

Eötvös Loránd University

Faculty of Informatics Department of Numerical Analysis

Using IoT Technology to Simulate Smart House Environment

*Supervisor:*

ILLÉS ZOLTÁN

Doctor/ Lecturer

*Author:*

LE THANH DUC

Computer Science BSc

*Budapest, 2021*

EÖTVÖS LORÁND UNIVERSITY

FACULTY OF INFORMATICS

Thesis Registration Form

Student's Data:

**Student's Name:** Le Thanh Duc

**Student's Neptun code:** YZ49Y8

Course Data:

**Student's Major:** Computer Science BSc

I have an internal supervisor

***Internal Supervisor's Name:*** *ILLÉS ZOLTÁN*

*Supervisor's Home Institution: ELTE University*

*Address of Supervisor's Home Institution: Room: 2.403 - 1117 Budapest, Pázmány Péter sétány 1 / c.*

*Supervisor's Position and Degree: Doctor /Lecturer*

**Thesis Title:** Using IoT Technology to simulate SmartHouse Environment

Topic of the Thesis:

*(Upon consulting with your supervisor, give a 150-300-word-long synopsis os your planned thesis. )*

Based on what Professor/Dr. Zoltán and I discussed through email plus the topic was also inspired to me by my father, I am going to simulate the smart house environment using IoT Technology and it will be related to AWS Cloud or any other cloud services. I plan to use Raspberry Pi 3 as a local server containing OpenHab2 which uses MQTT to communicate with for example SonOff switch and other sensor devices within the house. In order to simulate the communication with other devices within the house, I will also code NodeMCU Esp8266 using Arduino language to control some lights or small fans/ motors. In term of cloud, I am thinking about using Amazon Alexa or Google Home and its Cloud Service Provider to control the house by voice. All of the devices will be compacted and simulated by a big box to be presented as a small smart house. There will be a dynamic website to control it via internet from distances. So far, this is what I have planned to do. Hopefully, this would be an interesting topic and a successful thesis.

Budapest, 2020.10.17.

# Contents

[Contents 1](#_Toc71895172)

[Chapter 1 4](#_Toc71895173)

[Introduction 4](#_Toc71895174)

[1.1. Problem statement 4](#_Toc71895175)

[1.2. Aim specification 5](#_Toc71895176)

[Chapter 2 6](#_Toc71895177)

[Used Tools and Devices 6](#_Toc71895178)

[2.1. Gateway and protocols 6](#_Toc71895179)

[2.1.1. HTTP 6](#_Toc71895180)

[2.1.2. MQTT 8](#_Toc71895181)

[2.2. Devices 9](#_Toc71895182)

[2.2.1. Raspberry Pi 9](#_Toc71895183)

[2.2.2. Node MCU - ESP8266 10](#_Toc71895184)

[2.2.3. ESP32 CAMERA 13](#_Toc71895185)

[2.2.4. Sonoff 4CH Switches 15](#_Toc71895186)

[2.2.5. HR202 Sensor 15](#_Toc71895187)

[2.2.6. HC- SR505 Sensor 16](#_Toc71895188)

[2.2.7. Diffused RGB LED 17](#_Toc71895189)

[2.2.8. Google Home Mini 17](#_Toc71895190)

[2.3. Home automation system components 18](#_Toc71895191)

[2.3.1. openHAB 18](#_Toc71895192)

[2.3.2. myopenHAB 19](#_Toc71895193)

[2.3.3. Apache2 19](#_Toc71895194)

[Chapter 3 20](#_Toc71895195)

[User Documentation 20](#_Toc71895196)

[3.1. Installation 20](#_Toc71895197)

[3.2. Decision- making process 20](#_Toc71895198)

[3.2. Application requirements and deployment scenario 21](#_Toc71895199)

[3.2.1. Application requirements 21](#_Toc71895200)

[3.2.2. Deployment scenario 22](#_Toc71895201)

[Chapter 4 27](#_Toc71895202)

[Developer Documentation 27](#_Toc71895203)

[4.1. System plan 27](#_Toc71895204)

[4.2. Steps to install required components 28](#_Toc71895205)

[4.2.1. Raspberry Pi 3 28](#_Toc71895206)

[4.2.2. OpenHAB2 31](#_Toc71895207)

[4.2.3. Sonoff 4CH Switches 34](#_Toc71895208)

[4.2.4. ESP8266 37](#_Toc71895209)

[4.3. Web Application Program 42](#_Toc71895210)

[Chapter 5 45](#_Toc71895211)

[Testing 45](#_Toc71895212)

[5.1. Raspberry Pi 3 and related components 45](#_Toc71895213)

[5.2. ESP32- CAM 46](#_Toc71895214)

[5.3. ESP8266 and related sensors 47](#_Toc71895215)

[Chapter 6 49](#_Toc71895216)

[Discussion 49](#_Toc71895217)

[6.1. Implication 49](#_Toc71895218)

[6.2. Limitation 49](#_Toc71895219)

[6.3. Conclusion 50](#_Toc71895220)

[List of references 51](#_Toc71895221)

[List of figures 53](#_Toc71895222)

# Chapter 1

# Introduction

## Problem statement

The following work goes into detail on how Internet of Things technologies work and can aid in building smart house environment while minimizing the amount of electricity and resources used in order to give user the best experience in their own house.

The used tools and devices chapter covers the state of art and available technologies such as communication protocols and their characteristics, as well as commercially available Internet of Things devices and sensors as potentially viable candidates in helping to achieve the goal of this work.

With the knowledge gathered from many researches and testing, an educated decision was made on how to develop the project. The development plan goes into detail how an idea solution fitting the requirements would work for the given use case and outlines the functions of the system by specifying the system’s components and its main deployable elements. Additionally, the planning phase sets the guidelines for the user interface design and interaction, finishing with a sequential estimation of the development efforts needed to complete this project.

The development section shelters the main parts of understanding the created strategies and plans. Beginning with necessary steps for putting in place the primary infrastructure required to make the proof of concept work. After that, it goes into detail through the required configuration of the physical devices existing in the system, followed by the development of the web application which is used for user interface and user interacting with the system.

The works presented in this thesis are a part of my own project with deployed physical system in a realistic house. The system mainly demonstrates how Internet of Things solution works and could provide a way of controlling smart house’s environments also its behavior to make life more comfortable and easier with as little manual intervention as possible.

# Aim specification

There are many great opportunities that are available with the Internet of Things. The key promise of IoT is quick and continuous access to almost anywhere, which means we did not find valid in our research. It's all you need is a basic internet connection and a simple piece of hardware to do this. Let's say, for example, users could use Google Maps to figure out where they are instead of asking anyone on the street, make a reservation, or consulting our calendars automation is conveniently and inexpensively built into the Internet of Things. as a result, smart offices decrease the number of routine activities for the workers, and free them up their time and effort to engage in other tasks that are more demanding. In another point of view, it is better for time management, the Internet of Things is a fantastic time-saving device. It might be engaged in a conversation or have on the phone or reading our favorite news while we're in the car, we can also look for something in the online shops while out and websites, and place an order, and call it if we like. People will end up with plenty of time on our hands in the long run (Mudric, 2020).

By looking at various contexts in which things like the business world, the health care industry, and even people's own particular emotional lives, this study seeks to discover and extend principles of which can help people become universal practices. This is a simple illustration of how things like the Internet of Things function in the real world.

The aim of this project is to use Internet of Things (IoT) to make life at home smarter. This sensor is designed to gather information about the environment around the building, such as temperature and humidity, and note if there's movement, as well as any changes in it. For instance, if you want a particular theme to be automatically applied at a particular time of day or night, this will also provide you with the ability to allow that. It is probable that an extension of this strategy or anything similar might lead to an unusually abundant supply of resources in the local region.

# Chapter 2

# Used Tools and Devices

## 2.1. Gateway and protocols

### 2.1.1. HTTP

Hypertext Transfer Protocol – the protocol for the World Wide Web which is proposed by Tim Berners- Lee. It was a global information network allowing the flow of websites, and files to occur in a consistent manner through many languages. The W3C has maintained it ever since.

HTTP works by having servers and user agents that connect to them, enabling the agents to consume the data. Server location is finished by using a Uniform Resource Identifier (URI) or Uniform Resource Locator (URL) which are subset of the former set (Lafon, 2014).

The Hypertext Transfer Protocol follows the request-response model where a device, such as a web browser or app, sends a request to the HTTP server. either the response includes a status code stating whether the request was successful, or there was an internal error in the server, or not; in the latter case, the requested item might not be available.

HTTP specifies a number of methods for requesting server query, creating, modified, deleting or providing meaningful necessary data and accessibility for the server (R, 1999).

These methods are:

* GET: Requests the required resource representation. These applications should only get data and have no other impact.
* HEAD: However, the response body is not returned, similar to GET. Used to search for metadata and return a smaller payload size.
* POST: Adds to the server a new resource, as defined in the application body.
* PUT: Used in order to add a given URI resource. A new resource is provided if the URI is not available. The resource is changed if the URI is present.
* PATCH: used to implement partial resource modifications.
* DELETE: Deletes the resource listed.
* TRACE: Used for debugging via loopback testing of the message along the path towards the target resource.
* OPTIONS: Used on the specified URI to query the list of HTTP support methods. This is used to implement security restrictions on some resources by Cross-Origin Resource Sharing (CORS). The OPTIONS question is referred to as a pre-flight request here.
* CONNECT: Creates a TCP/IP tunnel, used to facilitate SSL-encrypted communication through an unencrypted HTTP proxy.

To ensure end-to-end security and to guard against man-in-the-middle attacks, a standard HTTP communication is developed over Secure Sockets Layer (SSL), or the more commonly used Transport Layer Security (TLS) (TLS). This enables the entire HTTP protocol to be encrypted and transmitted over SSL or TLS. These encryption schemes normally use long-term private and public keys to generate a short-term session key, which is then used to encrypt communication between the server and the client.

HTTPS is critical in publicly accessible networks, such as Wi-Fi hotspots found in cafes, libraries, and universities, since anyone linked to such a network can reasonably easily eavesdrop on communication transmitted over simple unencrypted HTTP. The same logic applies to situations in which Wireless Local Area Networks (WLANs) are needed to allow access to a large number of devices in an area that may be frequented by untrusted personnel.

Within the scope of this work, the HTTP(S) protocol can be used to create contact between components, most notably between a web-based application and a backend service that exposes application programming interfaces (APIs) through this protocol, as it is the de-facto requirement for web-based services at the time of writing this study.

The Remote Procedure Call (RPC) protocol is an alternative to HTTP. However, due to the RFC protocol's age and the greater prevalence of HTTP, HTTP support in modern programming languages is superior to RFC support (Sun Microsystems, 1988).

### 2.1.2. MQTT

MQTT- Message Queueing Telemetry Transport, is a publish-subscribe messaging protocol. It was developed as a lightweight messaging transport protocol atop the TCP/IP protocol suite. It is intended for use in situations involving machine-to-machine (M2M) communication and the Internet of Things. MQTT provides connections for remote areas that need a small code footprint or have very limited network bandwidth (Light, n.d.).

Clients interact with a server, which is sometimes referred to as a "broker," in a MQTT scheme. Any client may connect to the broker and either publish or subscribe to topics. Topics are classes of information that are coordinated and shared. A client subscribes to a subject, and when the broker receives a message published by a client within that topic, it distributes the information contained in the message to all topic subscribers. The publisher does not need to know anything about the subscribers, and the subscribers do not need to know anything about the publishers. Messages in a subject with no subscribers are discarded by the broker.

MQTT's data transmission is based on the TCP protocol. MQTT-SN is a version of MQTT that is used over insecure transport protocols like UDP or Bluetooth. MQTT does not have any encryption or authentication mechanisms and sends out link credentials in plain text format. The TCP protocol, which sits underneath MQTT, often provides these steps.

The ISO/IEC PRF 20922 specification defines this protocol. The Organization for the Advancement of Structured Information Standards (OASIS) is in charge of defining new standards enhancements, and it recently released the official MQTT v5.0 specification (OASIS , 2019), which represents a major step forward in the MQTT protocol's refinement. OASIS took special care to maintain backward compatibility when updating the standard to include features such as Message Properties, which adds metadata to the header of a MQTT message, Shared Subscriptions, which allows for load balancing across multiple clients, Message and Session Expiry, Topic Aliases, which provides mappings between topic strings in messages, and Message and Session Expiry.

Because of its low power consumption and small data packet size, MQTT is used by many IoT solutions to create communications between devices.

In addition, unlike HTTP, a MQTT link does not close after each request, leaving the communication channel open and incurring the overhead of opening and closing connections.

The MQTT protocol is simple to set up, and there are a variety of MQTT brokers on the market, making it an appealing candidate for choosing a protocol over which some of the IoT devices mentioned by this project could communicate (Krasnopolski, 2020). However, since the MQTT protocol is based on TCP, it necessitates several extra steps before a link capable of exchanging data can be created, which can affect “wake-up” and communication times. This may also have an effect on the device's power consumption, as many IoT devices have minimal processing power, making the process of creating a new TCP session more resource intensive and, as a result, increasing the device's total energy consumption.

## 2.2. Devices

### 2.2.1. Raspberry Pi

It serves as, in a sense, as a miniature, and effectively, a functioning machine intended for use in schools and developing countries, and was created by the Raspberry Pi Foundation with the aim of educating the next generation of computer science engineers and programmers about computing fundamentals. Moreover, the flexibility and practicality of the product surpassed the manufacturer's expectations in terms of sales and use cases (Upton, 2016).

By the time you are reading this, 4 big generations of Raspberry Pi have already been marketed to the public. With a Broadcom System-on-Chip (SoC) on which is an integrated Graphical Unit (GPU) to bring benefits to the table, all SoCs are assembled with a combination of a Broadcom SoCs and CPUs that provide the graphics processing benefit (GPU). There are from one to four USB ports on each board. This includes two HDMI and one composite video ports as well as a traditional 1/2.5" headphone and microphone port. Later versions of the Raspberry Pi added General Purpose Input/Output (GPIO) pins to make connecting and writing to additional devices easier and allow for greater programming flexibility. Later on, in the product development phase, B and B+ model versions were equipped with Ethernet networking capabilities. Since the Raspberry Pi 3 and Pi Zero have Wi-Fi access, it is also possible to add it to other versions. There are a variety of peripherals available for the Raspberry Pi, including cameras, screens, and other boards, it is not necessary to get any of these individually.

Moreover, Bluetooth and Bluetooth Low Energy communication are also supported by more recent Raspberry Pi devices. Some communications can be established by using add-in boards, commonly known as dongles, for the Raspberry Pi, or by attaching one of the available ports on the computer. Raspberry has powerful processing chips, which is a bonus for MQTT because of its minimal involvement on the performance of devices in general, allows single devices to handle multiple MQTT connections without requiring major disruption to their operations.

The Raspberry Pi Foundation provides Raspbian, a Debian-based Linux distribution, as the default operating system for these computers. The programming languages Python and Scratch are featured as the primary and default programming languages on this distribution due to their ease of learning, with support for several other languages provided by the operating system. The Raspberry Pi can also run a variety of other operating systems, including Windows 10 IoT Core, FreeBSD, NetBSD, RISC OS, Android Stuff, SUSE, Fedora, Xubuntu, and many others (Raspberry Pi, 2021).

The Raspberry Pi community is a fundamental component of the ecosystem as a whole. It is considered very rich and proactive because it is a relatively large group of developers, instructors, users, and enthusiasts who contribute to open source projects and maintain them, allowing companies and private individuals to actively invent and build new projects either for or with the help of the community.

Additionally, because of the Raspberry Pi devices' low price, they have become very popular among home automation enthusiasts. People who are interested in technology are changing and expanding the capabilities of Raspberry Pi devices in order to create more versatile, expandable, and cost-effective home automation solutions than those available on the commercial market.

