



I.I.S. POLO TECNOLOGICO IMPERIESE

I.T.I. "G. Galilei" - Imperia

Corso di Studi in Informatica & Telecomunicazioni

Articolazione: Telecomunicazioni

ELABORATO ESAME DI STATO

A.S. 2020/2021

*“My IoT Home V2.0”*

Candidato

MIHAIL DIMITRIU

Docente Referente

PROF. ZANELLA SIMONE

## ABSTRACT

My IoT Home V2.0 is a highly customizable computer system, which through a Mobile App enables the user to have real-time control of his home. The system involves the use of multiple sensors and actuators connected to WeMos D1 R2 WiFi microcontrollers equipped with the ESP8266 module. These ones, using the MQTT protocol, communicate the detected data to a single board computer: a Raspberry 3B+. The data reached the Raspberry are processed by a Python program that interacts with a Real-Time Database and a Storage, hosted in-cloud by the Firebase platform. The data detected and any photos and files are displayed through the dashboard of a Mobile App programmed in Flutter. The same app is able to report alarms thanks to the integration of Push Notification and the FCM technology. Every detail of this product has been developed to ensure the reliability of the system and a pleasant user experience.

**Keyword :** *Arduino, ESP8266, Mqtt, Raspberry, Firebase, Real Time Database*

My IoT Home V2.0 è un sistema informatico altamente customizzabile, che tramite Mobile App abilita l'utente ad avere il controllo in tempo reale della sua abitazione. Il sistema prevede l'impegno di più sensori e attuatori connessi a dei microcontrollori WeMos D1 R2 WiFi dotati del modulo ESP8266. Questi ultimi, mediante protocollo MQTT, comunicano i dati rilevati ad un single board computer: un Raspberry 3B+. I dati giunti al Raspberry vengono elaborati da un programma in Python che interagisce con un Real Time Database ed uno Storage, ospitati sulla piattaforma in-cloud Firebase. I dati rilevati ed eventuali foto e file, vengono visualizzati attraverso la dashboard di una Mobile App programmata in Flutter. La medesima app è in grado di segnalare allarmi grazie all'integrazione delle Push Notification e alla tecnologia FCM. Ogni dettaglio di questo prodotto è stato sviluppato al fine garantire l'affidabilità del sistema e di assicurare un'esperienza utente gradevole.

**Parole chiave :** *Arduino, ESP8266, Mqtt, Raspberry, Firebase, Real Time Database*

# ANALISI DI MERCATO

Negli ultimi anni è avvenuta un'importante innovazione a livello tecnologico ed oggigiorno smartphone, computer ed altri dispositivi offrono un costante collegamento ad Internet, ad un costo sempre più accessibile. Questo processo ha permesso lo sviluppo di nuove tecnologie e device, che sempre più si riconducono al campo dell'IoT (acronimo di *Internet of Thing*).

L'IoT definisce una rete di oggetti fisici, per l'appunto "things", integrati con sensori, software e altre tecnologie allo scopo di connettere e scambiare dati con altri dispositivi e sistemi connessi a Internet [23]. Per citare un esempio, un tipico sistema IoT è la "Smart Home" che abilita l'utente ad avere il controllo, anche remoto, sulla sua abitazione.

Per comprendere quanti sono i dispositivi IoT connessi ad Internet e quali sono le previsioni per il futuro, è possibile fare riferimento al seguente rapporto di novembre 2020 [14], stilato da IoT Analytics, azienda leader nel settore tecnologico e dell'analisi statistica.

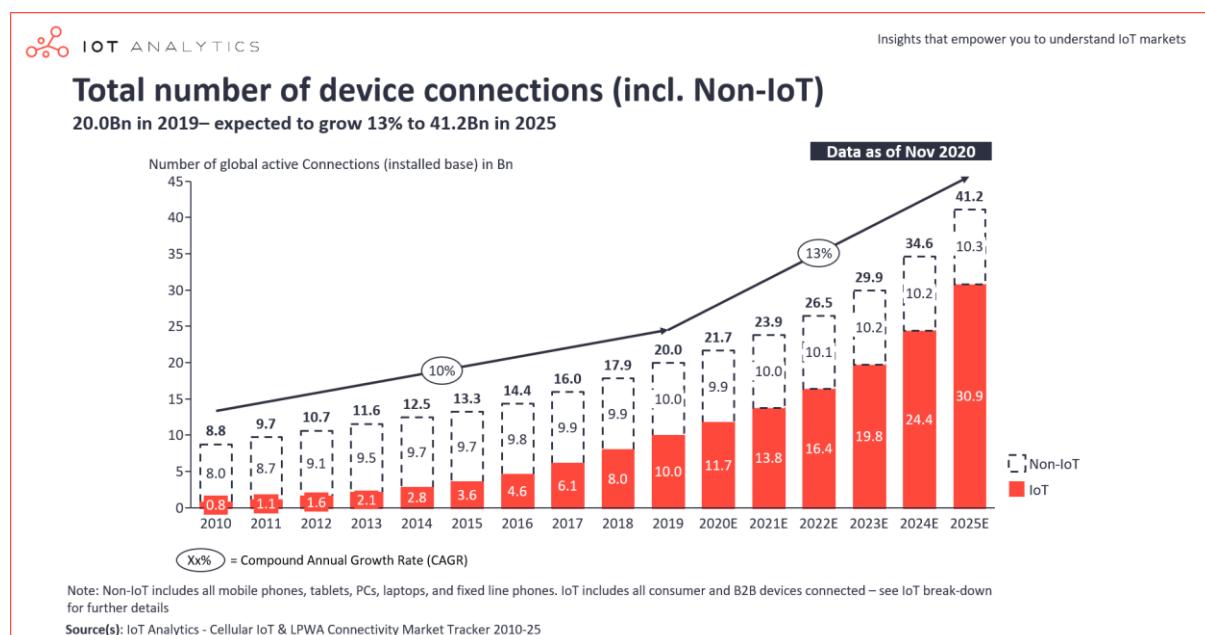


Grafico 1: Dispositivi IoT connessi e previsioni future

Il rapporto di IoT Analytics ha stimato, ad oggi, il raggiungimento di 12 miliardi di dispositivi IoT, che per la prima volta hanno superato il numero di dispositivi connessi ad Internet non-IoT.

Queste previsioni fanno ben sperare nel mercato delle tecnologie IoT e rappresentano un buon indice per le aziende e i privati che investono tempo e risorse nello sviluppo degli smart systems e delle loro integrazioni.

Nondimeno, è doveroso ricordare l'importanza di garantire che questi sistemi siano affidabili e al contempo sicuri, aspetti che richiedono attenta progettazione, studio e fasi di test.

# INDICE

ABSTRACT .....	2
ANALISI DI MERCATO .....	3
1. BACKGROUND UTILE ALLE SCELTE PROGETTUALI .....	1
1.1. MY IOT HOME V1.0.....	1
1.2. PROGETTO PCTO: SISTEMI EMBEDDED .....	3
1.3. TIROCINIO PRESSO DRAFINTECH SRL.....	3
2. GESTIONE DELLE FASI DI PROGETTO .....	4
2.1. GANTT CHART: FASI DI PROGETTO E MILESTONES.....	4
2.2. VCS: GIT CON GITHUB PRIVATE REPOSITORY .....	5
2.3. BACKUP MANUALI .....	6
2.4. SVILUPPO E TESTING.....	7
3. IL PROGETTO MY IOT HOME V2.0 .....	9
3.1. I REQUISITI STABILITI PER LA PROGETTAZIONE .....	9
3.2. PANORAMICA DEL SISTEMA .....	10
4. DISPOSITIVI HARDWARE .....	12
4.1. ARDUINO WEMOS D1 R2 V2.1.0 WiFi DOTATO DI ESP8266 ..	12
4.2. RASPBERRY Pi 3B+ .....	12
4.3. ALIMENTATORI E POWERBANK .....	13
4.4. SENSORI.....	14
4.4.1 Water Level Sensor .....	14
4.4.2 PIR Sensor HC-SR501.....	15
4.4.3 DHT11 Sensor .....	15
4.4.3 Webcam Logitech C120.....	16
4.5. ATTUATORI .....	16
4.5.1 Diodi Led (e dimensionamento dei resistori) .....	17

5. SCHEMI CIRCUITALI E COLLEGAMENTI FISICI .....	18
5.1. Wemos dotato di sensore Water Level .....	18
5.2. Wemos dotato di sensore PIR .....	19
5.3. Raspberry dotato del sensore DHT11 e Webcam .....	20
6. TECNOLOGIE E SOFTWARE IMPIEGATI .....	21
6.1. FASE DI RICERCA: AMAZON AWS E GOOGLE FIREBASE .....	21
6.2. SUITE TCP-IP E PROTOCOLLI DI APPLICAZIONE .....	22
6.2.1 STACK MQTT NELLA LAN .....	22
6.2.2 STACK HTTP/HTTPS PER I SERVIZI ONLINE .....	24
6.3. THINGSPEAK.COM .....	25
6.4. HEALTHCHECKS.IO .....	26
6.5. RealVNC .....	26
6.6. PYTHON .....	27
6.7. FLUTTER, BY GOOGLE .....	27
7. REALIZZAZIONE E SVILUPPO .....	28
7.1. PROGRAMMI ARDUINO .....	28
7.1.1 CODICE DEL WEMOS DOTATO DI SENSORE PIR .....	28
7.1.2 CODICE DEL WEMOS DOTATO DI SENSORE WATER LEVEL E LED .....	30
7.2. PROGRAMMA IN PYTHON ED INTEGRAZIONI API .....	31
7.2.1 SCHEMA A BLOCCHI DEL PROGRAMMA IN PYTHON .....	32
7.2.2 MODULI PYREBASE, PYFCM E FIREBASE PLATFORM .....	33
7.2.3 MODULO MQTT PHAO .....	35
7.2.4 MODULO Adafruit_DHT .....	37
7.2.5 FUNZIONE DI TRIGGER .....	38
7.2.6 FUNZIONI DI AUTODIAGNOSI .....	39
7.2.7 FUNZIONE DI LOG .....	40
7.3. AUTOMAZIONE DEL SISTEMA .....	42
7.4. FUNZIONE DI HEARTBEAT .....	43

7.5. MOBILE APP.....	45
7.5.1 SCHERMATA DELLA DASHBOARD .....	45
7.5.2 SCHERMATE DEI GRAFICI DI THINGSPEAK .....	46
7.5.3 SCHERMATA DELLE FOTO CATTURATE DALLA WEBCAM	47
7.5.4 MENU' LATERALE .....	48
7.5.5 SCHERMATA DEI LOG .....	48
7.5.6 SCHERMATA DELLE IMPOSTAZIONI.....	49
7.5.7 SERVIZIO DI NOTIFICHE PUSH MEDIANTE FCM .....	50
8. PENETRATION TEST E MESSA IN SICUREZZA.....	51
8.1. SCENARIO.....	51
8.2. VULNERABILITÀ DELLA CONFIGURAZIONE MQTT .....	51
8.3. ANALISI WIRESHARK SU RASPBERRY.....	52
8.4. MAN IN THE MIDDLE CON KALI OS.....	53
8.5. WIFI PASSWORD CRACKING.....	54
8.6. PROBLEMATICHE RISCONTRATE.....	55
8.7. SOLUZIONI ADOTTATE E ACCORGIMENTI FUTURI .....	56
8.8. SCELTA DI PASSWORD AD ALTO VALORE DI ENTROPIA ....	57
8.9. IL FATTORE UMANO ED IL SISTEMA.....	59
9. PIANO DI INDIRIZZAMENTO.....	61
9.1. CABLAGGIO STRUTTURATO .....	61
9.2. PROPOSTA DI INDIRIZZAMENTO .....	61
10. MESSA IN FUNZIONE SUL CAMPO .....	63
10.1. CORRETTA CONFIGURAZIONE DELL'AP .....	64
10.2. POSSIBILE PROBLEMATICA: COPERTURA DELLA WLAN.	66
10.3. POSSIBILI SOLUZIONI .....	66
11. IMPLEMENTAZIONI FUTURE .....	67
11.1. SENSORI E ATTUATORI FISICI .....	67
11.2. SEGNALE GPS COME TRIGGER .....	68
11.3. TEXT TO SPEECH E ASSISTENTI VOCALI.....	68

