INSTITUTE *of* TECHNOLOGY CARLOW

Institiúid Teicneolaíochta Cheatharlach

Bachelor of Science (Honours)
Software Development

# Home Automation System
# Design Specification

Course Code: CW238
Student ID: C00212235
Student Name: Xiaohui Ling
Supervisor: Paul Barry

# Abstract

This purpose of this document is to provide a detail of the system design of the Home Automation System. The system architecture and functional design outlined as core part on the system and describe each function in detail. This document first presents the architecture of Home Automation System and then given out a structure of whole system functions that presents the key modules.

## Contents

# Table of Figures

# 1. Introduction

Home automation as an important application of the IoT field has been constantly moving forward. In life, people can also easily buy smart devices and control them remotely through the App installed on the mobile phone. Meanwhile, there are many open-source home automation systems constantly emerging. However, these existing products or open source systems cannot fully meet the personalized customization needs of different users.

Therefore, I hope to develop a home automation system that is closer to people's lives. So, this project carried out preliminary research on home automation, and step by step to realize a system prototype involving management and control included appliances, security monitoring, sensors and entertainment.

# 2. Overview

## 2.1. Purpose

The purpose of the home automation system design is to break down the whole system into different modules and functions and describe them in detail conduct how the system should be implemented by the development stage.

## 2.2. Goal

The home automation system is designed used to remote control involved fields that included appliances, sensors, security monitoring, entertainment, through networks. This document describes the system architecture and system module design details, explains the system components and working principles.

# 3. System Architecture

## 3.1. IoT Three Layers Architecture

### 3.1.1. Overview

The three layers architecture is a classic design in the IoT field that simply describes how it works in its own responsibility of between different components.

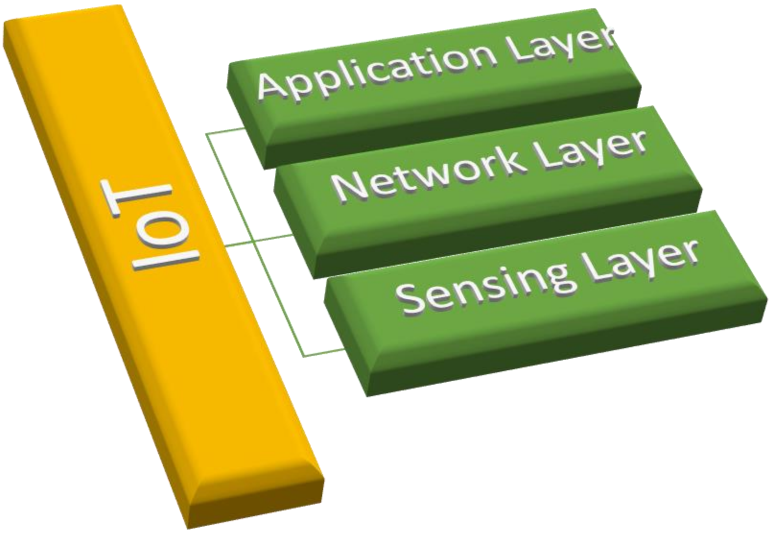


*Figure 1 IoT architecture*

### 3.1.2. Sensing Layer

#### *3.1.2.1. MEMS*

MEMS is an acronym for Micro-electromechanical System that is one of the critical techniques that was applied in various sensors or smart devices in the IoT field. The advantages are a microchip and circuit they have. In this project, the temperature and humidity sensor, DC motor driver controller and cameras are used.

- Temperature & Humidity Sensor

This sensor used to monitor the environment to obtain temperature and humidity in real-time. It is small size and works with low power.

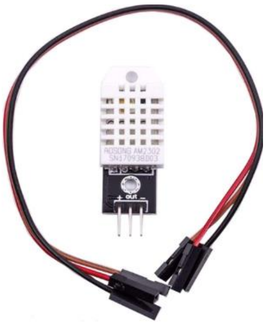


*Figure 2 temperature & humidity sensor*

- DC Motor Driver Speed Controller

The DC motor driver speed controller can convert each other between the analog signal and digit signal so that they can read and write data from the equipment.

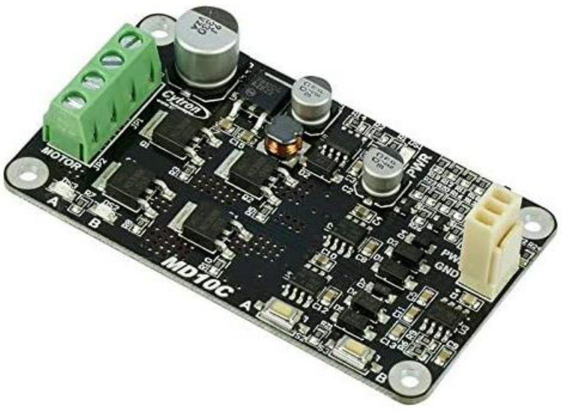


*Figure 3 DC Motor Driver 1*

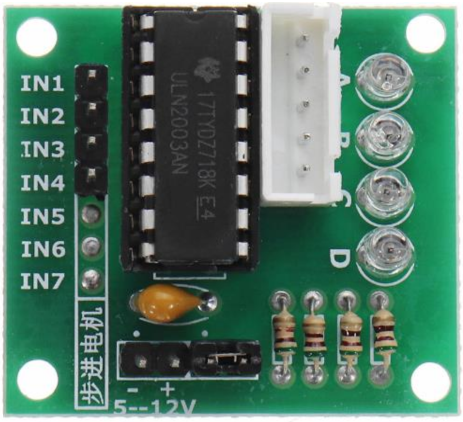


*Figure 4 DC Motor Driver 2*

- RaspberryPi Camera

This is a raspberry Pi camera V2 in the project. It has 8 megapixel native resolution sensor-capable of 3280 x 2464 pixel static images and supports high resolution video.

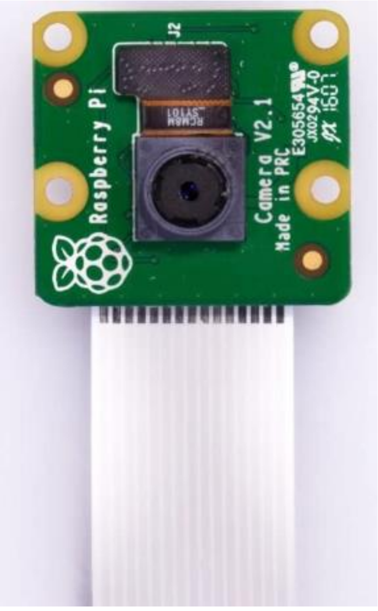


*Figure 5 Raspberry Pi Camera*

- USB Camera

This is a Microsoft Lifecam NX-3000 USB camera used in this project. It could be replaced with any type of USB camera.

*Figure 6 USB Camera*

### *3.1.2.2. Electronic Components*

Those components have a circuit control system that is controlled through the analog signal. The way they work based on electromagnetic induction and mechanical principles.

- Relay Module

It can control the home appliances turn on or off through the 5 to 12 voltage that powered by battery or raspberry Pi. Please read the Wikipedia if you would like to understand how the relay module works.

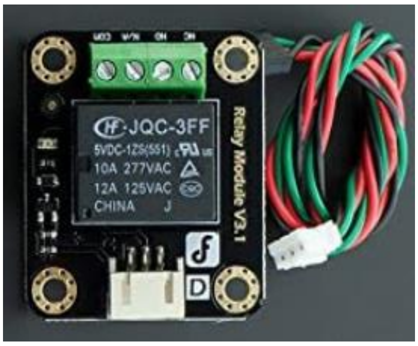


*Figure 7 Relay Module*

- DC Motor

The DC motors work with 5 to 12 voltage that the maximum of RMP is 15,000. They can be controlled by the DC motor driver controller which is MEMS.

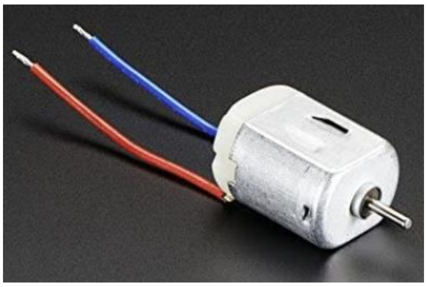


*Figure 8 Fan DC Motor*

*Figure 9 Shutter DC Motor 2*

- Buzzer

As a doorbell that is the low-level signal trigger in door access system.

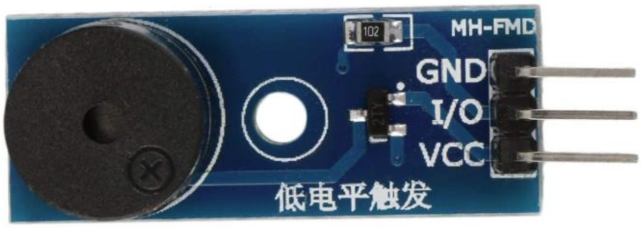


*Figure 10 Buzzer*

- Push Button

As a doorbell push button in door access system. It is a switch. It is switched on if it is pressed down then.

*Figure 11 Push Button*

- Desk Lamp

As a home appliance, it is turned on or off when a high- level signal is received through the relay module.

*Figure 12 Table Light*

*Figure 13 Bulb*

- E-Lock

As a lock in door access system, it is locked or unlocked when a high-level signal is received through the relay module.

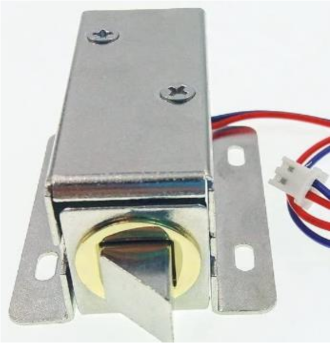


*Figure 14 E-Lock*

### 3.1.3. Network Layer

#### 3.1.3.1. Wi-Fi

Wi-Fi is a local network communication protocol that takes advantage of the 802.11 standards to defines service, clients, access points. It supports the maximum number of the client nodes is 32.

The Wi-Fi is used for communication between IoT devices in this project.

#### 3.1.3.2. Router

As a gateway that is a networking device, it forwards data packets between networks. It also provides the Wi-Fi feature which is local networks. In this project, It is necessary to support the OpenWRT that is an open-source project for an embedded operating system based on Linux used to route network traffic and We can login into it via SSH to install the software which I want to.

Here I am using a GL.iNET GL-MT300N-V2 Mini Travel Router, any router support OpenWRT can replace it in the project.

*Figure 15 Home Router*

### 3.1.4. Application Layer

#### *3.1.4.1. Raspberry Pi 4B*

It is an open-source single board hardware platform base on the Linux operating system that easily extends to develop a device's control system in the IoT field through the GPIO feature is provided.

In this project, the Raspberry Pi is a device control centre to manage the IoT devices through Python GPIO feature.

*Figure 16 Raspberry Pi 4*

#### *3.1.4.2. Mobile Phone*

The mobile phone as a terminal control device that remotely manages the IoT devices through a mobile app developed in this project. It could be an Android or Apple iOS operating system within the mobile phone, however, that just is tested with the Android operating system in this project so far.

Currently, I am using the brand of mobile phone that is Samsung SM-A320FL base on the Android 7.0 version. It can be replaced with the same Android version of any mobile phone.

#### *3.1.4.3. Mobile App*

It is an app was developed by Google Flutter mobile app development framework that used Dart programming language. Once the app development is complete, it can be easily deployed on both Android and iOS mobile operating systems. The app mainly provides features through the network to remotely control the IoT devices and obtain environment data both from the sensors.

#### 3.1.4.4. MQTT

It is a lightweight message queue mid-ware applied in the system. It was developed by IBM that widely used in the IoT field to solve the problems of real-time communication between devices. It consists of MQTT Client and Broker. The MQTT client could publish the message to or subscribe to the message from the broker.

In this project, the flutter app installed in the mobile phone and Raspberry Pi both are MQTT client and the router is a MQTT broker.

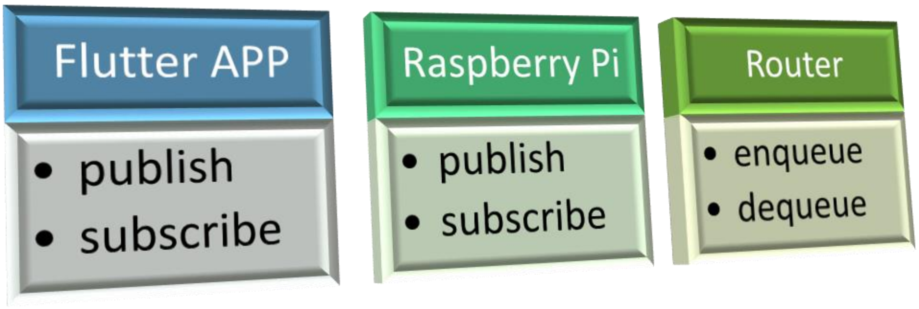


*Figure 17 MQTT Clients & Broker*

#### 3.1.4.5. AWS ActiveMQ

AWS ActiveMQ is a cloud service provided by Amazon corporate company. It bases on the open source 's Apache ActiveMQ used to communicate and exchange information between different systems. Its function is like MQTT and it also compatible with MQTT as a broker, so it as a service is used to communicate across the internet between devices in this project.

## 3.2. Topology Architecture

In the whole project, there are four components are mainly involved that listed below.

1. The smart phone as a control terminal has already installed an app that was developed base on Flutter mobile framework.
2. The AWS cloud provides the ActiveMQ service as a broker to solve the problems of across internet real-time communication between mobile phone and Raspberry Pi.
3. The wireless router in the home provides a local network service using Wi-Fi protocol between mobile phone and Raspberry Pi and it also as a gateway to communicate to the internet.
4. The Raspberry Pi as a control centre that manages and controls the devices which are home appliances, monitoring equipment, sensors and etc.

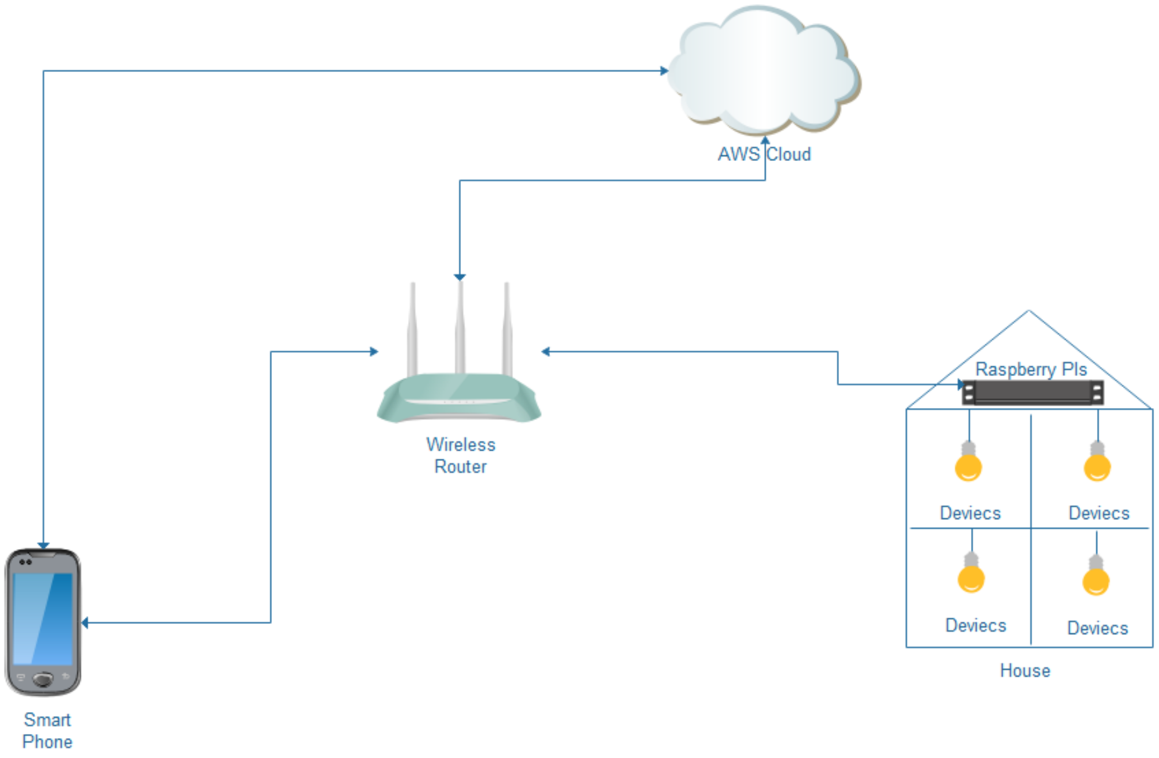


*Figure 18 Topology Architecture*

# 4. System Design

## 4.1. Modules and Functions Layout

The modules and functions of each module are listed below. It will be described in detail later in the document.

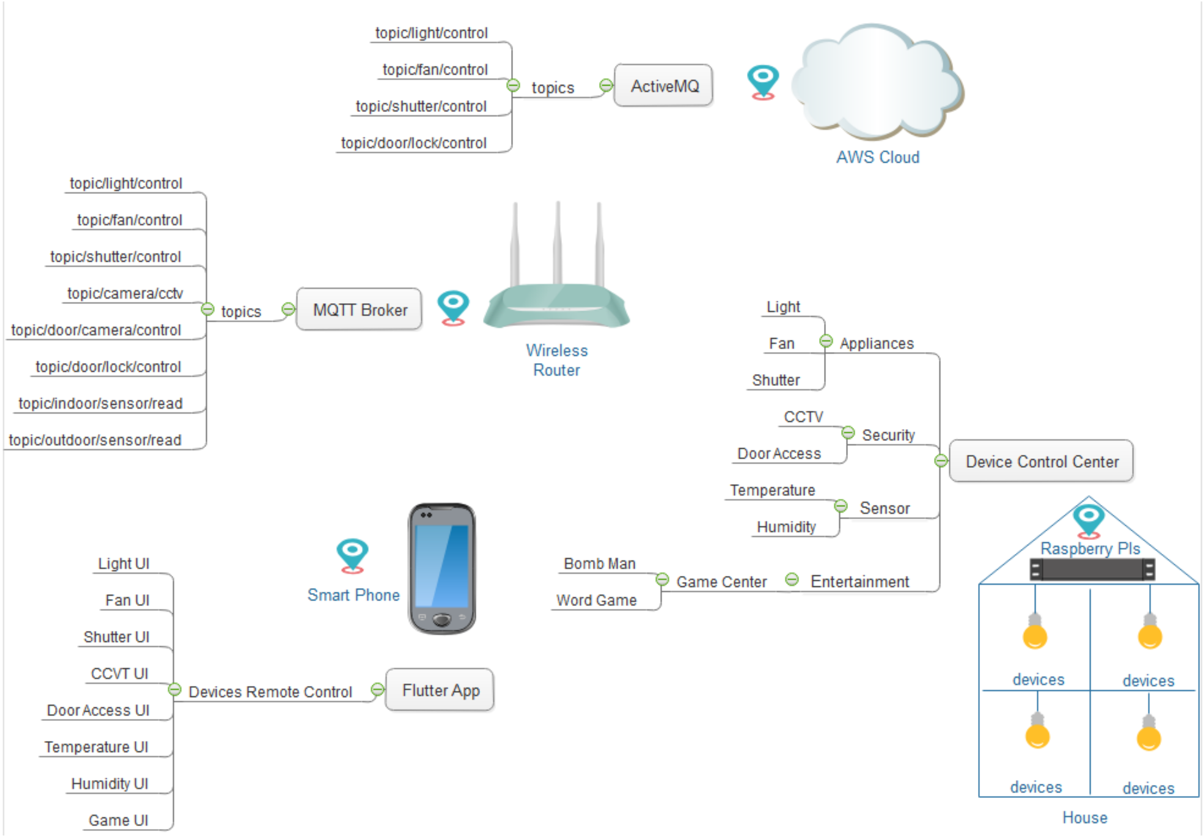


*Figure 19 Modules & Functions Layout*

## 4.2. UI Components Structure

The mobile app UI components and structure listed below. The widget is a concept in Flutter mobile framework that is abstracted as an object which is corresponding to a device control panel in this project.

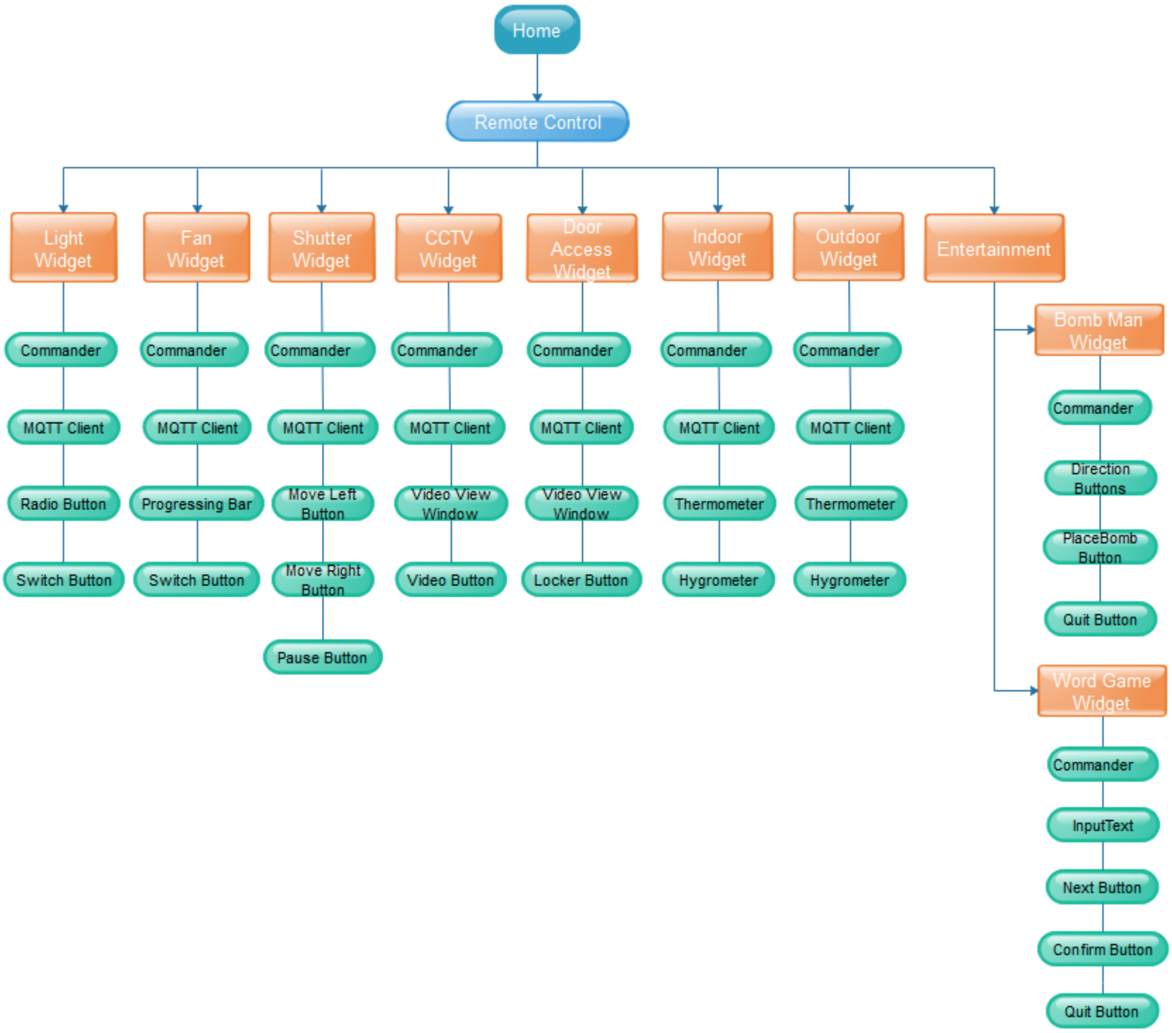


*Figure 20 UI Components Structure*

## 4.3. Device Control Structure

The Raspberry Pi as a device control centre has a controller to manage the terminal devices to communicate to a mobile app through the network. Those were implemented by python scripts.