Indeed, “ModBerry” is a computer based on the Raspberry Pi that was launched by TECHBASE in 2014, includes serial ports, CAN and Wire-1 buses, as well as digital and analog I/O, both of which are commonly used in the automation industry. This device's architecture makes it suitable for use in harsh industrial conditions, addressing the Raspberry Pi's shortcomings in terms of industrial IoT solutions (ModBerry, 2021).

The Raspberry Pi is considered to be potentially one of the most useful devices when bringing the concept behind this project to existence due to the excellent group driving the ecosystem, sponsorship, open-source software, and availability of Raspberry Pi devices, coupled with their affordability and ease of functional extension.

### 2.2.2. Node MCU - ESP8266

NodeMCU is an open-source Lua-based firmware and development board designed specifically for Internet of Things (IoT) applications. It includes firmware that runs on Espressif Systems' ESP8266 Wi-Fi SoC and hardware based on the ESP-12 board (Nodemcu, 2021).

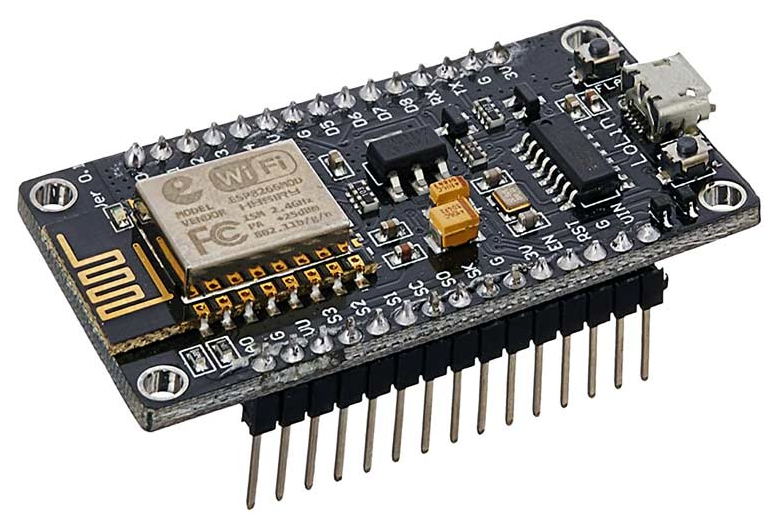


Figure 2.1.: ESP8266 device

The NodeMCU ESP8266 development board comes with a specific ESP-12 module which includes Tensilica 32-All RISC chip with Tensil-micro XTSU microprocessor (name of one of the supplier's chips). This microprocessor runs on an adjustable clock frequency of 80MHz to 160MHz and supports RTOS. To store data and programs, the NodeMCU has 128 KB of RAM and 4MB of Flash memory. It is suitable for IoT projects due to its high processing speed, built-in Wi-Fi / Bluetooth, and Deep Sleep Operating features. The NodeMCU is operated by a Micro USB jack and a VIN pin (External Supply Pin). It has interfaces for UART, SPI, and I2C (Last Minute Engineers, n.d.).

NodeMCU can be programmed using the Arduino IDE, which is made to be used by beginner programmers and less challenging for novices. Getting the Arduino IDE up and running with NodeMC will not take you longer than 5-10 minutes to do. You just need the Arduino IDE, a USB cable, and the NodeMCU, which comes with it, to use this kit. Here are the NodeMCU Development Board Pinout Configuration:

|  |  |  |
| --- | --- | --- |
| Pin Category | Name | Description |
| Power | Micro USB 3.3V, GND, Vin | Micro-USB: The Micro-USB NodeMCUU can be operated by plugging the device into a USB socket.  3.3V: 3.3V can be supplied to this pin to power the board  GND: Ground pins  Vin: External Power Supply |
| Control Pins | EN, RST | The pins and button to reset the microcontroller |
| Analog Pin | A0 | It is used to measure the analog voltage in the range of 0-3.3V |
| GPIO Pins | GPIO1 to GPIO16 | The microcontroller has 16 general purpose input-output pins |
| SPI Pins | SD1, CMD,SD0, CLK | NodeMCU has for pins for SPI communication |
| UART Pins | TXD0, RXD0, TXD2, RXD2 | It has 2 UART interfaces, UART0 (RXD0 & TXD0) and UART1 which is used to upload the firmware or program |
| I2C Pins |  | NodeMCU has I2C functionality support |

Table 2.1.: NodeMCU ESP8266 Development Board Pinout Configuration

NodeMCU ESP8266 Specifications and Features (Components101, 2020):

* Microcontroller: Tensilica 32-bit RISC CPU Xtensa LX106
* Operating Voltage: 3.3V
* Input Voltage: 7-12V
* Digital I/O Pins (DIO): 16
* Analog Input Pins (ADC): 1
* UARTs: 1
* SPIs: 1
* I2Cs: 1
* Flash Memory: 4 MB
* SRAM: 64 KB
* Clock Speed: 80 MHz
* USB-TTL based on CP2102 is included onboard, Enabling Plug n Play
* PCB Antenna
* Small Sized module to fit smartly inside your IoT projects

Here is the Pinout image of NodeMCU ESP8266:

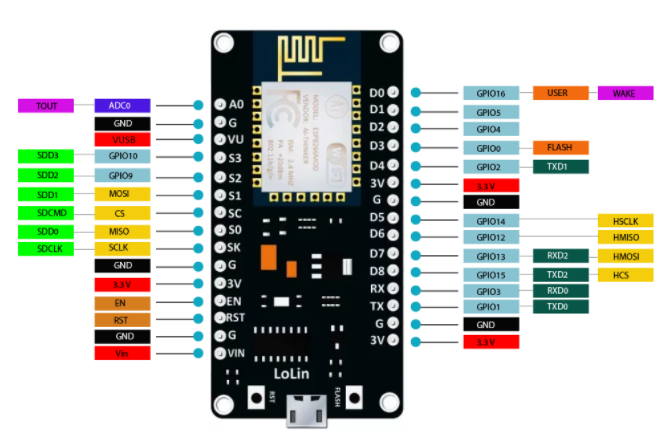


Figure 2.2.: ESP8266 Pinouts

### 2.2.3. ESP32 CAMERA

The ESP32 comes with integrated video and microSD socket for video, and acts as a microcontroller with an additional feature set of sensors. Camera purchases for use in general are very cheap and easy to acquire, but for cameras with advanced features like image tracking and recognition, this one is great value.

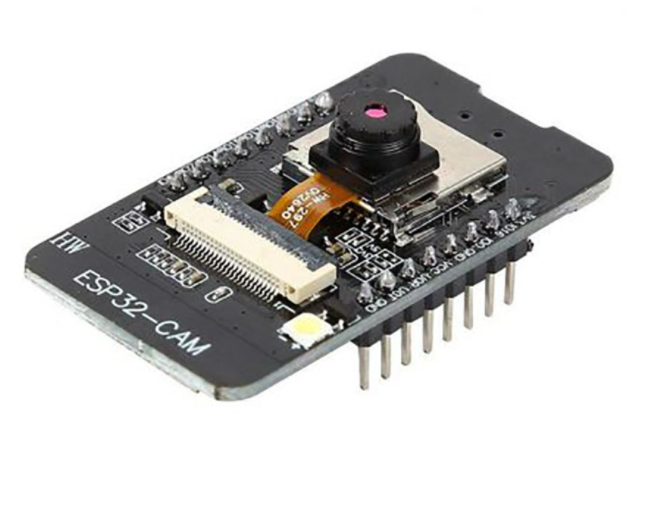


Figure 2.3.: ESP32-CAM device

The ESP32-CAM is based upon the ESP32-S module, so it shares the same specifications.  It has the following features (Dfrobot, n.d.):

* 802.11b/g/n Wi-Fi
* Bluetooth 4.2 with BLE
* UART, SPI, I2C and PWM interfaces
* Clock speed up to 160 MHz
* Computing power up to 600 DMIPS
* 520 KB SRAM plus 4 MB PSRAM
* Supports WiFi Image Upload
* Multiple Sleep modes
* Firmware Over the Air (FOTA) upgrades possible
* 9 GPIO ports
* Built-in Flash LED

There is one minus point for this device is that the ESP32- CAM board does not have any USB port, so it cannot connect to the computer and start loading program. Instead, it needs an FTDI adapter which is the same adapter used for modifying the Sonoff 4CH Switches in the next chapter.

### 2.2.4. Sonoff 4CH Switches

Sonoff4-gang switches (also referred to as 4CH & 4CH R2) are built into-in 4-gang din rail enclosures. This Sonoff switches (grouped together and controlled by buttons or via the ‘EweLink’ app normally) has four-electrical appliances with one switch that is compatible with the smartphones and tablets, allowing them to be controlled separately or in turn on or all at once (店长, 2017). Secondly, the multi gang wiring feature of App allows users to see the status of several devices on the screen simultaneously. Combining 4 Sonoff switches in a single ownership means that you will have the ability to have a single Sonoff WiFi-R2 unit.

Din rail support is used to hold the board inside the enclosure! Many circuit breakers and industrial control equipment installations use DIN rails, which are a typical D-rail system. Usually, these items are made of cold-rolled carbon steel with a chromium or zinc-plated steel.

The Sonoff 4CH Wi-Fi light switch is without a doubt a member of the family, it performs all the essential functions, users can switch on or off from anywhere at any moment, with the option to set to make schedules that run once or continuously, users can all-rearrange schedules to make things go on/off at the same time, or all together to set them to stay on a given time and to single/off together.



Figure 2.4.: Sonoff 4CH Switches

The Sonoff 4CH in this thesis is softly modified by the aim of usages to control all the devices such as light, TV and so on using MQTT communication technology. It could be reprogramed easily by using the FTDI adapter to expand its abilities.

### 2.2.5. HR202 Sensor

This sensor uses operating voltage in range of 3.3V- 5V and has the AO as analog output and DO as digital output. It is deal for custom humidity sensing applications, like meteorology, textile facility control, and storage building humidifying applications are those which require sensing the current humidity levels in the environment as well as the relative humidity. A current-dependent or voltage-dependent output is often called an analog or the output depending on the Comparator integrated circuit (LM393), is possible both as well as a digital. This sensor is based on HR202, having a greater linearity, wider measurement range, and lower power consumption than the hygrometric sensors are utilized in many previous designs. There is an LED indicator to show the amount of power input, and a digital indicator to show output. It is inexpensive and very low power consumption (Elecrow, n.d.).

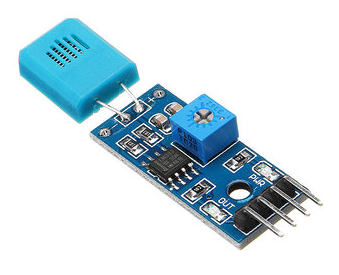


Figure 2.5.: HR202 sensor

### 2.2.6. HC- SR505 Sensor

HC- SR505 sensor is a smaller version of the PIR (Passive Infra-Red) motion sensor which accounts for that as the small room size and reduced ability to discern subtle changes in an infant in its presence. IR technology is used to expand the beam width and the intensity. This feature is highly efficient since it is able to maintain and/increase intensity when the system control is unattended. Because of its minimum size and low-power operation, it is used in battery-powered devices as well as in others.



Figure 2.6: HC-SR505 senso

HC- SR505 is a three pins device which are:

* Ground (GND)
* Signal Output (OUT)
* Positive Power Supply (+)

This sensor has wide range of operating voltage with minimum size and repeatable trigger (Components101, 2018). The module is used mainly in applications where the motion of a body emitting infaring radiations can be detected. The HC-SR505 is favored in battery systems as the unit is designed to be operated with a very low power consumption. The small size allows for the installation of the module in applications with consideration of device size. Because of its wide range detecting motion, it could be used as motion alarm, automatic lighting system, theft protection, etc. In this study, it will be acted as motion alarm to detect if any object is moving.

### 2.2.7. Diffused RGB LED



Figure 2.7.: Diffused RGB Led

As what it is called, diffused 10mm LED with 3 different separated red, green and blue LED chips inside. It has 4 pins: GND and RGB pins. The purpose of this led in this project is to simulate real life RGB light but smaller version.

### 2.2.8. Google Home Mini

The Google Home Mini was invented by Google, which is a voice-activated speaker that can play music, monitor smart home devices, answer trivia questions, add items to a shopping list, schedule appointments, and play video on a Chromecast-enabled screen. Google recently introduced the ability to make phone calls from the Home speakers, as well as use them to track down your phone if you've lost it. Compact, convenient, the Home Mini smart speaker offers much of the same smart speaker features as the Echo Dot for less than the full price at a smaller size. always-listening Wi-fi connected speaker is something you can fit in your home that is very compact, inexpensive and gives you a broad coverage (James, 2018).

Google Home Mini has the size of a “donut” with many good- looking colors such as light gray, dark gray and coral red. It could be turned on by using the “OK Google” or “Hey Google” wake words. There are many activities that could be done with Google Home, but in this thesis, it will be the device which listen to the command and control things using Google API connected to openHAB2 (Saxena, Jain, Arora, & Sharma, 2019).

## 2.3. Home automation system components

### 2.3.1. openHAB

Open Home Automation Bus (openHAB) is an open source project dedicated to creating a community-oriented and pluggable platform that helps you assemble IoT devices regardless of their type, whether they use a proprietary or open protocol.

Furthermore, the features of many mobile apps have been restricted in their use and deployment because of the lack of integration and application possibilities are inherent. Unlike with Android, however, the openHAB is centric, and centers on what is wanted by the end user, which is feasible and offers a variety of ways to accomplish these tasks (OpenHAB, n.d.).

Using so-called objects, openHAB abstracts away physical devices or virtual data sources. These things free the user from having to deal with different IP addresses, MAC addresses, or the like, allowing them to concentrate on what they represent, which is normally in a human-readable and understandable format. This makes it easy to replace devices in an IoT network.

When it comes to replaceability, it's important to remember that openHAB is built on a modular architecture, which allows for the safe and inexpensive addition of new features like bindings, which provide access to other technologies, extensions, automation rules, and so on. Because of these advantages, openHAB is extremely common among home automation enthusiasts who are eager to contribute to the open source project.

OpenHAB can run on a number of operating systems, including Windows, Mac OS X, and Linux. OpenHAB is also available as pre-configured, easy-to-install images for Raspberry Pi (openHABian), Docker, PINE64, and other platforms. This allows for a relatively quick and efficient installation of the openHAB platform, which is particularly useful when only preliminary testing is needed.

The openHAB program, on the other hand, is not cloud-ready on its own. There are a few options, including hosting the openHAB runtime in the cloud, but this would necessitate a complicated networking configuration to ensure that the gateways and their connected devices can communicate with the runtime, which requires that they be on the same logical network. Fortunately, the openHAB project has a cloud companion feature that allows you to link the openHAB runtime to the cloud and expose all of the platform's related features over the Internet.

Nowaday, openHAB is proved to be a good choice for managing the IoT platform for the project due to its ease of setup and the associated advantage of focusing development efforts on delivering better user experiences in less time.

### 2.3.2. myopenHAB

myopenHAB is a cloud-based service instance that connects a running openHAB system to the Internet through the Cloud Connector extension and provides functionality like remote control, push alerts, and integration with mobile and third-party apps like Amazon Alexa, IFTTT, Google Assistant, and others.

While the openHAB Foundation gives sufficient instances of the myopenHAB cloud service instance, the cloud service instance, like everything else in the openHAB project, is open source, which means that there are builds and images that can be deployed to any cloud service provider without many issues, usually resulting in better performing instances than those hosted by the openHAB Foundation (OpenHABFoundation, n.d.).

