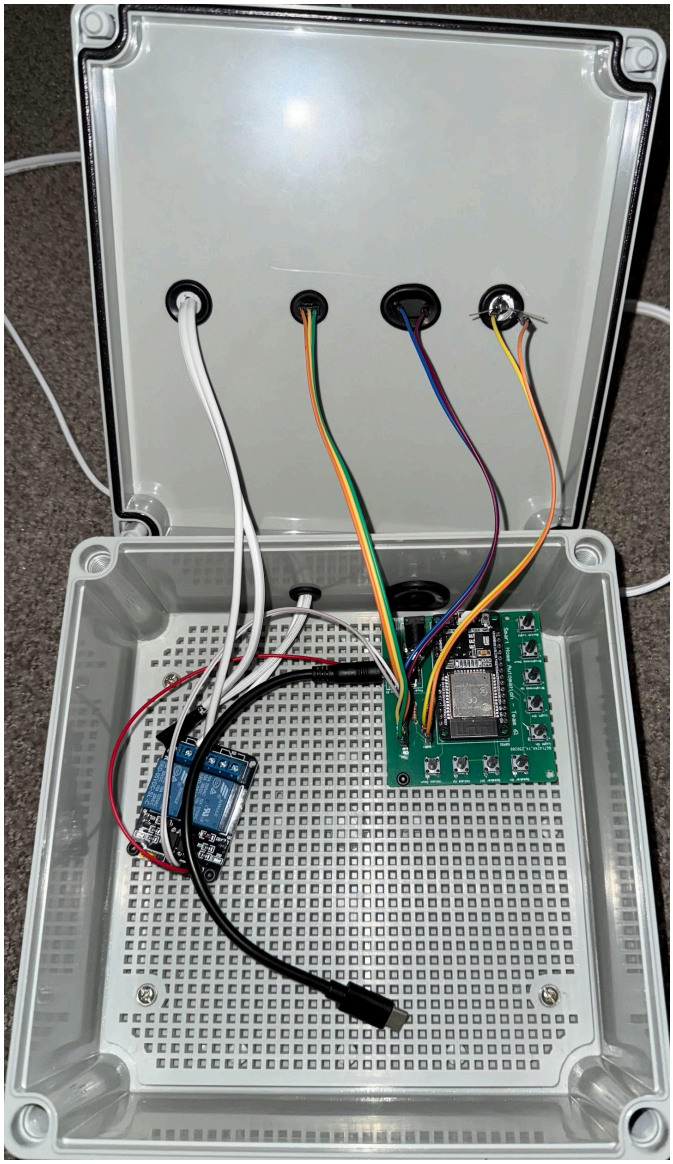


*Figure 2. Hardware Inside Housing Compartment*

In Figure 2, the hardware with the ESP32 is on the top right, the relay module is on the left, and the Raspberry PI would be placed on the bottom right where the black USB-C cord is.

### 2.2.2. Programming of the ESP32

To program the ESP32, the ESP-IDF from Espressif is used on VSCode. The ESP-IDF includes a library of different functions used to program the ESP32 as desired. The project can be compiled and flashed to the ESP32 using a UART. Prior to building the program, the ESP32 can be configured in the configuration menu to set up the device how it is needed. In the main C file, for this subsystem, the program contains functions for MQTT event handling, WiFi initialization, interrupts for the buttons, initialization of GPIOs and connected devices, and the DHT11 functions. The configuration of the GPIOs/pins of the ESP32 is shown above in Table 1. To change the volume and brightness of the speaker and light the duty cycle was changed to alter the output voltage. To read the data from the DHT11

sensor, the library of functions from the ESP-IDF are used and then the values of the temperature are converted into degrees Fahrenheit and a percent.

## *2.3 Subsystem Validation*

As shown in the Validation Plan, a variety of tests were conducted to validate the subsystem. Mainly a multimeter was used to validate components of the subsystem. The power supply, light, and speaker were validated using a multimeter to check the voltage of the components. The voltage of the light and speaker was checked at the max duty cycle to ensure they would not be over the rated voltage. The ESP-IDF allows the ESP32 to be monitored to check how the device is operating.

To validate the WiFi and MQTT connectivity and communication of the ESP32, the device monitor terminal was used. In Figure 3 below, the advertisement details of the ESP32 and the name of the device are shown. In Figure 4, the data “hello world” is read from the ESP32.