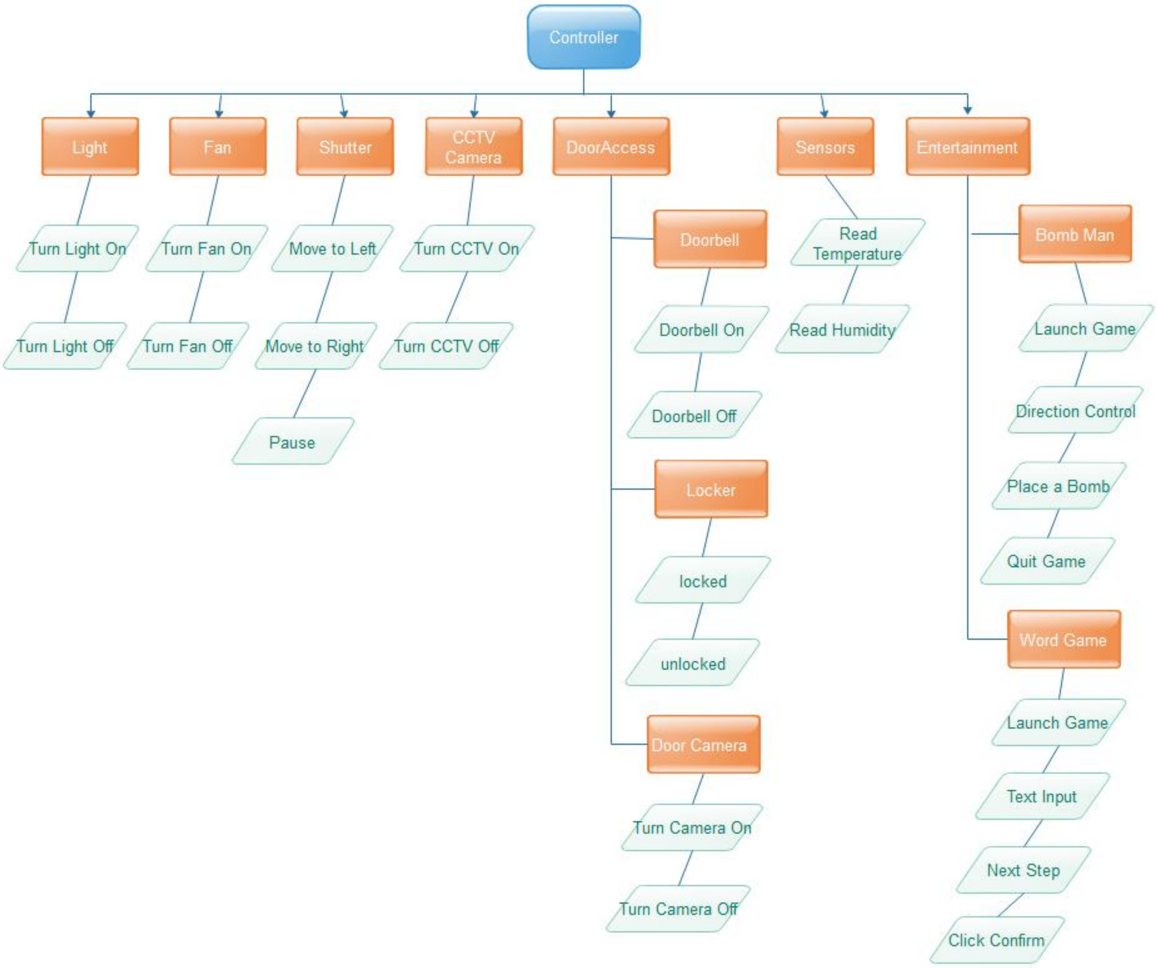


*Figure 21 Device Control Scripts Structure*

## 4.4. Networks Communication

### 4.4.1. Wi-Fi

In this project, it uses the WiFi protocol to provide the local network service between the mobile phone and Raspberry Pi to exchange information. The Raspberry Pi and mobile phone both have already had the WiFi feature that means it is easier to implement the project. The disadvantages that not secure enough and the number of nodes supported is small. However, in this stage of project that is not too matter.

### 4.4.2. MQTT

MQTT is a lightweight message transport protocol that provides the publish and subscribe method to exchange information through wireless between the mobile phone and the devices. It was widely applied to the IoT field that can communicate at low latency and high performance. In the MQTT protocol, it used the classical C/S architecture, the MQTT Client that either a publisher or subscriber, even both publisher and subscriber at the same time. The MQTT broker as a server between clients to transfer the information.

### 4.4.3. Internet

Due to the system requires that the home appliances should be able to control across the Internet, So this project will take advantage of AWS cloud to provide a capability which is able to control the devices between Raspberry Pi and mobile phone across the Internet. The AWS service called ActiveMQ that is a message queue mid-ware offer the publish and subscribe method to exchange information between networks. It is also compatible with the MQTT protocol.

# 5. Modules Design

In this chapter, it describes the functions of each module in detail to explain how the function works through the design planning.

## 5.1. Appliances Control

There are 3 types of home appliances are applied in the project, light, fan and shutter that will be described in the next sections.

### 5.1.1. Light

This section described how to control the light by the mobile app through both local network and internet.

#### 5.1.1.1. Use Case

The use case diagram below is clearly shown the process of the light control between the mobile phone, Raspberry Pi.

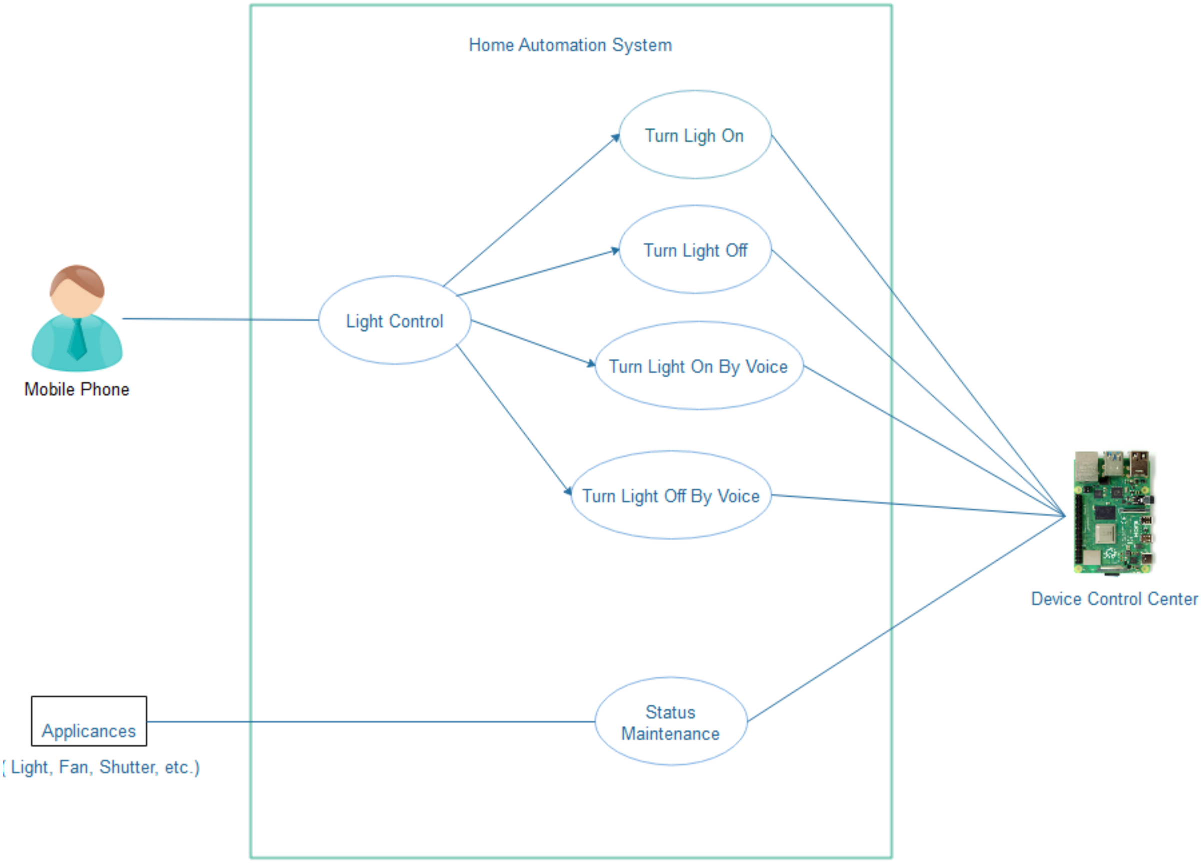


*Figure 22 Light Use Case*

### 5.1.1.2. Class Diagram

The light module classes are described using UML class diagram below.

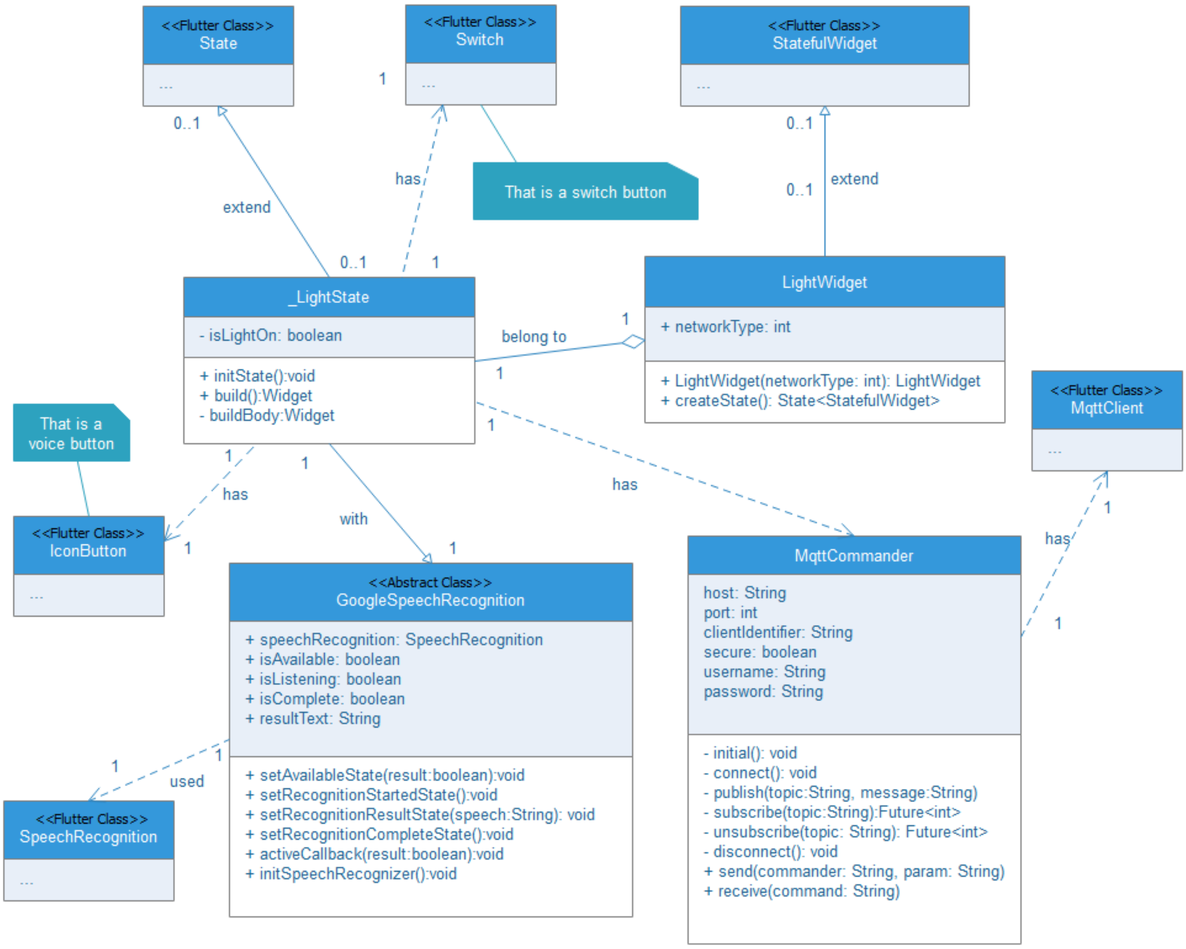


*Figure 23 Light Class Diagram*

### 5.1.1.3. Wiring Diagram

The diagram described how the components connect to each other. There is useful information provided below.

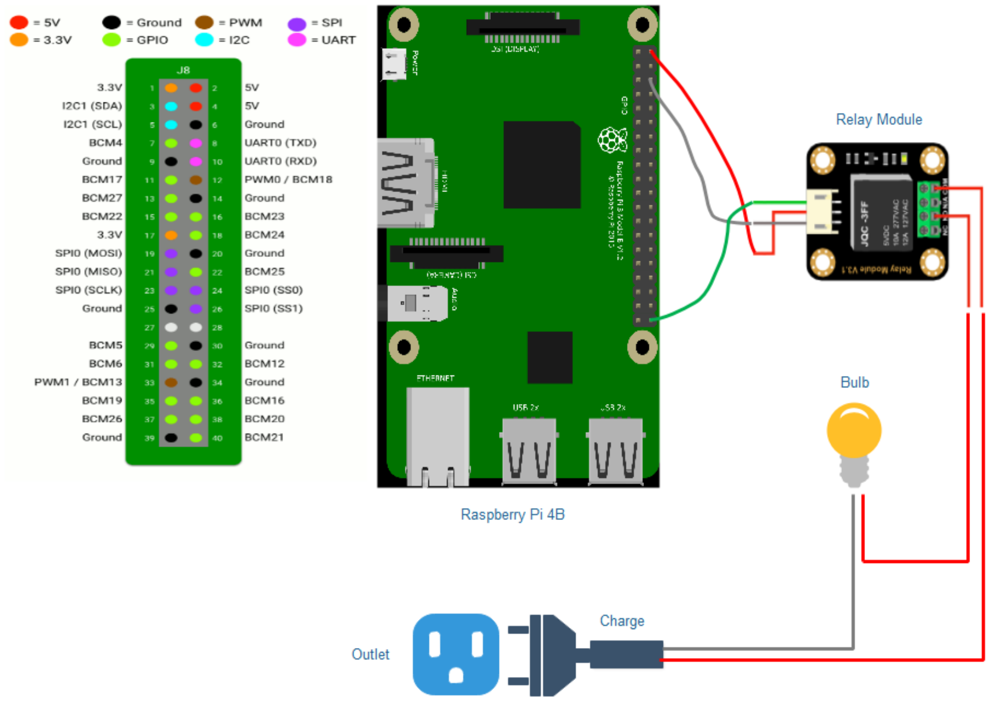


*Figure 24 Light Hardware Design*

1. Raspberry Pi and Relay Module

| **Raspberry Pi** | **Relay Module** | **Wire Colour** |
|---|---|---|
| 5v Pin | Positive Pin | Red |
| GND Pin | Negative Pin | Grey |
| BCM 21 | I/O Pin | Green |

2. Relay Module and Light

| **Relay Module** | **Light** | **Charge of Light** | **Wire Colour** |
|---|---|---|---|
| Normal Open Pin | Positive of Charge | *Not Used* | Red |
| COM Pin | *Not Used* | Positive of Charge | Red |

### 5.1.1.4. Device Control Script

There are Python scripts implements to control the Light device through the Raspberry Pi. The Python script is described using class diagram below.

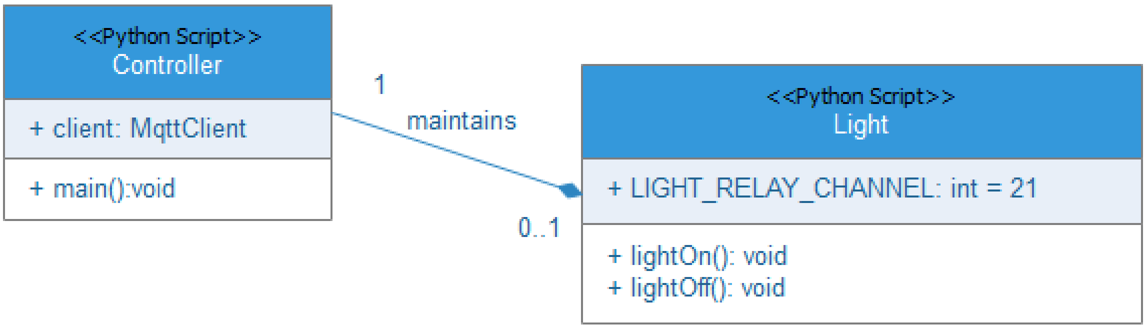


*Figure 25 Light Python Scripts Class Diagram*

### 5.1.1.5. Functional Description

In this section, it will describe that functions design of the light in detail.

#### 5.1.1.5.1. Turn Light On/Off via Wi-Fi

- *Description*

The householder could click the button on the mobile phone screen to remotely control the light to turn on or off through local network.

- *Key Parameters*

| MQTT Client | Role | Action | Network | Method | Topic Name | Message | Return Value |
|---|---|---|---|---|---|---|---|
| Flutter App | sender | switch button on | Wi-Fi | publish | topic/light/control | on | *[not applicate]* |
| Raspberry Pi | receiver | *[not applicate]* | | subscribe | topic/light/control | *[not applicate]* | on |
| Flutter App | sender | switch button off | | publish | topic/light/control | off | *not applicate* |
| Raspberry Pi | receiver | *[not applicate]* | | subscribe | topic/light/control | *[not applicate]* | off |

- *Sequence Diagram*

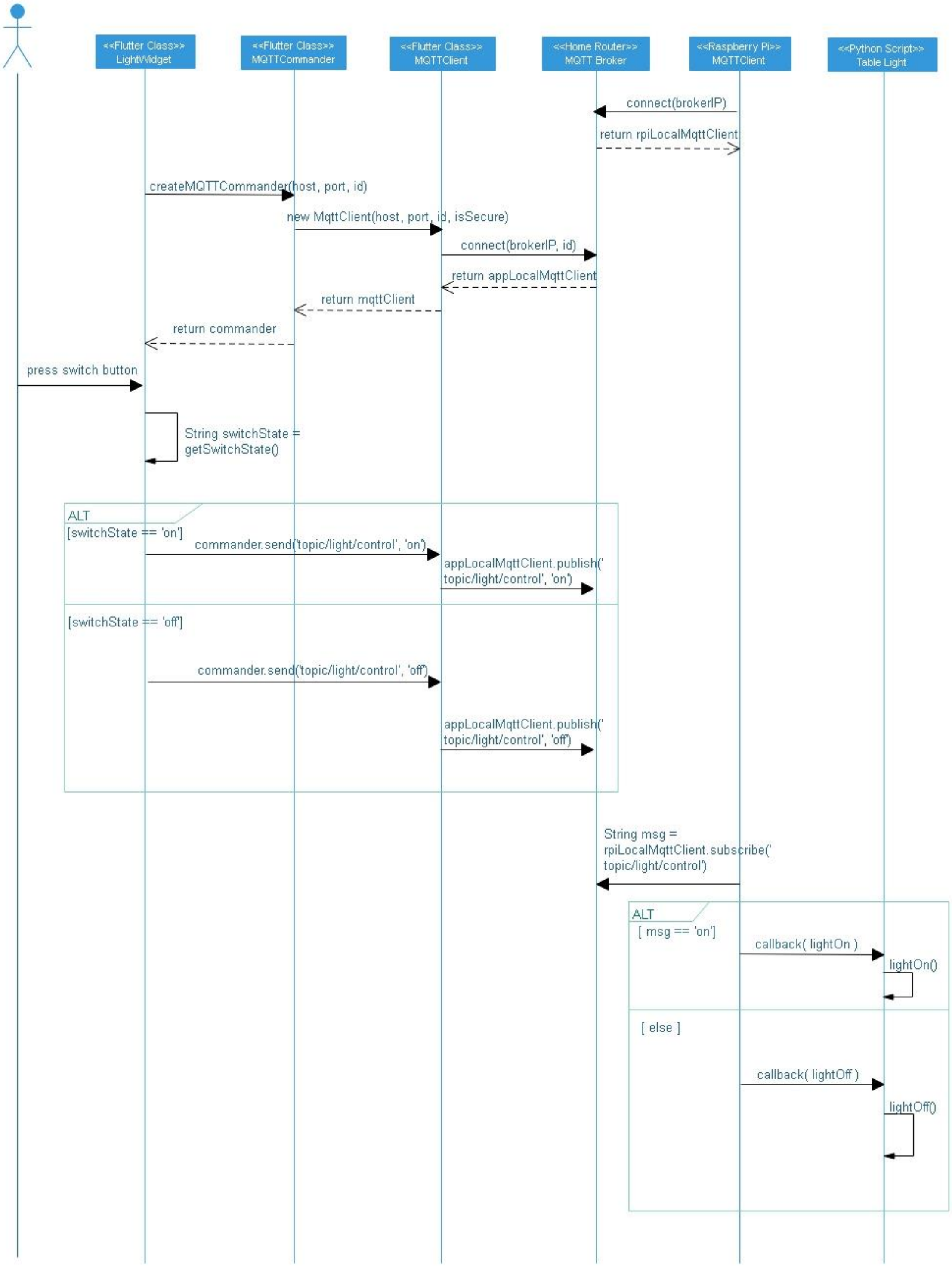

<<Flutter Class>>
LightWidget
<<Flutter Class>>
MQTTCommander
<<Flutter Class>>
MQTTClient
<<Home Router>>
MQTT Broker
<<Raspberry Pi>>
MQTTClient
<<Python Script>>
Table Light
connect(brokerIP)
return rpiLocalMqttClient
createMQTTCommander(host, port, id)
new MqttClient(host, port, id, isSecure)
connect(brokerIP, id)
return appLocalMqttClient
return mqttClient
return commander
press switch button
String switchState =
getSwitchState()
ALT
[switchState == 'on']
commander.send('topic/light/control', 'on')
appLocalMqttClient.publish('
topic/light/control', 'on')
[switchState == 'off']
commander.send('topic/light/control', 'off')
appLocalMqttClient.publish('
topic/light/control', 'off')
String msg =
rpiLocalMqttClient.subscribe('
topic/light/control')
ALT
[ msg == 'on']
callback( lightOn )
lightOn()
[ else ]
callback( lightOff )
lightOff()

*Figure 26 Turn Light On/Off Sequence Diagram 1*

### 5.1.1.5.2. Turn Light On/Off via Internet

- *Description*

The householder could click the button on the mobile phone screen to remotely control the light turn on or off between local network and Internet.