11.4.	AUTENTICAZIONE TRAMITE FIREBASE AUTH.....	68
11.5.	HEARTBEAT CON FCM E LINUX VM.....	69
11.6.	WEB APP .....	69
12.	NORMATIVE, STANDARD E LICENZE.....	70
12.1.	NORMATIVE SULLA PRIVACY: GPDR UE 2016/679 E DLGS 196/2003 30 GIUGNO 2003 n.196.....	70
12.2.	DISTRIBUZIONE SU PLAY STORE E APP STORE.....	70
12.3.	MODIFICA DEL PROGRAMMA “ADAFRUIT MQTT” .....	71
12.4.	FLUTTER E PACKAGES .....	71
12.5.	NORMATIVA EUROPEA RELATIVA AL CABLAGGIO STRUTTURATO: CABLAGGIO EN50175 .....	72
13.	ASSISTENZA CLIENTI E AUDITING .....	73
13.1.	COME RISALIRE AL PROBLEMA .....	73
13.2.	AGGIORNAMENTI PERIODICI DEL SISTEMA .....	73
14.	MARKETING, INVESTIMENTI E PREVENTIVI.....	74
14.1.	LOGO, SITO WEB E VIDEO ILLUSTRATIVO .....	74
14.2.	OPEN SOURCE E CODECANYON .....	75
14.3.	VALUTAZIONE DEL PREVENTIVO.....	75
	CONCLUSIONI.....	76
	ALLEGATI .....	77
	INDICE DELLE FIGURE .....	80
	INDICE DEI GRAFICI.....	83
	INDICE DELLE TABELLE .....	84
	BIBLIOGRAFIA E SITOGRAFIA.....	85
	RINGRAZIAMENTI .....	89

# 1. BACKGROUND UTILE ALLE SCELTE PROGETTUALI

Diverse delle scelte progettuali di My IoT Home V2.0 sono state prese basandomi su esperienze pregresse. Nello specifico, My IoT Home V2.0 si basa su una prima versione che è stata sviluppata nell'estate 2019. Alcune idee per la seconda versione provengono dalle attività di PCTO svolte nell'ambito scolastico. Altre ancora scaturiscono dal tirocinio svolto presso Drafintech srl nell'estate.

## 1.1. MY IOT HOME V1.0

My IoT Home V1.0 è un progetto che è stato realizzato nell'estate 2019 ed è presentato attraverso un video di digital story telling pubblicato su YouTube (<https://youtu.be/B1fcfAcfR64>):

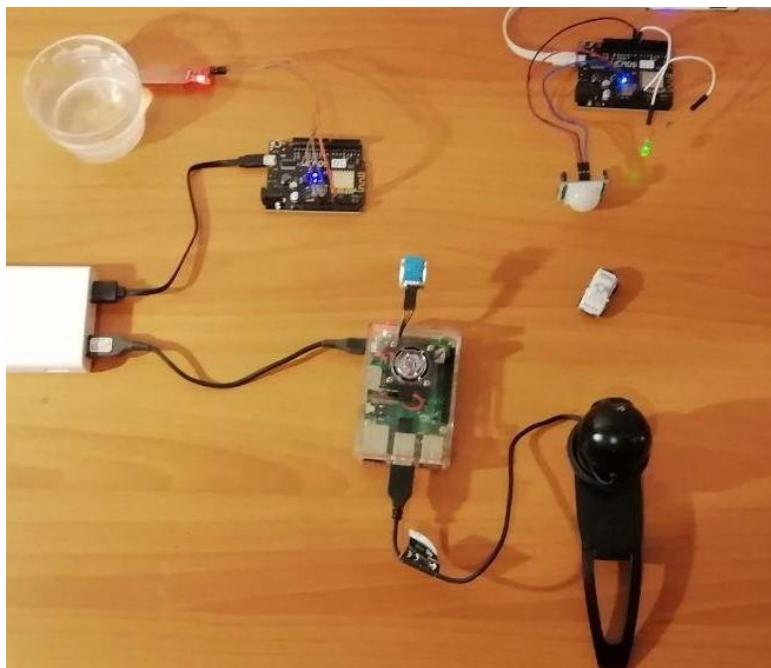


Figura 1: Progetto My IoT Home V1.0

Il sistema prevede:

1. Il campionamento dati attraverso un sensore di movimento PIR (acronimo di Passive InfraRed) ed un sensore Water Level gestiti da due microcontrollori Wemos D1 R2 dotati di modulo ESP8266.
2. La comunicazione wireless tramite protocollo MQTT tra i due microcontrollori ed un Raspberry.
3. L'elaborazione dati svolta da un programma in Python sul Raspberry, che aggiunge i dati rilevati da un sensore di temperatura DHT11 connesso al Raspberry e che talvolta esegue delle istantanee attraverso la webcam quando il sensore PIR rileva un movimento.
4. L'invio di foto e messaggi di avvertimento ad un bot Telegram, eseguito dal medesimo programma in Python.



Figura 2: Bot Telegram di My IoT Home V1.0

## **1.2. PROGETTO PCTO: SISTEMI EMBEDDED**

Il progetto “Sistemi Embedded”, svolto in ambito scolastico fin dal 2019, prevede l'utilizzo del Raspberry al fine di realizzare un gestore di carte per bar che ha come obiettivo la fidelizzazione del cliente.

Alcune funzioni implementate su My IoT Home V2.0 che prendono spunto dal progetto sono la funzione di backup automatizzati e l'avvio automatico del programma principale che avviene inserendo un file con estensione “.desktop” nella cartella *home/pi./config/autostart*.

Un altro punto caratterizzante che è stato assimilato dal progetto “Sistemi Embedded” riguarda la user experience e la necessità di rendere l'assistenza cliente il più immediata possibile. Questo ultimo punto è stato integrato nella Mobile App nella relativa schermata delle impostazioni.

## **1.3. TIROCINIO PRESSO DRAFINTECH SRL**

Il tirocinio svolto presso Drafintech srl nell'estate 2020 ha richiesto l'impiego di tecnologie lato software che si sono rivelate utili al progetto My IoT Home V2.0.

In particolar modo, una fase del tirocinio era stata dedicata allo sviluppo di Mobile App programmata in Flutter, sfruttando tecnologie in-cloud offerte dalla piattaforma Firebase by Google, quali Firebase FCM per le notifiche Push, Firebase Storage per l'archiviazione di file e Firebase Firestore per l'archiviazione di dati.

È stato anche appreso l'utilizzo di Git, un sistema di controllo di versione del codice o VCS (acronimo di *Version Control System*) comunemente in uso anche a livello aziendale per la gestione del codice.

## 2. GESTIONE DELLE FASI DI PROGETTO

Al fine di raggiungere gli obiettivi preposti sono state messe in atto diverse pratiche di project management, quali la redazione a priori del Gantt chart e delle milestones da raggiungere entro un determinato periodo. Sono stati impiegati anche un sistema di controllo del versionamento del codice o VCS, un sistema di backup e sono state attuate fasi di test sia durante lo sviluppo che a prodotto ultimato.

### 2.1. GANTT CHART: FASI DI PROGETTO E MILESTONES

L'impostazione di un Gantt chart in fase di progettazione attraverso il sito [prod.teamgantt.com](https://prod.teamgantt.com) si è rivelata fondamentale per lo sviluppo del progetto e la sua realizzazione, in quanto il Gantt chart determina l'individuazione dei legami logici presenti tra le attività ed il loro scheduling.

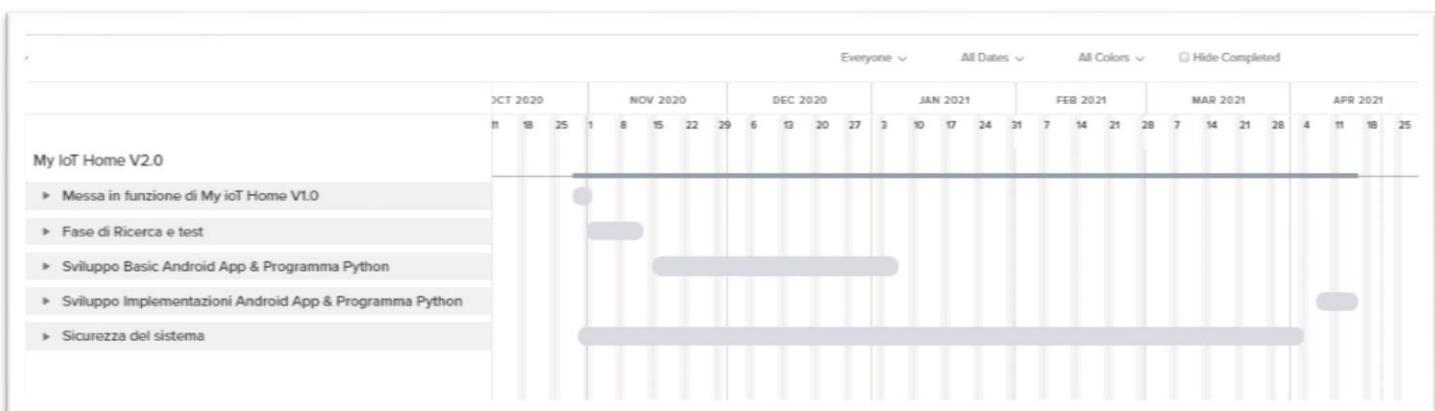
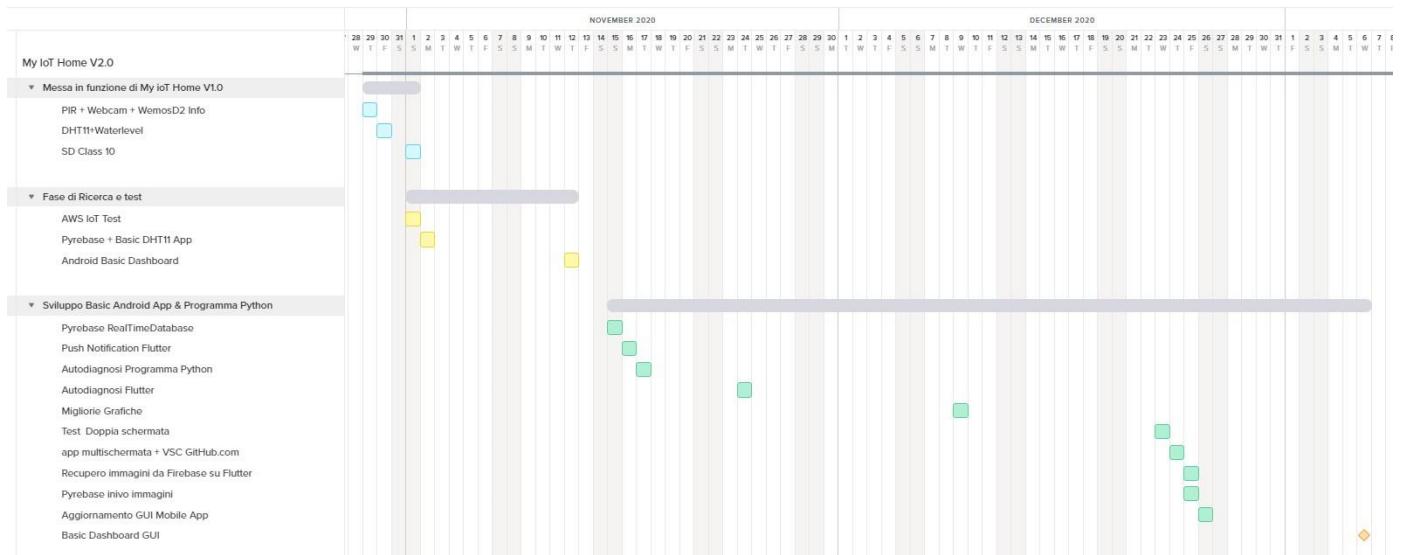