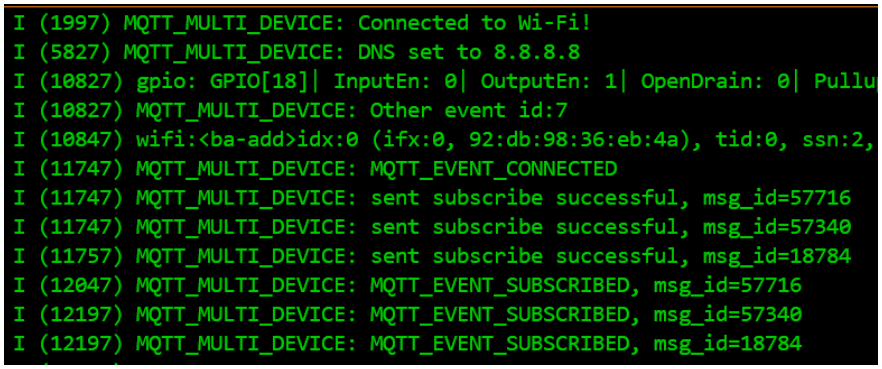



*Figure 3. ESP32 Connected to WiFi and Subscribed to Topics*

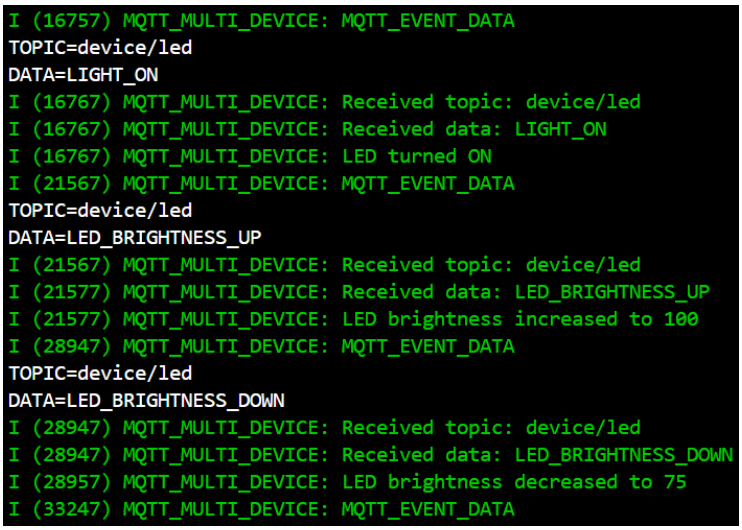



*Figure 4. ESP32 Receiving Data from the Server for LED Control*

```
TOPIC=device/relay
DATA=LIGHT_ON
I (33247) MQTT_MULTI_DEVICE: Received topic: device/relay
I (33247) MQTT_MULTI_DEVICE: Received data: LIGHT_ON
I (33257) MQTT_MULTI_DEVICE: Relay turned ON
I (53007) MQTT_MULTI_DEVICE: MQTT_EVENT_DATA
```

*Figure 5. ESP32 Receiving Data from the Server for Relay Control*

```
I (53007) MQTT_MULTI_DEVICE: MQTT_EVENT_DATA
TOPIC=device/speaker
DATA=SPEAKER_ON
I (53007) MQTT_MULTI_DEVICE: Received topic: device/speaker
I (53017) MQTT_MULTI_DEVICE: Received data: SPEAKER_ON
I (53017) MQTT_MULTI_DEVICE: duty_speaker is 30
I (53027) MQTT_MULTI_DEVICE: Speaker turned ON
I (58747) MQTT_MULTI_DEVICE: MQTT_EVENT_DATA
TOPIC=device/speaker
DATA=SPEAKER_VOLUME:80
I (58747) MQTT_MULTI_DEVICE: Received topic: device/speaker
I (58757) MQTT_MULTI_DEVICE: Received data: SPEAKER_VOLUME:80
I (58757) MQTT_MULTI_DEVICE: Setting volume to 80%
I (58767) MQTT_MULTI_DEVICE: duty_speaker is 32
I (63677) MQTT_MULTI_DEVICE: MQTT_EVENT_DATA
TOPIC=device/speaker
DATA=SPEAKER_VOLUME:90
I (63677) MQTT_MULTI_DEVICE: Received topic: device/speaker
I (63677) MQTT_MULTI_DEVICE: Received data: SPEAKER_VOLUME:90
I (63687) MQTT_MULTI_DEVICE: Setting volume to 90%
I (63697) MQTT_MULTI_DEVICE: duty_speaker is 36
I (65197) MQTT_MULTI_DEVICE: MQTT_EVENT_DATA
TOPIC=device/speaker
DATA=SPEAKER_VOLUME:100
I (65197) MQTT_MULTI_DEVICE: Received topic: device/speaker
I (65197) MQTT_MULTI_DEVICE: Received data: SPEAKER_VOLUME:100
I (65207) MQTT_MULTI_DEVICE: Setting volume to 100%
I (65207) MQTT_MULTI_DEVICE: duty_speaker is 40
I (78817) MQTT_MULTI_DEVICE: MQTT_EVENT_DATA
TOPIC=device/speaker
DATA=SPEAKER_VOLUME:10
I (78817) MQTT_MULTI_DEVICE: Received topic: device/speaker
I (78817) MQTT_MULTI_DEVICE: Received data: SPEAKER_VOLUME:10
I (78827) MQTT_MULTI_DEVICE: Setting volume to 10%
I (78827) MQTT_MULTI_DEVICE: duty_speaker is 4
I (83427) MQTT_MULTI_DEVICE: MQTT_EVENT_DATA
TOPIC=device/speaker
DATA=SPEAKER_VOLUME:30
I (83427) MQTT_MULTI_DEVICE: Received topic: device/speaker
I (83427) MQTT_MULTI_DEVICE: Received data: SPEAKER_VOLUME:30
I (83437) MQTT_MULTI_DEVICE: Setting volume to 30%
I (83437) MQTT_MULTI_DEVICE: duty_speaker is 12
I (84857) MQTT_MULTI_DEVICE: MQTT_EVENT_DATA
TOPIC=device/speaker
DATA=SPEAKER_OFF
I (84857) MQTT_MULTI_DEVICE: Received topic: device/speaker
I (84857) MQTT_MULTI_DEVICE: Received data: SPEAKER_OFF
I (84867) MQTT_MULTI_DEVICE: Speaker turned OFF
```