The key disadvantage of using the myopenHAB cloud service is that, unlike the solutions offered by AWS, GCP, or Microsoft Azure, all component management, such as updates, reliability, and any load balancing, must be done manually. But luckily, its downside of it does not affect the result of this project in general.

### 2.3.3. Apache2

On December 1, 1995, Apache, a web server, was launched for the first time. Under the sponsorship of the Apache Software Foundation, it is actively operated by open-community developers. Among other web servers, it is the most commonly used and popular today. Apache is a cross-platform, open-source web server that is widely used in Linux distributions. According to the Web Technology Survey, Apache is the most popular web server today, hosting 38.4% of all live websites. The fact that apache has a module-based framework makes it extremely scalable, and it has modules for authentication and URL rewriting are only a couple of the reasons why it is the most common server program (Istifanos & Tekahun, 2020).

# Chapter 3

# User Documentation

## 3.1. Installation

This section describes the initial mandatory steps to build the whole Internet of Things system to simulate smart house and also discuss how users could use the system.

My aim was for the application to include two systems. The first ensures that IoT devices can be handled and that data and commands can be retrieved and released. The second system will be in charge of showing the web control panel from which the user in the family could track and control the IoT devices.

## 3.2. Decision- making process

One important requirement for this project is to create a nice website application interface with a consistent look, secure and presence. It is based on JavaScript, HTML5, CSS and PHP. Moreover, it should provide abilities to support for RESTFUL API.

The table light will be chosen as the light in the real house's hallway for simulating purposes, low cost, and high practicality, while still being energy efficient and ideal for the project's scenario. Another lighting source was the RGB Diffused LED, which was chosen to demonstrate some of the power of MQTT communication.

The HR202 provides a practical humidity sensor solution for capturing the humidity of the house data due to its high reliability ratings and low cost. The SR505 sensor was selected to detect motion because of its ability to detect movement over a wide range of angles.

The Sonoff 4CH Switches acts as a relay and were chosen to control the other devices because of their ability to communicate via Wi-Fi. On the other hand, for the Sonoff switches, a MQTT service can be set up manually by customize its program that will publish their connected devices activities on unique channels and thus inform all interested parties.

Because of previous decisions to use commodity hardware and the project's tight budget, openHAB was chosen as the IoT platform on which the device will be hosted because it can operate on inexpensive hardware like the Raspberry Pi and offers a REST API. Because of the strong community behind openHAB, there is support for a wide range of consumer off-the-shelf devices that will be used in the project, making setup as simple as possible.

In addition, the myopenHAB cloud builds feature like remote control and third-party apps which can be linked to the Google Home Mini makes openHAB the most important choice for this project.

Many of the tedious configuration processes presented in the IoT Service Solutions for onboard IoT devices may show major developmental slowdowns. In addition, these solutions also have a subscription charge or usage fee, which cannot be distributed at phases if only proof of concept is presented.

## 3.2. Application requirements and deployment scenario

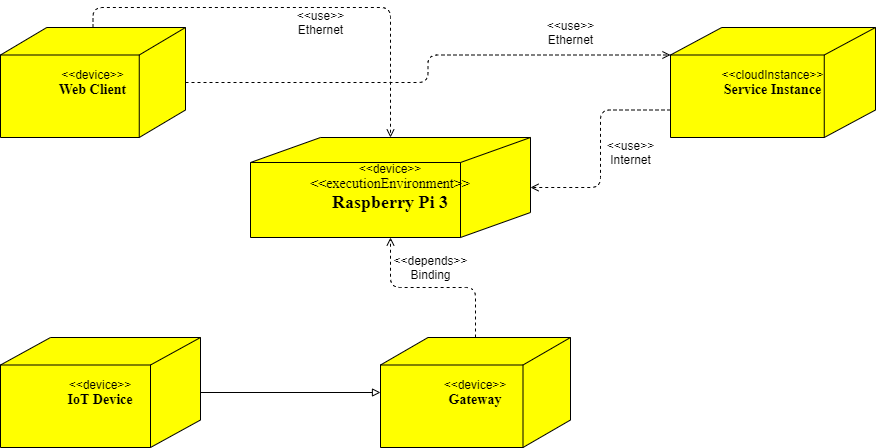


Figure 3.1.: Deployment Diagram

### 3.2.1. Application requirements

Upon getting interested in this idea and this topic specifically, the requirements had been discussed with my supervisor, where the following key components were defined:

Establishing two-way communication: Ensuring that the backend system is functioning properly and that the front-end wen application can send commands and receive status notifications from the connected IoT devices.

Providing automation systems: It is possible to automate one or more onboarded devices in response to various types of actuations, such as a specific time of day, or a theme to adjust the state of another IoT device in the system, and so on.

Allowing for simple device onboarding: When a device is connected to the system, the setup should be minimal, and the onboarding procedures should need as little feedback from the user as possible.

Pleasant control panel configuration: The user should be able to use the control panel with ease, according to their expectations and needs. The design of the control panel should be flexible enough to accommodate as many different configurations as the user requires. Security is also a top priority.

## 3.2.2. Deployment scenario

The use of technological systems, automated procedures, and wired, remote-controlled devices in apartments and houses is referred to as a smart house. The functions' key goal is to enhance the quality of life and ease of use in the home. Other objectives include improved protection and energy efficiency due to wired, remote-controllable devices.

Internet of Things is awesome, and this project may be used as a base for a time-controlled feature of home appliances such as the washing machine, lighting, or coffee maker. Motion sensors, cameras, shutters, and thermostats, for example, start user-programmed processes. The central control device, with which various smart components are connected and can be managed from a PC, smartphone, or tablet, is at the heart of the smart home. For communication or system control, popular wireless protocols such as Wi-Fi or Bluetooth are could be used.

More specifically about the project, the simulation should allow users or family members to interact with the system by sending commands to the devices and observing their actions and monitoring the data collected from said devices.

**3.2.3. User document of web application**

This section will discuss about general requirement of the web application and steps to use it.

In order to allow users to use, manage the smart house, it is required to create a small web application. The main idea is that it would help controlling and managing the device. In order to allow this, a web application would be hosted from Raspberry Pi 3 as a product of HTML, CSS, JavaScript and PHP that is accessible from anywhere in the world via the Internet. The web application is hosted in this project and can be accessed with a local address. The URL for the main page is http://192.168.1.110 (IP of the Raspberry Pi which is configured to be fixed in my network system). The below are the major pages and their functionalities that are part of the web application:

**Login Page**

Until users can do something, they must first navigate to the first main tab, which is the login page. It is accessed by directly accessing the Raspberry Pi's IP address in the local server.

To access the controlling tab, users must first create an administrator account." In this project, the username is "admin," and the password is also "admin". After entered, the browser will be directed to the home panel page.

**Home panel page**

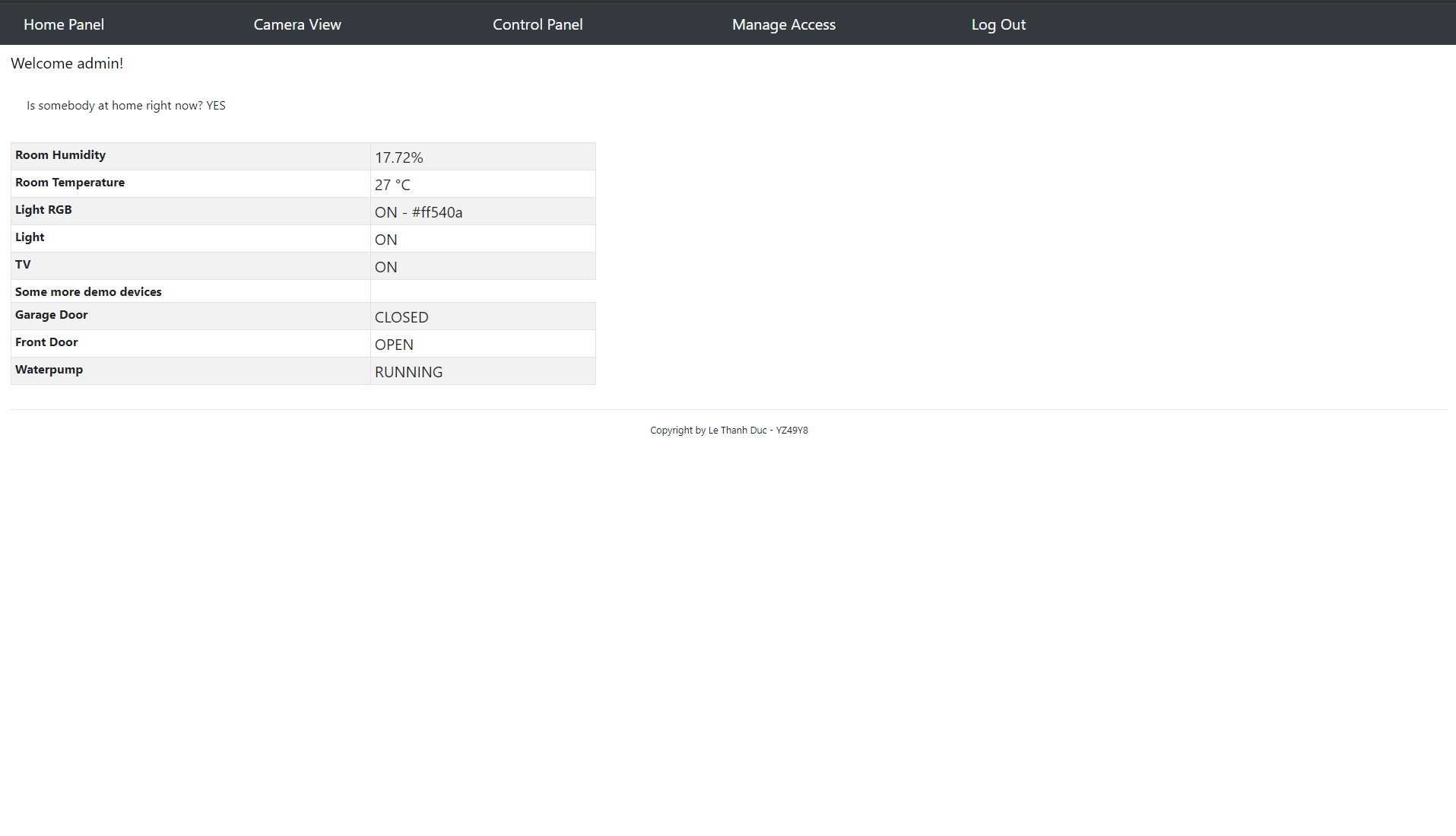


Figure 3.2.: Home Panel page

The home panel page is used to show the name and status of all related devices. Users could check if there is anyone at home at the moment using the data got from the motion detector. they may also check for example temperature, garage door, front door (in this project these value are put as a demo for demonstration purpose), humidity level, and so on depending on the devices setup. From this page, users could navigate to Camera View, Control Panel, Manage Access (in case of administrator account) and Log out.

**Camera view page**

Firstly after navigating to the webpage, a panel with camera setup options would appear for the user to choose. Users may change the resolution, image quality, brightness, contrast, and other settings. The camera's face detection and recognition technique can also detect whether or not a human is in range.

Secondly, when all of the settings are in place, the “Start Stream” button can be used to show the real-time video, and the “Get Still” button can be used to capture the image, which can then be saved to the local SD card for later use.

**Control panel page**

This page's main function is to control all of the gadgets in the smart house scheme. The user will pick choices for each linked device. In this project, for example, users might switch on and off lights or Television, change the RGB color, and so on. In the other hand, as part of future work, it may also monitor the volume of the TV by attaching to either of the IR sensors or monitoring the house's water pump, etc.

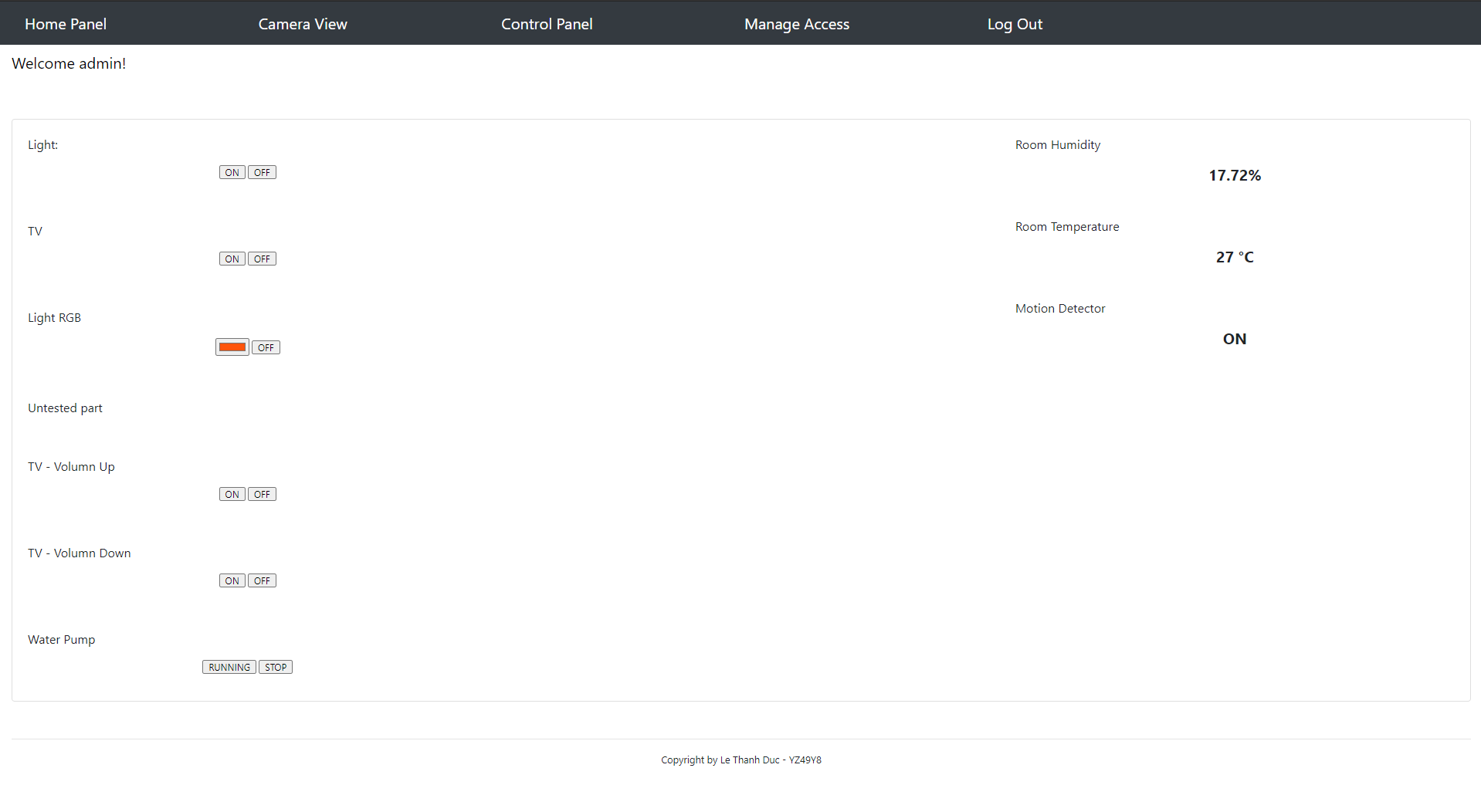


Figure 3.3.: Control Panel page

**Manage access page**

This website is only accessible to the user with administrator privileges in order to protect the privacy of the smart house system or, in this case, the account "admin." It enables the administrator to create new accounts for users to access the system. Users may also search who is already on the system's accessible list and manage their account.