- *Key Parameters*

| MQTT Client | Role | Action | Network | Method | Topic Name | Message | Return Value |
|---|---|---|---|---|---|---|---|
| Flutter App | sender | switch button on | Internet | publish | topic/light/control | on | *[not applicate]* |
| Raspberry Pi | receiver | *[not applicate]* | | subscribe | topic/light/control | *not applicate* | on |
| Flutter App | sender | switch button off | | publish | topic/light/control | off | *[not applicate]* |
| Raspberry Pi | receiver | *[not applicate]* | | subscribe | topic/light/control | *[not applicate]* | off |

- *Sequence Diagram*

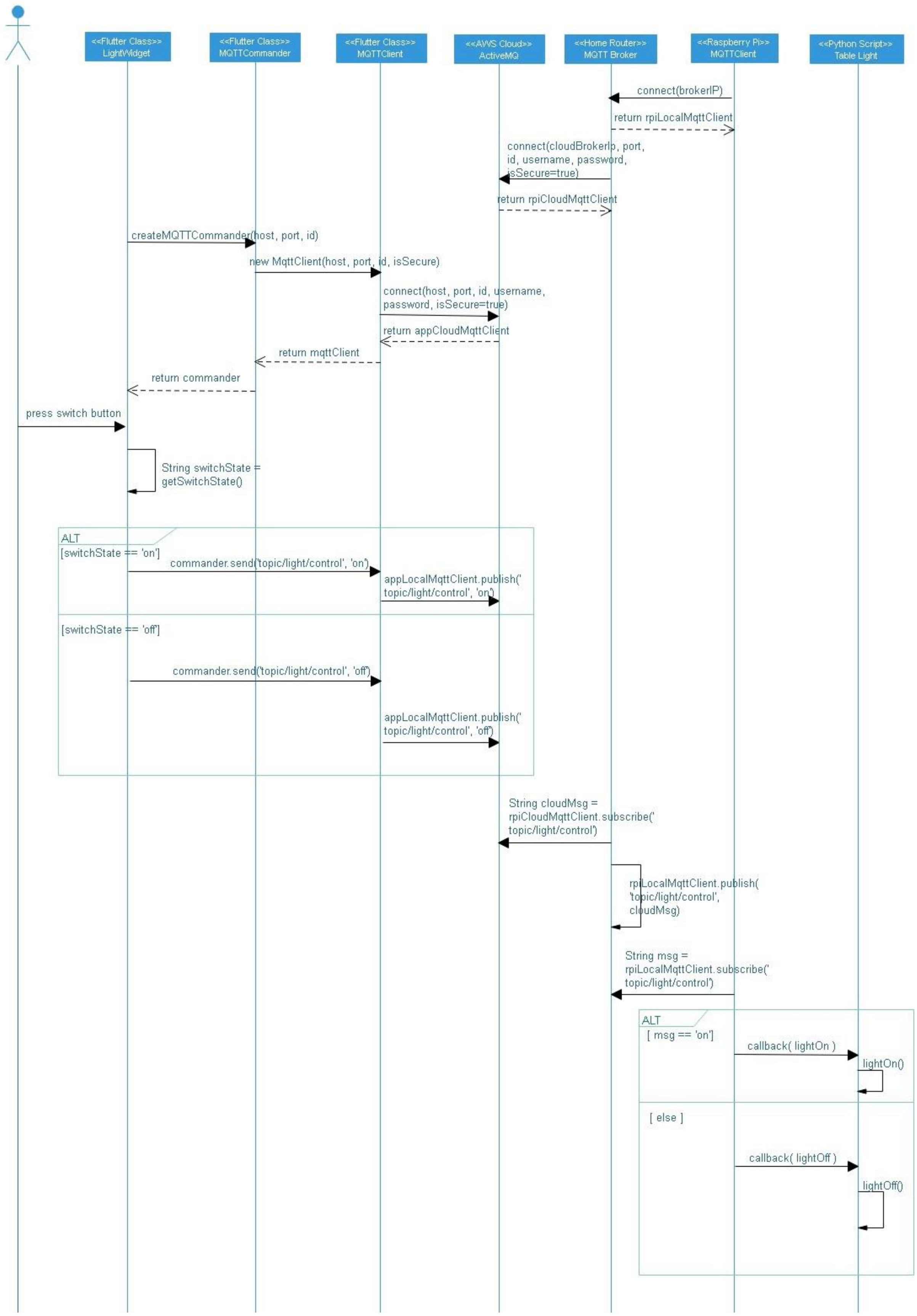


*Figure 27 Turn Light On/Off Sequence Diagram 2*

### 5.1.1.5.3. Turn Light On/Off with Voice via Wi-Fi

- *Description*

The householder could speak to the mobile phone to remotely control the light turn on or off between local network. The keywords are ‘light on’ or ‘light off’ should be spoke to mobile phone.

- *Key Parameters*

| MQTT Client | Role | Action | Network | Method | Topic Name | Message | Return Value |
|---|---|---|---|---|---|---|---|
| Flutter App | sender | Said ‘light on’ to phone | Wi-Fi | publish | topic/light/control | on | *[not applicate]* |
| Raspberry Pi | receiver | *[not applicate]* | | subscribe | topic/light/control | *[not applicate]* | on |
| Flutter App | sender | Said ‘light off’ to phone | | publish | topic/light/control | off | *[not applicate]* |
| Raspberry Pi | receiver | *[not applicate]* | | subscribe | topic/light/control | *[not applicate]* | off |

- *Sequence Diagram*

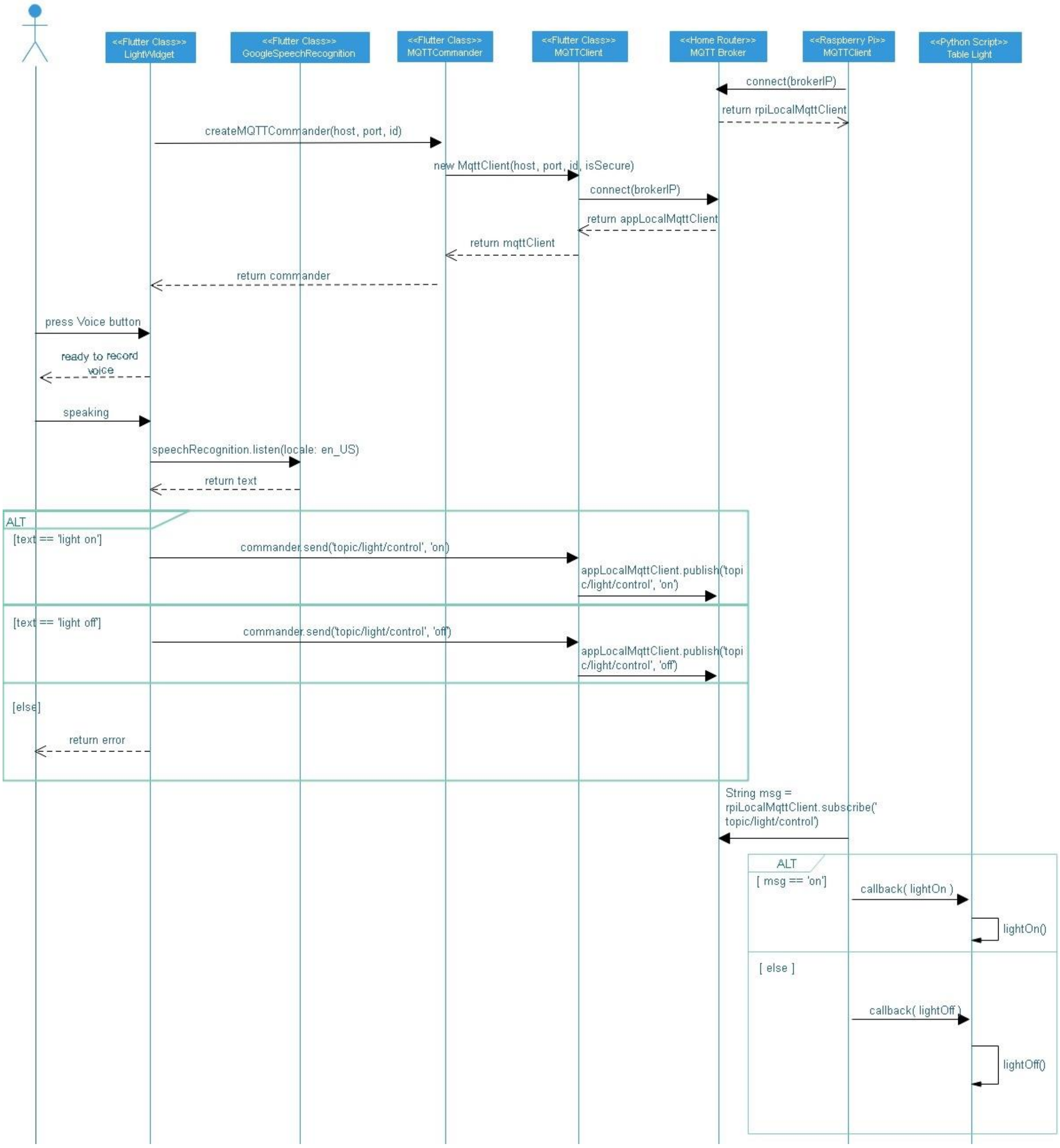


*Figure 28 Turn Light On/Off Sequence Diagram 3*

### 5.1.1.5.4. Turn Light On/Off with Voice via Internet

- *Description*

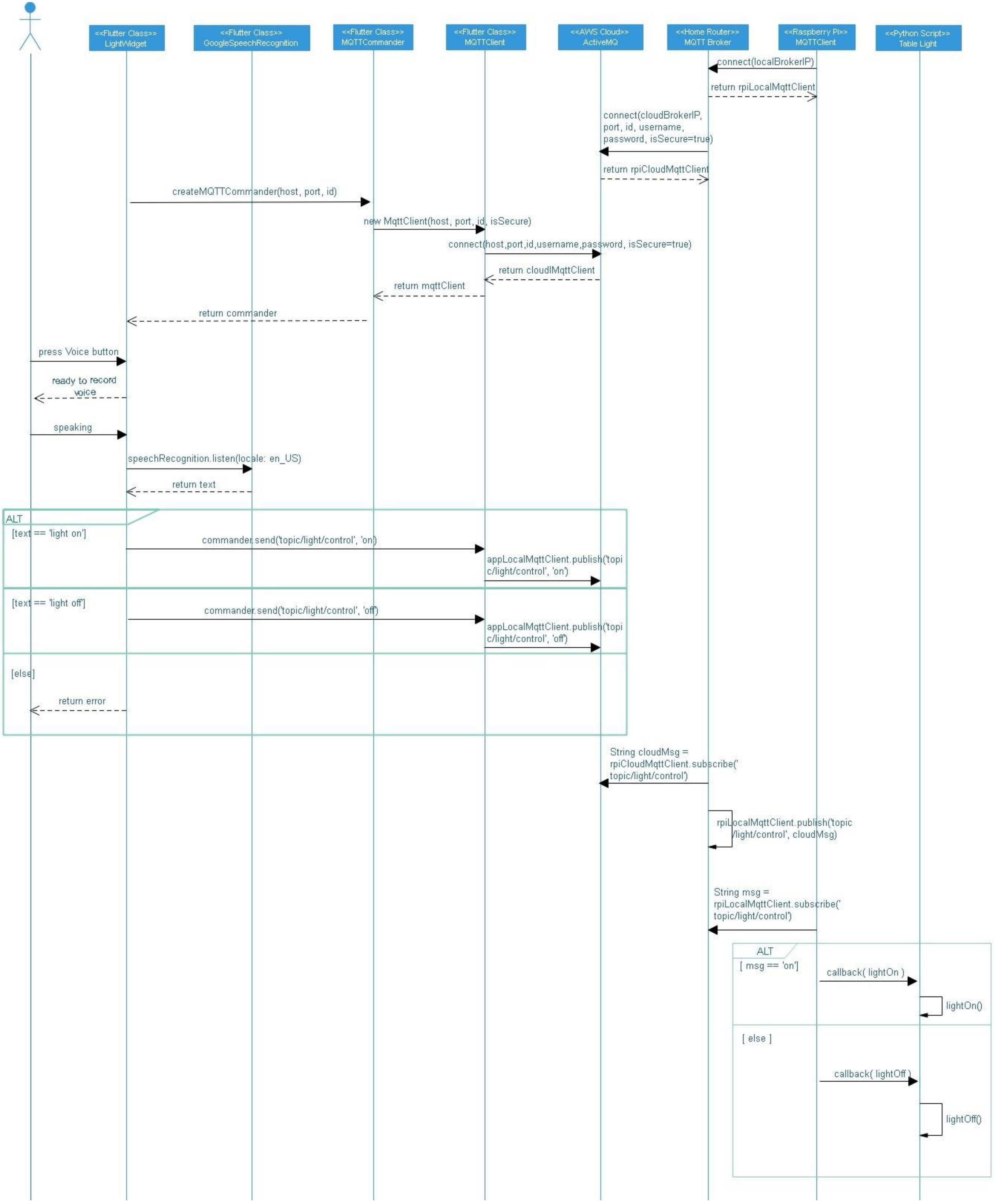
The householder could speak to the mobile phone to remotely control the light turn on or off through internet. The keywords are 'light on' or 'light off' should be spoke to mobile phone.

- *Key Parameters*

| MQTT Client | Role | Action | Network | Method | Topic Name | Message | Return Value |
|---|---|---|---|---|---|---|---|
| Flutter App | sender | Said 'light on' to phone | Internet | publish | topic/light/control | on | *[not applicate]* |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Raspberry Pi | receiver | *[not applicate]* | | subscribe | topic/light/control | *[not applicate]* | on |
| Flutter App | sender | Said ‘light off’ to phone | | publish | topic/light/control | off | *[not applicate]* |
| Raspberry Pi | receiver | *[not applicate]* | | subscribe | topic/light/control | *[not applicate]* | off |

- *Sequence Diagram*



<<Flutter Class>> LightWidget
<<Flutter Class>> GoogleSpeechRecognition
<<Flutter Class>> MQTTCommander
<<Flutter Class>> MQTTClient
<<AWS Cloud>> ActiveMQ
<<Home Router>> MQTT Broker
<<Raspberry Pi>> MQTTClient
<<Python Script>> Table Light
connect(localBrokerIP)
return rpiLocalMqttClient
connect(cloudBrokerIP, port, id, username, password, isSecure=true)
return rpiCloudMqttClient
createMQTTCommander(host, port, id)
new MqttClient(host, port, id, isSecure)
connect(host,port,id,username,password, isSecure=true)
return cloudlMqttClient
return mqttClient
return commander
press Voice button
ready to record voice
speaking
speechRecognition.listen(locale: en_US)
return text
ALT
[text == 'light on']
commander.send('topic/light/control', 'on')
appLocalMqttClient.publish('topic/light/control', 'on')
[text == 'light off']
commander.send('topic/light/control', 'off')
appLocalMqttClient.publish('topic/light/control', 'off')
[else]
return error
String cloudMsg = rpiCloudMqttClient.subscribe(' topic/light/control')
rpiLocalMqttClient.publish('topic /light/control', cloudMsg)
String msg = rpiLocalMqttClient.subscribe(' topic/light/control')
ALT
[ msg == 'on']
callback( lightOn )
lightOn()
[ else ]
callback( lightOff )
lightOff()

*Figure 29 Turn Light On/Off Sequence Diagram 4*

### 5.1.2. Fan

The Fan module that manages their status is to turn on, turn off and adjust running speed.

#### 5.1.2.1. Use Case

The use case diagram below is clearly shown the process of the Fan control between the Mobile app, Raspberry Pi.

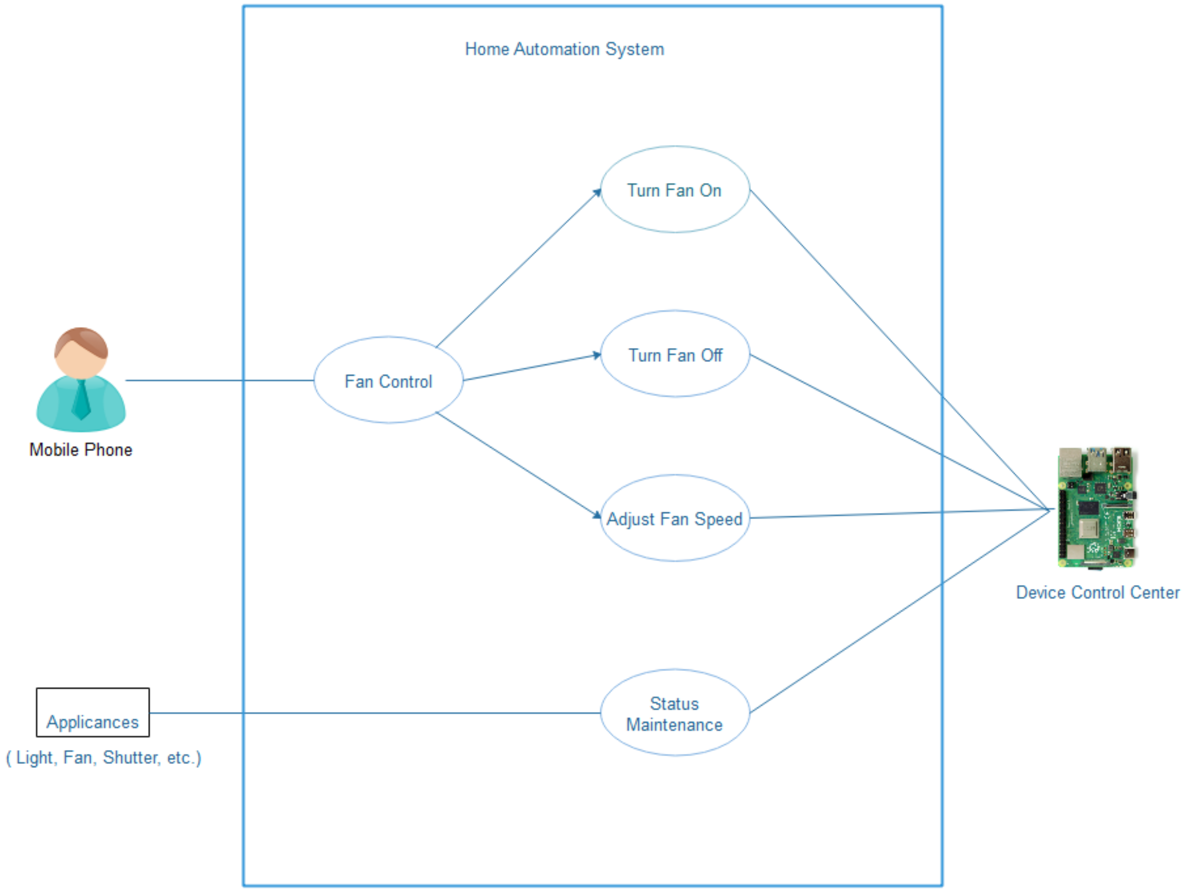


*Figure 30 Fan Use Case*

#### 5.1.2.2. Class Diagram

The Fan module classes are described using UML class diagram below.

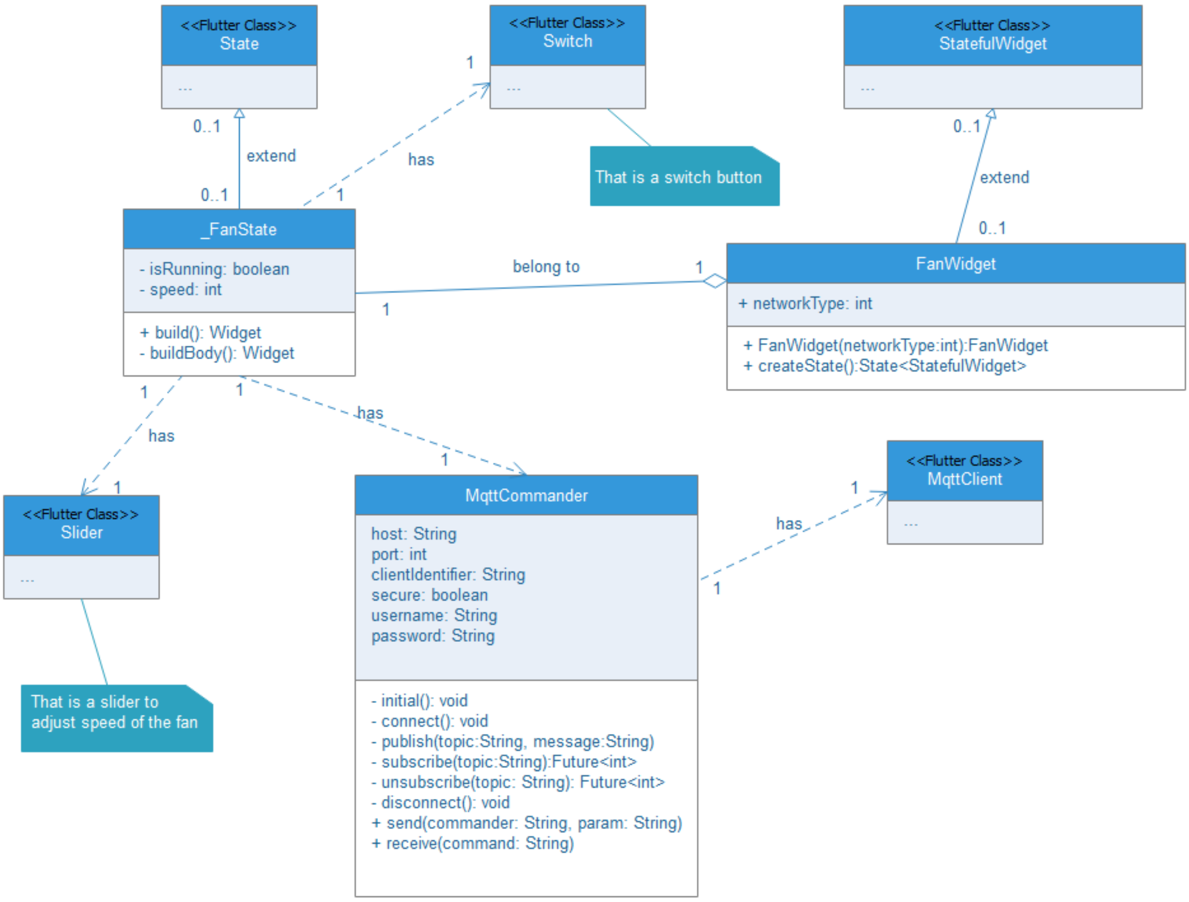


*Figure 31 Fan Class Diagram*

### 5.1.2.3. Wiring Diagram

The diagram described how the components connect to each other. There is useful information provided below.