*Figure 6. ESP32 Receiving Data from the Server for Speaker Control*

```
I (61298) MQTT_MULTI_DEVICE: Published sensor data: temperature:69 (msg_id=32495)
I (61448) MQTT_MULTI_DEVICE: MQTT_EVENT_PUBLISHED, msg_id=32495
```

*Figure 7. ESP32 Publishing Temperature to the Server*

### *2.4 Subsystem Conclusion*

Each component of the subsystem works as intended. The main communication between the ESP32 and the server is MQTT. The hardware subsystem controls a LED, speaker, relay/light bulb, and a temperature sensor by commands from the server. This system has the ability to control more devices and these few devices were used for demonstration purposes. The things that changed from 403 are: a relay and light bulb were added. Switched from Bluetooth to MQTT for communication with the server. Updated PCB due to addition of relay.

## 3. Mobile and web app Subsystem Report

### *3.1. Subsystem Introduction*

The web and mobile app are designed to be the user interface through which users will control the devices. The mobile app is a native android application and the web app will be able to be accessed by any device connected to the server. The layout of the app is sleek, intuitive, and simple and functions as needed.

### *3.2. Subsystem Details*

The project structure of the mobile and web apps can be seen in the figure below.

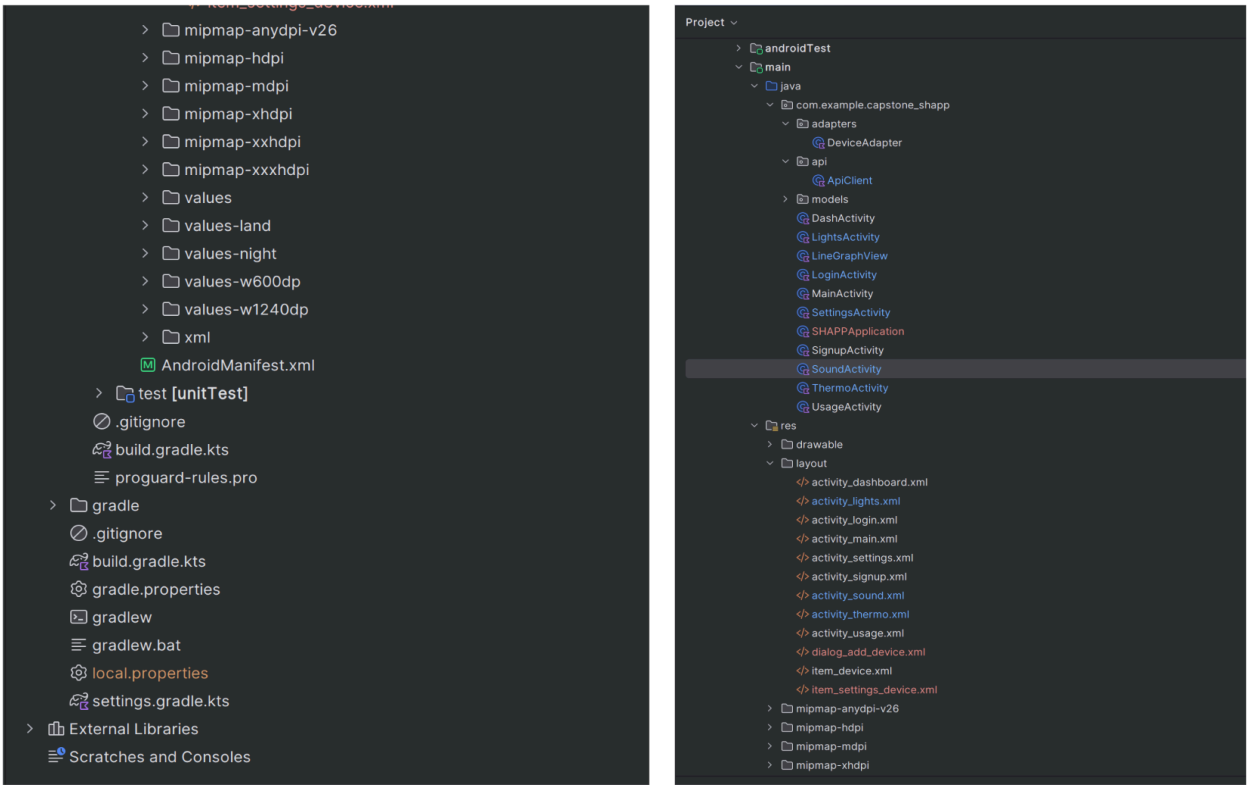



*Figure 8. Project Structure For Mobile App*

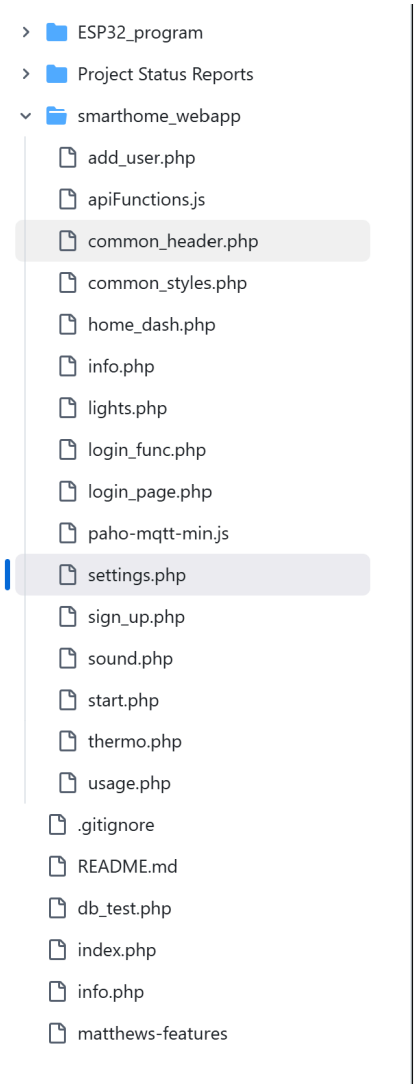



*Figure 9. Project Structure For Web App*

The contents of each of these files can be found on our capstone GitHub.

## *3.3 Subsystem Validation*

Validation for the Web and Mobile app consisted of ensuring every desired function operates by using the app and attempting to break it. Below are images containing every page in the mobile and web application. The Second aspect of the Validation was the synchronization with the server and database subsystem. The details proving that full synchronization and integration was accomplished can be found in the next section.