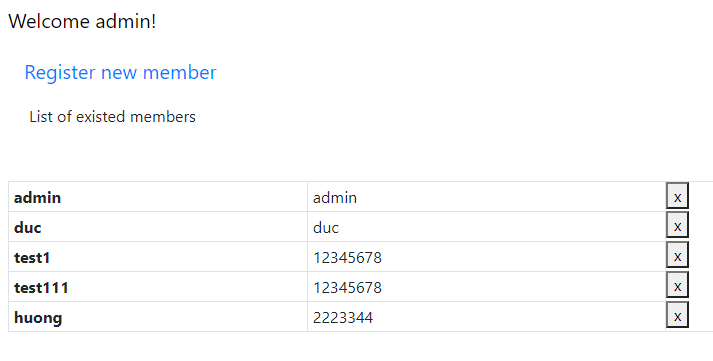


Figure 3.4.: Manage access page

**Register page**

This page is also accessible only to users with the administrator role. It simply serves to register new members for the smart house application. It is necessary to enter a new username and password. To ensure that the user entered the password correctly, the password field must be double checked by repeating the password field. After clicking the "register" button, the browser would search the database to ensure that the username is unique. In the case of a successful registration, the user will be forwarded to the login page.

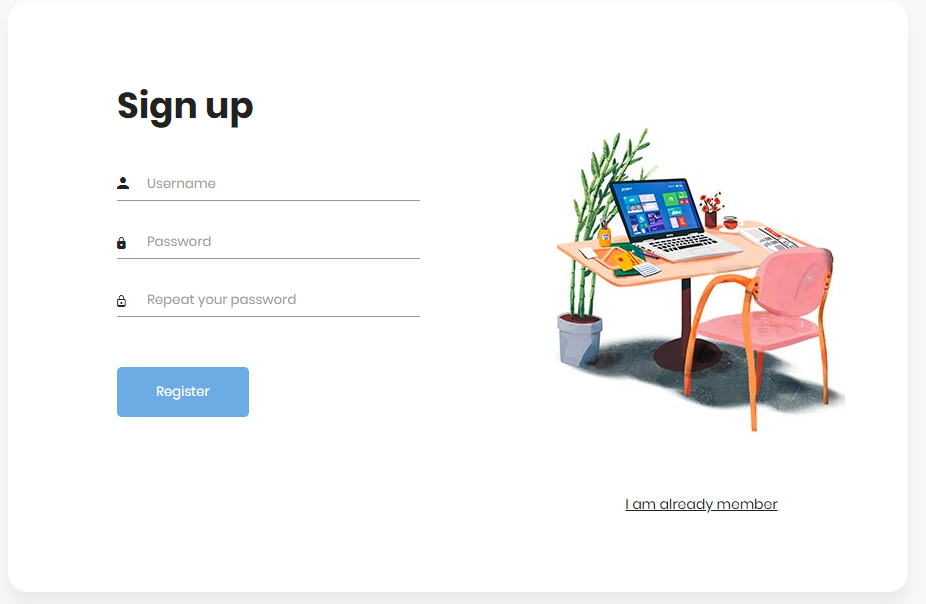


Figure 3.5.: Register new user page

**Log Out page**

The current user is simply logged out and redirected to the login tab on this page after 3 seconds.

Overall, the Raspberry Pi 3 serves as the primary component, hosting this web application from Apache2. The PHP, HTML, and Javascript programming languages are used to create all of the websites. All user data is contained in a JSON format, which can be transferred to a database in the future for privacy and security reasons. In general, the openHAB RESTFUL API manages correspondence between the webserver and the openHAB MQTT broker. Following a good login to the main tab, all necessary information will be accessed using the PHP feature using the GET request method sent to the IP of the openHAB server with the “/rest/items/” subpart. Furthermore, JavaScript is used to track and control the ON/OFF button.by sending POST requests to the corresponding device’s URL.

Due to the small budget, it was decided that the IoT devices such as sensors and light are enough to simulate the communication and demonstrate the real-life devices. They are all physically located in the house with easy deployment to give potential service efforts of Internet of Things.

# Chapter 4

# Developer Documentation

Installing Raspberry Pi 3with openHABian, setting up MQTT Broker, openHAB2, customizing firmware for Sonoff 4CH Switches using available Tasmota firmware, setting up sensors(ESP32-CAM, HR202, HC-SR505, etc), and developing a web application are all covered in this chapter.

## 4.1. System plan

The system plan below illustrates by the main components comprising the small house. All the sensors will be managed by the ESP8266 chip on the bread board. Google Home Mini would be used to control all the things which are connected on the same MQTT system. Raspberry Pi 3 will be the MQTT broker providing connection and communication, also hosting a web application server.

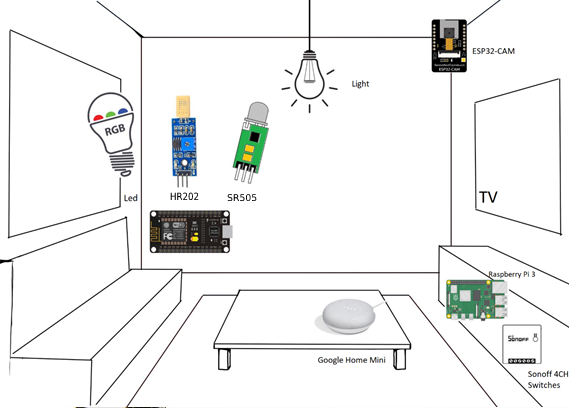


Figure 4.1.: System plan illustration image

The room light and TV are controlled by Sonoff 4CH Switches together with all the sensors which are IoT-enabled, which means they can be operated from any Wi-Fi connected devices that can track their status and issue commands to them. The gateways, as well as any other devices that are responsible for the connectivity and control of these devices, are in appropriate areas within the house.

## 4.2. Steps to install required components

### 4.2.1. Raspberry Pi 3

**openHABian**

Firstly, I decided to take a Raspberry Pi 3 model B for the initial phase of this project and for the proof-of-concept stage because I had it available already and the solution for deployment scenarios like these is fast and simple, especially when the costs of the system themselves are relative small. This is also a very smart device with a really low energy efficient.

Secondly, the Raspberry Pi 3 model B could easily use a ready-made image of the Raspbian Lite Linux distribution named openHABian made available by the openHAB project, which takes care of all setup phases and configuration through shell scripts included in the image. Writing the openHABian image file available on the openHAB project website to an SD card and inserting the card into the Raspberry Pi's slot are the only steps required to deploy a functioning instance of openHAB on a Raspberry Pi utilizing this process (Smart Home Blog, n.d.).

Following that, I needed to connect the Raspberry Pi to the Internet, which could be done with a UDP cable or, if the Raspberry Pi has Wi-Fi capabilities, the network's service-set identifier (SSID) and pre-shared-key (PSK) should be filled out in the openhabian.conf file on the SD card before inserting it into the Raspberry Pi as values for the Wi-Fi SSID and Wi-Fi PSK respectively. The next move is to add power to the Raspberry Pi and wait 15 to 45 minutes for the openHAB instance to be mounted on the Raspberry Pi, depending on the speed of the Internet connection. The progress of the installation can be tracked using the http://openhabian or http://openhab URLs, depending on the image version, or by connecting to the Raspberry Pi's local IP address through HTTP.

Since the openHABian operating system is installed with default usernames and passwords, there is a disadvantage to this simplified version of the openHABian operating system on the Raspberry Pi. Fortunately, the openHABian Configuration Tool makes it easy to adjust these settings. The “sudo openhabian-config” command can be used to use this utility from the terminal. OpenHABian can also be mounted on other Linux distributions in the same way as every other software is. Since we'll be pulling the openHABian repository from GitHub, the only prerequisites is that we have git built on our operating system.

These commands are to manually install the openHABian:

|  |
| --- |
| sudo git clone https://github.com/openhab/openhabian.git /opt/openhabian  sudo ln -s /opt/openhabian/openhabian-setup.sh /usr/local/bin/openhabian-config  sudo openhabian-config |

**Apache2**

Firstly, I need to make sure we have an up-to-date computer after installed the openHABian instance and before downloading the Apache2 server. To do so, we will need administrator privileges, which we can get by using the sudo command. After the Raspberry is up to date, it is enough requirement to install Apache server now. These commands are for update and install the Apache server:

|  |
| --- |
| sudo apt update  sudo apt upgrade  sudo apt update  sudo apt install apache2 |

Following that, we will need to consider changing the server’s files permissions so that we could modify the inventory easier:

|  |
| --- |
| sudo chown -R pi:www-data /var/www/html  sudo chmod -R 770 /var/www/html/ |

The website's root directory is /var/www/html, which is used by Apache. This means that Apache searches for the file in "/var/www/html" when we call the Raspberry on port 80 (http).

The home page of Apache which we could access is the IP of the Raspberry Pi in the local network. Moreover, if users type http://”IPofRaspberryPi”/example into the browser, Apache can search for the "example" file in the "/var/www/html" directory.

After all, we can now use your Raspberry to build an internal website using HTML, CSS, and JavaScript. Those who can, however, want to make easy interactions between the site and the customer. To allow the user to log, for example. This will necessitate the use of PHP.

**PHP**

PHP is a free programming language that is maintained by the PHP Foundation, Zend Enterprise, and a number of other organizations.

PHP is mostly used to render a website interactive, which means that the user sends data to the server, which then returns updated responses based on that data. A static site, on the other hand, does not respond to the information supplied by the customer. It will still provide the same content since it is stored as a file once. It's one of the most popular programming languages, and it's still the most popular for web development.

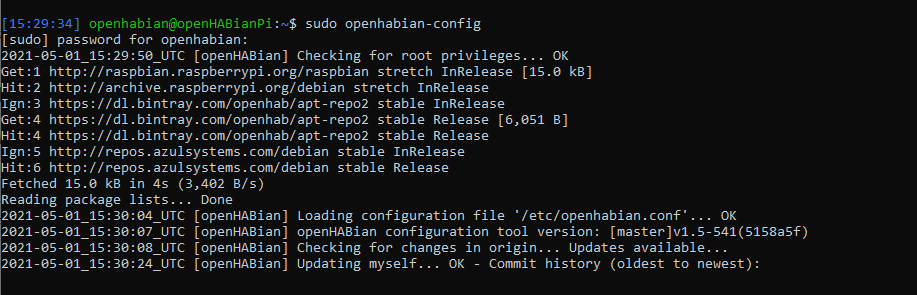
To install the PHP on the Raspberry Pi 3, I simply used these following commands:

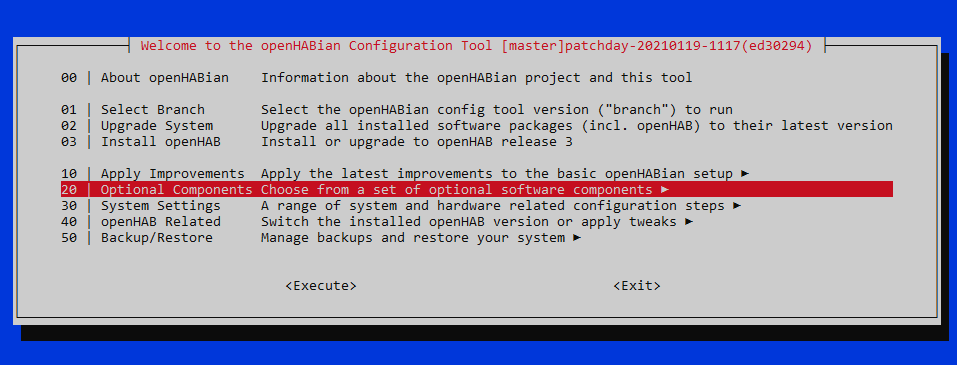
|  |
| --- |
| cd /var/www/html/  sudo apt install php php-mbstring  sudo rm index.html  echo “<?php phpinfo();?>” > index.php |

Because we already had the index page of Apache2 after the installation, we would need to delete the index file after in order to create a new php main page. After all, we could check the page using the same method used for Apache2, typing this link to the browser: http://”IPofRaspberry”. The webserver home page should display the version and information of the PHP installed.

**Mosquitto**

This sections describe the steps to install Mosquitto on Raspberry Pi 3. After setup the openHABian and its component successfully, we could use command “sudo openhabian-config” to check around the setting and configure it based on purpose of usages. Here I choose “Optional Components” and choose Mosquitto to setup the MQTT Broker:





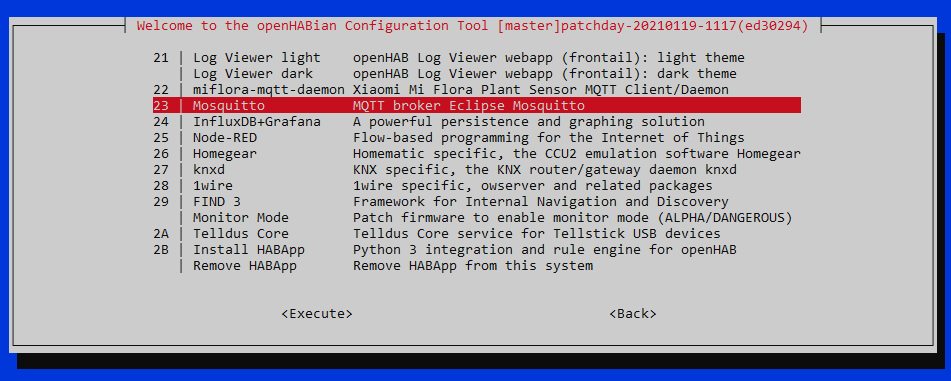


Figure 4.2.: Steps to install Mosquitto Broker

After all the installed, the Raspberry restart step is required to apply changes.

### 4.2.2. OpenHAB2

It's time to link the openHAB server to the MQTT Broker now that it is been built on the server. The Raspberry Pi's IP with port 8080 could be used as the openHAB's IP. The best feature of openHAB based on opinion is the Log Viewer, which allows one to see what's going on through adjustment and processing. Port 9001 could be used to access the Log Viewer.