Figura 3: Attività principali del Gantt Chart

In primo luogo sono state impostate le attività e le sottoattività. Seguono le attività principali (alcune delle quali sono state svolte in parallelo):

- La messa in funzione di My IoT Home V1.0
- Fase di ricerca e test rispetto alle soluzioni in cloud per MQTT
- Lo sviluppo del programma in Python e della Mobile App con la versione base della dashboard
- Lo sviluppo di implementazioni riguardanti il programma in Python e la Mobile App
- La sicurezza del sistema: messa a punto tramite penetration test e attuazione delle soluzioni alle problematiche riscontrate



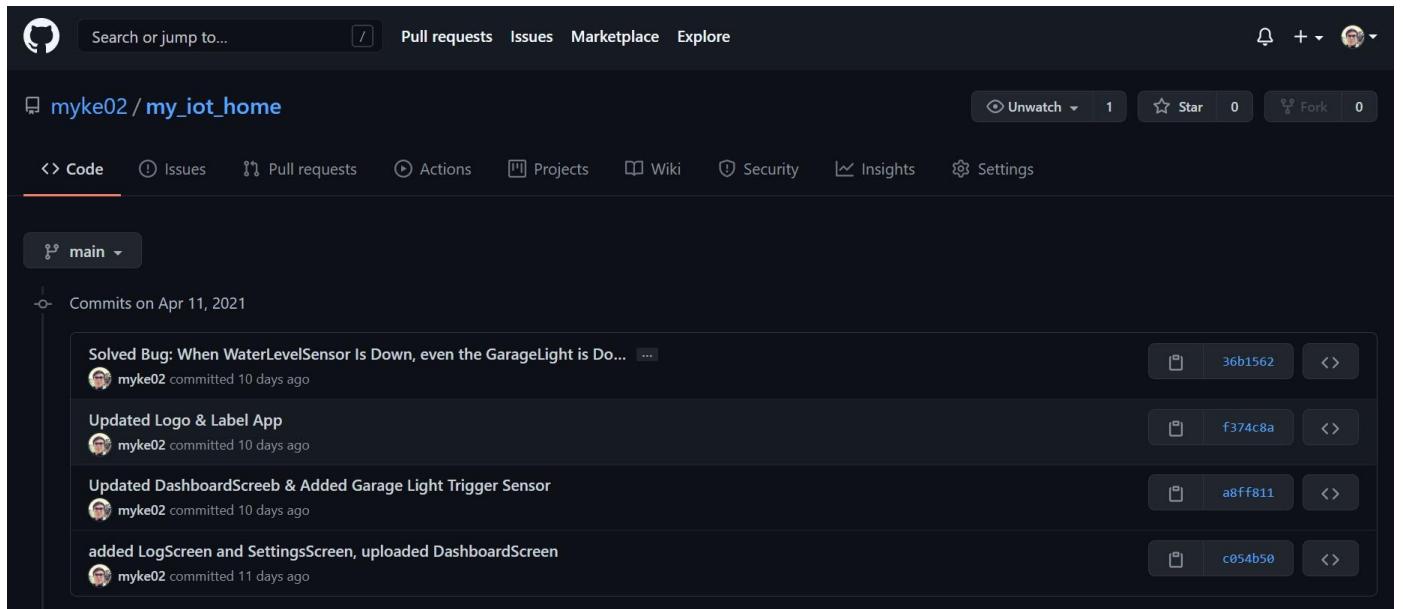
**Figura 4: Sottoattività e Milestones del Gantt Chart**

Inoltre, come mostrato dalla figura 4 sono state stabilite delle sottoattività. Mentre in corrispondenza dei momenti fondamentali e degli obiettivi preposti sono state definite le milestones (ad esempio la scadenza per la realizzazione della “Basic Dashboard GUI”).

## 2.2. VCS: GIT CON GITHUB PRIVATE REPOSITORY

Per il versionamento del codice si è optato per la tecnologia Git e l'utilizzo di un repository privato su GitHub.com. In questo modo il codice non viene distribuito

evitando che vengano resi pubblici dati sensibili e parti di codice che potrebbero rendere attaccabile il sistema.



**Figura 5: Repository privato su GitHub.com e relativi commits**

I vantaggi principali dell'utilizzo di Git e GitHub consistono:

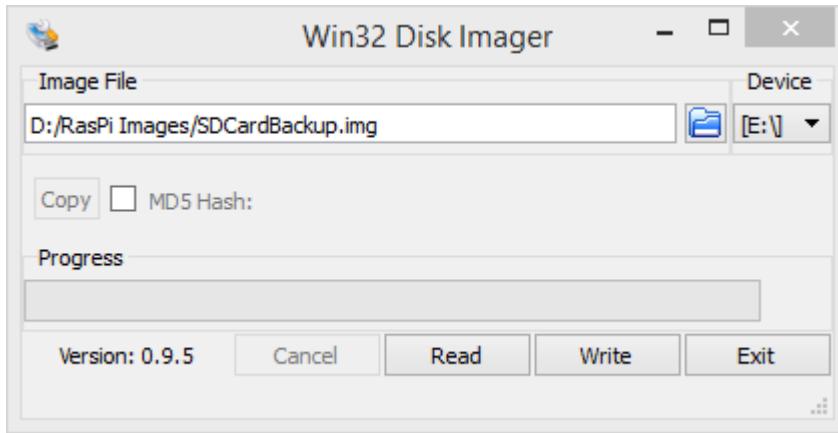
- Nel mantenere una cronologia del codice
- Avere la possibilità di mettere al confronto differenti versioni di codice
- Nel risparmio di tempo
- Nella possibilità di collaborare e sviluppare codice in team e di rendere open-source il progetto

## 2.3. BACKUP MANUALI

Siccome il Raspberry, almeno nella fase di prototipazione e test utilizza una SD card come memoria statica, è presente il rischio che si corrompa nel tempo.

A questa eventualità si è posto rimedio adottando due procedure di sviluppo:

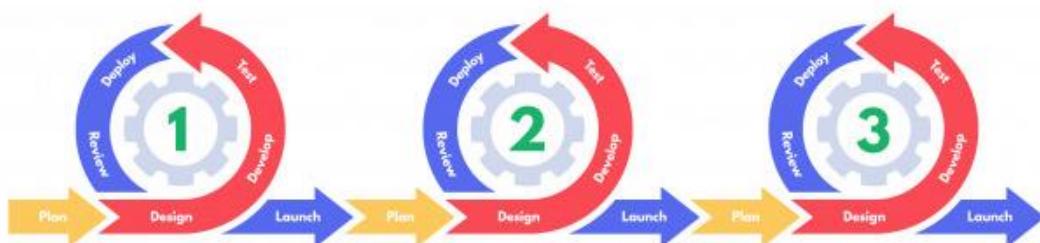
1. Backup periodici del sistema e dei files attraverso la creazione di un file con estensione “.IMG” (tramite il programma Win32DiskImager) che talvolta possono essere salvati in cloud, su un account di Google Drive protetto da autenticazione a due fattori.



**Figura 6:** SD Backup attraverso Win32DiskImager

2. Backup manuale dei files, salvataggio su unità esterna e su un account di Google Drive protetto da autenticazione a due fattori. In particolar modo, questa metodologia è stata impiegata per il salvataggio del programma principale scritto in Python.

## 2.4. SVILUPPO E TESTING



**Figura 7:** Sviluppo iterativo e incrementale secondo il principio dell’agile development

Il progetto è stato sviluppato secondo i criteri dell'agile development. Dunque una volta pianificati gli sviluppi e le integrazioni da aggiungere, volta per volta si sono susseguite le fasi di sviluppo e test a scatola bianca. Infine, le modifiche sono salvate salvate attraverso attraverso backup manuali o l'utilizzo di VCS.

## **3. IL PROGETTO MY IOT HOME V2.0**

My IoT Home V2.0 è un sistema embedded che ha l'obiettivo di sostituire l'interfaccia e le notifiche provvedute dal bot Telegram nella prima versione, con una Mobile App che si basa sui servizi provveduti dalla piattaforma Firebase.

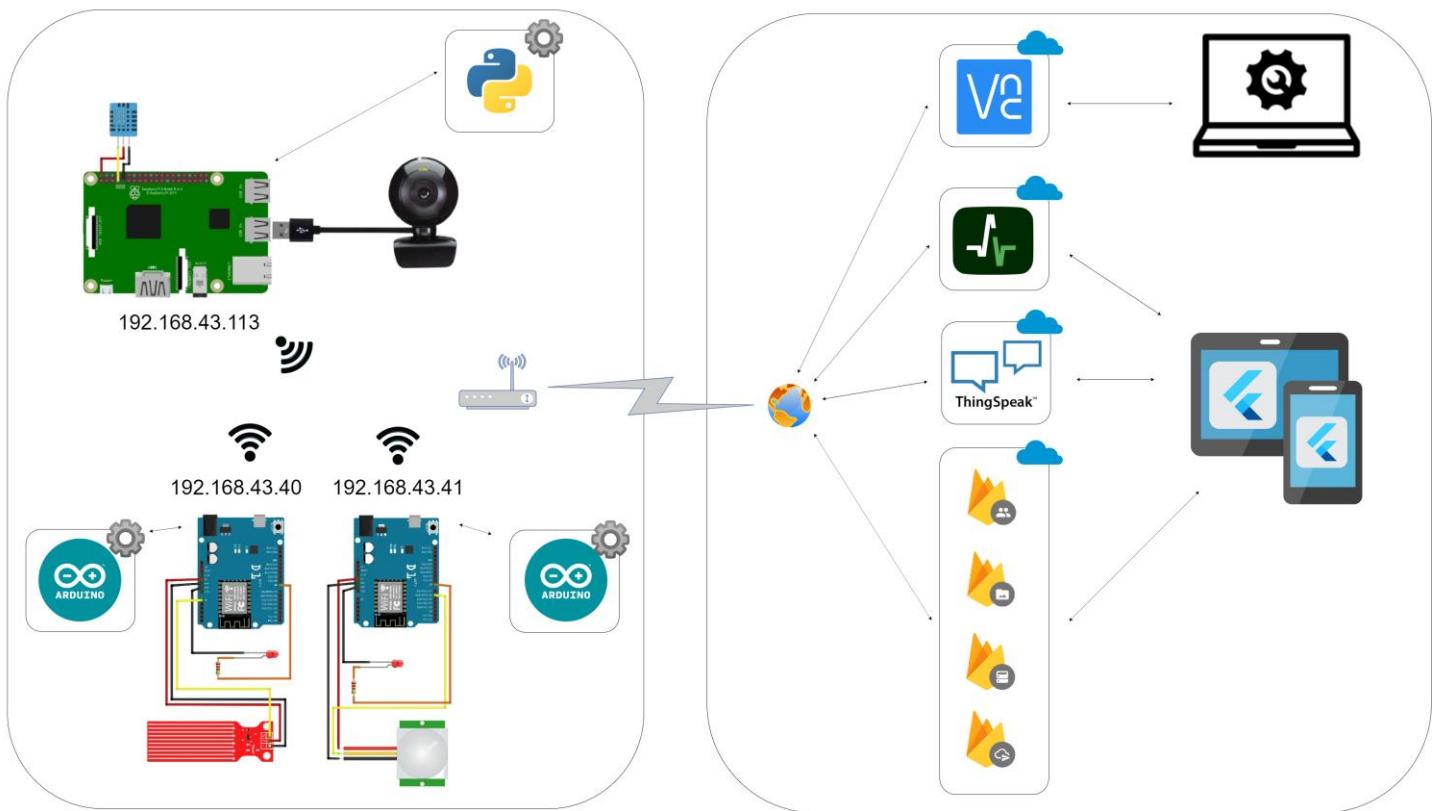
### **3.1. I REQUISITI STABILITI PER LA PROGETTAZIONE**

In fase di progettazione sono state stabiliti dei requisiti riguardanti il sistema, che hanno caratterizzato l'intero sviluppo del progetto:

- Customizzazione: il cliente ha la possibilità di richiedere modifiche relative all'intero sistema, sia lato hardware che lato software.
- Sicurezza: Tutti gli elementi e le connessioni stabilite dal sistema devono garantire confidenzialità, integrità, autenticità dei dati trasmessi ed aderenza alle normative e agli standard vigenti in Italia ed Europa.
- Real-time: I dati vengono aggiornati in tempo reale, con un gap massimo di 15 secondi di ritardo, dovuto all'elaborazione e all'interazione tra le connessioni.
- Affidabilità: Il sistema deve essere dotato di funzioni di diagnosi, funzioni di alert in tempo reale e di un sistema di heartbeat per verificare che il Raspberry sia costantemente alimentato e connesso alla rete. Si rende necessaria anche un sistema di log e di backup periodico del log.
- User experience gradevole: È stato stabilito che il programma principale deve essere eseguito automaticamente non appena il Raspberry viene connesso alla rete e che l'applicazione mobile deve presentare nella schermata delle impostazioni i contatti diretti per l'assistenza (numero di telefono, email, ecc.).