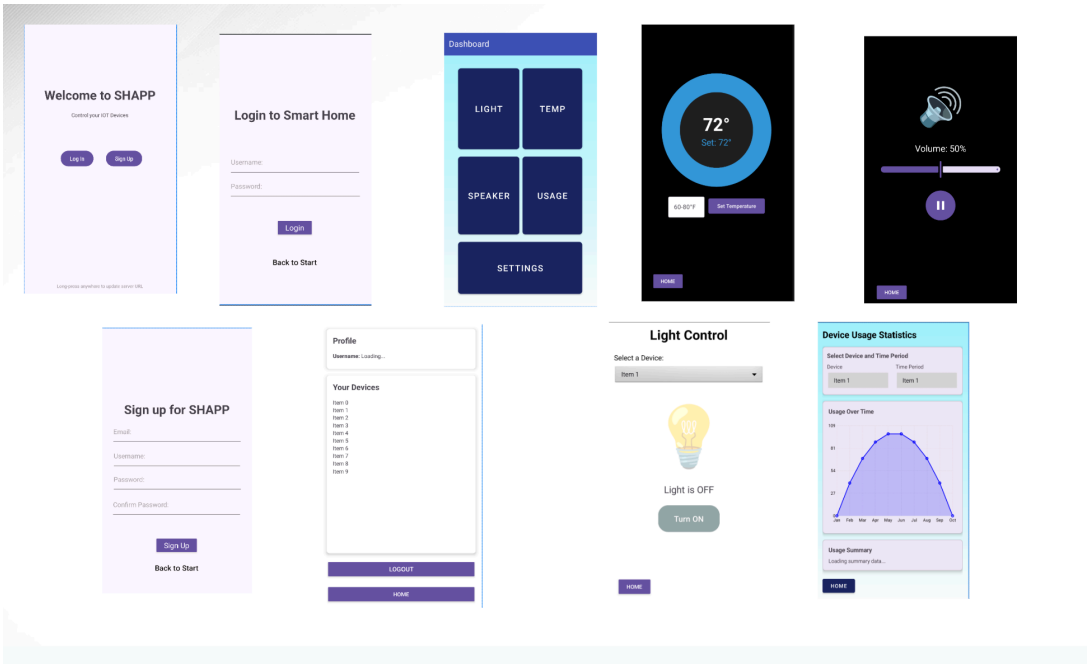



*Figure 10. Mobile App images*

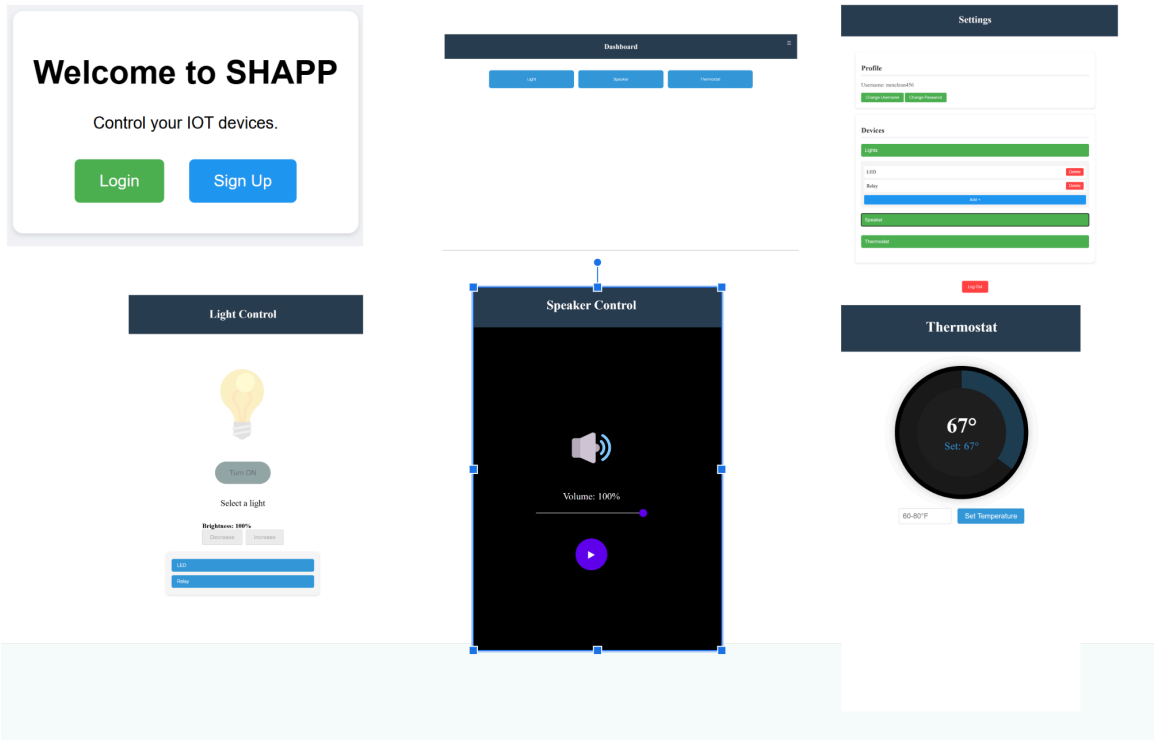



*Figure 11. Web App images*

### *3.4 Subsystem Conclusion*

This subsystem was designed and integrated to fulfill the requirements of the project. The beginning of 404 was spent updating the UI, preparing security features, and increasing the complexity of account and device control. The second half was spent getting it to work with our team's server and database. I had built my own server and database for testing in 403, but a lot needed to change for everything to our team's server. This was the most challenging and rewarding part of the project as the end result was full communication from app to devices (from anywhere in the world) and controlling the devices worked perfectly by the end of the semester. I began this project with zero knowledge or experience with web or mobile app development and have developed countless new skills. The final result of our devices being monitored and controlled through a fully synchronized communication loop from phone and computer through raspberry pi's database to the esp 32 controlling the devices and back to both the phone and computer was a great accomplishment. In conclusion we developed a working system that accomplishes what this project set out to do: provide user control over smart home devices.