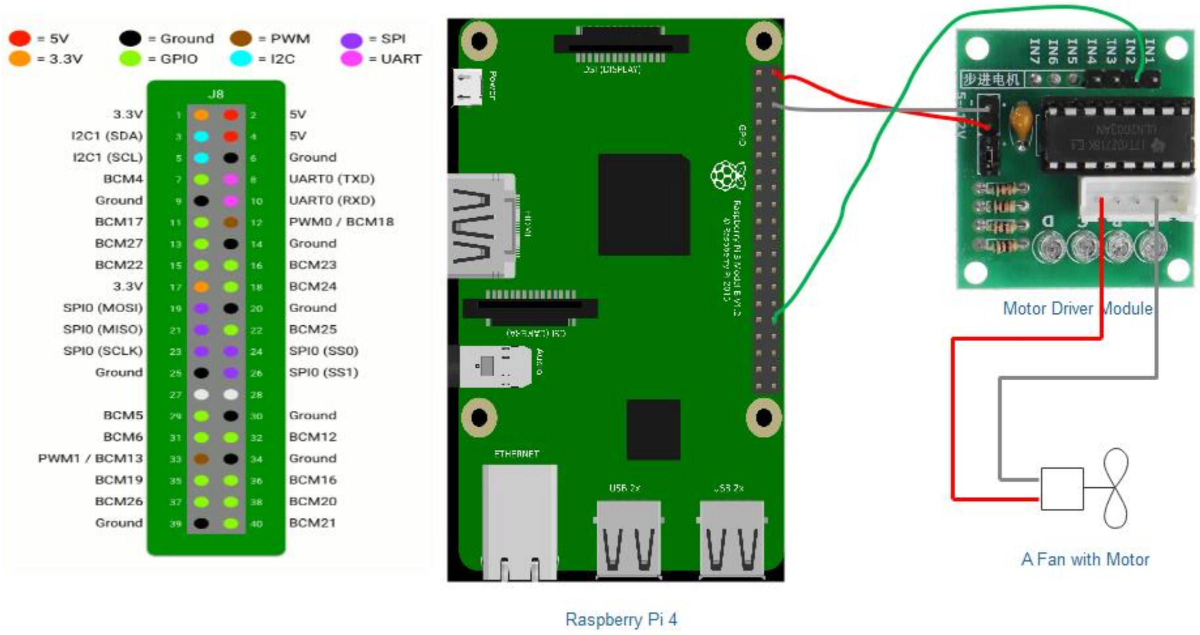


*Figure 32 Fan Hardware Design*

1. Raspberry Pi and Motor Driver Module

| **Raspberry Pi** | **Motor Driver Module** | **Wire Colour** |
|---|---|---|
| 5v Pin | Positive Pin | Red |
| GND Pin | Negative Pin | Grey |
| BCM 12 | In2 Pin | Green |

2. Motor Driver Module and Fan Motor

| **Motor Driver Module** | **Fan Motor** | **Wire Colour** |
|---|---|---|
| First Pin in Slot | Positive | Red |
| Fourth Pin Slot | *Negative* | Grey |

### 5.1.2.4. Device Control Script

There are Python scripts implements to control the Light device through the Raspberry Pi. The Python script is described using class diagram below.

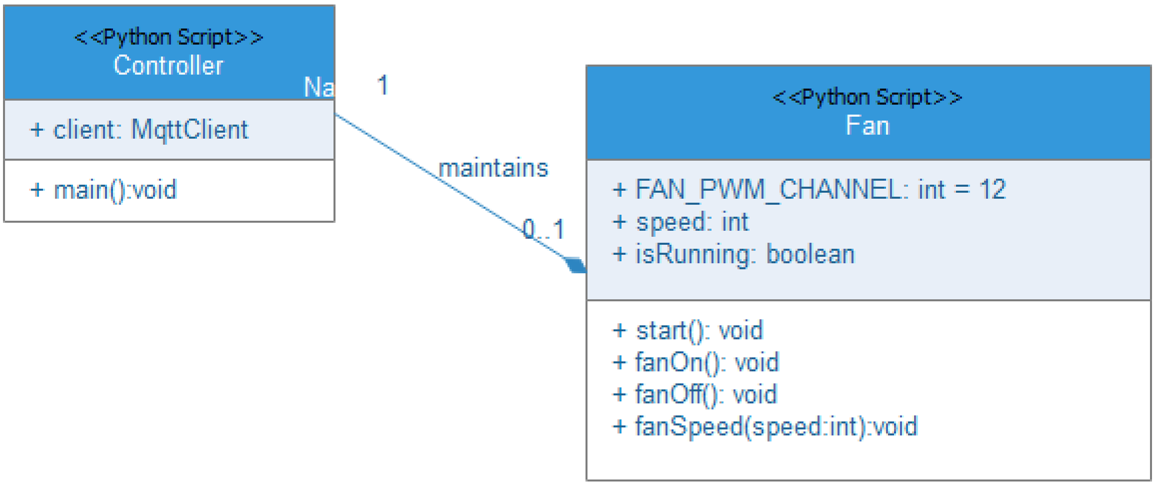


*Figure 33 Fan Python Scripts Class Diagram*

### 5.1.2.5. Functional Description

In this section, it will describe that functions design of the Fan in detail.

#### 5.1.2.5.1. Turn Fan On/Off

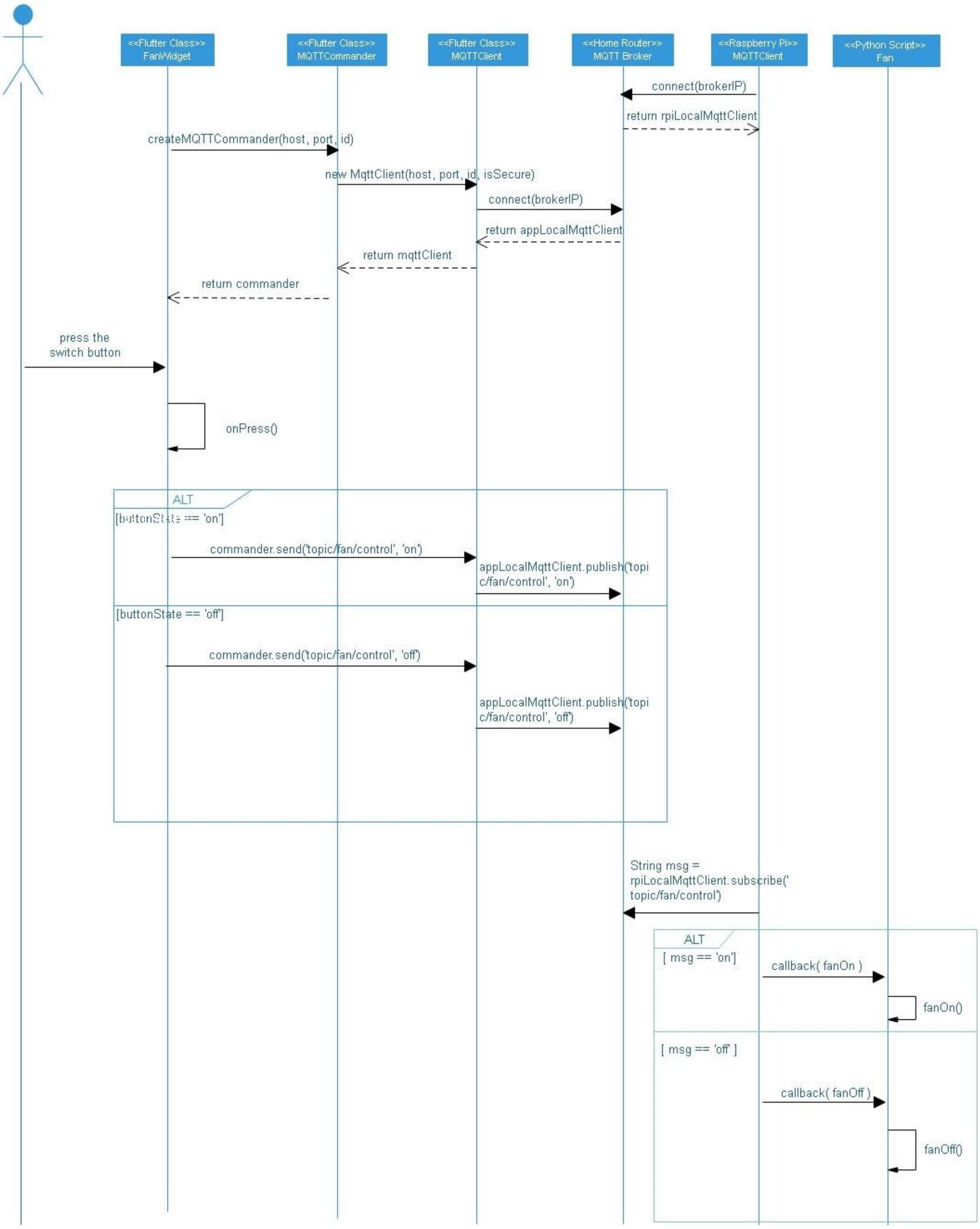
- *Description*

The householder could click the button on the mobile phone screen to remotely control the fan to turn on or off through local network.

- *Key Parameters*

| MQTT Client | Role | Action | Network | Method | Topic Name | Message | Return Value |
|---|---|---|---|---|---|---|---|
| Flutter App | sender | switch button on | Wi-Fi | publish | topic/fan/control | on | *[not applicate]* |
| Raspberry Pi | receiver | *[not applicate]* | | subscribe | topic/fan/control | *[not applicate]* | on |
| Flutter App | sender | switch button off | | publish | topic/fan/control | off | *not applicate* |
| Raspberry Pi | receiver | *[not applicate]* | | subscribe | topic/fan/control | *[not applicate]* | off |

- *Sequence Diagram*



<<Flutter Class>> FanWidget
<<Flutter Class>> MQTTCommander
<<Flutter Class>> MQTTClient
<<Home Router>> MQTT Broker
<<Raspberry Pi>> MQTTClient
<<Python Script>> Fan
connect(brokerIP)
return rpiLocalMqttClient
createMQTTCommander(host, port, id)
new MqttClient(host, port, id, isSecure)
connect(brokerIP)
return appLocalMqttClient
return mqttClient
return commander
press the switch button
onPress()
ALT
[buttonState == 'on']
commander.send('topic/fan/control', 'on')
appLocalMqttClient.publish('topic/fan/control', 'on')
[buttonState == 'off']
commander.send('topic/fan/control', 'off')
appLocalMqttClient.publish('topic/fan/control', 'off')
String msg = rpiLocalMqttClient.subscribe('topic/fan/control')
ALT
[ msg == 'on']
callback( fanOn )
fanOn()
[ msg == 'off' ]
callback( fanOff )
fanOff()

*Figure 34 Turn Fan On/Off Sequence Diagram*

### 5.1.2.5.2. Adjust Fan Speed

- *Description*

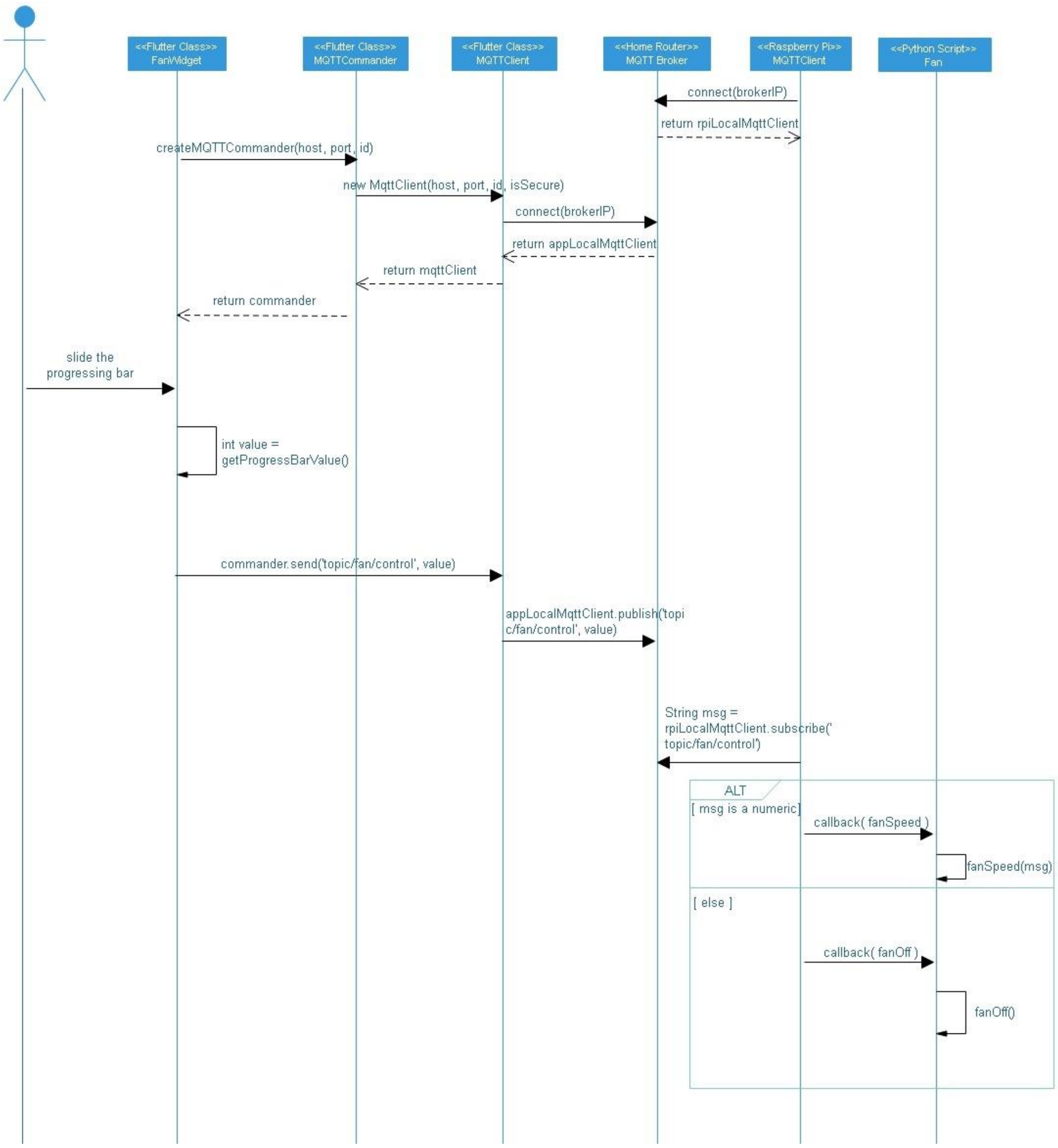
The householder could slide the progressing bar on the mobile phone screen to remotely control the fan running speed through local network.

- *Key Parameters*

| MQTT Client | Role | Action | Network | Method | Topic Name | Message | Return Value |
|---|---|---|---|---|---|---|---|

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Flutter App | sender | Slide the progressing bar | Wi-Fi | publish | topic/fan/control | Numeric value | *[not applicate]* |
| Raspberry Pi | receiver | *[not applicate]* | | subscribe | topic/fan/control | *[not applicate]* | Numeric value |

*Sequence Diagram*



<<Flutter Class>> FanWidget
<<Flutter Class>> MQTTCommander
<<Flutter Class>> MQTTClient
<<Home Router>> MQTT Broker
<<Raspberry Pi>> MQTTClient
<<Python Script>> Fan
connect(brokerIP)
return rpiLocalMqttClient
createMQTTCommander(host, port, id)
new MqttClient(host, port, id, isSecure)
connect(brokerIP)
return appLocalMqttClient
return mqttClient
return commander
slide the progressing bar
int value = getProgressBarValue()
commander.send('topic/fan/control', value)
appLocalMqttClient.publish('topic/fan/control', value)
String msg = rpiLocalMqttClient.subscribe('topic/fan/control')
ALT
[ msg is a numeric]
callback( fanSpeed )
fanSpeed(msg)
[ else ]
callback( fanOff )
fanOff()

*Figure 35 Adjust Fan Speed Sequence Diagram*

### 5.1.3. Shutter

The Shutter module that manages their directions to move and pause itself.

### 5.1.3.1. Use Case

The use case diagram below is clearly shown the process of the Shutter control between the Mobile phone, Raspberry Pi.

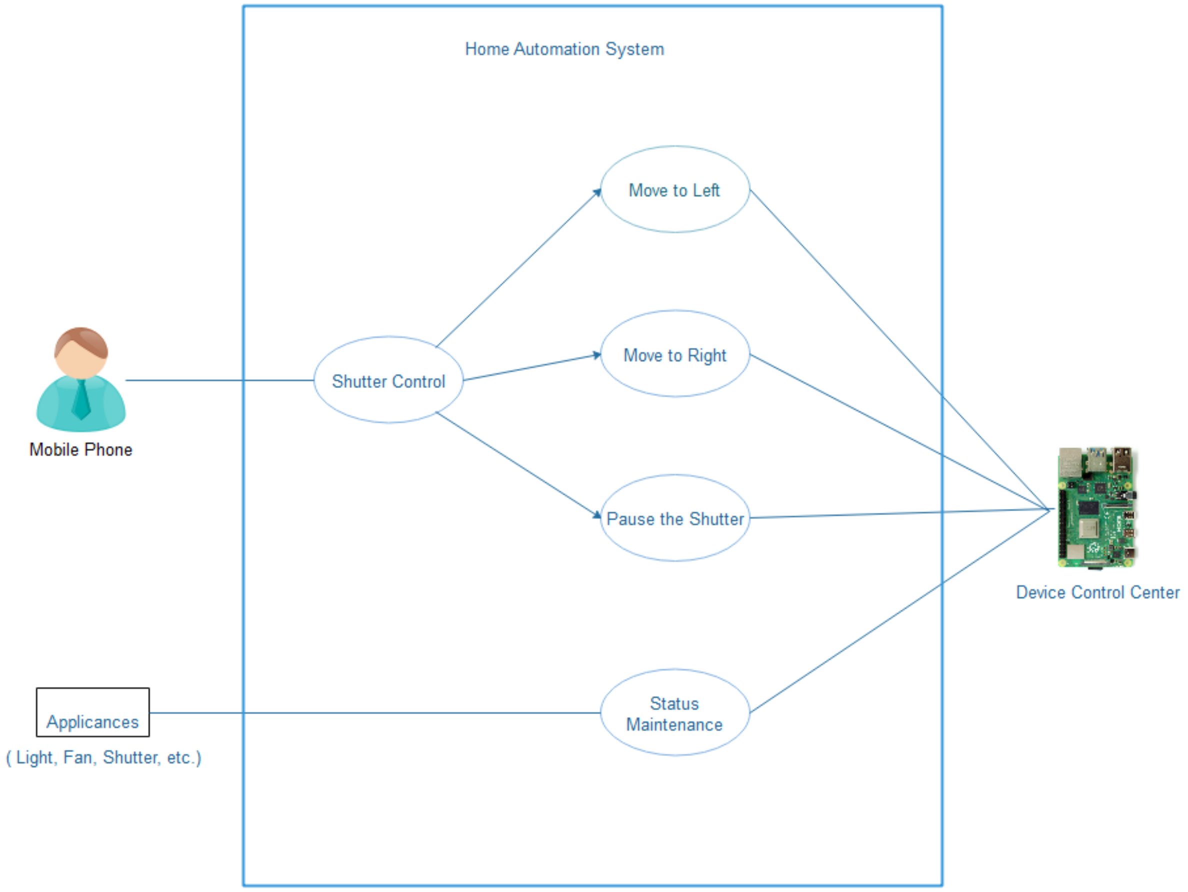


*Figure 36 Shutter Use Case*

### 5.1.3.2. Class Diagram

The Shuter module classes are described using UML class diagram below.

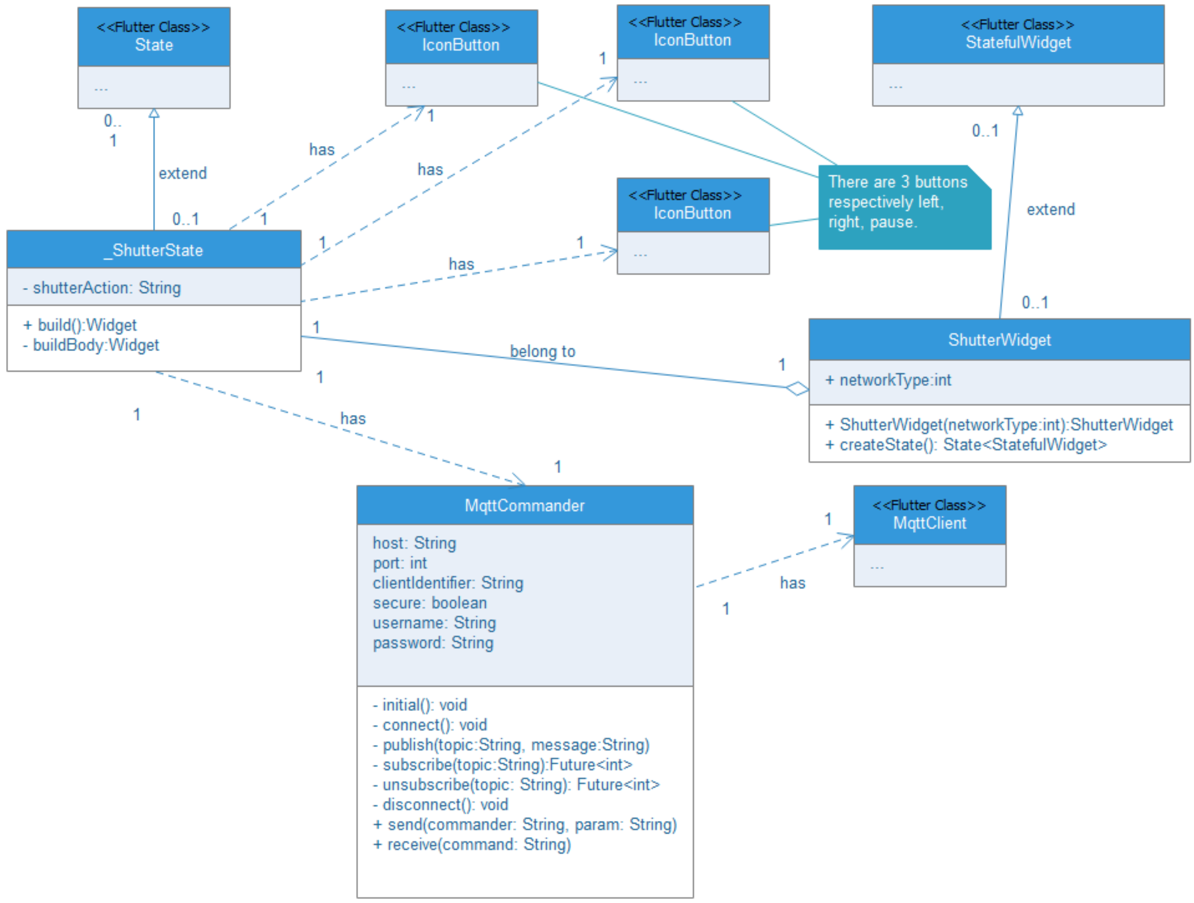


*Figure 37 Shutter Class Diagram*

### 5.1.3.3. Wiring Diagram

The diagram described how the components connect to each other. There is useful information provided below.