Benchè queste caratteristiche possano rappresentare delle limitazioni, e abbiano richiesto una maggiore quantità di tempo per lo sviluppo, il risultato ottenuto è un prototipo che rispetta le normative software e che aderisce agli standard di progettazione: aspetti che rendono il prodotto commerciabile e accattivante all'acquirente.

### 3.2. PANORAMICA DEL SISTEMA



**Figura 8: Schema a blocchi di My IoT Home V2.0**

Come illustrato nella figura 8 il sistema My IoT Home V2.0 si articola in due parti principali:

1. La prima è pressoché identica dal punto di vista hardware a My IoT Home V1.0 e prevede la comunicazione con la rete in locale tramite protocollo MQTT. In particolare, il Raspberry espone un broker MQTT di tipo Mosquitto

attraverso il quale avviene la comunicazione bidirezionale con due microcontrollori Wemos D1 R2 dotati di ESP8266, in cui si differenziano i messaggi:

- Trasmessi dal Raspberry ai microcontrollori → Rappresentano i trigger, ovvero richieste effettuate ai microcontrollori (ad esempio la richiesta di accensione di un led)
  - Trasmessi dai microcontrollori al Raspberry → Riguardano i dati raccolti dai sensori (PIR o Water Level) o segnali di Ack ad un trigger (ad esempio l'avvenuta accensione di un led al verificarsi della richiesta)
2. La seconda parte richiede che il dispositivo con funzionalità di Access Point, (quindi un router o una saponetta) permetta al Raspberry l'accesso ad Internet e ai servizi online VNC Viewer, Healthchecks.io, i servizi in cloud offerti da Thingspeak.com e dalla piattaforma Firebase by Google. (Le comunicazioni in questione impiegano i protocolli HTTPS e VNC + TLS). Infine, è previsto che gli stessi servizi comunichino con i dispositivi del cliente (la mobile app e la ricezione di notifiche di Down del Raspberry) e con quelli dello sviluppatore per eseguire modifiche da remoto (VNC Viewer).

## 4. DISPOSITIVI HARDWARE

### 4.1. ARDUINO WEMOS D1 R2 V2.1.0 WiFi DOTATO DI ESP8266

Il microcontrollore Wemos D1 R2 WiFi viene distribuito da Geekcreit ed è stato progettato sulla base di ESP-8266EX , acquistabile ad un costo variabile intorno ai 6€. È compatibile con Arduino ed è programmabile su Arduino IDE. Presenta 11 pin I/O digitali ed 1 pin analogico di input. L'alimentazione massima supportata in ingresso è di 24[V], mentre l'uscita presenta una tensione di 5[V] ad una intensità di corrente di 1[A].

Il datasheet [30] e lo schema logico del circuito [31][31] sono recuperabili mediante Archive.org sul sito internet ufficiale del produttore [wiki.wemos.cc](http://wiki.wemos.cc).

Nel caso in cui il numero di pin necessari a soddisfare le richieste del cliente fosse inferiore al necessario, sarebbe possibile impiegare la versione Wemos Mega R3 Atmega2560, anch'essa dotata di ESP8266, reperibile ad un costo variabile intorno agli 11€.

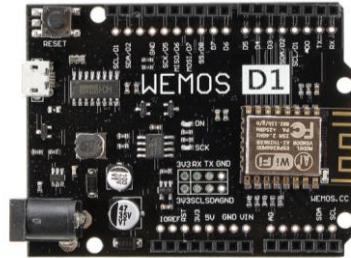


Figura 9: Wemos D1 R2 WiFi

### 4.2. RASPBERRY Pi 3B+



Raspberry Pi 3B+ è un single board computer dotato di 1Gb di memoria RAM e un processore Broadcom @1.4GHz con un Soc a 64 bit. Presenta un modulo WiFi con tecnologia IEEE 802.11.b/g/n/ac e connettività a 2.5GHz e 5GHz.

Figura 10: Raspberry Pi 3B+

Presenta anche 40 pin di GPIO, 4 porte USB 2.0, una porta con interfaccia HDMI ed una porta Gigabit Ethernet con throughput massimo di 300 Mbps. Rpi 3B+ richiede un'alimentazione a 5[v] di tensione e 2.5[A] di intensità di corrente.

Il Raspberry necessita di una SD card con il sistema operativo pre-flashato basato su Linux. Per il progetto My IoT Home V2.0 si è optato per una SD card da 32 Gb di classe U1 garantendo una velocità minima di scrittura di 10 MB/s.

Il costo del singolo dispositivo, esclusi componenti aggiuntivi, si aggira intorno ai 50€.

Il datasheet e la documentazione ufficiale sono recuperabili dal sito ufficiale della Raspberry Pi Foundation che li produce [26].

### **4.3. ALIMENTATORI E POWERBANK**

A seconda del dispositivo e del suo impegno sono state selezionate diverse fonti di alimentazione:

1. Per il Raspberry Pi 3B+ è necessario impiegare un alimentatore a 5[V] di tensione e almeno 2,5[A] di intensità di corrente, con interfaccia micro USB. Nel caso in cui si decida di impiegare un Raspberry 4, è consigliabile una fonte di alimentazione con almeno 3[A] di intensità di corrente, in quanto le componenti hardware, a seconda delle operazioni in esecuzione, richiedono una maggiore potenza. Il costo si aggira intorno ai 10 €.
2. Per un microcontrollore Wemos, in vicinanza di una presa elettrica, è necessario impiegare un alimentatore a 5 [V] di tensione e almeno 1 [A] di intensità di corrente, con interfaccia micro USB. Il costo è di circa 10 €.
3. Per un microcontrollore Wemos, in mancanza di una presa elettrica, come accade per quello collocato in garage nella prototipazione del sistema, è necessario utilizzare un powerbank che eroga 5 [V] di tensione ad almeno 1 [A] di intensità di corrente. A seconda della capacità (se ne consiglia uno con capacità minima di 20000 mAh, in modo da massimizzare l'autonomia) il costo si aggira intorno ai 15 €. Un'altra soluzione “out of the box” è rappresentata

dall'utilizzo di powerbank dotati di pannello solare, ad un costo che si aggira intorno ai 30 €.

In base alle caratteristiche del sistema installato e al grado di affidabilità di cui si necessita potrebbe rendersi necessaria la spesa di un generatore di corrente.

## 4.4. SENSORI

Per quanto riguarda i sensori connessi ai microcontrollori sono stati utilizzati un Water Level sensor ed un sensore di movimento PIR. Mentre i sensori connessi al Raspberry sono il DHT11 ed una Webcam.

### 4.4.1 Water Level Sensor

Il Water Level Sensor è un sensore di riconoscimento di alto e basso livello dell'acqua piovana.

Oltre ai pin di alimentazione, operanti ad una tensione di 3-5[V] e un'intensità di corrente di 20[mA], è presente un pin analogico che trasmette il segnale. Il principio di rilevamento si basa sulla presenza dei fili paralleli esposti alle gocce d'acqua o ad un volume di acqua. La superficie di rilevamento è di 40mm x 16mm, la temperatura di funzionamento comprende il range 10-30 [°C] ed il range dell'umidità di funzionamento varia dal 10% al 90% senza condensa, altrimenti verrebbero falsate le letture.

Il costo si aggira intorno a 4 €.

Il datasheet [10] è consultabile presso il sito dell'Emartee, un fornitore professionale di componenti elettrici.

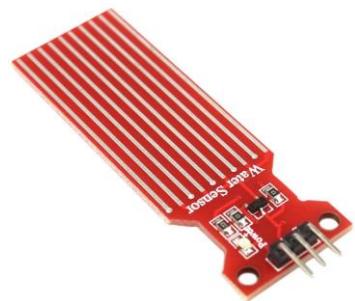


Figura 11: Water Level Sensor

#### 4.4.2 PIR Sensor HC-SR501

Il sensore PIR (acronimo di *Passive InfraRed*) è in grado di rilevare movimenti nell'ambiente circostante attraverso la radiazione infrarossa.

La sensibilità può essere regolata attraverso un potenziometro fisco e test in loco. La tensione di funzionamento è pari a 5[V] e l'intensità di corrente è inferiore ai 20[mA], limitata dalle caratteristiche elettriche del microcontrollore.

Il costo si aggira intorno ai 5 €.

Il datasheet [1] è consultabile presso il sito dell'Adafruit Industries, una società di hardware open-source, leader riconosciuto nel settore della produzione di componenti elettrici ed elettronici.



Figura 12: PIR Sensor HC-SR501

#### 4.4.3 DHT11 Sensor

Il sensore DHT11 è un sensore che misura la temperatura con una sensibilità di 1 [°C] e l'umidità con una sensibilità del 5%. Il range di misurazione della temperatura va da 0 a 50 [°C], mentre quello dell'umidità dal 20% al 90%. Il sensore necessita di un'alimentazione a 3.3-5 [V] ad un'intensità di corrente di 20[mA].

Il costo si aggira intorno ai 4 €.

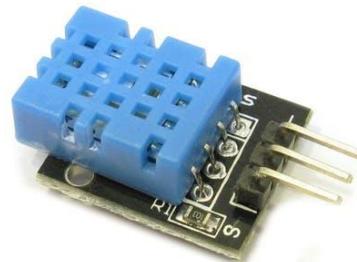


Figura 13: Sensore DHT11

Per una sensibilità maggiore delle letture effettuabili si consiglia di optare per il sensore DHT22.

Il datasheet [2] è consultabile presso il sito dell'Adafruit Industries, una società di hardware open-source, leader riconosciuto nel settore della produzione di componenti elettrici ed elettronici.

#### 4.4.3 Webcam Logitech C120



Figura 14: Webcam Logitech C120

La Webcam Logitech C120 offre un collegamento USB 2.0 con una risoluzione pari a 1,3 MP 640 x 480 Pixel.

Si tratta di un sensore utile ai fini di prototipazione, ma per gli sviluppi futuri del progetto My IoT Home V2.0 è necessario l'impiego di webcam o videocamere di sorveglianza dotate di funzioni quali la visione notturna, la messa a fuoco automatica ed una maggiore risoluzione. A seconda della scelta

che si effettuerà, il costo varia indicativamente tra 20 e 100 €.

Per l'accertamento delle specifiche tecniche si consiglia di fare riferimento alla casa produttrice del dispositivo scelto.

### 4.5. ATTUATORI

Per quanto riguarda gli attuatori connessi ai microcontrollori sono stati impiegati dei diodi led. Il diodo di colore rosso connesso al Wemos dotato di sensore PIR ha la funzione di feedback visivo nel caso in cui venga rilevato un movimento, mentre il diodo di colore giallo simula altri trigger personalizzabili come ad esempio la modifica dello stato di un relè.