## 4. Host Server and Database Subsystem Report

### *4.1. Subsystem Introduction*

The host server and database are designed to serve as the back end of the system, handling interactions between the hardware and the user interface. The system models a LAMP stack, and runs on a Raspberry Pi 5 with a 128GB MicroSD card, and the system will be hosted on a public IP address for access from any location. The host can handle a variety of HTTP requests from either the user interface or from analog inputs from the hardware. The database stores and manages any information about the users and devices in the system. The backend code can handle operations such as email verification, changing username and password, adding and deleting devices, turning devices on and off, and changing device settings, such as brightness and volume.

### *4.2. Subsystem Details*

The LAMP stack operates on Linux, hosting the Apache server for managing incoming HTTP requests. MariaDB serves as the database management system, storing critical information about users, devices, and system logs. PHP acts as the glue, processing requests and interacting with the database. Figure 13 shows the directory structure on the Raspberry Pi for the lamp stack. Table 2 shows the structure of the users and devices table, as well as the data types for each column of the table. Note that some tables do not use as many columns, so some columns are left blank.

```
/var/www/backend
├── api
│   ├── current_temperature.json    # Stores the latest temperature reading in JSON
│   ├── currentTemperature.php      # Endpoint to fetch current temperature
│   ├── devices.php                 # Handles device-related API endpoints
│   ├── login.php                   # Manages user login/authentication
│   ├── mqtt_client.py              # Publishes/subscribes via MQTT
│   ├── publish.php                 # Publishes device updates
│   ├── responses.log               # Logs API responses for debugging
│   ├── users.php                   # Manages user-related API endpoints
│   └── verifyEmail.php             # Handles email verification workflow
├── includes
│   ├── databaseConnection.php      # Establishes database connection
│   └── functions.php               # Contains reusable backend helper functions
└── tests
    └── validationTest.php          # PHPUnit tests for API validation
```

*Figure 12. Directory Structure of LAMP stack*

| **Table** | **users** | **hardware_devices** | **user_devices** |
|---|---|---|---|
| **Column 1** | user_id (int) | hardware_device_id (int) | user_device_id (int) |
| **Column 2** | username (string) | device_type (string) | user_id (int, taken from users table) |
| **Column 3** | email (string) | device_name (string) | hardware_device_id (int, taken from hardware_devices table) |
| **Column 4** | password (hashed string) | created_at (datetime) | custom_name (string) |
| **Column 5** | status (string) | status (string) | created_at (datetime) |
| **Column 6** | created_at (datetime) | device_settings (JSON) | |
| **Column 7** | verificationToken (string) | | |
| **Column 8** | tokenExpiration (datetime) | | |

| | | | |
|---|---|---|---|
| **Column 9** | email_verified (boolean) | | |

*Table 2. Structure of data tables*

The host communicates using the following HTTP request methods: GET, POST, and DELETE, and each of these request methods are used in the handling of users and devices in the API. The hardware_devices table provides mapping for each of the hardware devices that are integrated into the system, and the user can only create as many devices as there are hardware devices, to prevent out of bounds errors with a user trying to change a device status that does not exist. The API handles user and device related endpoints separately due to the differences in the structures of the tables, as well as the devices table being a child of the users table. This is due to the fact that when any updates are made to a device, the user id of the user that made the change is associated with that device in the devices table. In the functions.php file, there are functions written to handle each of the different interactions that the API might have, such as creating or deleting a device. In the API directory, the devices.php and users.php files both rely on these redeployable functions that were written in the functions.php. While the API handles these cases separately, the functions of each case are so similar that it made more sense to use redeployable functions for any HTTP requests that the API might need to process. Figure 14 shows a snippet of the functions.php file, including the function for email verification, and Figure 15 shows how these functions are deployed in devices.php.

```php
function sendVerificationEmail($userEmail, $token) {
        $subject = "Verify your email for Smart Home";
        $verificationLink = "https://absolutely-vocal-lionfish.ngrok-free.app/api/verifyEmail.php?token=" . urlencode($token);

        $message = "Hello, \n\n";
        $message .= "Thank you for signing up for Smart Home.\n";
        $message .= "Please click the link below to verify your email:\n";
        $message .= $verificationLink . "\n\n";
        $message .= "If you did not sign up for Smart Home, ignore this email.";

        $headers = "From: smarthomecapstone@gmail.com\r\n";
        $headers .= "Reply-To: smarthomecapstone@gmail.com\r\n";
        $headers .= "X-Mailer: PHP/" . phpversion();

        $sent = mail($userEmail, $subject, $message, $headers);
        error_log("sendVerificationEmail: mail() returned " . ($sent ? "true" : "false") . " for email: $userEmail");
        return $sent;
}

function getRecord($conn, $table, $conditions = []) {
        //Build sql query to select all records
        $query = "SELECT * FROM $table";
        $params = [];
        $types = "";

        if (!empty($conditions)) {
                $whereClause = [];
                foreach ($conditions as $key => $value) {
                        $whereClause[] = "$key = ?";
                        $params[] = $value;
                        $types .= is_int($value) ? "i" : "s"; //Determines if type is integer or string dynamically
                }
                $query .= " WHERE " . implode(" AND ", $whereClause);
        }

        $stmt = $conn->prepare($query);

        if (!empty($params)) {
                $stmt->bind_param($types, ...$params);
        }

        $stmt->execute();
        $result = $stmt->get_result();

        //Check if connection was correct, if so get records
        if ($result) {
```