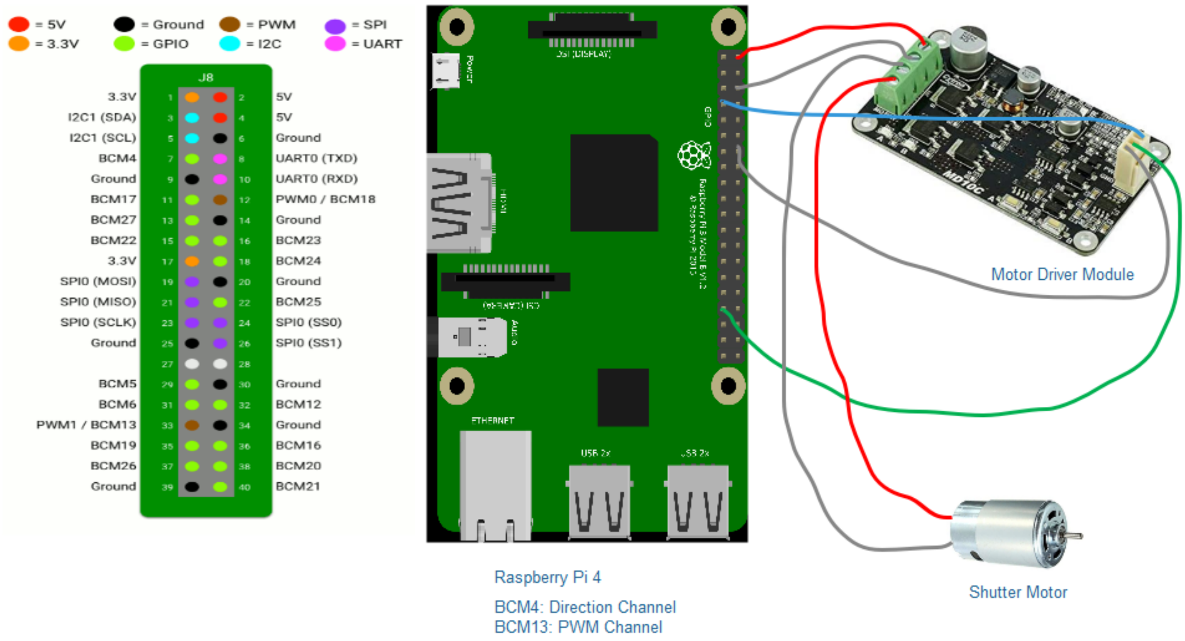


*Figure 38 Shutter Hardware Design*

1. Raspberry Pi and Motor Driver Module

| Raspberry Pi | Motor Driver Module | Wire Colour | Description |
|---|---|---|---|
| 5v Pin | Positive Pin | Red | |
| GND Pin | Negative Pin | Grey | |
| BCM 4 | DIR Pin | Blue | Control motor direction |
| BCM 13 | PWM Pin | Green | Control motor speed |
| GND Pin | GND Pin | Grey | |

2. Motor Driver Module and Fan Motor

| Motor Driver Module | Shutter Motor | Wire Colour |
|---|---|---|
| A Pin | Positive | Red |
| B Pin | *Negative* | Grey |

### 5.1.3.4. Device Control Script

There are Python scripts implements to control the Shutter device through the Raspberry Pi. The Python script is described using class diagram below.

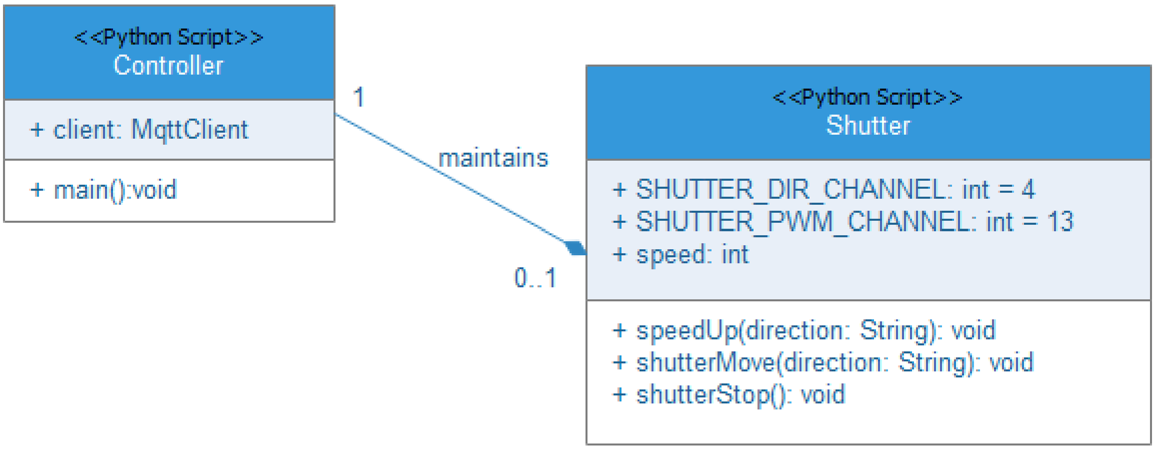

<<Python Script>>
Controller
+ client: MqttClient
+ main():void
1
maintains
0..1
<<Python Script>>
Shutter
+ SHUTTER_DIR_CHANNEL: int = 4
+ SHUTTER_PWM_CHANNEL: int = 13
+ speed: int
+ speedUp(direction: String): void
+ shutterMove(direction: String): void
+ shutterStop(): void

*Figure 39 Shutter Python Scripts Class Diagram*

#### 5.1.3.5. Functional Description

In this section, it will describe that functions design of the Shutter in detail.

##### 5.1.3.5.1. Move to Left/Right

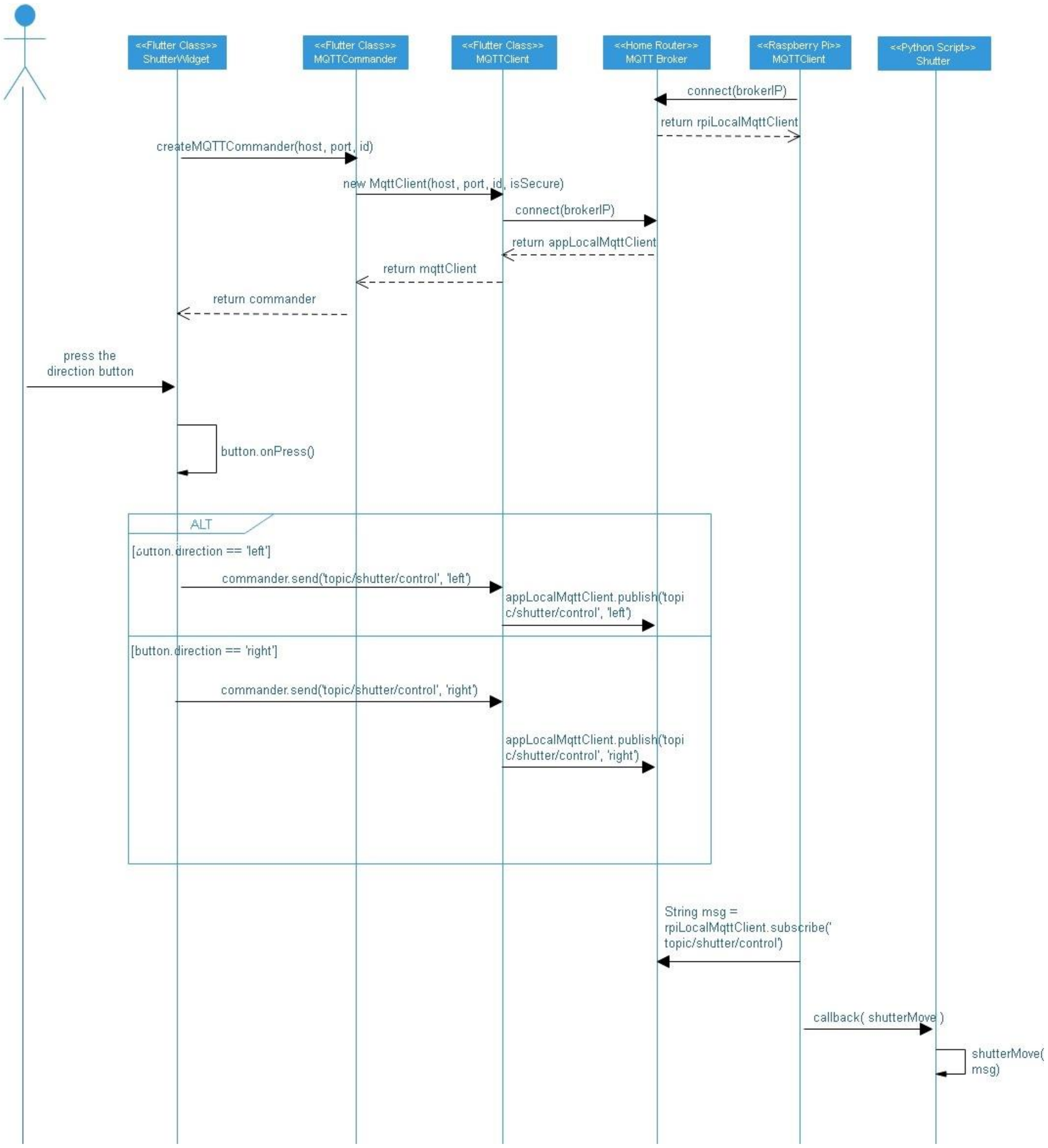
- *Description*

The householder could click the left or right arrow button on the mobile phone screen to remotely control the Shutter moving to left or right through local network.

- *Key Parameters*

| MQTT Client | Role | Action | Network | Method | Topic Name | Message | Return Value |
|---|---|---|---|---|---|---|---|
| Flutter App | sender | click the left arrow button | Wi-Fi | publish | topic/shutter/control | left | *[not applicate]* |
| Raspberry Pi | receiver | *[not applicate]* | | subscribe | topic/shutter/control | *[not applicate]* | left |
| Flutter App | sender | click the right arrow button | | publish | topic/shutter/control | right | *not applicate* |
| Raspberry Pi | receiver | *[not applicate]* | | subscribe | topic/shutter/control | *[not applicate]* | right |

- *Sequence Diagram*



<<Flutter Class>> ShutterWidget
<<Flutter Class>> MQTTCommander
<<Flutter Class>> MQTTClient
<<Home Router>> MQTT Broker
<<Raspberry Pi>> MQTTClient
<<Python Script>> Shutter
connect(brokerIP)
return rpiLocalMqttClient
createMQTTCommander(host, port, id)
new MqttClient(host, port, id, isSecure)
connect(brokerIP)
return appLocalMqttClient
return mqttClient
return commander
press the direction button
button.onPress()
ALT
[button.direction == 'left']
commander.send('topic/shutter/control', 'left')
appLocalMqttClient.publish('topic/shutter/control', 'left')
[button.direction == 'right']
commander.send('topic/shutter/control', 'right')
appLocalMqttClient.publish('topic/shutter/control', 'right')
String msg = rpiLocalMqttClient.subscribe('topic/shutter/control')
callback( shutterMove )
shutterMove( msg)

*Figure 40 Move Shutter to Left/Right Sequence Diagram*

### 5.1.3.5.2. Pause the Shutter

- *Description*

The householder could click pause button on the mobile phone screen to remotely control the Shutter to stop the shutter through local network.

- *Key Parameters*

| MQTT Client | Role | Action | Network | Method | Topic Name | Message | Return Value |
|---|---|---|---|---|---|---|---|
| Flutter App | sender | click the pause button | Wi-Fi | publish | topic/shutter/control | stop | *[not applicate]* |
| Raspberry Pi | receiver | *[not applicate]* | | subscribe | topic/shutter/control | *[not applicate]* | stop |

- *Sequence Diagram*

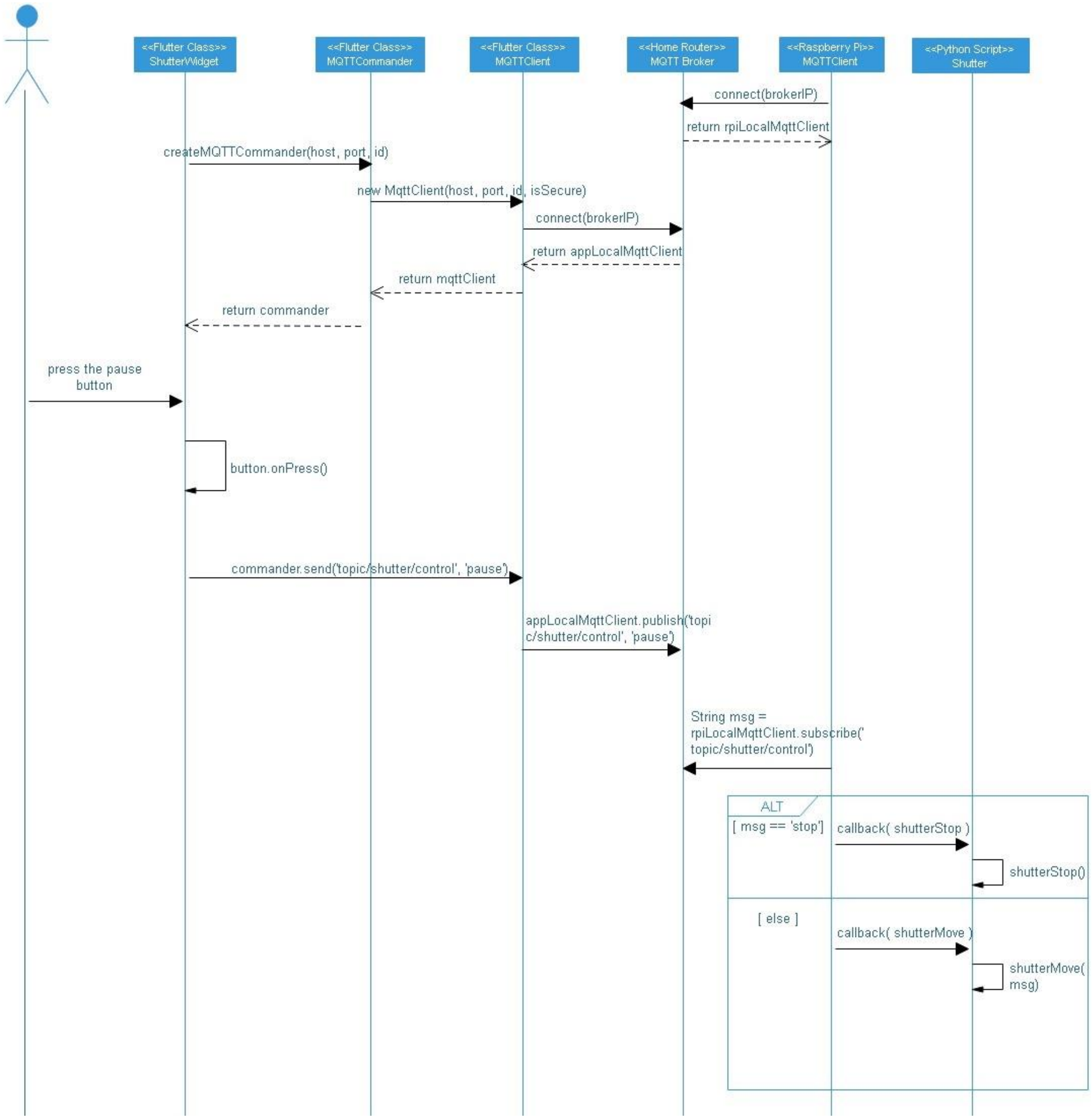


*Figure 41 Pause Shutter Class Diagram*

## 5.2. Security Monitoring

There are two parts in this module, one part is a CCTV to capture the live view and transfer to mobile phone. another one is a Door Access to manage the doorbell, camera, and lock that will be described in the next sections.

### 5.2.1. CCTV

The CCTV module that manages their status is to turn on and turn off.

### 5.2.1.1. Use Case

The use case diagram below is clearly shown the process of the CCTV control between the mobile phone, Raspberry Pi.

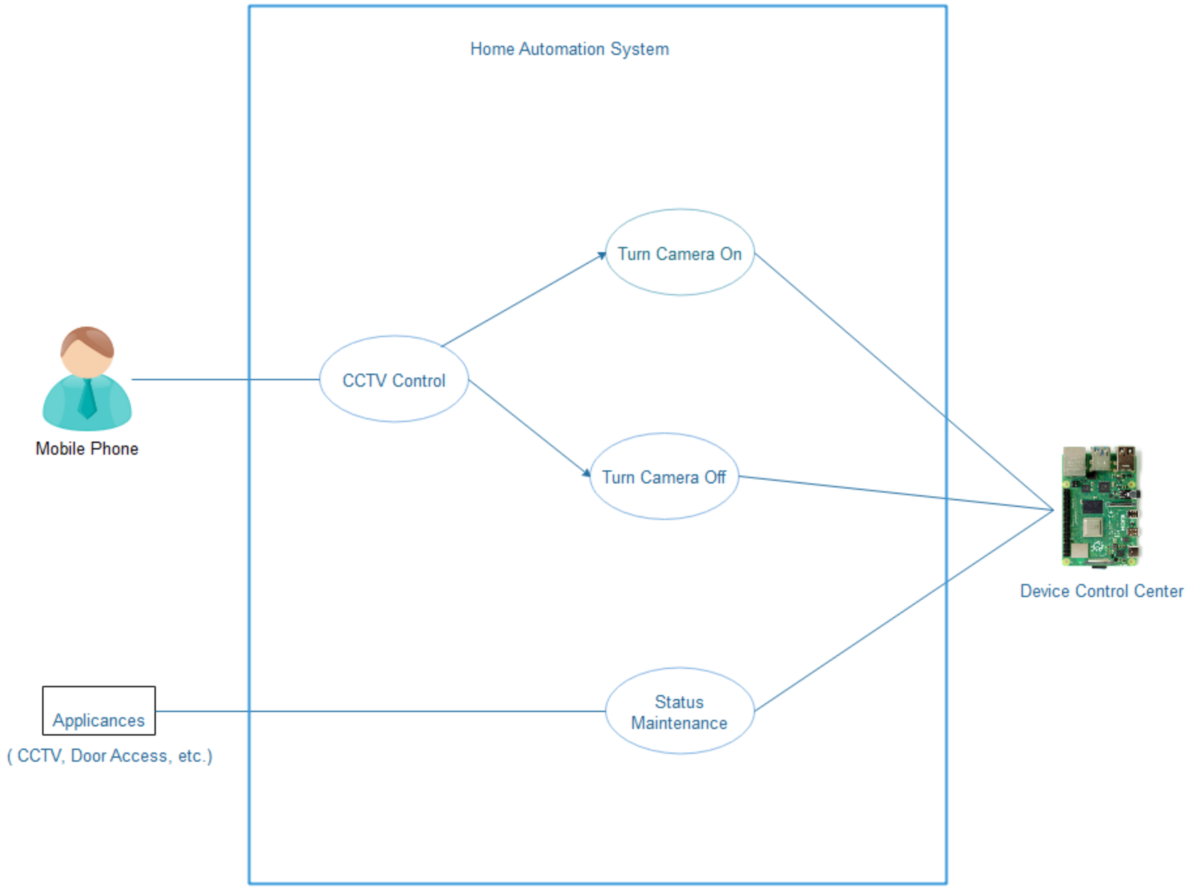


*Figure 42 CCTV Use Case*

### 5.2.1.2. Class Diagram

The CCTV module classes are described using UML class diagram below.

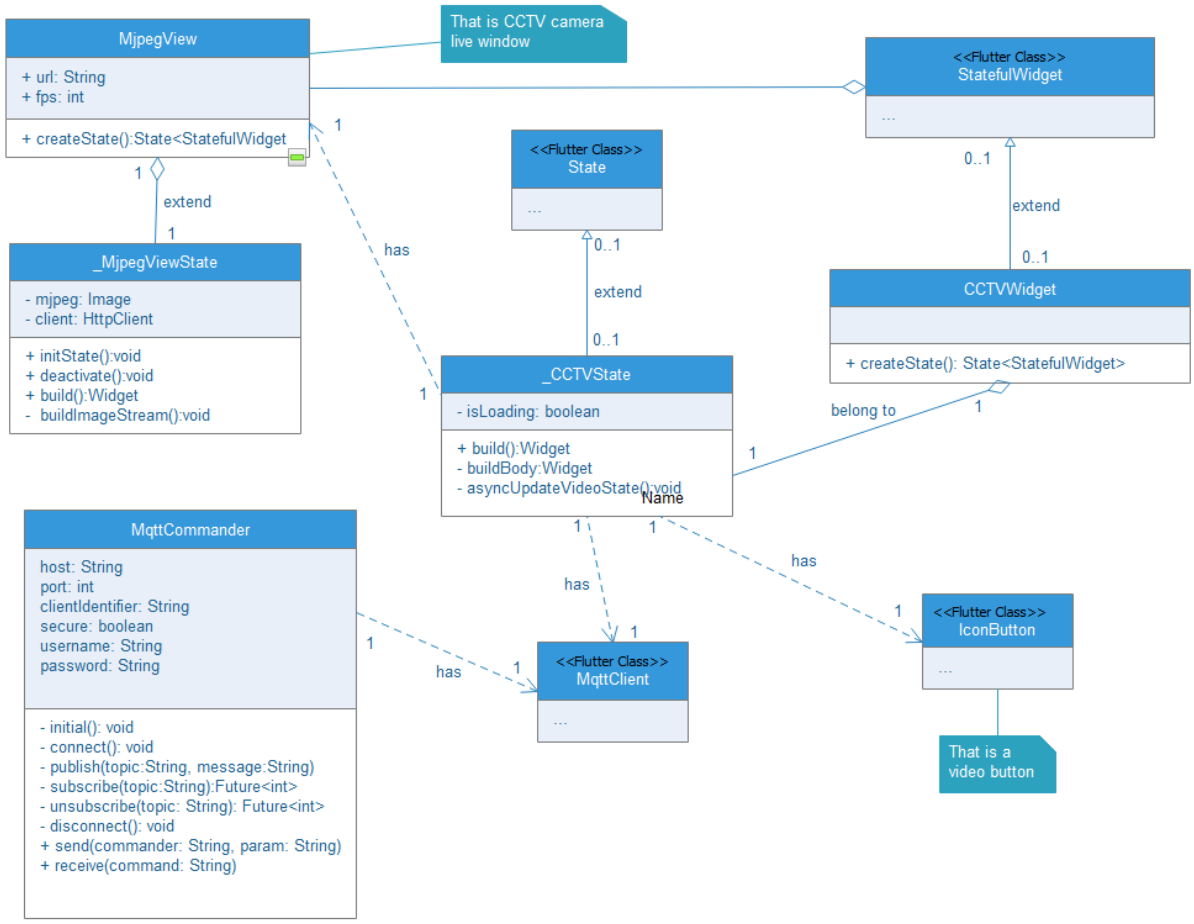


*Figure 43 CCTV Class Diagram*

### 5.2.1.3. Wiring Diagram

The diagram described how the components connect to each other. There is useful information provided below.