#### 4.5.1 Diodi Led (e dimensionamento dei resistori)

I valori di tensione utilizzata per i diodi differisce a seconda del colore, mentre l'intensità di corrente necessaria è compresa tra 10 e 15 [mA].

Siccome il diodo led rosso richiede 1.8 [V] è possibile valutare il valore della resistenza necessaria tramite i seguenti calcoli:



Figura 15: Diodi led e relativi resistori da 220 [ohm]

$$R_{\min} = \frac{\text{Tensione di alimentazione} - \text{Caduta di Tens.su led}}{\text{Corrente max passante}} = \frac{5 - 1,8}{0,015} = \frac{3,2}{0,015} = 213[\Omega]$$

$$R_{\max} = \frac{\text{Tensione di alimentazione} - \text{Caduta di Tens.su led}}{\text{Corrente min passante}} = \frac{5 - 1,8}{0,010} = \frac{3,2}{0,010} = 320[\Omega]$$

Mentre per il diodo led giallo, che necessita di un'alimentazione a 1,9 [V] si fa riferimento ai seguenti calcoli:

$$R_{\min} = \frac{\text{Tensione di alimentazione} - \text{Caduta di Tens.su led}}{\text{Corrente max passante}} = \frac{5 - 1,9}{0,015} = \frac{3,1}{0,015} = 207[\Omega]$$

$$R_{\max} = \frac{\text{Tensione di alimentazione} - \text{Caduta di Tens.su led}}{\text{Corrente min passante}} = \frac{5 - 1,9}{0,010} = \frac{3,1}{0,010} = 310[\Omega]$$

Ne consegue che i resistori scelti dalla scala E12, hanno una resistenza nominale tarata a 220 [ $\Omega$ ]  $\pm$  5%.

Il costo dei componenti in questione è inferiore all'unità di euro.

Il datasheet del componente [3] è consultabile presso il sito dell'Adafruit Industries, una società di hardware open-source, leader riconosciuto nel settore della produzione di componenti elettrici ed elettronici.

## 5. SCHEMI CIRCUITALI E COLLEGAMENTI FISICI

Al fine di documentare il progetto e di presentarlo ad un ipotetico cliente sono stati realizzati i seguenti schemi circuitali corredati dalla specifica dei collegamenti fisici.

### 5.1. Wemos dotato di sensore Water Level

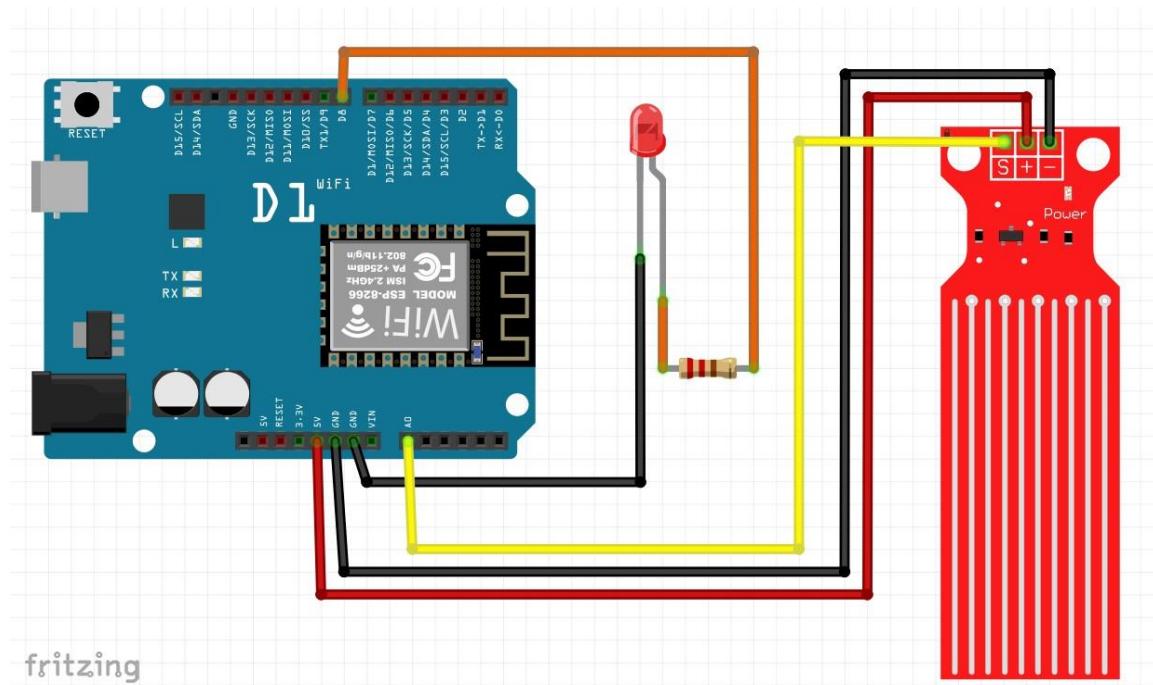


Figura 16: Wemos dotato di sensore Water Level

Sensore/Attuatore	Pin Sensore/Attuatore	Pin Wemos
Water Level	Segnale (S)	A0
Water Level	Vcc (+)	5V
Water Level	GND (-)	GND
Diodo Led	Vcc (+)	D8
Diodo Led	GND (-)	GND

Tabella 1: Collegamenti fisici del Wemos dotato del sensore Water Level

Il microcontrollore in questione ha lo scopo di segnalare attraverso il modulo ESP8266 eventuali allagamenti del garage e di controllare lo stato un led (simulando un relè o altri attuatori).

## 5.2. Wemos dotato di sensore PIR

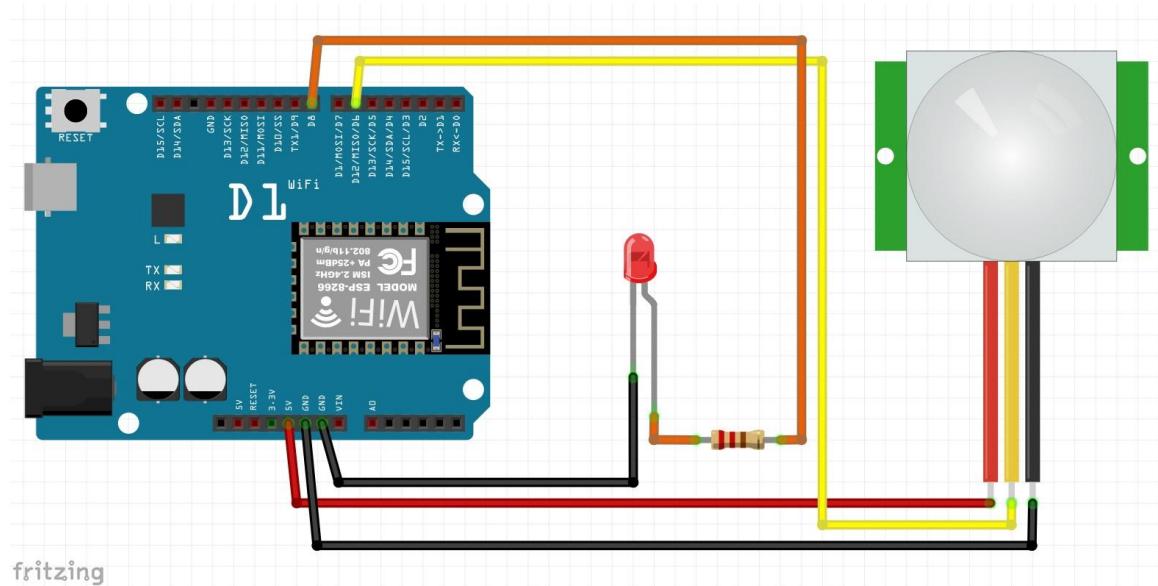


Figura 17: Wemos dotato di sensore PIR

Sensore/Attuatore	Pin Sensore/Attuatore	Pin Wemos
PIR	Segnale (S)	D6
PIR	Vcc (+)	5V
PIR	GND (-)	GND
Diodo Led	Vcc (+)	D8
Diodo Led	GND (-)	GND

Tabella 2: Collegamenti fisici del Wemos dotato del sensore PIR

Il microcontrollore in questione ha lo scopo di segnalare attraverso il modulo ESP8266 i movimenti sospetti captati dal sensore PIR. Il led connesso ha la funzione di feedback visivo.

### 5.3. Raspberry dotato del sensore DHT11 e Webcam

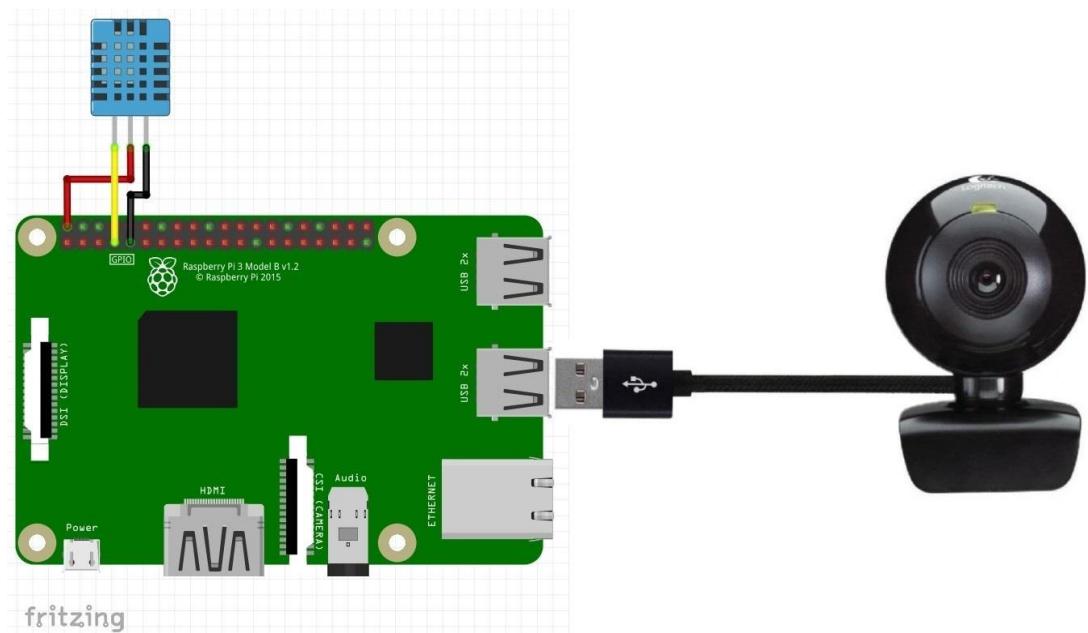


Figura 18: Raspberry dotato del sensore DHT11 e Webcam

Sensore	Pin Sensore	Pin Raspberry
DHT11	Segnale (S)	GPIO 4 / (N°7)
DHT11	Vcc (+)	5V / (N°2)
DHT11	GND (-)	GND / (N°9)

Tabella 3: Collegamenti fisici del Raspberry e del sensore DHT11

Fisicamente il Raspberry instaura una connessione fisica, tramite GPIO, con un sensore DHT11 al fine di rilevare temperatura e umidità dell'ambiente circostante. Un secondo collegamento, tramite porta USB 2.0, permette al “main program” in Python di richiedere alla webcam la cattura di un'immagine e poi di inviarla sul Firebase Storage.

## **6. TECNOLOGIE E SOFTWARE IMPIEGATI**

Dopo la fase di assemblaggio e test della prima versione del progetto, è seguita la fase di ricerca che ha permesso la scelta delle tecnologie effettivamente utilizzate.

### **6.1. FASE DI RICERCA: AMAZON AWS E GOOGLE FIREBASE**

Sia Amazon che Firebase sono dei servizi ad architettura serverless, perciò a differenza delle architetture tradizionali, che prevedono una struttura a microservizi. Questo significa che a seconda del tipo delle operazioni richieste, AWS o Firebase, faranno riferimento a diversi componenti della propria struttura.

I vantaggi principali dei servizi serverless secondo Dashbird (AWS partner network) sono l'estrema scalabilità e il pagamento solo per il runtime o determinati servizi [7].