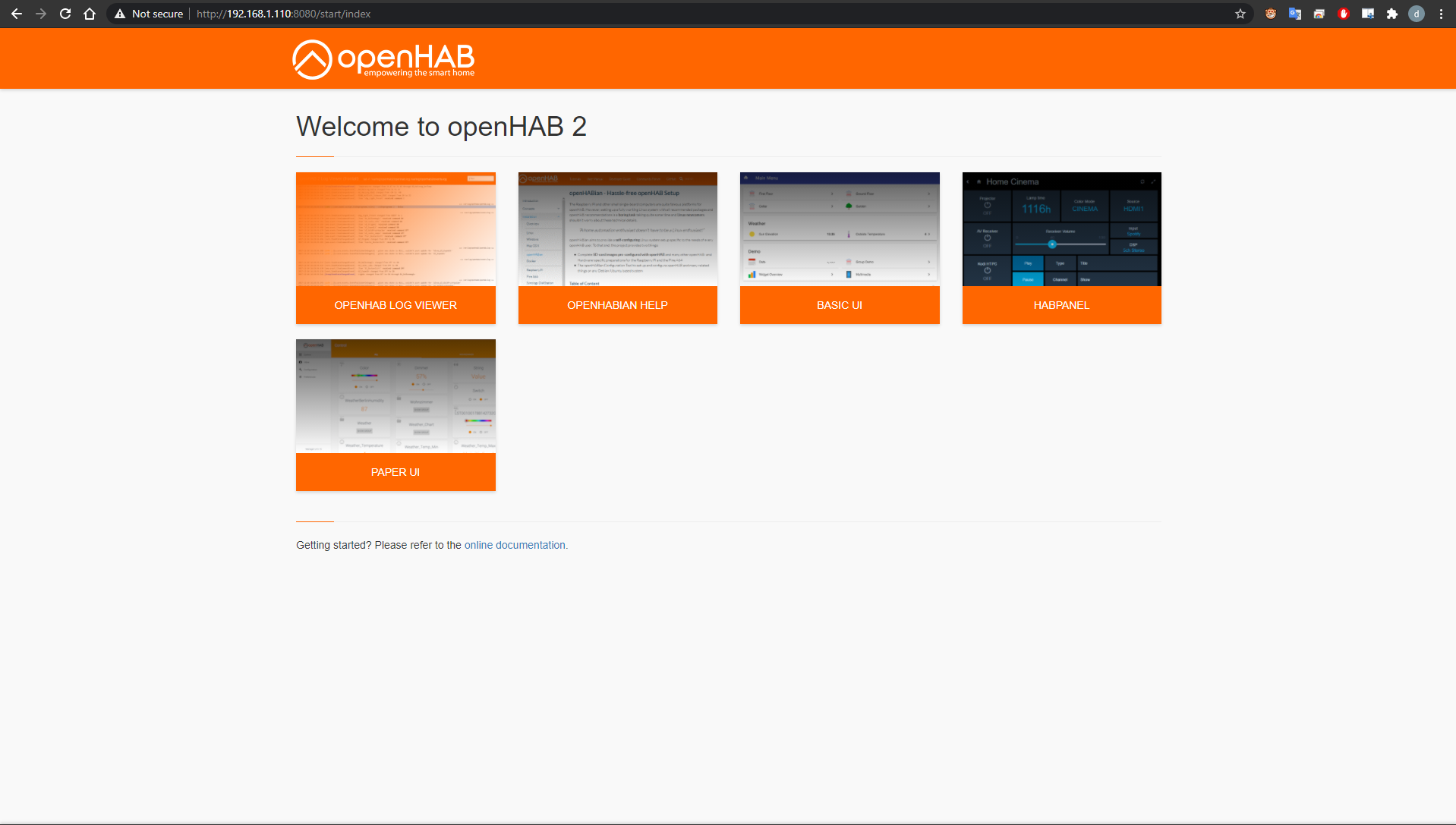


Figure 4.3.: OpenHAB webserver homepage

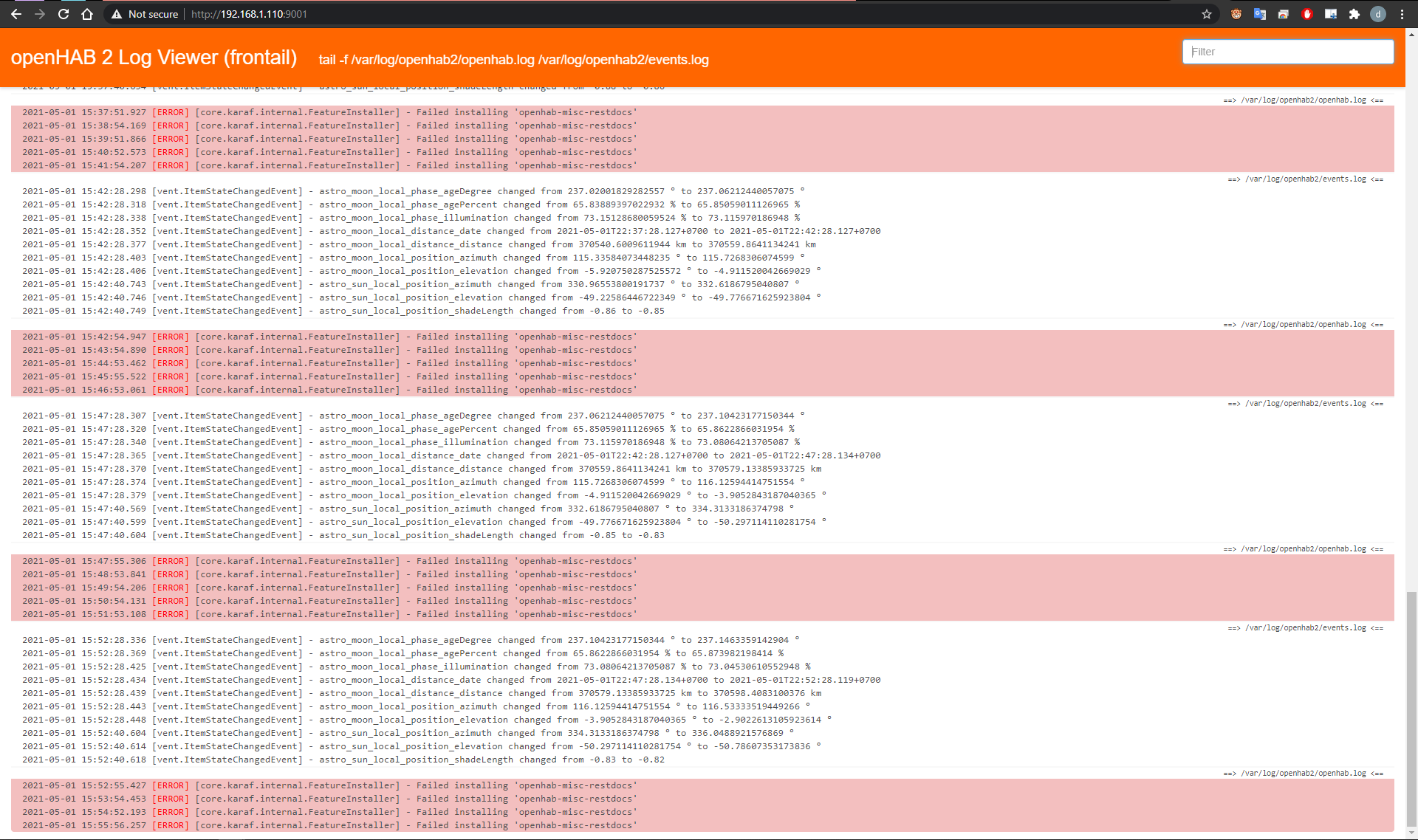


Figure 4.4.: OpenHAB LogViewer

The openHAB is connected to the Broker by Binding, which might be turned on by going to the PaperUI - Addons – search for MQTT Binding and install it. Once it is installed, I need to config its config file so that the openHAB knows how and where to connect to the MQTT Broker by accessing the /etc/openhab2/services/mqtt.cfg file and change value of broker.url = IP\_of\_the\_RaspberryPi : port 1883. Here is the result image:

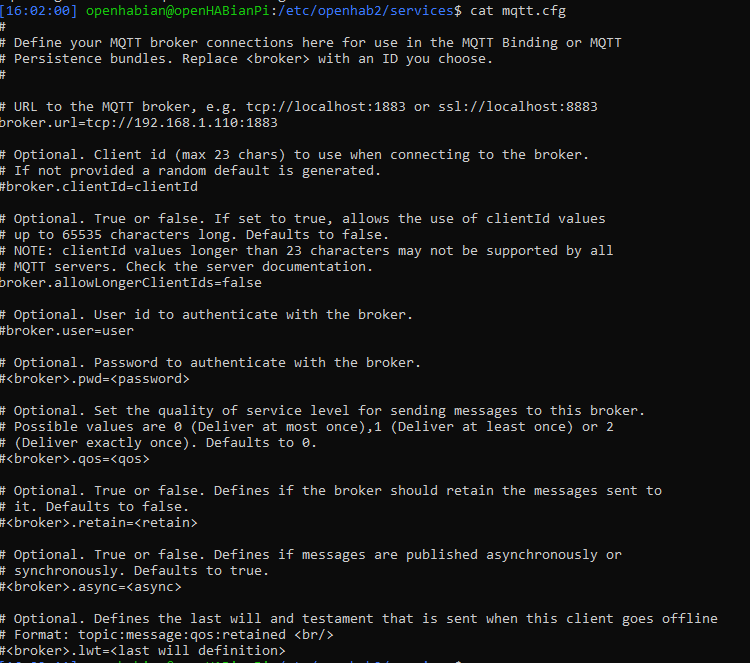


Figure 4.5.: mqtt.cfg file

The openHAB Cloud Connector extension must be enabled and modified in order to configure the Raspberry Pi and the openHAB instance running on it to link to a cloud-based instance of myopenHAB. This is achieved easily with openHAB's Paper UI, where we can install the openHAB Cloud Connector extension from the Add-ons menu in the "MISC" tab. The next move is to set up the openHAB Cloud Connector to interface with a cloud-based myopenHAB service. By default, the openHAB Foundation provides a free alternative, which can be found at https://www.myopenhab.org/. If using the openHAB Foundation's choice, we need to register and/or login to the web application, as well as have two key files from the openHAB framework running on our controller computer. The two files are called “UUID” and “Secret” and can be found in the folders /var/lib/openhab2/uuid and /var/lib/openhab2/openhabcloud/secret, respectively. They both contain strings that should be copied and pasted into the web UI of the operating myopenHAB cloud instance to ensure that connectivity between the cloud provider and the controller interface is functioning and encrypted. Finally, the openHAB could be accessed externally now via Internet.

In this project, openHAB is the center of the home automation system. All of the properties and capabilities of the devices and sensors are processed through openHAB to the user interface. Any device linked to openHAB is unique in terms of functionality and logic. OpenHAB determines the following foundation modules to reflect all of these:

* Add-ons - There are a variety of Add-ons available to connect with the devices.
* Things - openHAB's representation of devices and sensors
* Items - organized into groups, which are lists or categories
* Transformations - Data transformation helper functions
* Persistence - Services that keep data for a long time
* Rules – Automation logic
* JSR223 is a type of JSR. Scripting - Use Javascript, Jython, or Groovy to define rules and other runtime items.

All of these are located inside folder at “/etc/openhab2/” and ready to be configured after.

### 4.2.3. Sonoff 4CH Switches

As the previous article illustrated, the Sonoff 4CH Pro is a control box that allows user to control up to four relays using Wi-Fi or radio 433MHz. The steps to mount the Sonoff 4CH Switches in hardware and software are shown in this section. The Sonoff 4CH may be present as part of the project's key controller components. Because the original firmware does not allow MQTT communication or set it up, a customization and reinstalment is needed. It is necessary to search for a replacement firmware and I came across “Tasmota” on Github. The only requirement is that it needs an FTDI adapter to change the software, which is very low-cost device with many advantages.

Theo Arends created and maintains the Tasmota firmware, it is based on the same principles as the Homie firmware. It adds a web-based configuration interface that can be accessed from any device, tablet, or smartphone connected to the same local network. It then communicates with a home automation server, an internet server, a smartphone device (iOS, Android), and so on, using MQTT messaging (Arends, n.d.).

The process requires linking to the ESP8266 chip's serial programming interface. This is accomplished by attaching the TX and RX pins of our serial-to-USB adapter to the RX and TX pins of the ESP8266 and powering the chip with the 3.3V and GND pins. Pin headers or jumper wires must be soldered or otherwise added in most situations so such pins are available on the PCB in the shape of pin holes or solder pads. In certain cases, soldering wires directly to the chip's pins would be also needed:

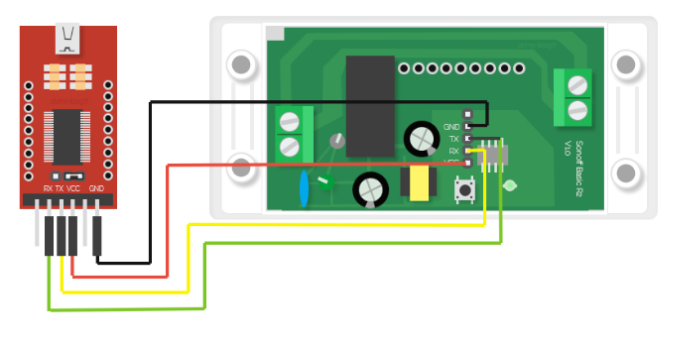


Figure 4.6.: Mapping from FTDI to Sonoff 4CH

Until the firmware can be uploaded, the ESP8266 of the board must be put into programming mode or flash mode. Pulling the GPIO0 pin to GND when the chip is booting accomplishes this. The mounted control button on most devices is attached to GPIO0 and GND, making accessing Programming Mode very easy. On some case, it will need to use a jumper wire to bridge the pins on the PCB or directly on the chip. After downloaded from Tasmota’s Github repositories, some coding modification is needed by using Arduino code with the same name of software. I was really struggled at the first step to understand its configuration and code, which resulted in several test failures. The most challenging challenge here is that the connection between the PC and the computer keeps breaking when uploading the new firmware to the board; it often fails because the wire was too loose, the flash mode was not switched on, or the coding change was incorrect. Before it can be uploaded to the board, the user must override the “user config.h” header file. The user must overwrite the “user config.h” header file before it can be uploaded to the board. The Wi-Fi section of the settings should be updated, followed by the MQTT section, which includes the subject name, sub pub prefix, topic, port number, and MQTT status on and off, which will be send or receive to the Broker. Below are few screenshots of my project’s Wi-Fi and MQTT setup:

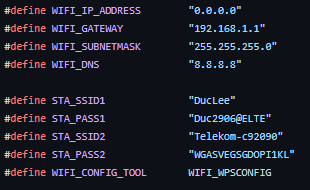


Figure 4.7.: Wi-Fi modified settings

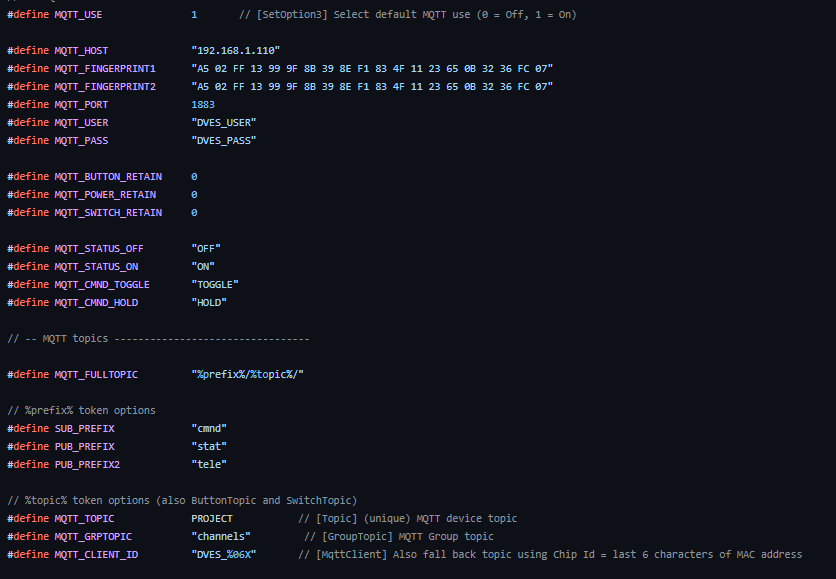


Figure 4.8.: MQTT modified settings

Finally, the Sonoff's web server is fairly simple and can be reached from the local network if the latest firmware was successfully uploaded; it includes several buttons to monitor four relay channels as well as new settings that allow a variety of helpful configurations such as MQTT, KNX, Timer, Logging, and so on. In this project, we will use two channels to control the room light and TV.

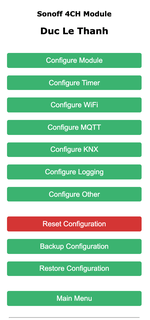
 

Figure 4.9.: Sonoff 4CH web interface

### 4.2.4. ESP8266

This section will go into the Arduino software structure and how to use the Arduino IDE to use the ESP8266 in the Arduino environment. Arduino's app is free and open source. The Java environment's source code is published under the GPL, while the C/C++ microcontroller libraries are released under the LGPL. Structure, Values (variables and constants), and Functions are the three main parts of an Arduino program. Setup() and Loop() are the two main functions in the software structure. When a sketch begins, the Setup() function is called. It can be used to set up variables, pin modes, and start using libraries, among other things. After each power up or reset of the Arduino board, the configuration operation can only run once. Following the Setup() function, which initializes and sets the initial values, the Loop() function does exactly as its name implies: it loops repeatedly, causing the program to answer and modify. It can also be used to power the Arduino board directly.