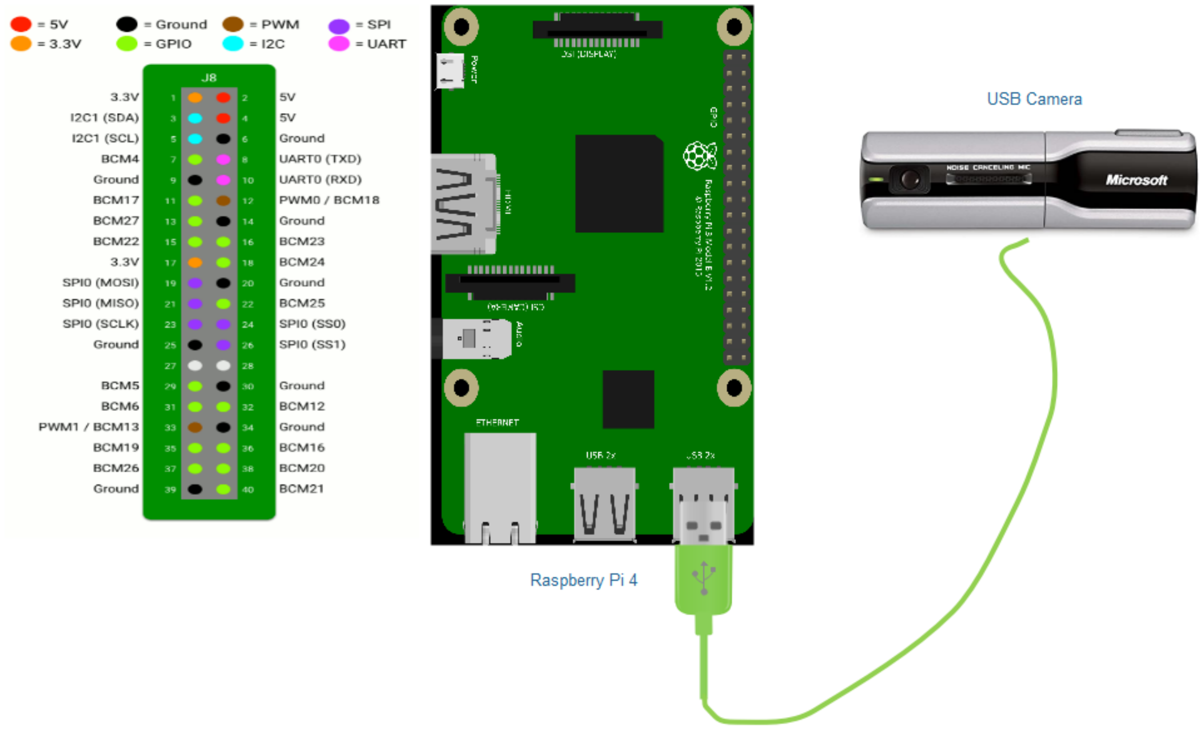


*Figure 44 CCTV Hardware Design*

1. Raspberry Pi and USB Camera

| **Raspberry Pi** | **USB Camera** | **Wire Colour** |
|---|---|---|
| USB Interface | USB Cable | Green |

### 5.2.1.4. Device Control Script

There are Python scripts implements to control the CCTV camera through the Raspberry Pi. The Python script is described using class diagram below.

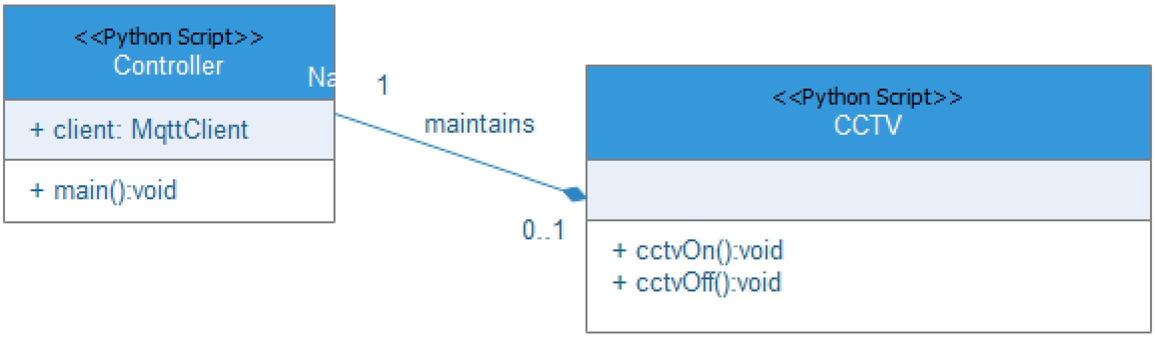


*Figure 45 CCTV Python Scripts Class Diagram*

### 5.2.1.5. Functional Description

In this section, it will describe that functions design of the CCTV in detail.

#### 5.2.1.5.1. Turn CCTV On/Off

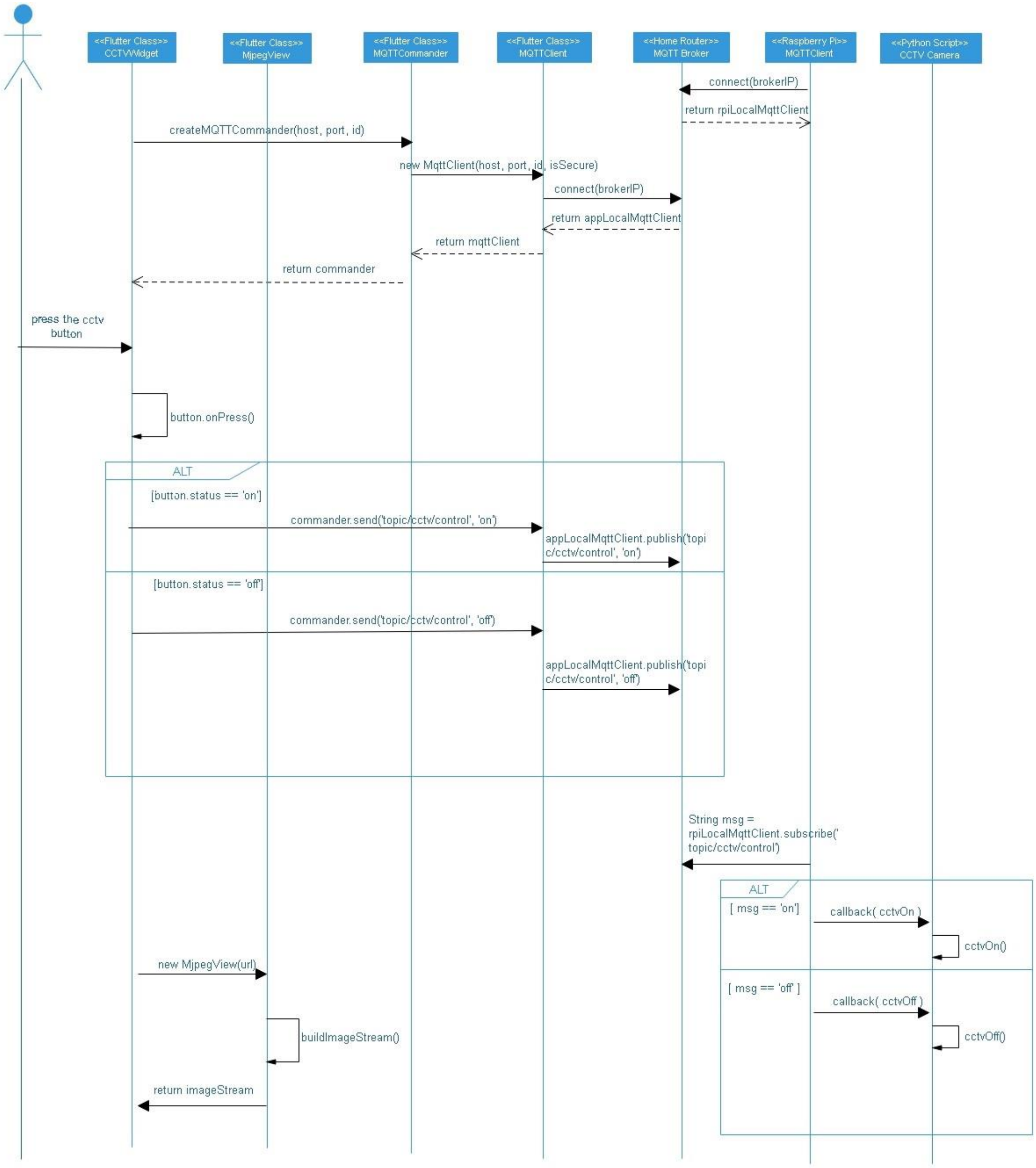
- *Description*

The householder could click the button on the mobile phone screen to remotely control the CCTV camera to turn on or off through local network.

- *Key Parameters*

| MQTT Client | Role | Action | Network | Method | Topic Name | Message | Return Value |
|---|---|---|---|---|---|---|---|
| Flutter App | sender | switch button on | Wi-Fi | publish | topic/cctv/control | on | *[not applicate]* |
| Raspberry Pi | receiver | *[not applicate]* | | subscribe | topic/cctv/control | *[not applicate]* | on |
| Flutter App | sender | switch button off | | publish | topic/fan/control | off | *not applicate* |
| Raspberry Pi | receiver | *[not applicate]* | | subscribe | topic/cctv/control | *[not applicate]* | off |

- *Sequence Diagram*



<<Flutter Class>> CCTVWidget
<<Flutter Class>> MjpegView
<<Flutter Class>> MQTTCommander
<<Flutter Class>> MQTTClient
<<Home Router>> MQTT Broker
<<Raspberry Pi>> MQTTClient
<<Python Script>> CCTV Camera
connect(brokerIP)
return rpiLocalMqttClient
createMQTTCommander(host, port, id)
new MqttClient(host, port, id, isSecure)
connect(brokerIP)
return appLocalMqttClient
return mqttClient
return commander
press the cctv button
button.onPress()
ALT
[button.status == 'on']
commander.send('topic/cctv/control', 'on')
appLocalMqttClient.publish('topic/cctv/control', 'on')
[button.status == 'off']
commander.send('topic/cctv/control', 'off')
appLocalMqttClient.publish('topic/cctv/control', 'off')
String msg = rpiLocalMqttClient.subscribe('topic/cctv/control')
ALT
[ msg == 'on']
callback( cctvOn )
cctvOn()
[ msg == 'off' ]
callback( cctvOff )
cctvOff()
new MjpegView(url)
buildImageStream()
return imageStream

*Figure 46 Turn CCTV On/Off Sequence Diagram*

### 5.2.2. Door Access

The Door Access module that manages the door status when a visitor stands at the door.

#### 5.2.2.1. Use Case

The use case diagram below is clearly shown the process of the Door Access control between the mobile phone, Raspberry Pi.

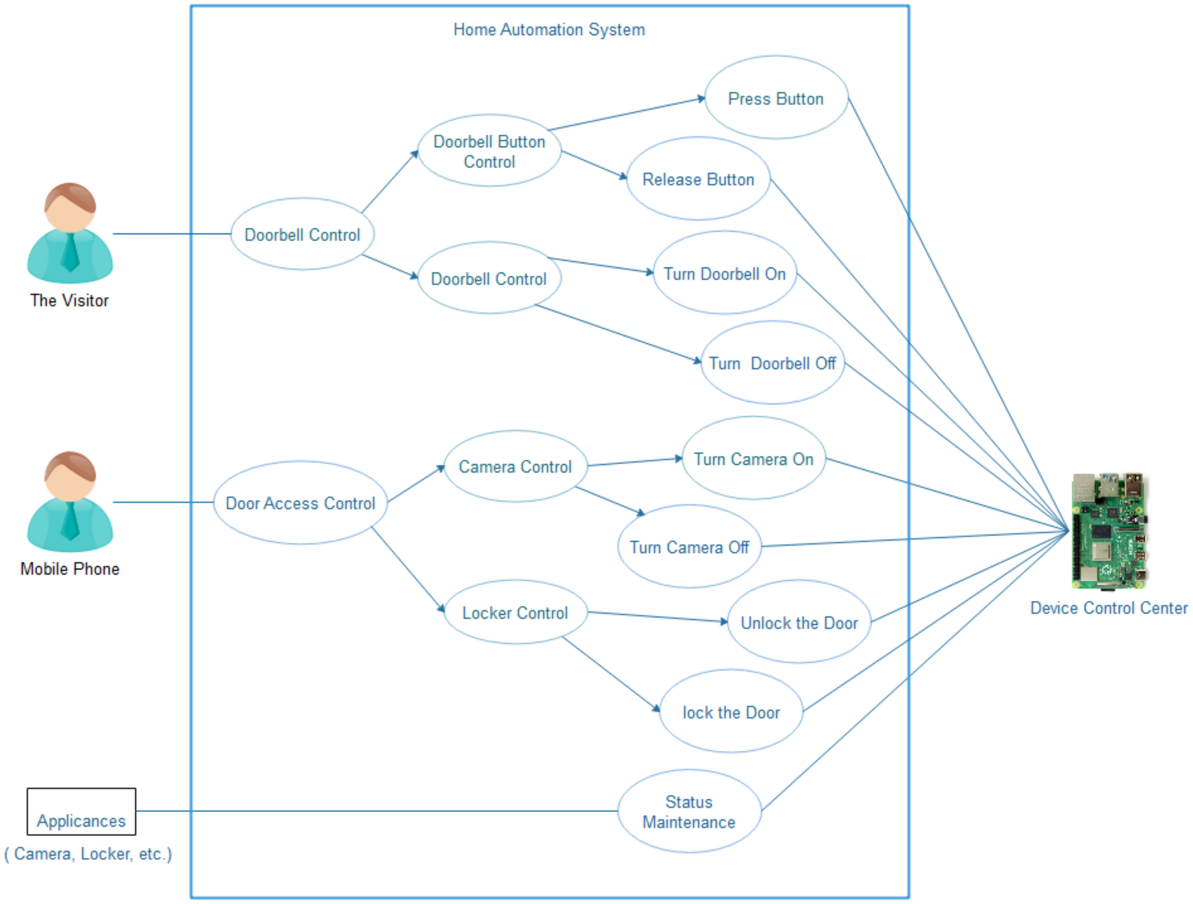


*Figure 47 Door Access Use Case*

### 5.2.2.2. Class Diagram

The CCTV module classes are described using UML class diagram below.

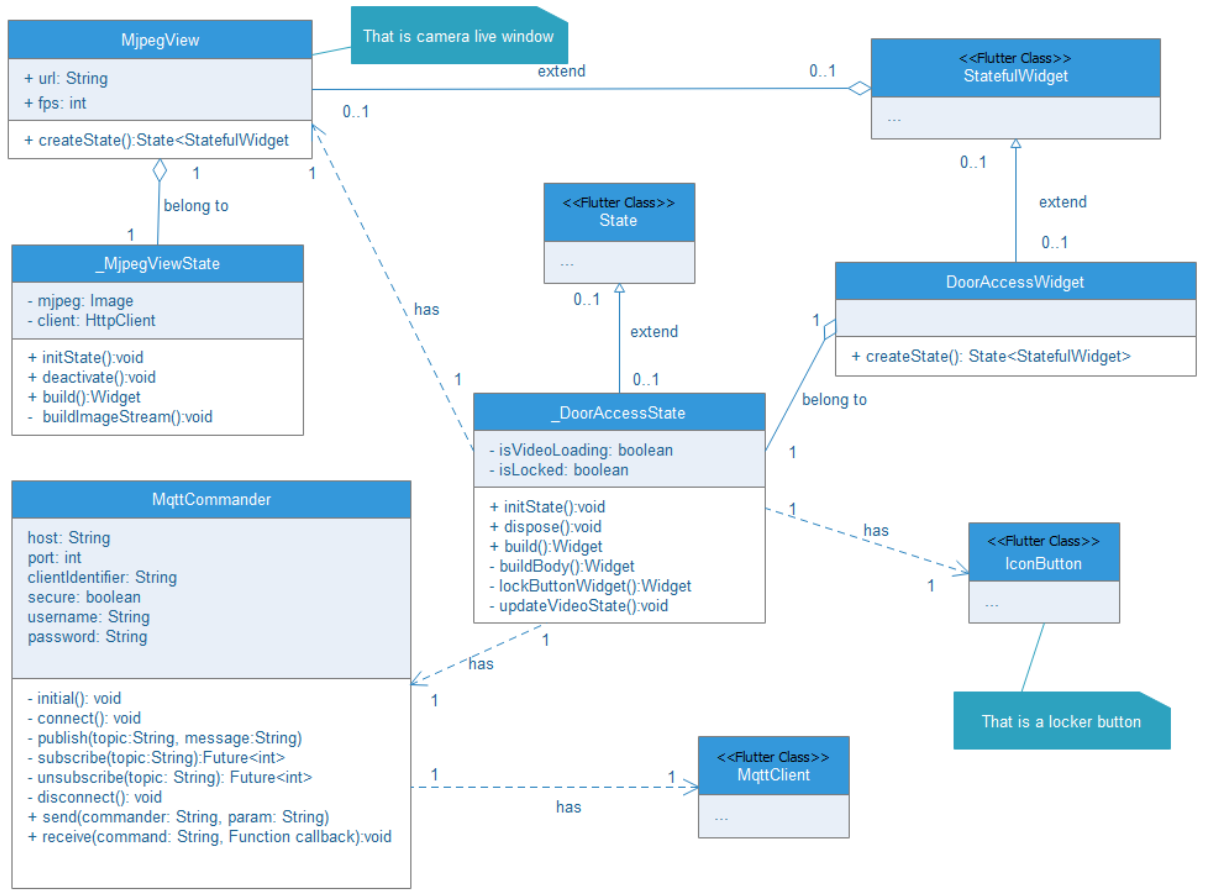


*Figure 48 Door Access Class Diagram*

### 5.2.2.3. Wiring Diagram

The diagram described how the components connect to each other. There is useful information provided below.

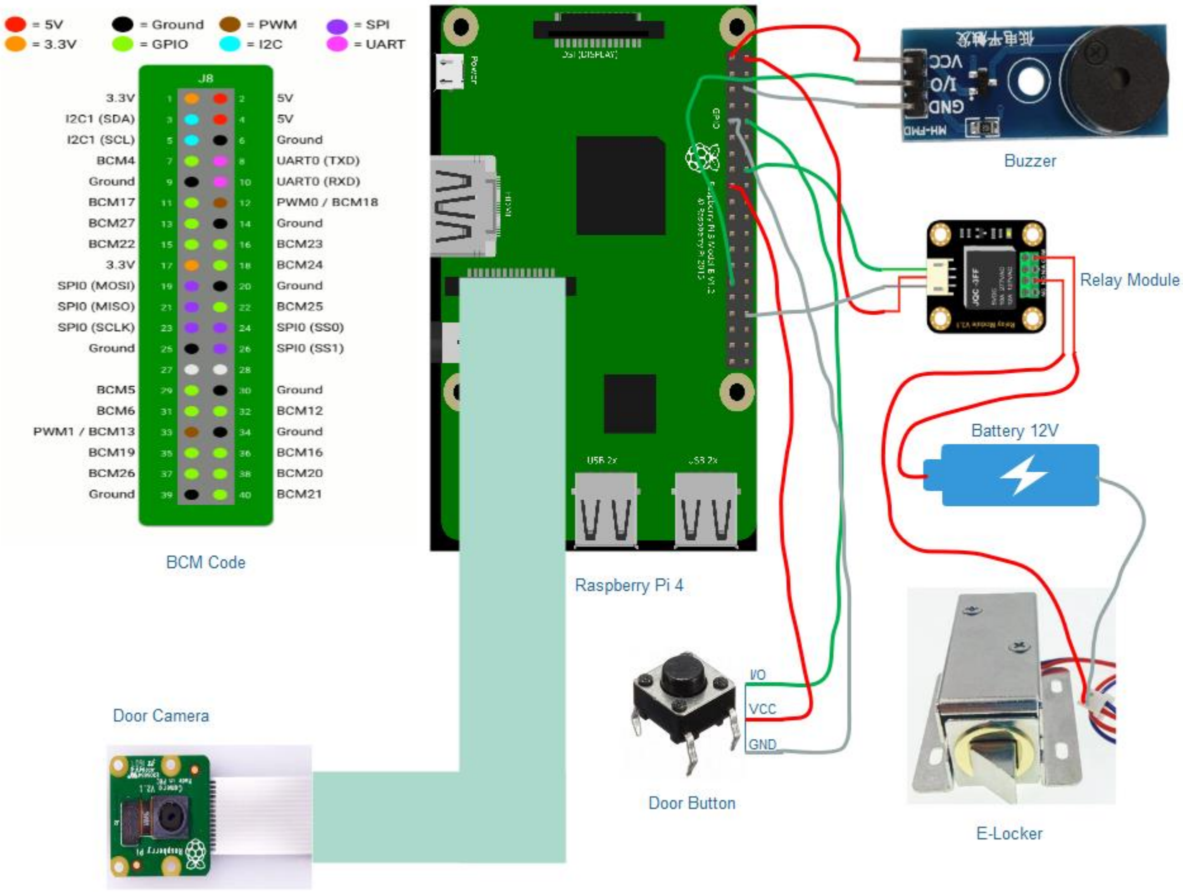


*Figure 49 Door Access Hardware Design*

1. Raspberry Pi and Buzzer (Doorbell)

| Raspberry Pi | Buzzer (Doorbell) | Wire Colour |
|---|---|---|
| 3.3v Pin | VCC Pin | Red |
| GND Pin | GND Pin | Grey |
| BCM 5 | I/O Pin | Green |

2. Raspberry Pi and Doorbell Button

| Raspberry Pi | Doorbell Button | Wire Colour |
|---|---|---|
| 3.3v Pin | VCC Pin | Red |
| GND Pin | GND Pin | Grey |
| BCM 15 (RXD) | I/O Pin | Green |

3. Raspberry Pi and Relay Module

| Raspberry Pi | Relay Module | Wire Colour |
|---|---|---|
| 5v Pin | Positive Pin | Red |
| GND Pin | Negative Pin | Grey |
| BCM 23 | I/O Pin | Green |

4. Relay Module, Lock and Battery

| **Relay Module** | **Lock** | **Battery** | **Wire Colour** |
|---|---|---|---|
| Normal Open Pin | Positive of Charge | *Not Used* | Red |
| COM Pin | *Not Used* | Positive | Red |

5. Raspberry Pi and Raspberry Camera

| **Raspberry Pi** | **Raspberry Camera** | **Wire Colour** |
|---|---|---|
| Raspberry Camera Interface | Camera Cable | Green |

### 5.2.2.4. Device Control Script

There are Python scripts implements to control the Door Access through the Raspberry Pi. The Python script is described using class diagram below.

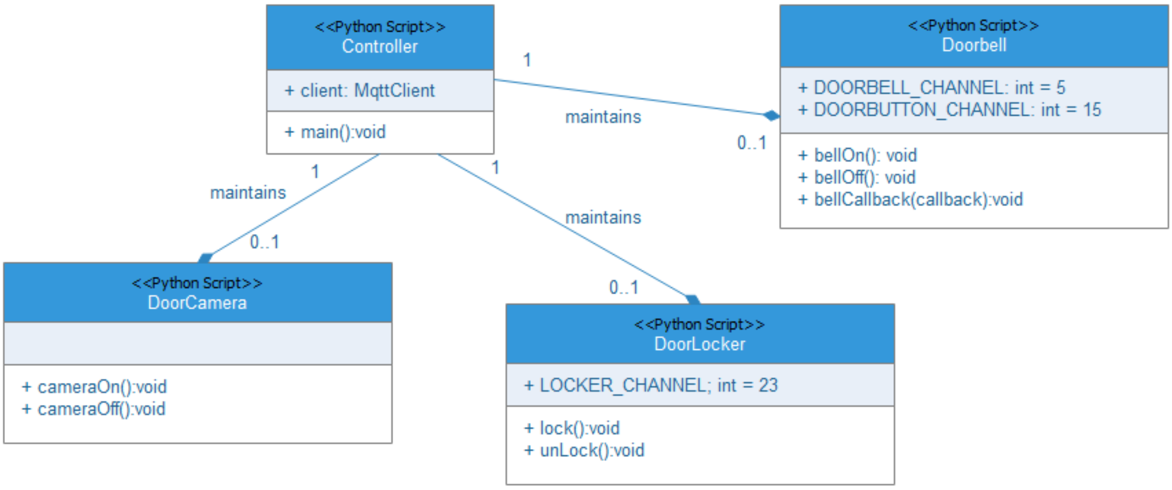


*Figure 50 Door Access Python Scripts Class Diagram*

### 5.2.2.5. Functional Description

In this section, it will describe that functions design of the Door Access in detail.

#### 5.2.2.5.1. Doorbell On/Off

- *Description*

The visitor could press the doorbell button on the door to enable the doorbell beep that can notify the householder to hear it.