I test eseguiti necessari alle scelte di progetto sono stati:

- Test del broker MQTT esposto da AWS con crittografia e test della comunicazione con Raspberry
- Test del realtime database esposto da Firebase con crittografia e test della comunicazione con Raspberry

Entrambi i sistemi hanno dimostrato di essere affidabili e al termine si è optato per la piattaforma Firebase per i seguenti motivi:

- I servizi di Firebase necessari al progetto non sono a pagamento, mentre Amazon AWS presenta un costo pay-per-use.
- Firebase implica una curva di apprendimento inferiore rispetto ad AWS
- Firebase offre servizi più innovativi, quali l'API delle immagini, di dati e di audio/voce. È prevista inoltre la presenza di più servizi esclusivi.

## 6.2. SUITE TCP-IP E PROTOCOLLI DI APPLICAZIONE

Per i dispositivi IoT, in genere è essenziale instaurare delle connessioni, che siano a livello di rete locale o che richiedano la comunicazione con determinati servizi, come avviene per la comunicazione con l'architettura serverless di Google Firebase.

### 6.2.1 STACK MQTT NELLA LAN

Le soluzioni più comuni per l'instaurazione di connessioni di questo tipo prevedono l'utilizzo di protocolli rilasciati oppure proprietari per la comunicazione al livello fisico della pila ISO/OSI.

Segue poi lo stack di protocolli: TCP/IP a livello di rete e trasporto e MQTT (*Message Queue Telemetry Transport*) o MQTTS (*MQTT Secure*, se eseguito su SSL/TLS) a livello di applicazione.

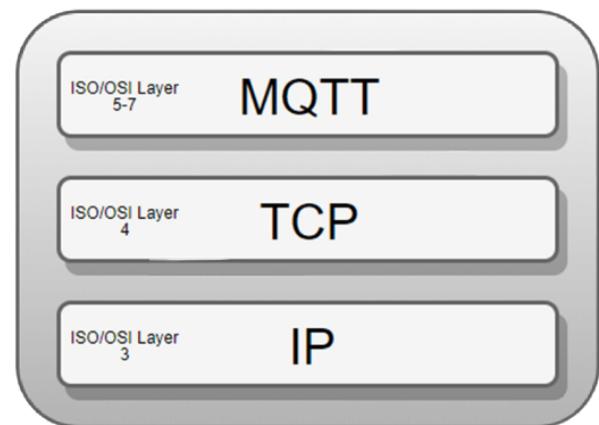


Figura 19: Stack MQTT

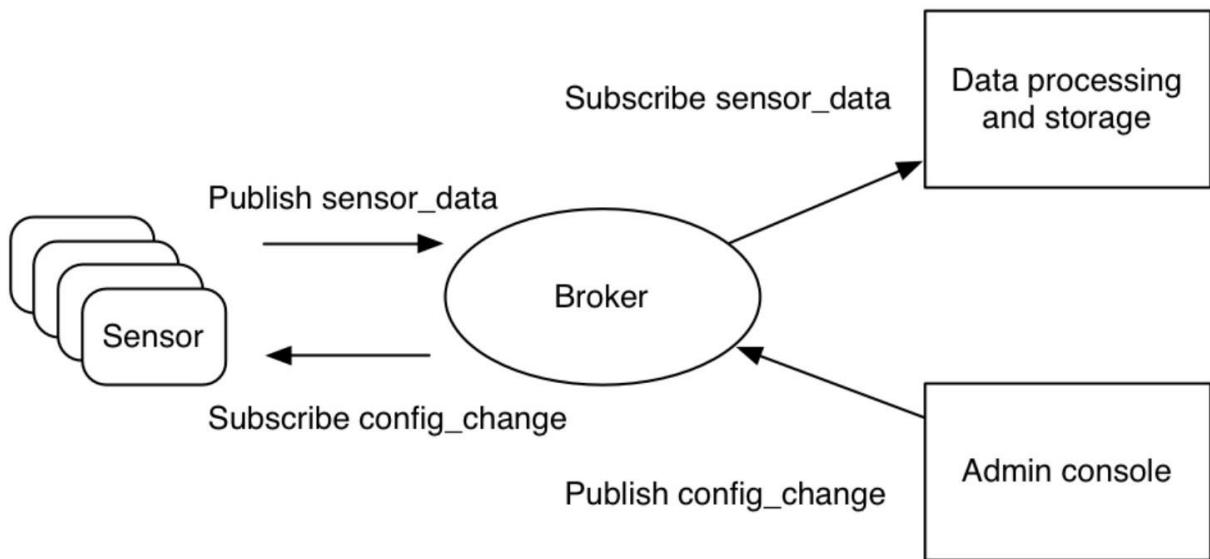
MQTT è stato ideato e sviluppato da IBM alla fine degli anni '90 e come indica l'acronimo, è un protocollo di messaggistica che supporta la comunicazione asincrona tra le parti, rendendola scalabile in ambienti inaffidabili. Alla fine del 2014 è divenuto uno standard open OASIS e ad oggi supportato dai linguaggi di programmazione più diffusi [13].

I vantaggi principali per cui si è scelto di utilizzare il protocollo MQTT per la comunicazione tra i microcontrollori Wemos dotati di modulo ESP8266 ed il Raspberry sono i seguenti:

- Il protocollo MQTT prevede la trasmissione di brevi messaggi in asincronia e consente la comunicazione full-duplex.

- I messaggi MQTT sono organizzati per argomento
- La sua estrema flessibilità consente di supportare diversi scenari applicativi.

Ne consegue che il sistema My IoT Home V2.0 segue il modello di pubblicazione e sottoscrizione MQTT per i microcontrollori IoT:



**Figura 20: Modello di pubblicazione e sottoscrizione MQTT**

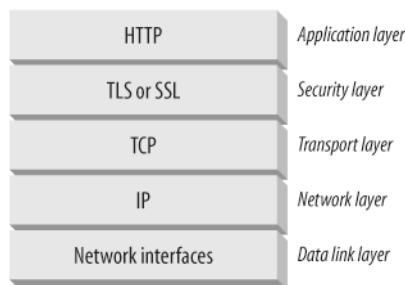
Nello specifico il sistema My IoT Home V2.0 prevede che nella rete locale:

1. Il Raspberry esponga un servizio MQTT con autenticazione mettendo a disposizione un broker di tipo Mosquitto. (Tecnologia sviluppata da Eclipse Foundation [8])
2. I client (3 in totale: 2 microcontrollori Wemos e il programma in Python sul Raspberry stesso) inviano i messaggi siglati da un argomento o “topic”.
3. Il broker inoltra il messaggio a tutti i clienti iscritti a quell’argomento o “topic”.

## 6.2.2 STACK HTTP/HTTPS PER I SERVIZI ONLINE

Il progetto prevede l'impiego dei seguenti servizi online che si basano sullo stack HTTPS:

- Firebase Services by Google
- Thingspeak.com
- Healthchecks.io
- RealVNC



**Figura 21: Stack HTTPS (HTTP + SSL/TLS)**

Ciò significa che tutti i servizi online utilizzati, prevedono il layer SSL/TLS, e dunque la crittografia dei messaggi e dei file inviati tramite il protocollo HTTP.

Ne consegue che le richieste HTTP avvengono in maniera sincrona, ovvero il client attende che il server risponda. Questo comporta un minimo di delay nella fase di upload dei dati, che comunque è più che accettabile, perché il campionamento dei dati rilevati dai microcontrollori Wemos prevede un intervallo di tempo pari a 5[s].

Un altro motivo per cui si è optato per l'impiego di servizi con protocollo HTTPS è il fatto che esso rappresenta uno standard. Dunque, nel caso in cui il progetto evolovesse fino al punto di diventare scalabile ed essere distribuito su scala globale, il deployment dell'applicativo attraverso Play Store e App Store non richiederà documentazioni aggiuntive riguardanti i protocolli impiegati.

### 6.3. THINGSPEAK.COM

ThingSpeak è una piattaforma di IoT Analytics che consente di aggregare, analizzare e visualizzare flussi di dati in tempo reale mediante un servizio in cloud [29].

Nello specifico si è scelto di utilizzare ThingSpeak in quanto provvede accesso a MATLAB, una piattaforma di data analysis che abilita l'utente a visualizzare i dati di un determinato canale avendo la possibilità di stabilirne le caratteristiche.

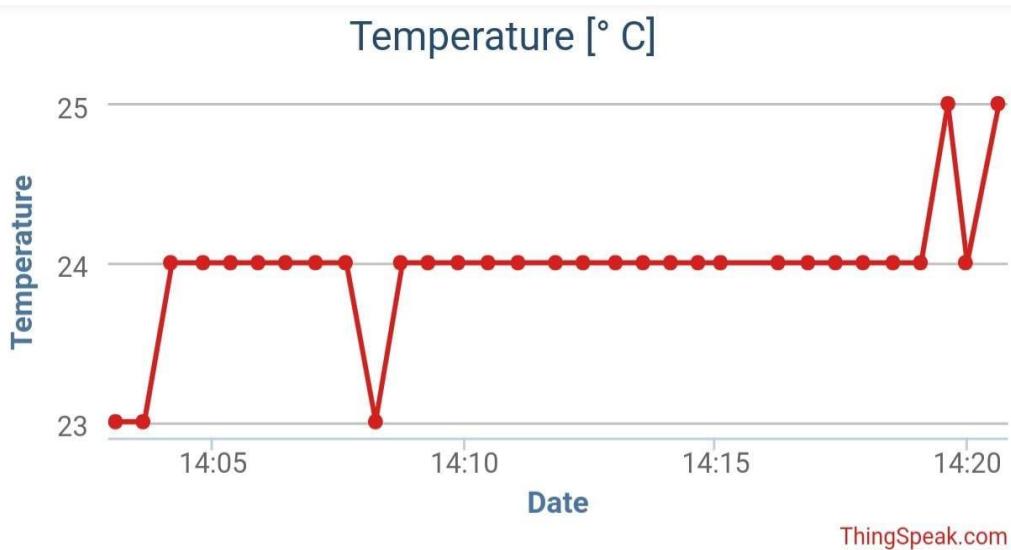


Grafico 2: Visualizzazione della Temperatura mediante ThingSpeak.com e MATLAB

Nello specifico il progetto My IoT Home V2.0 impiega ThingSpeak per la realizzazione e la fruizione di grafici 2D. Uno dei motivi principali per cui si è optato per Thingspeak è il fatto che dimensiona dinamicamente l'unità di misura dell'asse delle ascisse (le ore, i minuti o i giorni), in base alla regolarità con cui vengono inviati i dati.

Oltre a questo, ThingSpeak è risultato idoneo ai requisiti di sicurezza preposti, in quanto fa uso del protocollo crittografato HTTPS e dell'autenticazione mediante Api Key, resa necessaria ad ogni richiesta.

## 6.4. HEALTHCHECKS.IO

Healthchecks.io è un prodotto open-source di SIA Monkey See Monkey Do, una società di software e consulenza individuale. Healthchecks.io offre un servizio di avviso quando un evento non si verifica nel tempo previsto, ed è basato sui Docker Containers, garantendo così estrema scalabilità e affidabilità del servizio stesso [27].

Name	Ping URL	Integrations	Period Grace	Last Ping
MyIoTHome RPI	https://hc-ping.com/b0ff4e42[REDACTED]	@	***** 2 minutes	2 weeks ago

Figura 22: Configurazione di base si Healthchecks.io

Nello progetto My IoT Home V2.0, Healthchecks.io viene utilizzato per verificare che il Raspberry sia costantemente alimentato e connesso alla rete Internet, in caso contrario, se passati 2 minuti tra un ping e quello previsto, avviene l'avviso mediante mail.

## 6.5. RealVNC

RealVNC è un servizio che consente l'accesso e l'assistenza remota attraverso il protocollo HTTPS, che prevede crittografia dei dati. Ad incrementare la sicurezza del servizio vi è anche l'autenticazione a due fattori che avviene mediante accesso tramite password all'account e conferma in tempo reale via mail.