The ESP8266WiFi and PubSubClient libraries are used to set up the Wi-Fi and MQTT to communicate with the internet network and the broker before doing something else. After that, it is ready to handle the programming part for the sensors. During the processing, the serial console at 115200 baud will be used to display as logging of the system.

**HR202 sensor**

As stated in the previous article, this sensor is used to measure and detect the humidity of the surrounding environment. The first step is to bind GND to GND. The signal's pinout will be wired to the pin A0 on the ESP8266, with the power Vin ranging from 3.3 to 5.5 volts. The illustration below exemplifies this:

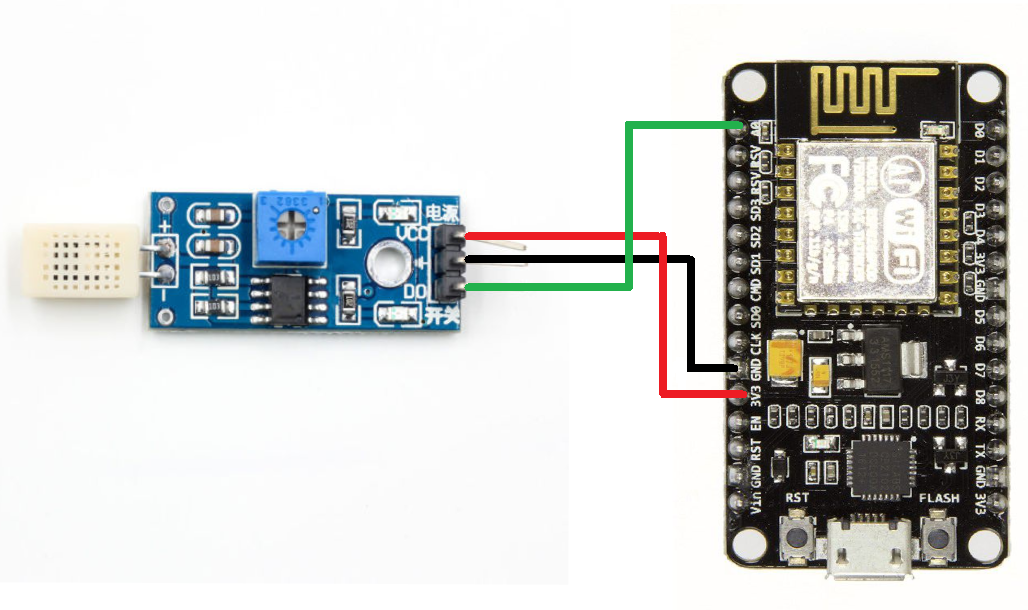
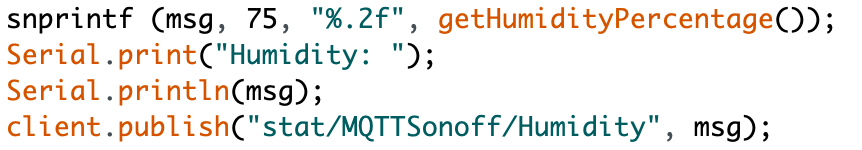


Figure 4.10.: Mapping from HR202 sensor to ESP8266

In the setup() function, this sensor is defined as INPUT attribute of pinMode() and its output value could be read by using analogRead() function. In this project, I will use HumidityHIH4040 library to read and process the value received from the sensor. Following the configuration, the humidity results will be sent to the openHAB server to change its state every 10 seconds under the same topic name:



**HC- SR505**

Based on infrared technology, the HC-SR505 Mini PIR Motion Sensor can automatically power itself with high sensitivity and reliability. It is commonly used in a variety of automated electronic equipment, especially battery-powered automatic products, due to its small size and low-power operating mode.

The steps to implement and process signal received by SR505 are as simple as the HR202. The pinout signal of the SR505 in this project will be mapped to D8 (GPIO15) on the ESP8266. The sensor consumes 3.3 voltages and also has one GND pin as the following:

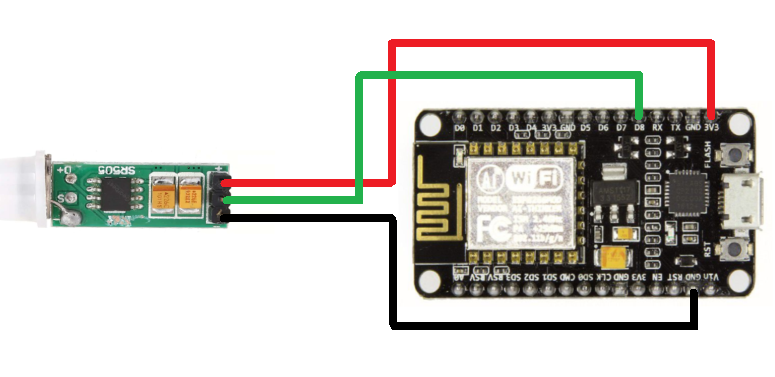
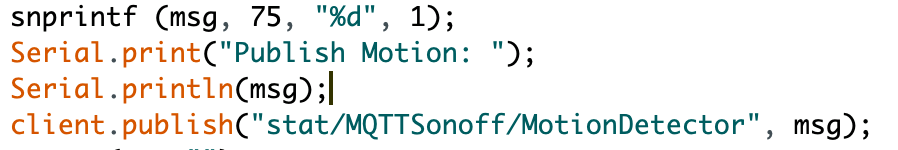


Figure 4.11.: Mapping from SR505 sensor to ESP8266

As an INPUT unit, the SR505 is similar to the HR202. Its data findings are either HIGH or LOW, with HIGH indicating that motion has been detected and LOW indicating that no motion has been detected at the moment. The result could be read by DigitalRead() function and send to MQTT broker with the topic “/MotionDetector” in this project.



**LED RGB Diffused**

The RGB Led Diffused serve as a realistic RGB light simulation in this project. It has four pins: GND, red, green, and blue. D5 (GPIO14) is connected to the red pin, D6 (GPIO12) to the green wires, and D7 (GPIO13) to the blue wires:

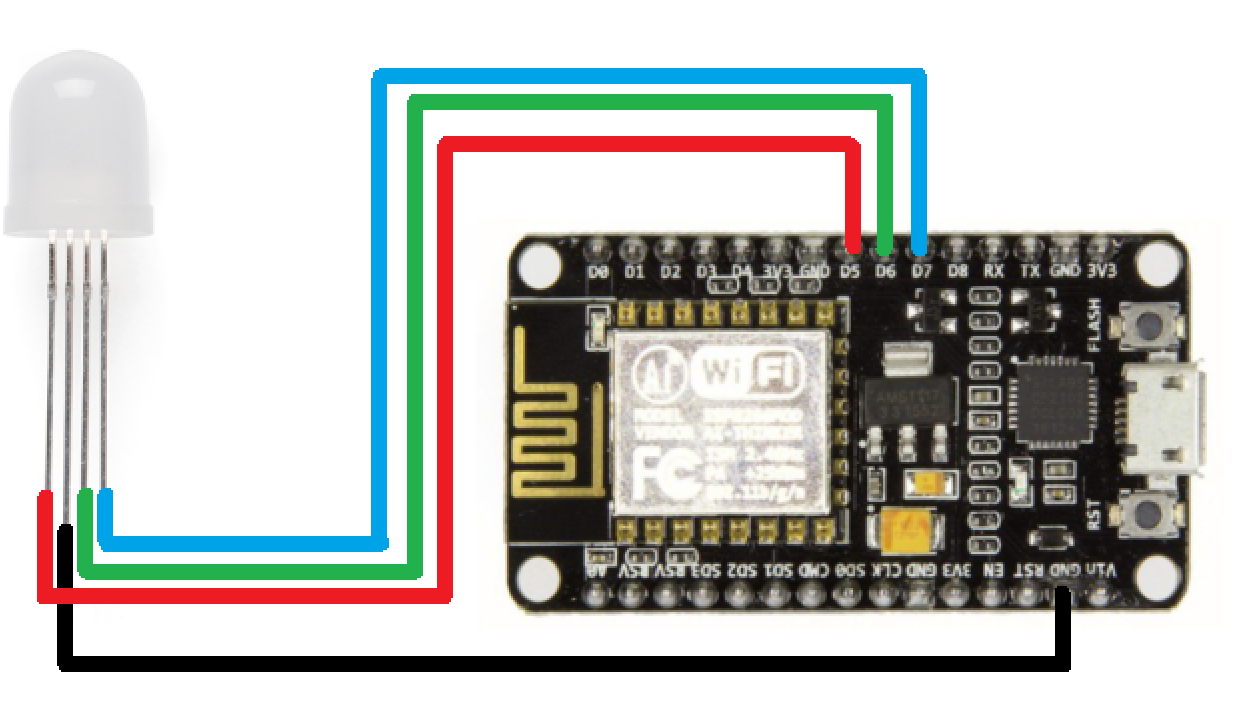
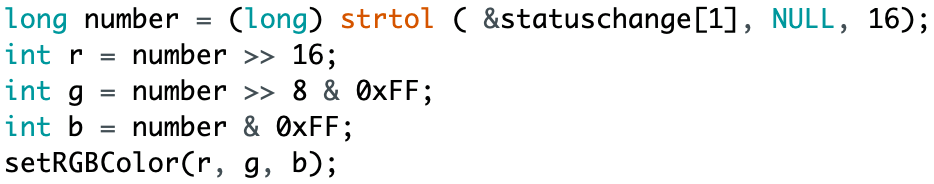


Figure 4.12.: Mapping from diffured RGB led to ESP8266

The obtained values for red, green, and blue are in the range of 0-255 and are sent using the analogWrite() function. The MQTT server will receive the hexadecimal data from the user interface and forward it to the ESP8266 in this project. When a new message arrives on the MQTT topic ‘cmnd/MQTTSonoff/RGBColor,' the ESP8266 is programmed to convert the hexadecimal value into an integer value and adjust the RGB color accordingly.



**ESP32-CAM**

The ESP32-CAM is a small camera module that runs on the ESP32-S chip with very low cost and energy efficient. It is the best option to be as a part of simulating smart house environment. Aside from the OV2640 camera and many GPIOs for connecting peripherals, it also has a microSD card slot for storing images captured by the camera or files to serve to clients. The ESP32- CAM does not come with a USB connector, so FTDI programmer is needed to upload its own code via Arduino IDE. The following picture illustrates how to bind FTDI to the ESP32-CAM:

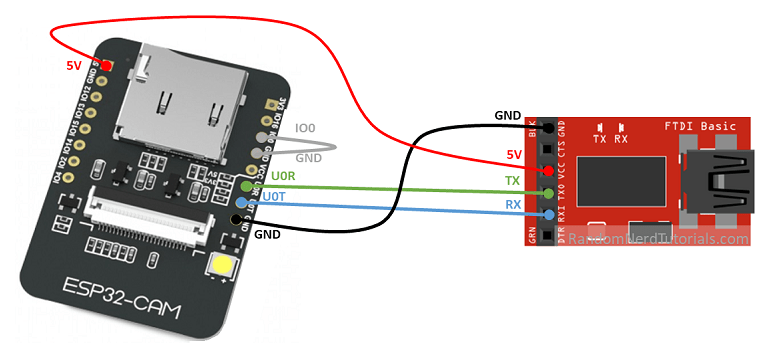


Figure 4.13.: Mapping from ESP32-CAM to FDTI connector

In this project, the ESP32-CAM acts as a standalone camera webserver with a fixed static IP address. The camera also has a slot for an external SD card, which can be used to store the recordings.

With its own ESP32 library and sketch, setting up this camera module is simple; all users have to do is change the SSID and password of the Wi-Fi it will link to. The camera is ready to use at the IP given in the Serial Monitor of the Arduino IDE after uploading the sketch:

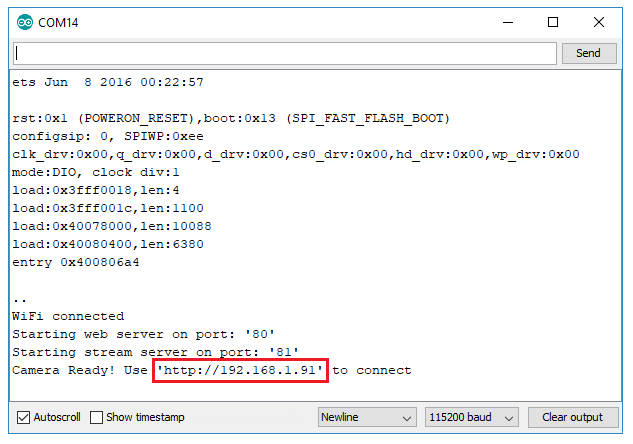


Figure 4.14.: Serial Monitor result

**Google Home Mini**

This section illustrates steps to set up the communication of Google Home Mini to openHAB2.

Recently, Google has approved the openHAB 2 Google Assistant Action, which helps anyone to connect your openHAB to Google Assistant through the myopenHAB.org cloud service and monitor it with voice commands. Users may use this Action to monitor objects that respond to on/off and, for example, brighten the lighting. The action has been checked for the English language, but it can also be used with other languages (depending on Google Assistant language support). In order to connect the devices and sensors to the Google Home, it is needed to modify the ‘items’ file in openHAB2 folder (“/etc/openhab2/items/”). Based on the Googles API document, items that wish to expose to Google Assistant will need to be tagged. Currently the following Tags are supported for openHAB2 but only Lighting and Switchable are supported for Google Home:

* ["Lighting"]
* ["Switchable"]
* ["CurrentTemperature"]
* ["Thermostat"]
* ["Blinds"]

Unfortunately, in this project, light and TV are only two devices that could be used to simulate and communicate with voice control. Google Home, on the other hand, is a very intelligent device that can do or search virtually anything in the Google database. To wake it up, simply say the command "OK! Google”. These are the code that is modified in items of openHAB2 to handle the status Light and TV:

|  |
| --- |
| Switch Button\_Light "Light" <light> (SW1) ["Lighting"] { mqtt=">[broker:cmnd/MQTTSonoff/POWER2:command:\*:default],<[broker:stat/MQTTSonoff/POWER2:state:default]" }  Switch Button\_TV "TV" <light> (SW1) ["Switchable"] { mqtt=">[broker:cmnd/MQTTSonoff/POWER1:command:\*:default],<[broker:stat/MQTTSonoff/POWER1:state:default]" } |

Afterwards, it's possible to connect openHAB devices to the Google Home app by creating a new system and entering an openHAB Cloud login credential. There are screenshots of the Google Home interface since I have attached the devices to the list:

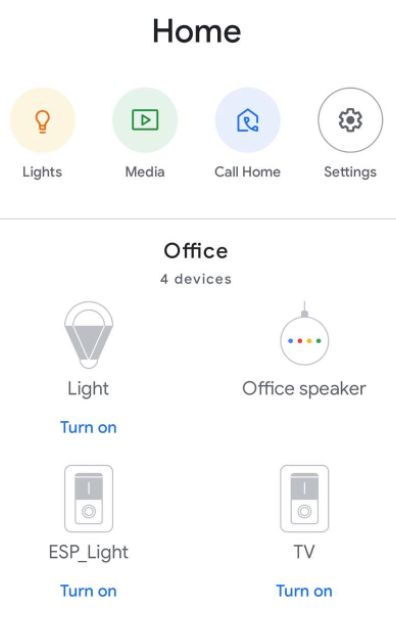


Figure 4.15.: Google Home application interface

## 4.3. Web Application Program

This section will talk about how to setup and configure for web application to manage and control the devices which hosted on Raspberry Pi 3 using Apache2.