- *Key Parameters*

| Device | Role | Action | Method |
|---|---|---|---|
| Doorbell Button | sender | Press the doorbell button | |
| Raspberry Pi | receiver | *Notify the doorbell beep* | *bellOn()* |
| Doorbell Button | sender | Release the doorbell button | |

|  |  |  |  |
|---|---|---|---|
| Raspberry Pi | receiver | *Notify the doorbell quiet* | *bellOff()* |

- *Sequence Diagram*

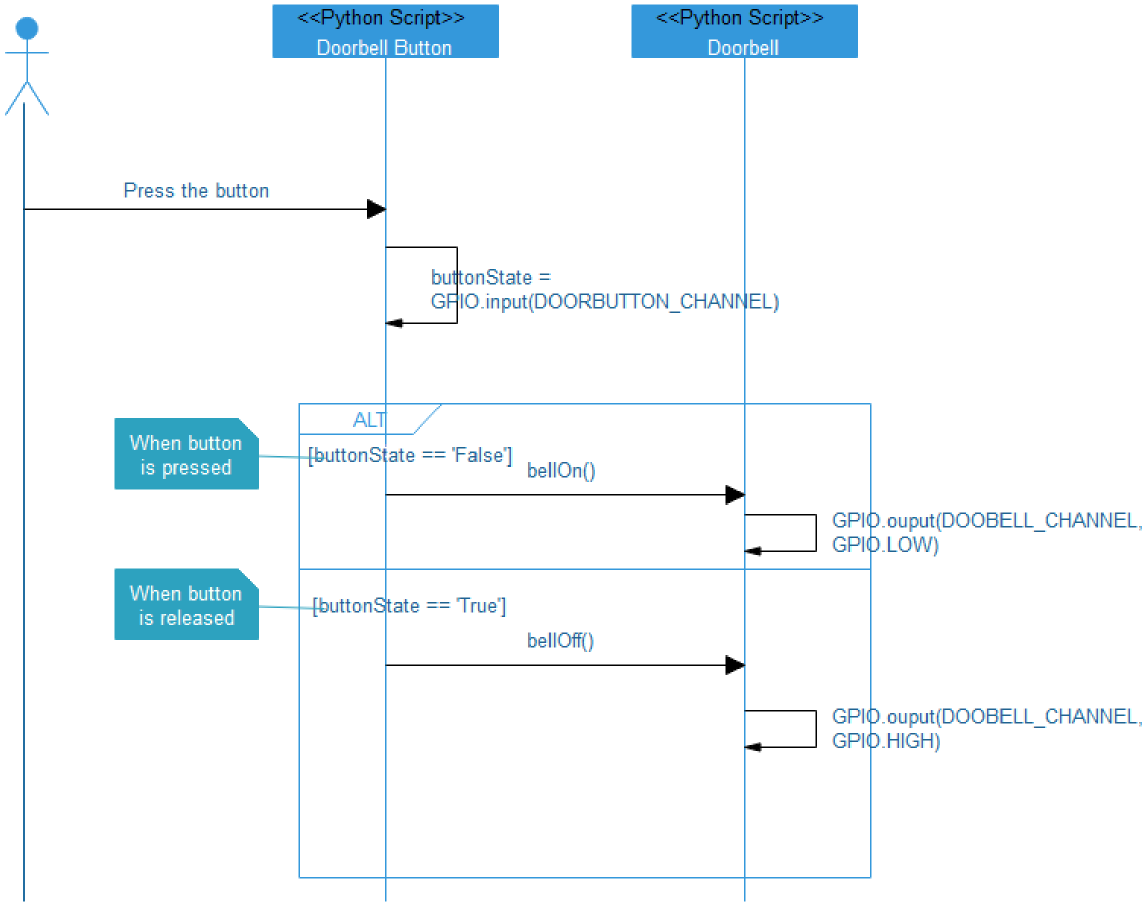


*Figure 51 Doorbell Sequence Diagram*

### 5.2.2.5.2. E-Lock Locked/Unlocked

- *Description*

When the householder hears the doorbell is beeping, the householder could click the lock button on the mobile phone screen to remotely control the door lock to unlock and lock through the local network.

- *Key Parameters*

| MQTT Client | Role | Action | Network | Method | Topic Name | Message | Return Value |
|---|---|---|---|---|---|---|---|
| Flutter App | sender | Click lock button on | Wi-Fi | publish | topic/door/lock/control | on | *[not applicate]* |
| Raspberry Pi | receiver | *[not applicate]* |  | subscribe | topic/door/lock/control | *[not applicate]* | on |
| Flutter App | sender | Click lock button off |  | publish | topic/door/lock/control | off | *not applicate* |
| Raspberry Pi | receiver | *[not applicate]* |  | subscribe | topic/door/lock/control | *[not applicate]* | off |

- *Sequence Diagram*

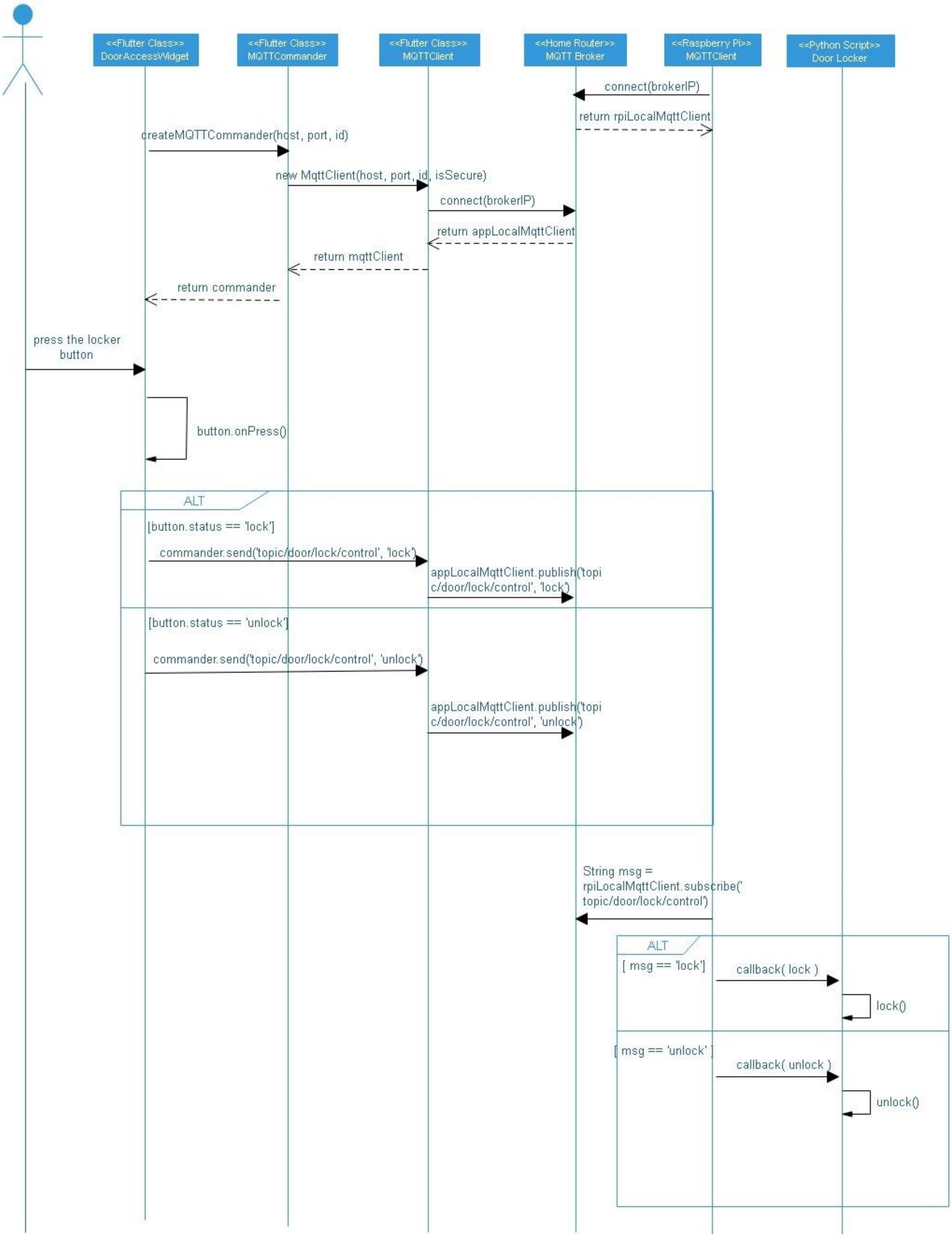

<<Flutter Class>>
DoorAccessWidget
<<Flutter Class>>
MQTTCommander
<<Flutter Class>>
MQTTClient
<<Home Router>>
MQTT Broker
<<Raspberry Pi>>
MQTTClient
<<Python Script>>
Door Locker
connect(brokerIP)
return rpiLocalMqttClient
createMQTTCommander(host, port, id)
new MqttClient(host, port, id, isSecure)
connect(brokerIP)
return appLocalMqttClient
return mqttClient
return commander
press the locker button
button.onPress()
ALT
[button.status == 'lock']
commander.send('topic/door/lock/control', 'lock')
appLocalMqttClient.publish('topic/door/lock/control', 'lock')
[button.status == 'unlock']
commander.send('topic/door/lock/control', 'unlock')
appLocalMqttClient.publish('topic/door/lock/control', 'unlock')
String msg = rpiLocalMqttClient.subscribe('topic/door/lock/control')
ALT
[ msg == 'lock']
callback( lock )
lock()
[ msg == 'unlock' ]
callback( unlock )
unlock()

*Figure 52 Door Lock Sequence Diagram*

### 5.2.2.5.3. Turn Door Camera On/Off

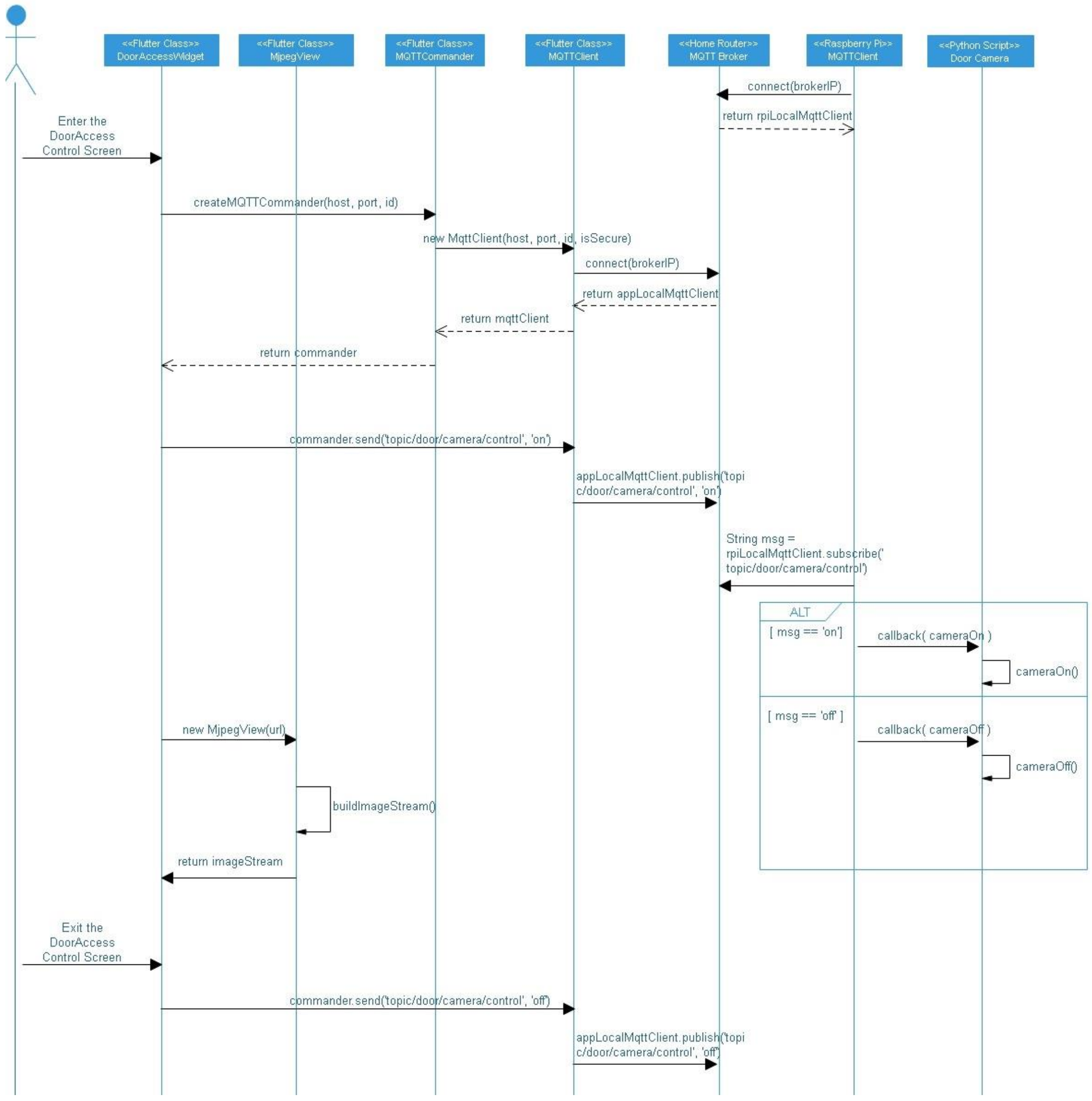
- *Description*

The householder enters the door access control panel screen on the mobile phone that could remotely control the door camera to turn on or off through the local network.

- *Key Parameters*

| MQTT Client | Role | Action | Network | Method | Topic Name | Message | Return Value |
|---|---|---|---|---|---|---|---|
| Flutter App | sender | Enter the door access control panel | Wi-Fi | publish | topic/door/camera/control | on | *[not applicate]* |
| Raspberry Pi | receiver | *[not applicate]* | | subscribe | topic/door/camera/control | *[not applicate]* | on |
| Flutter App | sender | Exit the door access control panel | | publish | topic/door/camera/control | off | *not applicate* |
| Raspberry Pi | receiver | *[not applicate]* | | subscribe | topic/door/camera/control | *[not applicate]* | off |

- *Sequence Diagram*



<<Flutter Class>> DoorAccessWidget
<<Flutter Class>> MjpegView
<<Flutter Class>> MQTTCommander
<<Flutter Class>> MQTTClient
<<Home Router>> MQTT Broker
<<Raspberry Pi>> MQTTClient
<<Python Script>> Door Camera
connect(brokerIP)
return rpiLocalMqttClient
Enter the DoorAccess Control Screen
createMQTTCommander(host, port, id)
new MqttClient(host, port, id, isSecure)
connect(brokerIP)
return appLocalMqttClient
return mqttClient
return commander
commander.send('topic/door/camera/control', 'on')
appLocalMqttClient.publish('topic/door/camera/control', 'on')
String msg = rpiLocalMqttClient.subscribe('topic/door/camera/control')
ALT
[ msg == 'on']
callback( cameraOn )
cameraOn()
[ msg == 'off' ]
callback( cameraOff )
cameraOff()
new MjpegView(url)
buildImageStream()
return imageStream
Exit the DoorAccess Control Screen
commander.send('topic/door/camera/control', 'off')
appLocalMqttClient.publish('topic/door/camera/control', 'off')

*Figure 53 Door Camera Sequence Diagram*

## 5.3. Sensors Control

The Sensor module maintains the sensor status which interacts with the Raspberry Pi. Currently, there are two data from sensors will be delivered to Raspberry Pi that is temperature and humidity both measurements respectively indoor and outdoor.

### 5.3.1. Temperature

The temperature measurement can be read from the temperature sensor by Raspberry Pi then publish it to the MQTT Broker. The mobile app also can subscribe to the message which temperature measurement sent by Raspberry Pi

#### 5.3.1.1. Use Case

The use case diagram below is clearly shown the process of the Temperature Sensor data exchange between the mobile phone, Raspberry Pi.

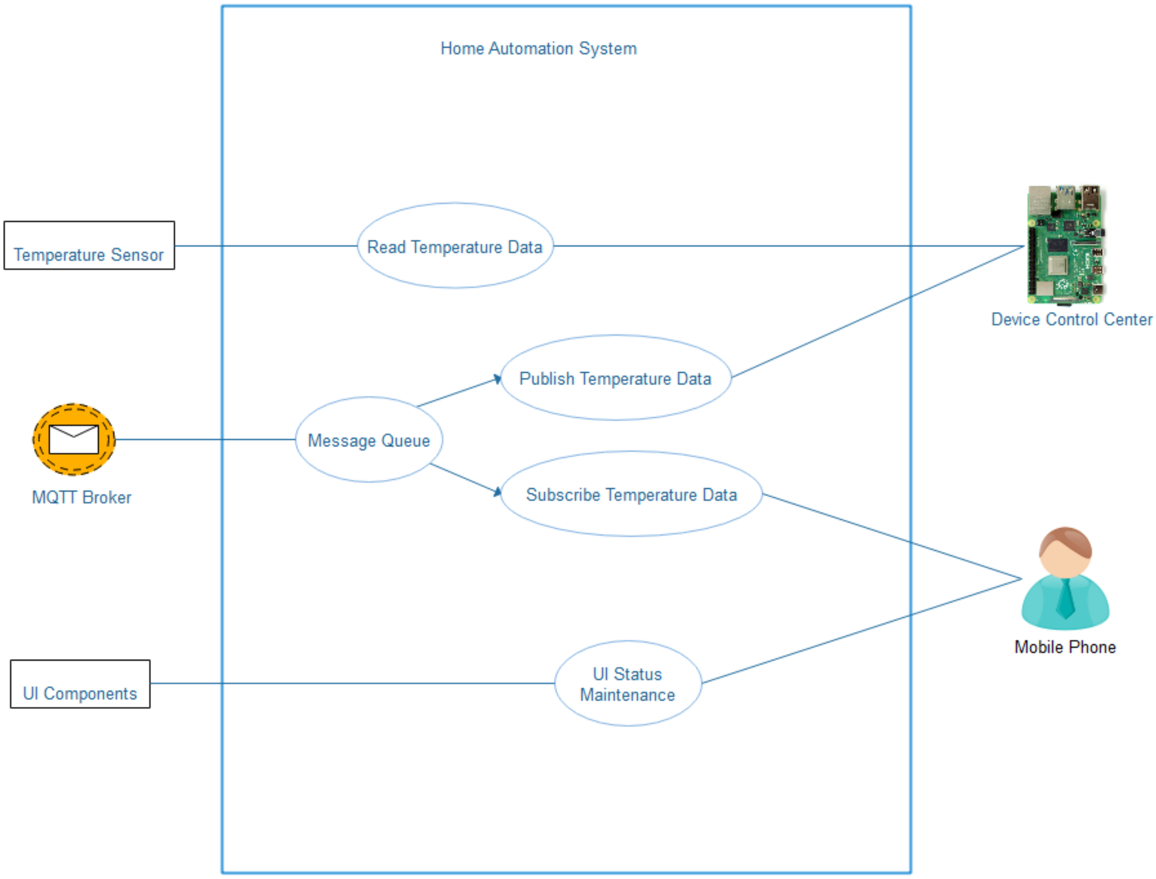


*Figure 54 Temperature Sensor Use Case*

### 5.3.1.2. Class Diagram

The temperature module classes are described using UML class diagram below.

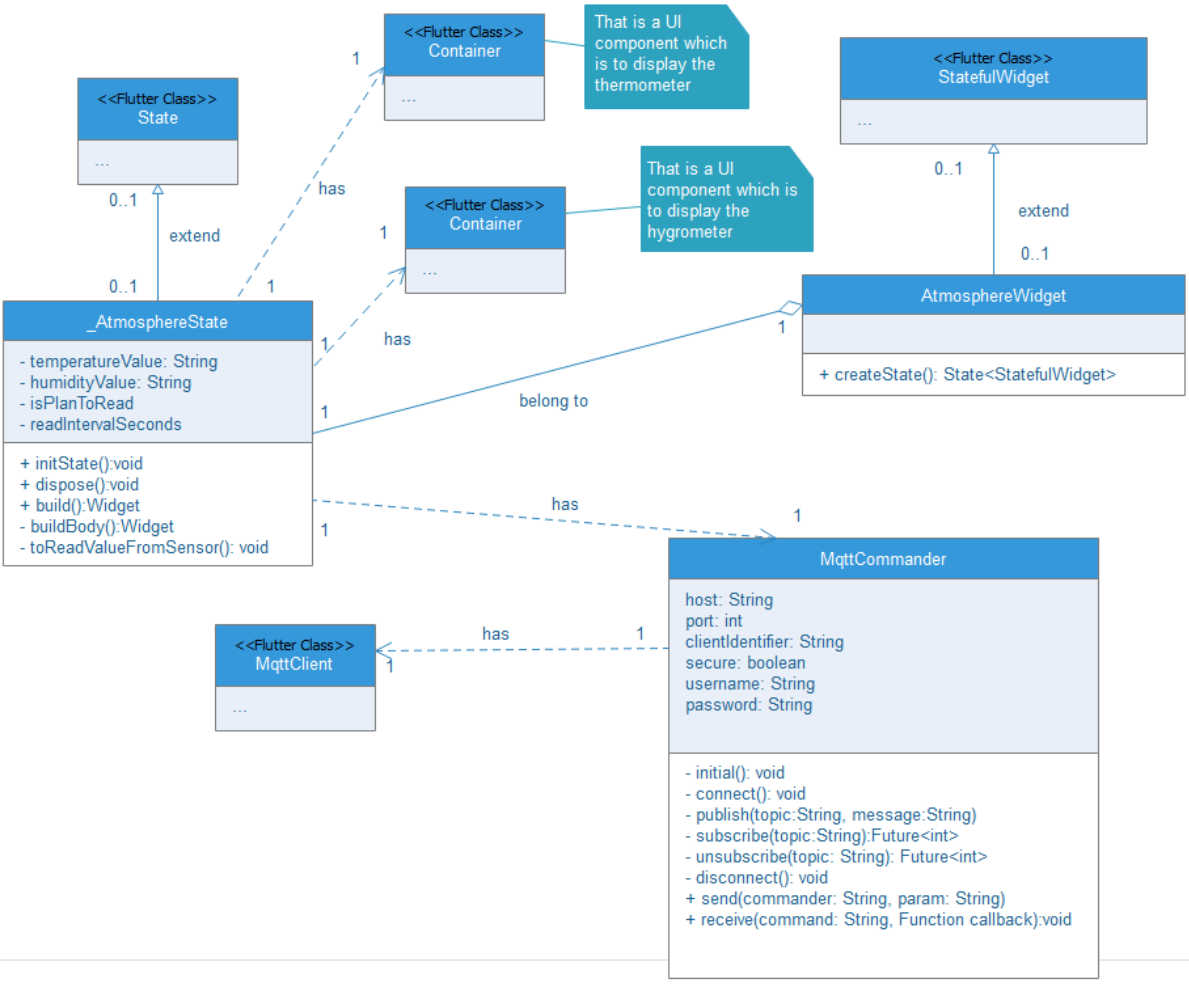


*Figure 55 Temperature & Humidity Sensor Class Diagram*

### 5.3.1.3. Wiring Diagram

The diagram described how the components connect to each other. There is useful information provided below.