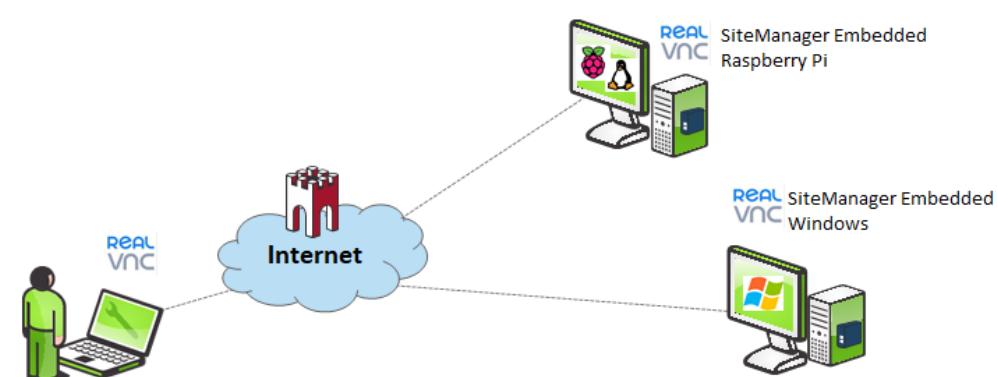


Figura 23:  
Controllo remoto  
e assistenza  
tramite RealVNC

Lo stesso programma RealVNC è estremamente utile anche per la fase di sviluppo su Raspberry, in quanto una volta connesso il dispositivo alla LAN è possibile accedervi senza l'impiego di un monitor dedicato, bensì sfruttando il protocollo proprietario VNC.

## 6.6. PYTHON

Python è un linguaggio di programmazione di alto livello estremamente versatile attualmente sviluppato in maniera open-source dalla Python Software Foundation.

Tra i vari utilizzi vi è lo sviluppo web, la programmazione di GUI dedicate, la generazione di programmi riguardanti il machine learning (viste le numerose integrazioni disponibili) e la creazione di programmi per sistemi embedded, come accade nel caso di My IoT Home V2.0.

## 6.7. FLUTTER, BY GOOGLE

Flutter è un framework reso open-source, ideato e sviluppato da Google nel 2017. L'obiettivo principale di Flutter è abilitare allo sviluppo di applicazioni multipiattaforma (per Android, iOS) utilizzando un singolo codice sorgente scritto in Dart, un linguaggio di programmazione ad oggetti.

La scelta dell'utilizzo di Flutter è stata motivata anche dal fatto che online si dispone di una documentazione piuttosto ampia ed è un dato di fatto che la community sia molto attiva!

Un altro motivo per cui è stato impiegato questo framework è l'integrazione offerta con Firebase by Google per il back-end attraverso packages sviluppati e rilasciati da Google.

Infine, la recente stable release di “Flutter 2”, genera codice JS per una Web App, anche se non ancora tutti i packages disponibili su Flutter Pub includono la funzione.

## 7. REALIZZAZIONE E SVILUPPO

### 7.1. PROGRAMMI ARDUINO

Il codice scritto in Arduino e utilizzato per i due microcontrollori Wemos, è stato distribuito da Adafruit Industries con licenza MIT [22]. I codici in questione permettono ai Wemos dotati di ESP-8266 di iscriversi ai topics MQTT e pubblicare sui topics MQTT, in quanto sono clients autenticati del broker Mosquitto MQTT configurato sul Raspberry.

#### 7.1.1 CODICE DEL WEMOS DOTATO DI SENSORE PIR

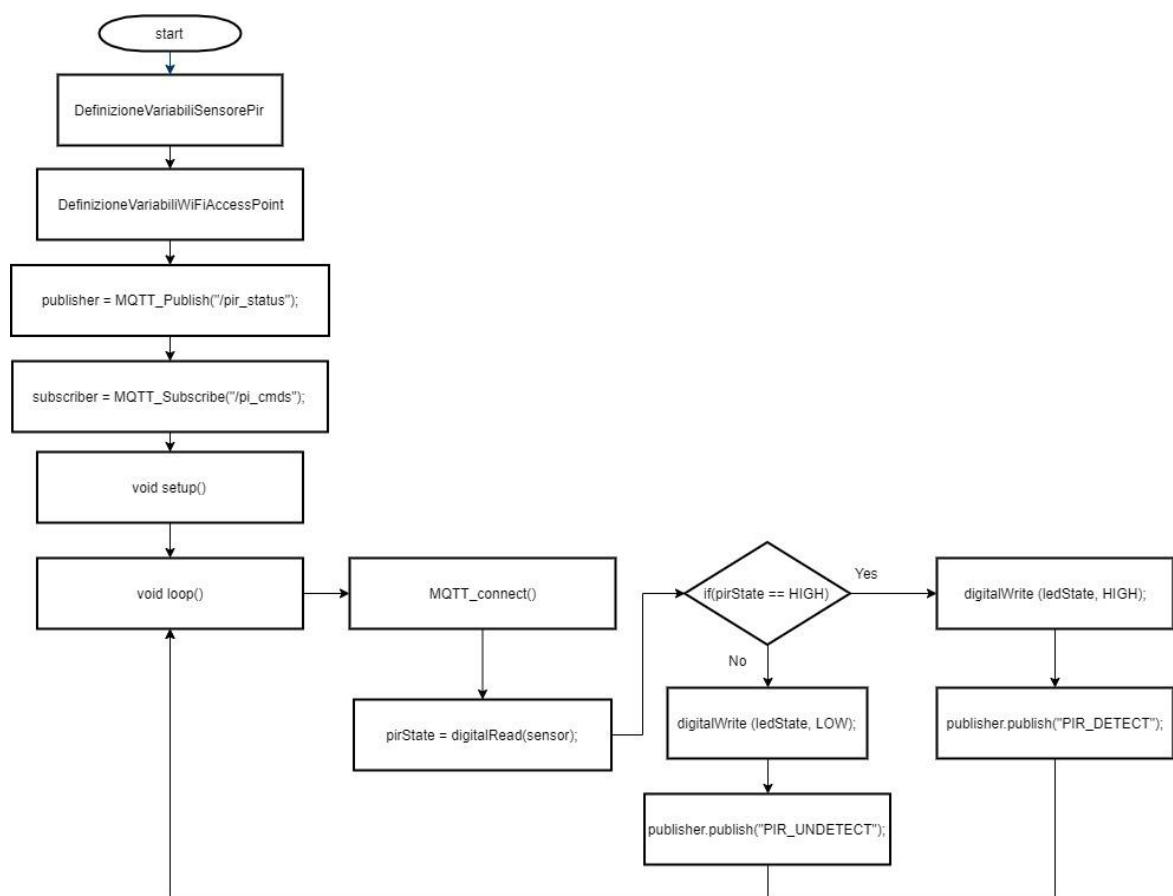


Figura 24: Flowchart a blocchi del programma in Arduino del Wemos dotato di sensore PIR

La prima parte del codice è caratterizzata dalla definizione delle variabili utilizzate per la gestione del sensore e di quelle necessarie alla sottoscrizione del Broker MQTT ospitato su Raspberry:

```
***** Libs *****
#include <ESP8266WiFi.h>
#include "Adafruit_MQTT.h"
#include "Adafruit_MQTT_Client.h"
***** WiFi Access Point *****
#define WLAN_SSID      "████████"
#define WLAN_PASS     "████████"
#define MQTT_SERVER    "192.168.████" //IP address of Raspberry MQTT Server
#define MQTT_PORT      1883
#define MQTT_USERNAME   "homeuser"
#define MQTT_PASSWORD  "████████"
```

**Figura 25: Definizione delle variabili utili alla sottoscrizione con autenticazione del Broker MQTT**

In seguito sono stati definiti publisher e subscriber MQTT, indicando i rispettivi topics:

```
// Setup a feed called 'publisher' for publishing.
// Notice MQTT paths for AIO follow the form: <username>/feeds/<feedname>
Adafruit_MQTT_Publish publisher = Adafruit_MQTT_Publish(&mqtt, "/pir_status");
// Setup a feed called 'subscriber' for subscribing to changes.
Adafruit_MQTT_Subscribe subscriber = Adafruit_MQTT_Subscribe(&mqtt, "/pi_cmds");
***** Sketch Code *****
```

**Figura 26: Definizione di publisher e subscriber MQTT del Wemos dotato di sensore PIR**

Un'altra parte essenziale del codice è la funzione void loop(), che permette la pubblicazione dello stato del sensore PIR attraverso il subscriber del canale "/pir\_status":

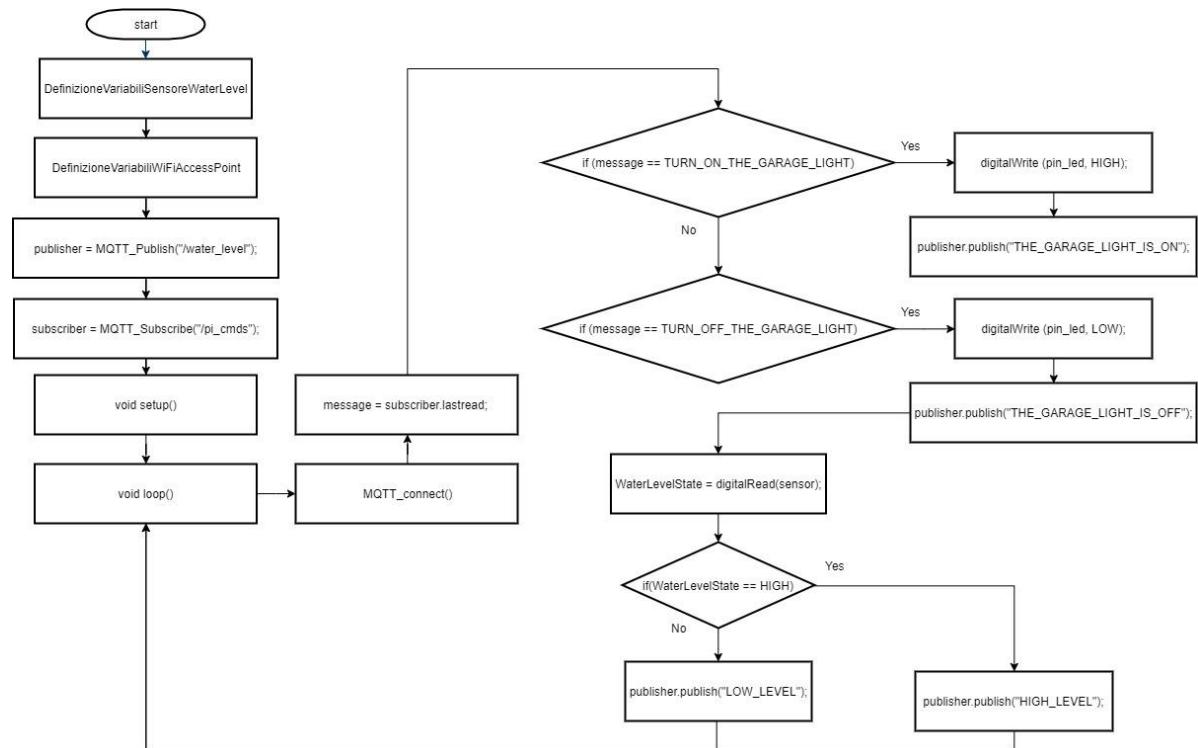
```

void loop() {
    // Ensure the connection to the MQTT server is alive
    MQTT_connect();
    // Motion Check and Publish Pir State
    long pirState = digitalRead(sensor);
    if(pirState == HIGH){
        digitalWrite (ledState, HIGH);
        Serial.println("Motion detected!");
        publisher.publish("PIR_DETECT");
    } else {
        digitalWrite (ledState, LOW);
        Serial.println("Motion absent!");
        publisher.publish("PIR_UNDETECT");
    }
    delay(5000);
}