The web application was created mainly by HTML, CSS, Javascript and PHP programming languages. It is accessible by modified the /var/html/www/ folder. There are 5 main parts:

* Login page
* Home panel page
* Camera View page
* Control Panel page
* Manage Access page

**Login Page**

To enter their "smarthouse", users must have an accessible username and password. This action ensures that the owners' privacy is protected. The data of the users is stored in a json format, which will be checked at each attempt. Only the user "admin" is allowed to create and manage accounts that are accessible to their house control.

**Home panel page**

This page contains a list of the smart house’s currently linked openHAB devices. The states of all attached devices are collected and shown in a table using the OpenHAB REST APi. This PHP function is used to send GET request to get the status of a particular item, including current Humidity value and RGB’s light colors:

|  |
| --- |
| function getStatus($item)  {  $url = "http://192.168.1.110:8080/rest/items/" . $item;  $options = array(  'http' => array(  'header' => "Content-type: text/plain\r\n",  'method' => 'GET',  ),  );  $context = stream\_context\_create($options);  $json = file\_get\_contents($url, false, $context);  $state = json\_decode($json);  $result = $state->{'state'};  return $result;  } |

**Camera View page**

This page triggers a JavaScript function on click to redirect to the ESP32-CAM webserver.

|  |
| --- |
| function RedirectToCamera() {  location.href = "http://192.168.1.118";  } |

**Control Panel page**

This page includes a list of openHAB devices that are currently connected to the smart home, as well as toggles to control them. A POST request would be submitted using the JavaScript programming language and the openHAB REST APi to adjust their status. This includes modifying the RGB light's HEX value.

|  |
| --- |
| function Switch(items, command){  var url = 'http://192.168.1.110:8080/rest/items/'.concat(items);  try{  fetch(url , { method: 'POST', mode:'no-cors', body: command });  location.href = "http://192.168.1.110/controlpanel.php";  } catch(err){  console.log("Error");  }  } |

**Manage Access page**

This page is only for user with username “admin” to manage all the sub accounts accessibilities.

# Chapter 5

# Testing

This chapter would go into how each of the components are tested.

In general, the system is deployed and usable in my own apartment. The web interface allows users to easily use and monitor all of the relevant elements, such as appearing the humidity percentage and adjusting the RGB light color, among other things. They may also use their voice to switch off lights or the television.

## 5.1. Raspberry Pi 3 and related components

I have set up the static IP for the Raspberry Pi on address 192.168.1.110 which could be used to access via SSH connection. The openHABian is running perfectly after deployment with version 2.5.11 together with Mosquitto.

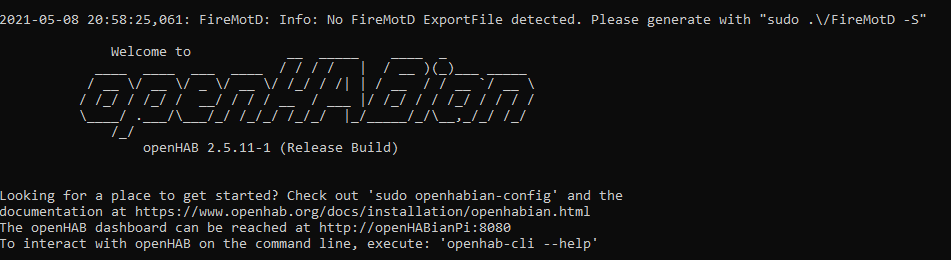


Figure 5.1.: OpenHABian interface

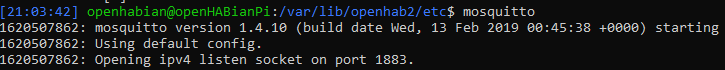


Figure 5.2.: Mosquitto broker version

Moreover, the PHP login page and the others webpages are hosted on the Apache2 which is accessible using the IP of the Raspberry Pi via web browser as discussed above:

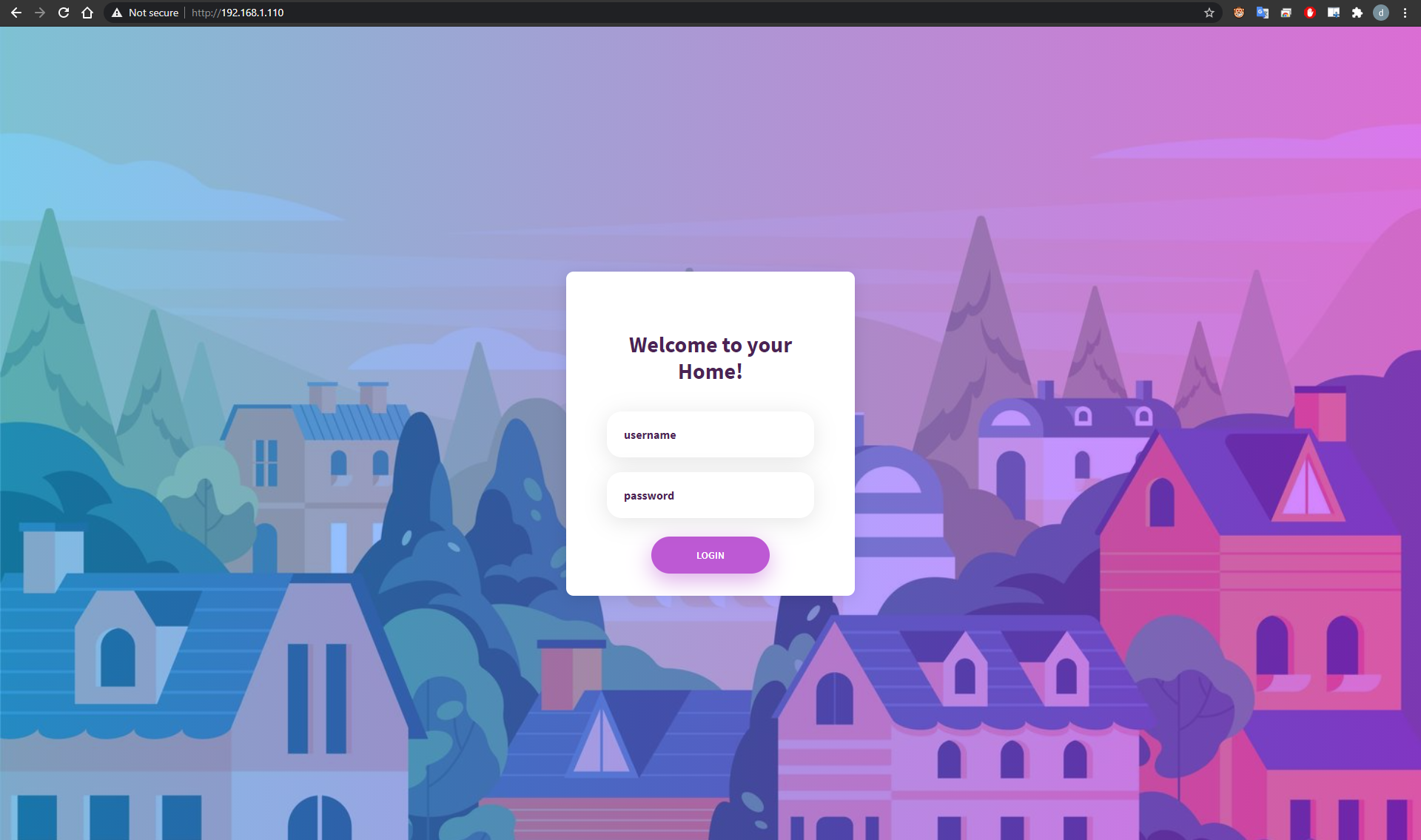


Figure 5.3.: Login page

## 5.2. ESP32- CAM

I was very struggling with uploading its program for quite a long time trying to set up this camera sensor, but it eventually succeeded. I was able to configure the program's web server to static IP 192.168.1.118 as it was streaming. The camera could also adjust its own settings, such as resolution, brightness, contrast, saturation, and so on, through its own web interface:

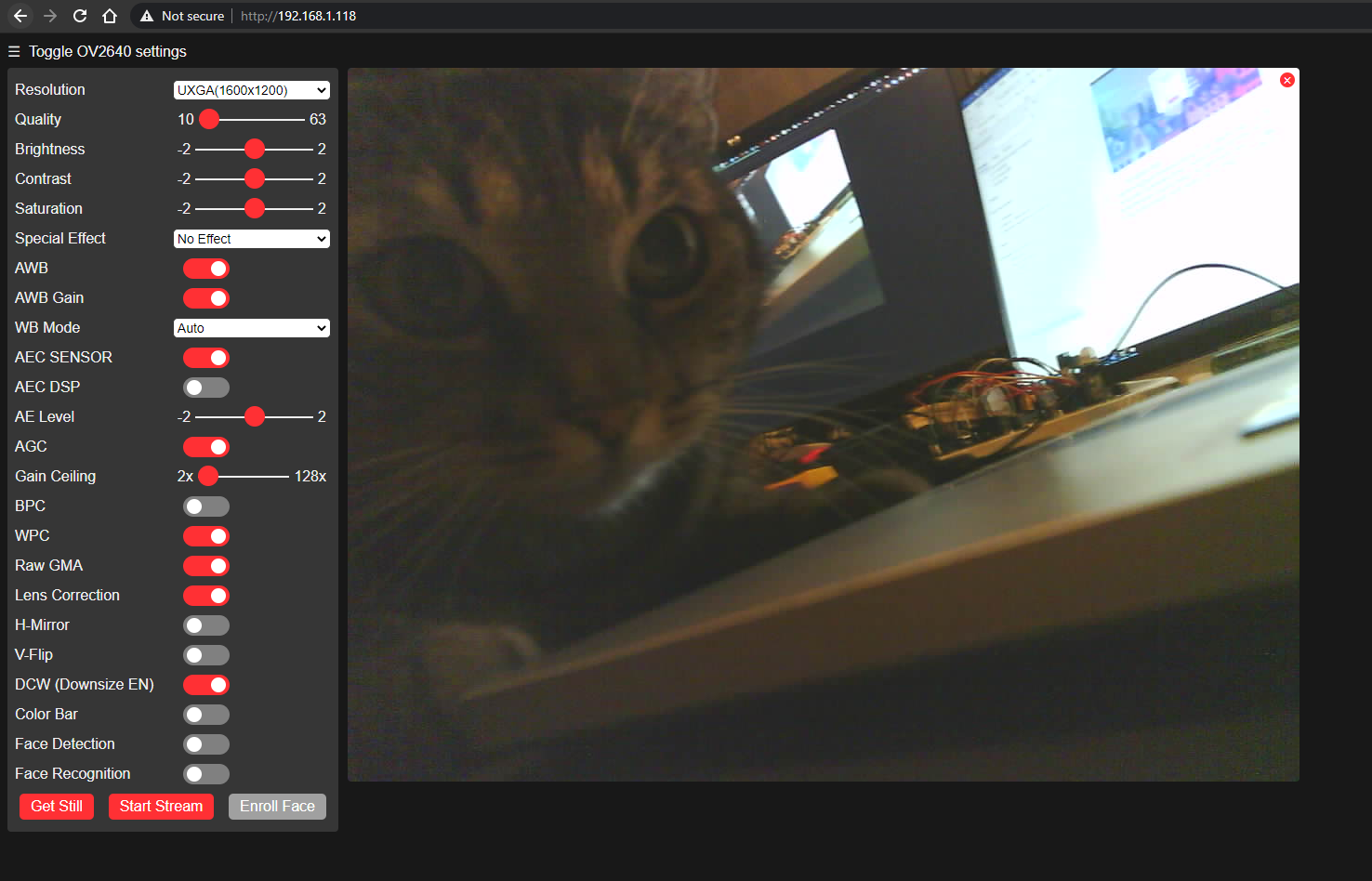


Figure 5.4.: ESP32-CAM website interface

## 5.3. ESP8266 and related sensors

Since the ESP8266 has several pinouts, I was able to wire all of the sensors to it and connect it as a compact controller device. The ESP8266 can connect to a Wi-Fi network as well as a MQTT connection. Whenever it is turned on, it is loaded automatically the results from the previous session. I also use a small LCD display showing its own IP and showing if any of the connections are connected. ESP8266 can finally run individually with power from power bank or with whatever power supply:

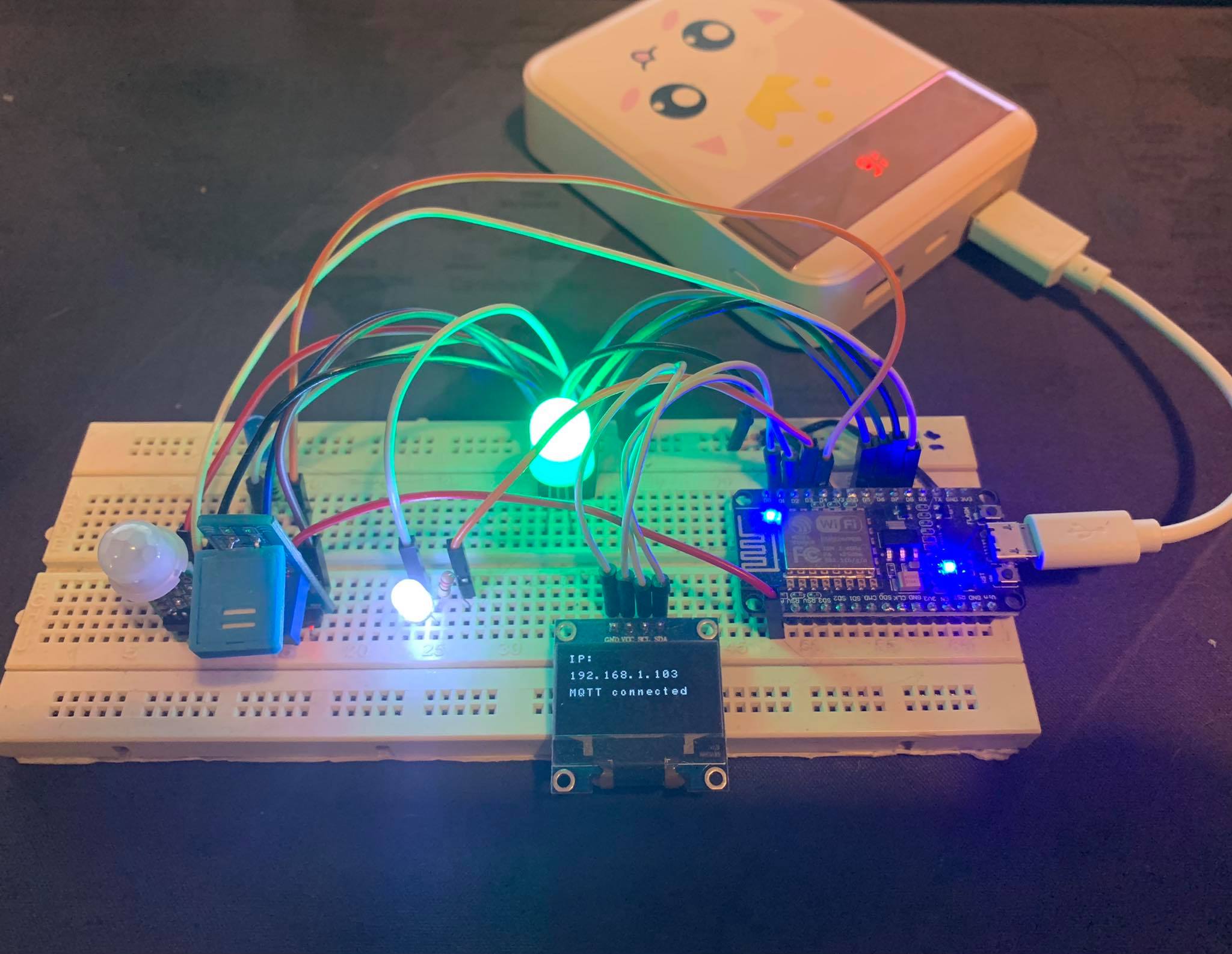


Figure 5.5.: ESP8266 system

Furthermore, to verify the publishing and receipt of MQTT messages from all sensors, such as the SR202, HR505, etc. I could use the Serial Monitor of the Arduino IDE together with the Log Viewer of the openHAB to see if their messages had been received or not:

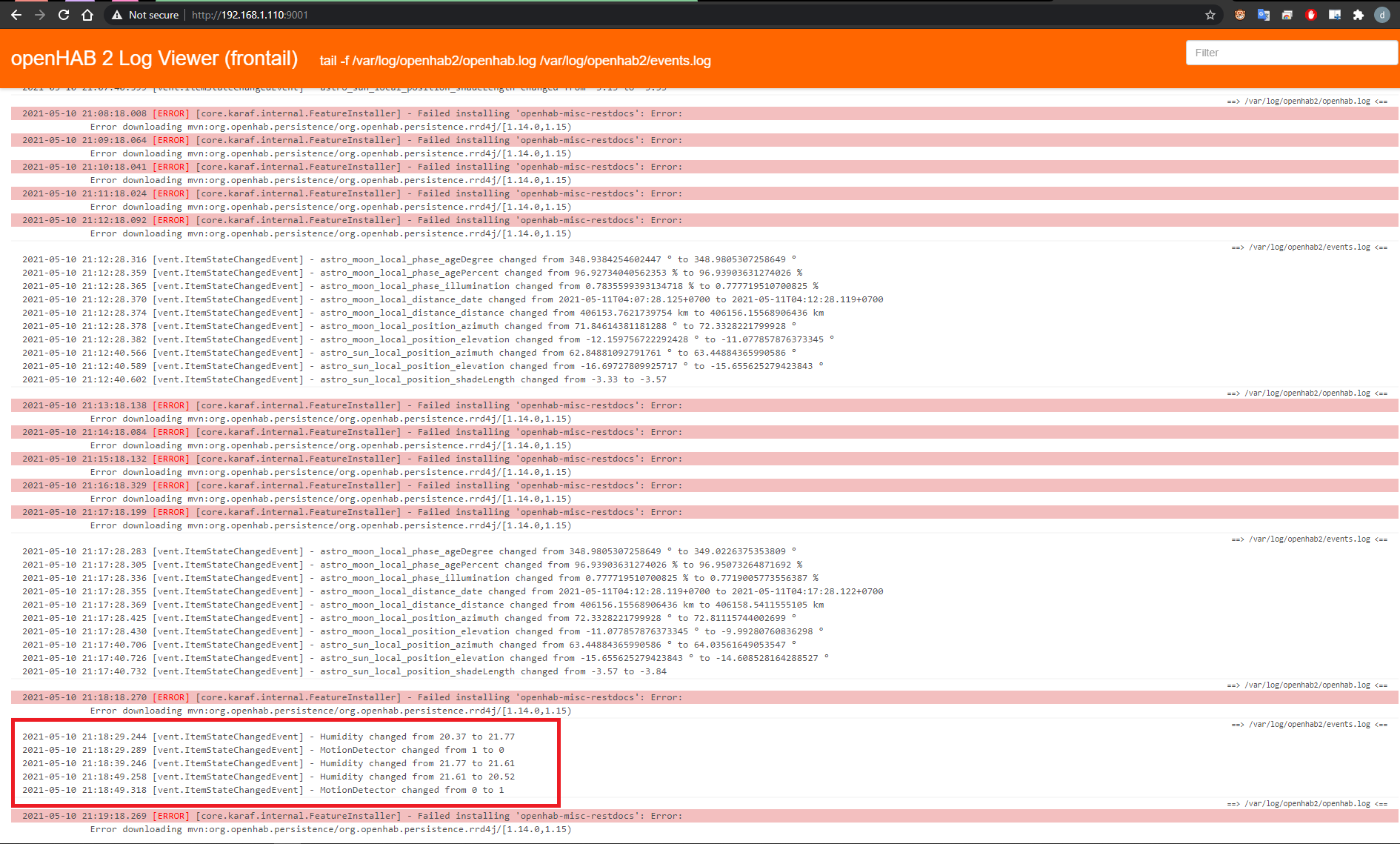


Figure 5.6.: Log Viewer with received data confirmation

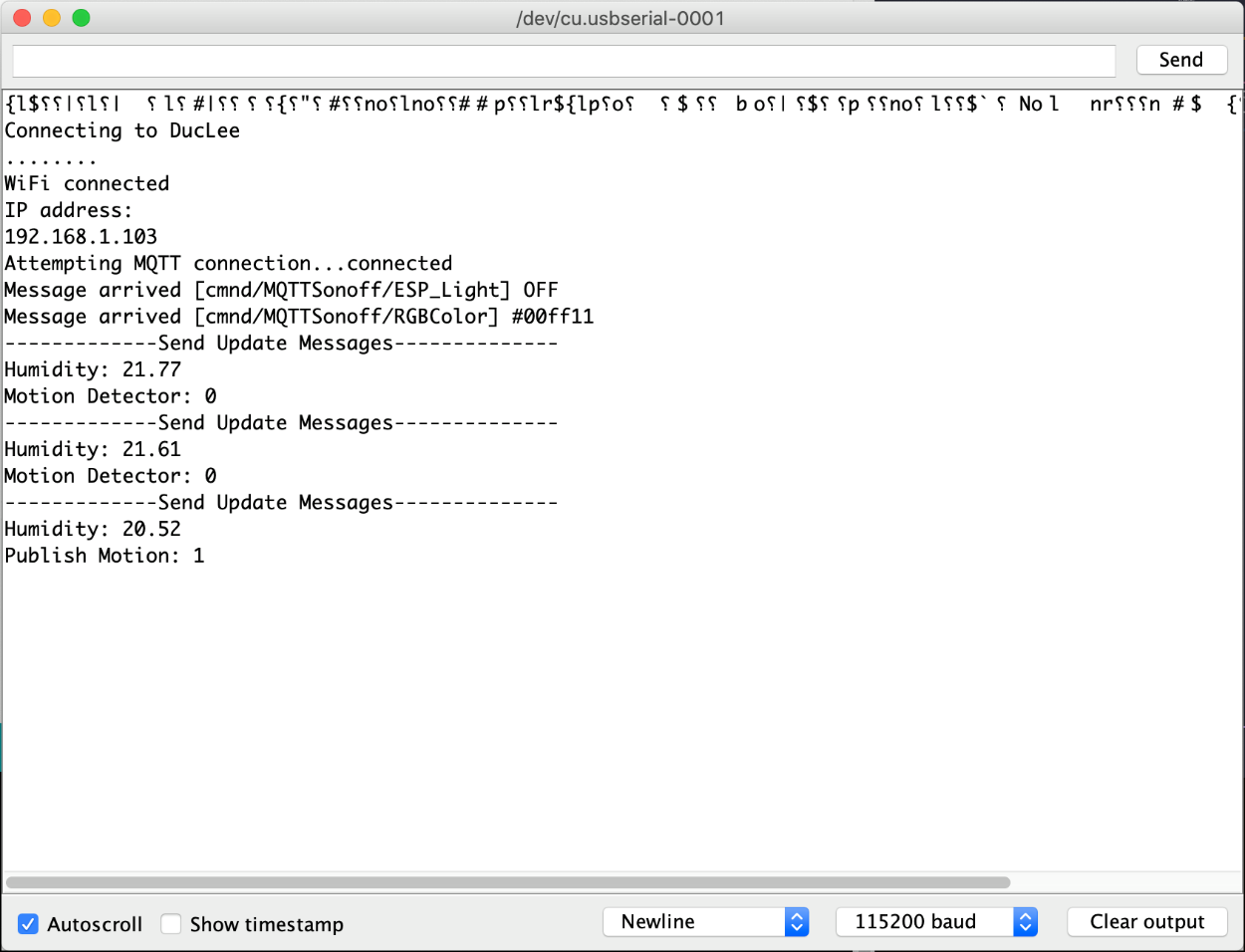


Figure 5.7: Serial Monitor with MQTT published messages

# Chapter 6

# Discussion

This project was completed in five months. I had no prior knowledge of this except for a fascinating concept that my father had given me at the time. After a bit of research as well as some low-budget experiments, I was able to bring the concept to life. As I mentioned at the beginning of this document, this project could be used to simulate and serve as a foundation for any smart house environment, including a working office or a broader range of applications.

## 6.1. Implication

Throughout the development process, I created a system whose physical devices are placed as a demonstration of what can be achieved by using Internet of Things.

From the web application, an operator is able to control the system environment and gather data from related devices. All of the sensors can interact and work with the openHAB MQTT broker. To remotely control the device from a distance, the user now still have to use the openHAB cloud service. This is also something to keep in mind for upcoming work.

## 6.2. Limitation

Based on my point of view, since the owner of the system does all of the work, the cost of the smart home system is reduced to only the cost of the units. Users just need to purchase the required components and hardware. This means that, in comparison to the solutions that many businesses offer, the solution itself is very inexpensive. Since the owner is fully responsible for keeping the solution up and running, certain issues can become difficult to resolve.

The most significant issue with this custom smart home solution is its complexity. Both changes and configurations must be done manually to keep the solution going. Manual work requires the users' time and specific skills.

## 6.3. Conclusion

Throughout the writing of this thesis, I encountered numerous difficulties of varying degrees of difficulty. The difficulties were broad, ranging from a lack of hardware to lack of time. Working on the thesis was really entertaining for me because of the aspects of the thesis and the technologies used, which made me feel inspired and encouraged to address the problems that arose as a result of my progress. During the planning and development phases, I gained valuable insight into how a project like the one outlined in this thesis should be carried out.

Furthermore, I learned a lot of new and useful things about web application creation, especially about sensor programming and how the Internet of Things works in general.

# List of references

Arends, T. (n.d.). *Github*. Retrieved from https://github.com/arendst/Tasmota

Components101. (2018). *HC-SR505 PIR Motion Sensor Module*. Retrieved from https://components101.com/sensors/hc-sr505-pir-sensor-pinout-datasheet

Components101. (2020). *NodeMCU ESP8266*. Retrieved from https://components101.com/development-boards/nodemcu-esp8266-pinout-features-and-datasheet

Dfrobot. (n.d.). *ESP32-CAM Development Board* . Retrieved from https://media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/DFR0602\_Web.pdf

Elecrow. (n.d.). *Resistive humidity sensor, Model: HR202*. Retrieved from https://www.elecrow.com/download/HR202%20Humidity%20Sensor.pdf

Istifanos, M., & Tekahun, I. (2020). *Performance evaluation of.* Karlskrona: Blekinge Institute of Technology.

James, J. I. (2018). Preliminary Study of a Google Home Mini. *Journal of Digital Forensics*.

Krasnopolski, A. (2020). *Servers*. Retrieved from github.com: https://github.com/mqtt/mqtt.org/wiki/servers

Lafon, Y. (2014). *HTTP - Hypertext Transfer Protocol.* Retrieved from W3: https://www.w3.org/Protocols/

Last Minute Engineers. (n.d.). *Insight Into ESP8266 NodeMCU Features & Using It With Arduino IDE*. Retrieved from https://lastminuteengineers.com/esp8266-nodemcu-arduino-tutorial/

Light, R. (n.d.). *mqtt — MQ Telemetry Transport*. Retrieved from mosquitto.org: http://mosquitto.org/man/mqtt-7.html

ModBerry. (2021). *Modberry*. Retrieved from https://modberry.techbase.eu/

Mudric, M. (2020). *The Biggest Advantages and Disadvantages of IoT*. Retrieved from Readwrite: https://readwrite.com/2020/06/24/the-biggest-advantages-and-disadvantages-of-iot/

Nodemcu. (2021). *NodeMCU Documentation*. Retrieved from https://nodemcu.readthedocs.io/en/release/

OASIS . (2019). *MQTT Version 5.0- OASIS Standard*. Retrieved from https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

OpenHAB. (n.d.). *OpenHAB: Our Vision and Philosophy*. Retrieved from https://www.openhab.org/about/who-we-are.html

OpenHABFoundation. (n.d.). *OpenHABFoundation: Become a Member*. Retrieved from https://www.openhabfoundation.org/join/

R, F. (1999). *Hypertext Transfer Protocol -- HTTP/1.1*. Retrieved from https://tools.ietf.org/html/rfc2616

Raspberry Pi. (2021). *Raspberry Pi OS*. Retrieved from https://www.raspberrypi.org/software/

Saxena, S., Jain, S., Arora, D., & Sharma, P. (2019). Implications of MQTT Connectivity Protocol for IoT based Device Automation using Home Assistant and OpenHAB. *2019 6th International Conference on Computing for Sustainable Global Development (INDIACom).*

Smart Home Blog. (n.d.). *Openhab Raspberry Pi*. Retrieved from https://www.smarthomeblog.net/openhab-raspberry-pi/

Sun Microsystems. (1988). RPC: Remote Procedure Call. Retrieved from https://tools.ietf.org/html/rfc1057

Upton, E. (2016). *Ten millionth Raspberry Pi, and a new kit*. Retrieved from Raspberry Pi Blog: https://www.raspberrypi.org/blog/ten-millionth-raspberry-pi-new-kit/

店长. (2017). *Sonoff 4CH Pro (R2) User Guide*. Retrieved from Ewelink: https://ewelink.coolkit.cc/?p=458

# List of figures

|  |  |
| --- | --- |
| Figure 2.1 | ESP8266 device |
| Figure 2.2. | ESP8266 Pinouts |
| Figure 2.3. | ESP32-CAM device |
| Figure 2.4. | Sonoff 4CH Switches |
| Figure 2.5. | HR202 sensor |
| Figure 2.6 | HC-SR505 senso |
| Figure 2.7. | Diffused RGB Led |
| Figure 3.1 | Deployment Diagram |
| Figure 3.2 | Home Panel page |
| Figure 3.3 | Control Panel page |
| Figure 3.4 | Manage access page |
| Figure 3.5 | Register new user page |
| Figure 4.1 | System plan illustration image |
| Figure 4.2 | Steps to install Mosquitto Broker |
| Figure 4.3 | OpenHAB webserver homepage |
| Figure 4.4 | OpenHAB LogViewer |
| Figure 4.5 | mqtt.cfg file |
| Figure 4.6 | Mapping from FTDI to Sonoff 4CH |
| Figure 4.7 | Wi-Fi modified settings |
| Figure 4.8 | MQTT modified settings |
| Figure 4.9 | Sonoff 4CH web interface |
| Figure 4.10 | Mapping from HR202 sensor to ESP8266 |
| Figure 4.11 | Mapping from SR505 sensor to ESP8266 |
| Figure 4.12 | Mapping from diffured RGB led to ESP8266 |
| Figure 4.13 | Mapping from ESP32-CAM to FDTI connector |
| Figure 4.14 | Serial Monitor result |
| Figure 4.15 | Google Home application interface |
| Figure 5.1 | OpenHABian version |
| Figure 5.2 | Mosquitto broker version |
| Figure 5.3 | Login page |
| Figure 5.4 | ESP32-CAM website interface |
| Figure 5.5 | ESP8266 system |
| Figure 5.6 | Log Viewer with received data confirmation |
| Figure 5.7 | Serial Monitor with MQTT published messages |