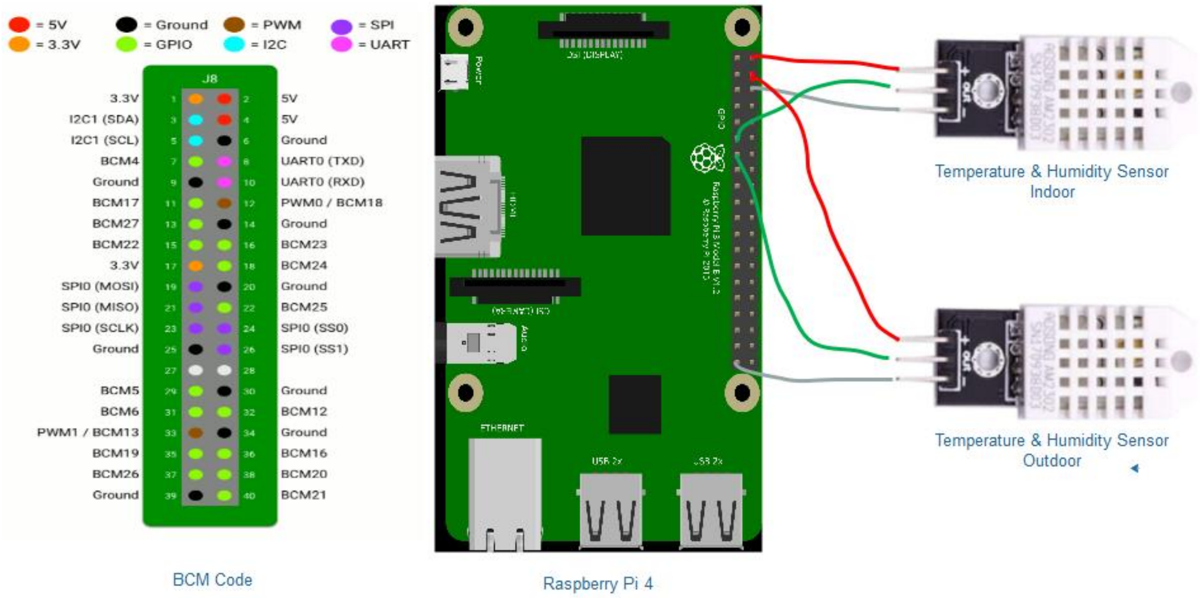


*Figure 56 Temperature & Humidity Sensor Hardware Design*

1. Raspberry Pi and Indoor Sensor

| **Raspberry Pi** | **Indoor Sensor** | **Wire Colour** |
|---|---|---|
| 5v Pin | Positive Pin | Red |
| GND Pin | Negative Pin | Grey |
| BCM 17 | I/O Pin | Green |

2. Raspberry Pi and Outdoor Sensor

| **Raspberry Pi** | **Indoor Sensor** | **Wire Colour** |
|---|---|---|
| 5v Pin | Positive Pin | Red |
| GND Pin | Negative Pin | Grey |
| BCM 27 | I/O Pin | Green |

### 5.3.1.4. Device Control Script

There are Python scripts implements to control the Temperature Sensor through the Raspberry Pi. The Python script is described using class diagram below.

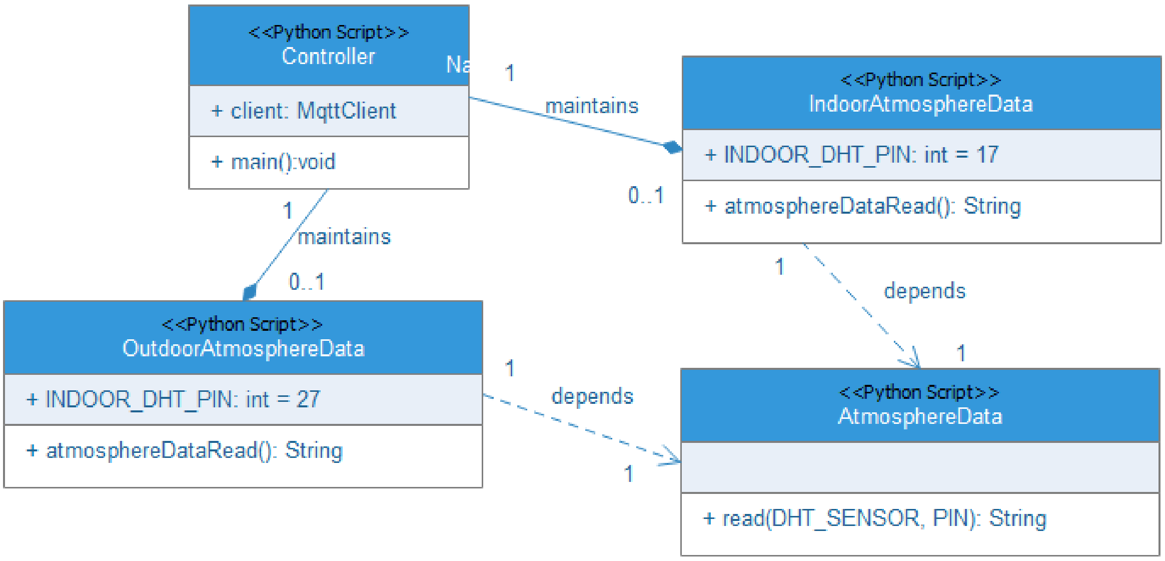


*Figure 57 Temperature & Humidity Sensor Scripts Class Diagram*

### 5.3.1.5. Functional Description

In this section, it will describe that functions design of the Temperature Sensor in detail.

#### 5.3.1.5.1. Read Indoor Temperature Measurement

- *Description*

The householder enters the temperature panel screen on the mobile phone that should display the indoor temperature measurement in real-time.

- *Key Parameters*

| MQTT Client | Role | Action | Network | Method | Topic Name | Message | Return Value |
|---|---|---|---|---|---|---|---|
| Flutter App | sender | Enter the temperature screen | Wi-Fi | publish | topic/indoor/measurement/read | on | *[not applicate]* |
| Raspberry Pi | receiver | *[not applicate]* | | subscribe | topic/indoor/measurement/read | *[not applicate]* | on |
| Raspberry Pi | sender | Read the measurement from sensor | | publish | topic/indoor/measurement/data | measurement | *[not applicate]* |
| Raspberry Pi | receiver | *[not applicate]* | | subscribe | topic/indoor/measurement/data | *[not applicate]* | measurement |
| Flutter App | sender | Click lock button off | | publish | topic/indoor/measurement/read | off | *[not applicate]* |
| Raspberry Pi | receiver | *[not applicate]* | | subscribe | topic/indoor/measurement/read | *[not applicate]* | off |

Note: when the Raspberry Pi received the message of the topic 'topic/indoor/measurement/read' is 'on', then the Raspberry Pi will continuously publish the data to the topic 'topic/indoor/measurement/data' interval 1 second until it received the message which is 'off' of the topic 'topic/indoor/measurement/read'.

- *Sequence Diagram*

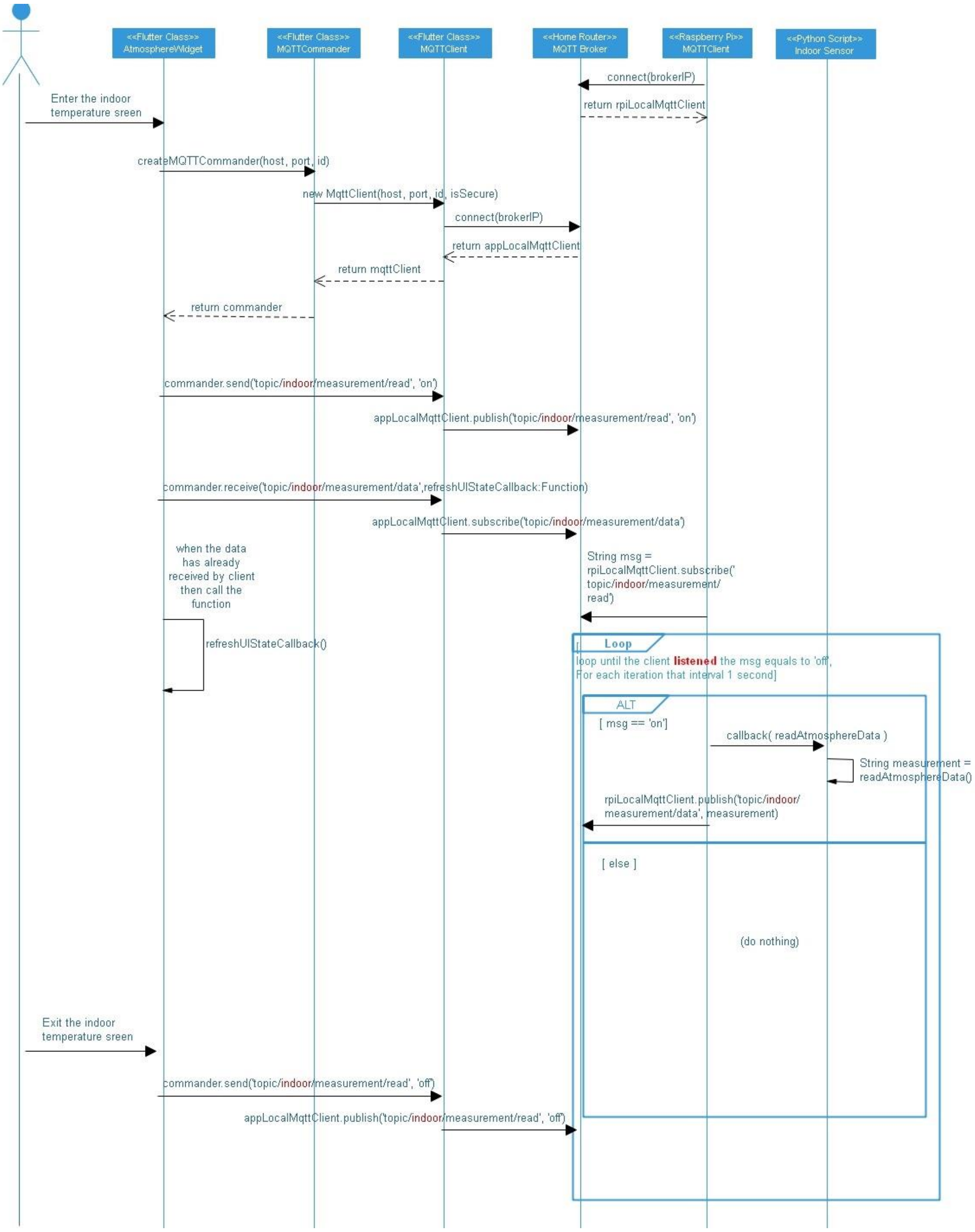

<<Flutter Class>>
AtmosphereWidget
<<Flutter Class>>
MQTTCommander
<<Flutter Class>>
MQTTClient
<<Home Router>>
MQTT Broker
<<Raspberry Pi>>
MQTTClient
<<Python Script>>
Indoor Sensor
connect(brokerIP)
return rpiLocalMqttClient
Enter the indoor temperature sreen
createMQTTCommander(host, port, id)
new MqttClient(host, port, id, isSecure)
connect(brokerIP)
return appLocalMqttClient
return mqttClient
return commander
commander.send('topic/indoor/measurement/read', 'on')
appLocalMqttClient.publish('topic/indoor/measurement/read', 'on')
commander.receive('topic/indoor/measurement/data',refreshUIStateCallback:Function)
appLocalMqttClient.subscribe('topic/indoor/measurement/data')
when the data has already received by client then call the function
String msg = rpiLocalMqttClient.subscribe('topic/indoor/measurement/read')
refreshUIStateCallback()
Loop
[loop until the client listened the msg equals to 'off',
For each iteration that interval 1 second]
ALT
[ msg == 'on']
callback( readAtmosphereData )
String measurement = readAtmosphereData()
rpiLocalMqttClient.publish('topic/indoor/measurement/data', measurement)
[ else ]
(do nothing)
Exit the indoor temperature sreen
commander.send('topic/indoor/measurement/read', 'off')
appLocalMqttClient.publish('topic/indoor/measurement/read', 'off')

*Figure 58 Indoor Temperature & Humidity Sensor Sequence Diagram*

### 5.3.1.5.2. Read Outdoor Temperature Measurement

- *Description*

The householder enters the temperature panel screen on the mobile phone that should display the outdoor temperature measurement in real-time.

- *Key Parameters*

| MQTT Client | Role | Action | Network | Method | Topic Name | Message | Return Value |
|---|---|---|---|---|---|---|---|
| Flutter App | sender | Enter the temperature screen | Wi-Fi | publish | topic/outdoor/measurement/read | on | *[not applicate]* |
| Raspberry Pi | receiver | *[not applicate]* | | subscribe | topic/outdoor/measurement/read | *[not applicate]* | on |
| Raspberry Pi | sender | Read the measurement from sensor | | publish | topic/outdoor/measurement/data | measurement | *[not applicate]* |
| Raspberry Pi | receiver | *[not applicate]* | | subscribe | topic/outdoor/measurement/data | *[not applicate]* | measurement |
| Flutter App | sender | Exit the temperature screen | | publish | topic/outdoor/measurement/read | off | *[not applicate]* |
| Raspberry Pi | receiver | *[not applicate]* | | subscribe | topic/outdoor/measurement/read | *[not applicate]* | off |

Note: when the Raspberry Pi received the message of the topic 'topic/outdoor/measurement/read' is 'on', then the Raspberry Pi will continuously publish the data to the topic 'topic/outdoor/measurement/data' interval 1 second until it received the message which is 'off' of the topic 'topic/outdoor/measurement/read'.

- *Sequence Diagram*

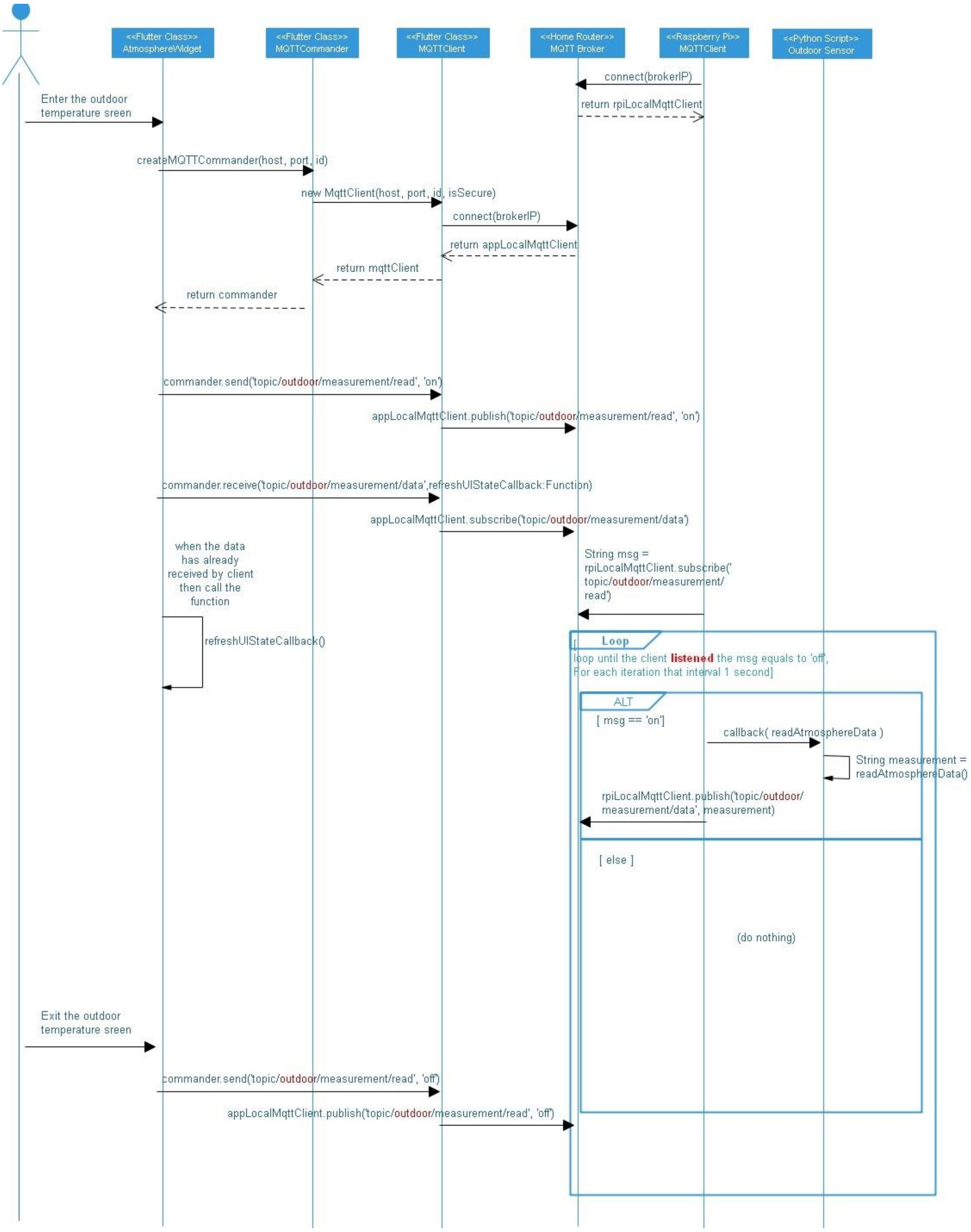

<<Flutter Class>>
AtmosphereWidget
<<Flutter Class>>
MQTTCommander
<<Flutter Class>>
MQTTClient
<<Home Router>>
MQTT Broker
<<Raspberry Pi>>
MQTTClient
<<Python Script>>
Outdoor Sensor
connect(brokerIP)
Enter the outdoor temperature sreen
return rpiLocalMqttClient
createMQTTCommander(host, port, id)
new MqttClient(host, port, id, isSecure)
connect(brokerIP)
return appLocalMqttClient
return mqttClient
return commander
commander.send('topic/outdoor/measurement/read', 'on')
appLocalMqttClient.publish('topic/outdoor/measurement/read', 'on')
commander.receive('topic/outdoor/measurement/data',refreshUIStateCallback:Function)
appLocalMqttClient.subscribe('topic/outdoor/measurement/data')
when the data has already received by client then call the function
String msg = rpiLocalMqttClient.subscribe('topic/outdoor/measurement/read')
refreshUIStateCallback()
Loop
[loop until the client listened the msg equals to 'off', For each iteration that interval 1 second]
ALT
[ msg == 'on']
callback( readAtmosphereData )
String measurement = readAtmosphereData()
rpiLocalMqttClient.publish('topic/outdoor/measurement/data', measurement)
[ else ]
(do nothing)
Exit the outdoor temperature sreen
commander.send('topic/outdoor/measurement/read', 'off')
appLocalMqttClient.publish('topic/outdoor/measurement/read', 'off')

*Figure 59 Outdoor Temperature & Humidity Sensor Sequence Diagram*

### 5.3.2. Humidity

The humidity measurement can be read from the humidity sensor by Raspberry Pi then publish it to the MQTT Broker. The mobile app also can subscribe to the message which humidity measurement sent by Raspberry Pi

#### 5.3.2.1. Use Case

The use case diagram below is clearly shown the process of the Humidity Sensor data exchange between the mobile phone, Raspberry Pi.

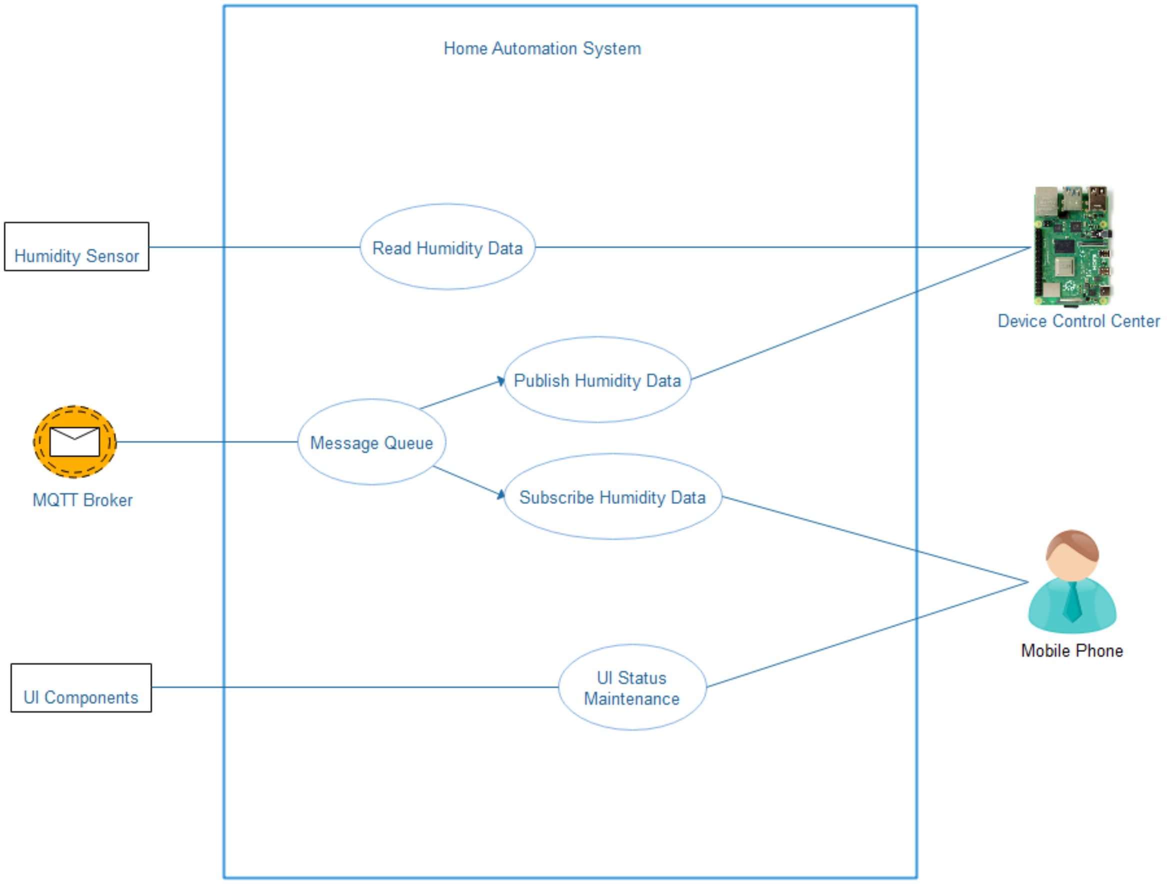


*Figure 60 Humidity Sensor Use Case*

#### 5.3.2.2. Class Diagram

The temperature module classes are described using UML class diagram that is the same to temperature module classes diagram.

Please back to [Temperature Class Diagram]

#### 5.3.2.3. Wiring Diagram

The diagram described how the components connect to each other. There is useful information provided is the same to Temperature module's wiring diagram.

Please back to [Temperature Wiring Diagram]

#### 5.3.2.4. Device Control Script

There are Python scripts implements to control the Temperature Sensor through the Raspberry Pi. The Python script is described using class diagram that is the same to Temperature's.

Please back to Temperature Device Control Script

#### 5.3.2.5. Functional Description

In this section, it will describe that functions design of the Humidity Sensor in detail.

##### 5.3.2.5.1. Read Indoor Humidity Measurement

The householder enters the humidity panel screen on the mobile phone that should display the indoor humidity measurement in real-time. The detail design which is the same to Temperature module.

Please back to Read Indoor Temperature Measurement

##### 5.3.2.5.2. Read Outdoor Humidity Measurement

The householder enters the humidity panel screen on the mobile phone that should display the outdoor humidity measurement in real-time. The detail design which is the same to Temperature module.

Please back to Read Outdoor Temperature Measurement