*Figure 13. Snippet of Code in functions.php*

```php
switch ($_SERVER['REQUEST_METHOD']) {

    case 'GET':
        // If a user_id is provided, retrieve mapped devices for that user
        if (isset($_GET['user_id'])) {
            $user_id = $_GET['user_id'];
            $query = "SELECT
                        ud.user_device_id,
                        ud.user_id,
                        hd.hardware_device_id,
                        hd.device_type,
                        hd.device_name,
                        hd.status,
                        hd.device_settings,
                        ud.custom_name
                    FROM user_devices ud
                    JOIN hardware_devices hd ON ud.hardware_device_id = hd.hardware_device_id
                    WHERE ud.user_id = ?";
            if (isset($_GET['device_type'])) {
                $query .= " AND hd.device_type = ?";
            }
            $stmt = $conn->prepare($query);
            if (isset($_GET['device_type'])) {
                $stmt->bind_param("is", $user_id, $_GET['device_type']);
            } else {
                $stmt->bind_param("i", $user_id);
            }
            $stmt->execute();
            $result = $stmt->get_result();
            $devices = $result->fetch_all(MYSQLI_ASSOC);
            jsonResponse(true, "Devices retrieved successfully.", $devices);
            $stmt->close();
        } else if (isset($_GET['device_id'])) {
            getRecord($conn, 'hardware_devices', ['hardware_device_id' => $_GET['device_id']]);
        } else {
            jsonResponse(false, "User ID or Device ID required.");
        }
        break;
```

*Figure 14 . Snippet of devices.php*

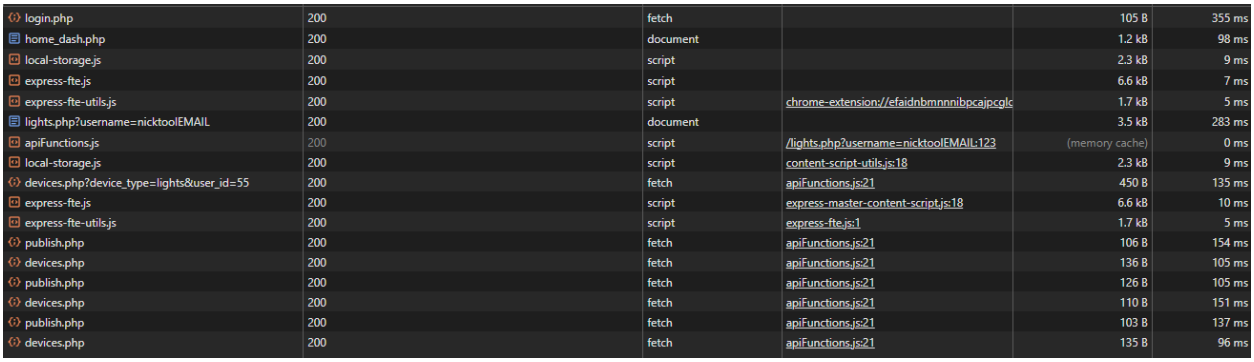
## *4.3 Subsystem Validation*

To complete the validation testing for the subsystem, a test script was created to simulate both SQL queries and cURL requests to test both the database and the HTTP responses. Figure 16 shows a snippet of the code for the first 3 validation tests and their structure. Figure 17 shows the output results of the validation testing the web server output from the IP address of the Raspberry Pi. Once the system was integrated, network tools were used to verify that all of the HTTP requests were being processed correctly, and their approximate run times were measured. Figure 18 shows an example of the web developer tools showing the requests made when logging in, going to the lights page, and updating the status of a light bulb. Note that a status code of 200 signifies that everything executed as planned, as is common in HTTP protocol. Table 3 shows the approximate run time of different operations on the web server.

```php
//Test 1: Check if users table exists
function testCheckUsersTable($conn) {
        $query = "SHOW TABLES LIKE 'users'";
        $result = $conn->query($query);

        return $result && $result->num_rows > 0
                ? ["status" => "pass", "message" => "'users' table exists."]
                : ["status" => "fail", "message" => "'users' table does not exist."];
}

//Test 2: Check if devices table exists
function testCheckDevicesTable($conn) {
        $query = "SHOW TABLES LIKE 'devices'";
        $result = $conn->query($query);

        if (!$conn) {
                echo "No connection";
        }
        return $result && $result->num_rows > 0
                ? ["status" => "pass", "message" => "'devices' table exists."]
                : ["status" => "fail", "message" => "'devices' table does not exist."];
}

//Test 3: Insert a new user
function testInsertUser($conn) {
        //Remove existing test user if it exists
        $queryDelete = "DELETE FROM users WHERE username = 'test_user'";
        $conn->query($queryDelete);

        //Insert new user
        $queryInsert = "INSERT INTO users (username, email, password, status)
                VALUES ('test_user', 'test@example.com', 'hashed_password', 'active')";

        return $conn->query($queryInsert) === TRUE
                ? ["status" => "pass", "message" => "New user inserted successfully."]
                : ["status" => "fail", "message" => "Error inserting new user: " . $conn->error];
}
```

*Figure 15 . Snippet of validation testing code*


*Figure 16. Output on Web Server after running Validation Test*



| | | | | | |
|---|---|---|---|---|---|
| login.php | 200 | fetch | | 105 B | 355 ms |
| home_dash.php | 200 | document | | 1.2 kB | 98 ms |
| local-storage.js | 200 | script | | 2.3 kB | 9 ms |
| express-fte.js | 200 | script | | 6.6 kB | 7 ms |
| express-fte-utils.js | 200 | script | chrome-extension://efaidnbmnnnibpcajpcglc | 1.7 kB | 5 ms |
| lights.php?username=nicktoolEMAIL | 200 | document | | 3.5 kB | 283 ms |
| apiFunctions.js | 200 | script | /lights.php?username=nicktoolEMAIL:123 | (memory cache) | 0 ms |
| local-storage.js | 200 | script | content-script-utils.js:18 | 2.3 kB | 9 ms |
| devices.php?device_type=lights&user_id=55 | 200 | fetch | apiFunctions.js:21 | 450 B | 135 ms |
| express-fte.js | 200 | script | express-master-content-script.js:18 | 6.6 kB | 10 ms |
| express-fte-utils.js | 200 | script | express-fte.js:1 | 1.7 kB | 5 ms |
| publish.php | 200 | fetch | apiFunctions.js:21 | 106 B | 154 ms |
| devices.php | 200 | fetch | apiFunctions.js:21 | 136 B | 105 ms |
| publish.php | 200 | fetch | apiFunctions.js:21 | 126 B | 105 ms |
| devices.php | 200 | fetch | apiFunctions.js:21 | 110 B | 151 ms |
| publish.php | 200 | fetch | apiFunctions.js:21 | 103 B | 137 ms |
| devices.php | 200 | fetch | apiFunctions.js:21 | 135 B | 96 ms |

*Figure 17. Output on Web Developer Tools after Logging In, going to the Lights Endpoint, and Turning On/Off a Light Bulb*

| **Operation** | **Time To Execute** |
|---|---|
| Load Page | ~100 ms |
| Load Devices | ~100 ms |
| Change Device Status | ~200 ms |
| Add/Delete Device | ~200 ms |
| Login/Sign Up | ~300 ms |
| Change Username/Password | ~250 ms |

*Table 3. Operations and Execution Times of Different Operations on Website*

## *4.4 Subsystem Conclusion*

All of the components of the subsystem work as intended, and all of the validation tests passed as expected. The only changes that need to be made when the full integration of the system occurs is allowing access on a public IP address rather than the current private address that is being used, and also being able to accommodate any extra data or request types in the database and host server, respectively. These updates should not change any of the overall design of the subsystem, and the logic and functionality will remain the same with any added changes to the database or HTTP request methods.