```

**Figura 27: Funzione void loop() del Wemos dotato di sensore PIR**

### 7.1.2 CODICE DEL WEMOS DOTATO DI SENSORE WATER LEVEL E LED



**Figura 28: Flowchart a blocchi del programma in Arduino del Wemos dotato di sensore Water Level e LED**

Il programma del Wemos dotato di sensore Water Level è molto simile nella struttura a quello del Wemos dotato di sensore PIR (dunque fare riferimento al codice precedentemente riportato), con la sola differenza che anche la ricezione dei comandi di trigger viene gestita e all'interno della funzione di loop:

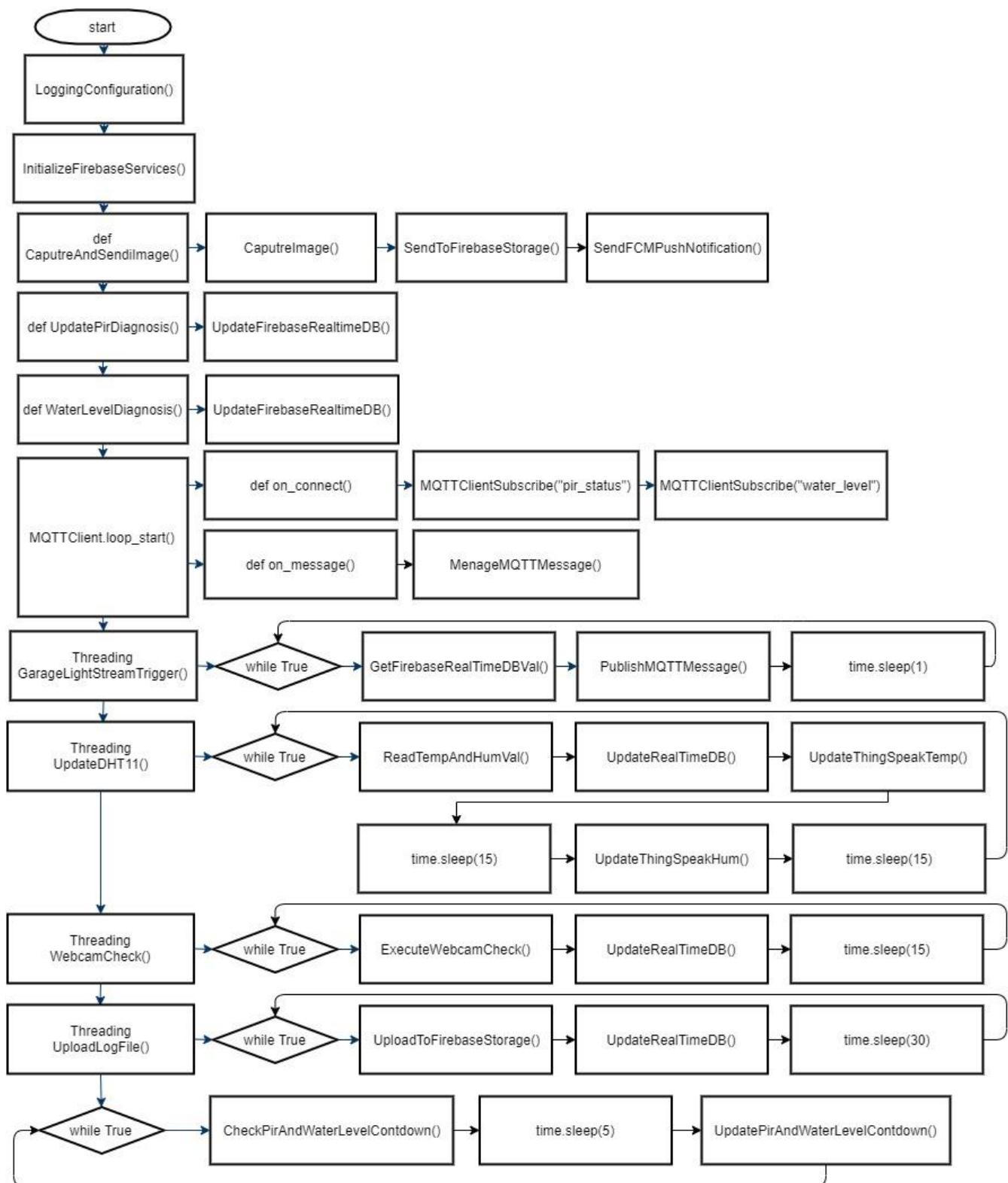
```
//Manage MQTT subscriber's messages
Adafruit_MQTT_Subscribe *subscription;
while ((subscription = mqtt.readSubscription()) != NULL) {
    if (subscription == &subscriber) {
        char *message = (char *)subscriber.lastread;
        Serial.print(F("Received Rpi Command: "));
        Serial.println(message);
        // Check if the message was ON, OFF, or TOGGLE.
        if (strcmp(message, "TURN_ON_THE_GARAGE_LIGHT", 24) == 0) {
            // Turn the LED on.
            Serial.println("THE GARAGE LIGHT IS ON");
            digitalWrite(pin_led, HIGH);
            if(digitalRead(pin_led) == HIGH)
                publisher.publish("THE_GARAGE_LIGHT_IS_ON");
        }
        else if (strcmp(message, "TURN_OFF_THE_GARAGE_LIGHT", 25) == 0) {
            // Turn the LED off.
            Serial.println("THE GARAGE LIGHT IS OFF");
            digitalWrite(pin_led, LOW);
            if(digitalRead(pin_led) == LOW)
                publisher.publish("THE_GARAGE_LIGHT_IS_OFF");
        }
    }
}
```

Figura 29: Codice di gestione dei trigger rilevati dal subscriber MQTT

## 7.2. PROGRAMMA IN PYTHON ED INTEGRAZIONI API

Il programma principale eseguito sul Raspberry si occupa della ricezione e dell'invio di messaggi MQTT tramite la configurazione di un client MQTT esposto dallo stesso Raspberry. Lo stesso programma si interfaccia con i servizi offerti dalla piattaforma Firebase e ThingSpeak, e gestisce le comunicazioni di back-end mediante chiamate HTTPS secondo le indicazioni della documentazione dalle API utilizzate.

## 7.2.1 SCHEMA A BLOCCHI DEL PROGRAMMA IN PYTHON



**Figura 30: Flowchart a blocchi del programma principale in Python**

Il programma principale in Python esegue più operazioni in parallelo attraverso la programmazione concorrente ed il modulo di Threading [25].

### 7.2.2 MODULI PYREBASE, PYFCM E FIREBASE PLATFORM

```
import time
import pyrebase
import fcm
import subprocess

firebaseConfig = {
    "apiKey": "AI[REDACTED]",
    "authDomain": "[REDACTED].firebaseapp.com",
    "databaseURL": "https://[REDACTED].firebaseio.com",
    "projectId": "[REDACTED]",
    "storageBucket": "[REDACTED].appspot.com",
    "messagingSenderId": "10[REDACTED]",
    "appId": "1:[REDACTED]:web:[REDACTED]",
    "measurementId": "G-[REDACTED]"
}

firebase = pyrebase.initialize_app(firebaseConfig)

realtime_db = firebase.database()

storage = firebase.storage()
```

Figura 31: Pyrebase config e definizione dei servizi utilizzati

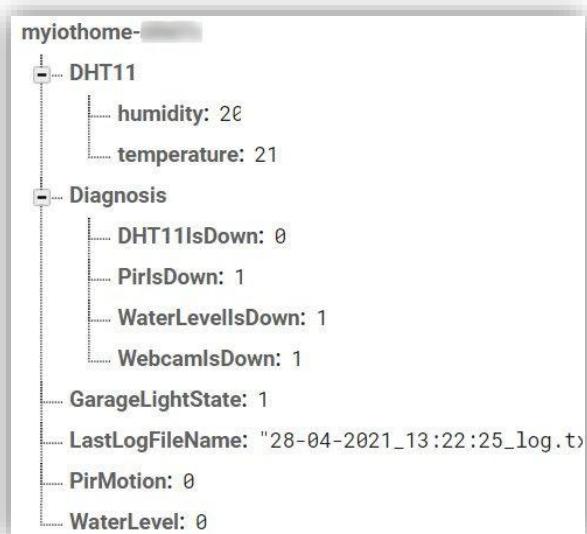


Figura 32: Real Time Database

Nella prima parte del programma vengono importate le variabili utili all'autenticazione e alla definizione di servizi. Mediante il modulo Pyrebase vengono inizializzati i servizi del Real Time Database e del Firebase Storage. Inoltre, viene importato il modulo Fcm, necessario per inviare notifiche push attraverso Firebase Cloud Messaging.

```

from pyfcm import FCMNotification

server_key="AAAA7SZ_BuI:APA91
home_flood_url_img = "https://tinyurl.com/home-flood-image"
pir_url_image = "https://tinyurl.com/pir-image"

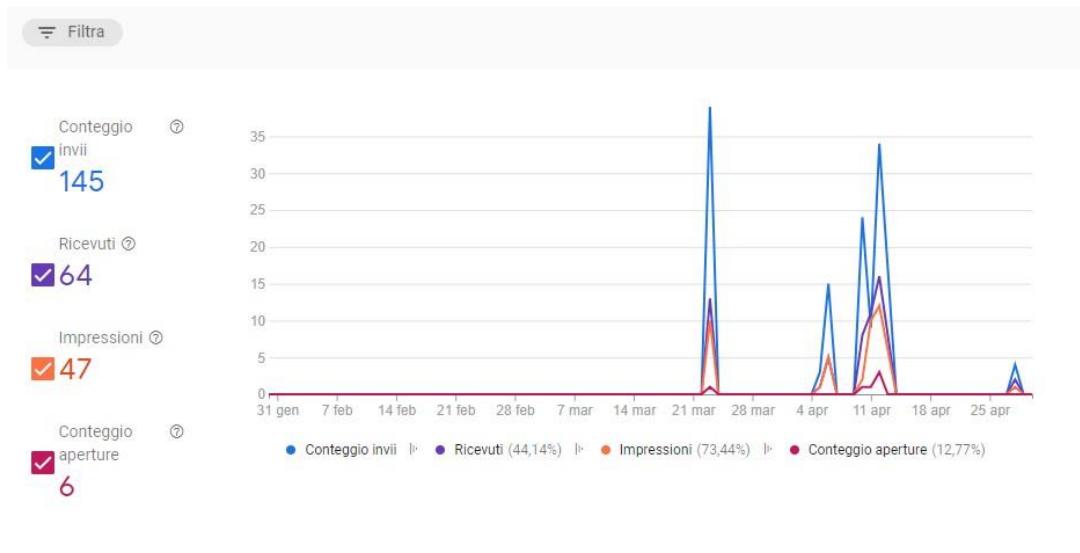
def send_push_notification(text_message, url_image):
    push_service = FCMNotification(api_key=server_key)
    title = "My IoT Home Alert!"
    message = text_message
    extra_notification_kwargs = { 'image': url_image }
    push_service.notify_topic_subscribers(message_title=title,topic_name="alert",message_body=message,extra_notification_kwargs=extra_notification_kwargs)
    print("Send Push Notification")

def water_level_alert():
    send_push_notification("Garage Flooding was detected!", home_flood_url_img)

def motion_sensor_alert():
    send_push_notification("Motion was detected!", pir_url_image)

```

**Figura 33:** Definizione delle funzioni relative al modulo Fcm



**Figura 34:** Servizio di analitica relativo al servizio FCM

Il servizio FCM (*Firebase Cloud Messaging*), permette la visualizzazione delle statistiche riguardanti le notifiche inviate. In particolare, vengono evidenziati il numero di messaggi inviati, ricevuti, il numero delle notifiche visualizzate e delle aperture.

### 7.2.3 MODULO MQTT PHAO

```
# Create MQTT client and connect to localhost
MQTT_USER = 'homeuser'
MQTT_PASSWORD = 'mqttpwauth'

client = mqtt_client.Client()
client.username_pw_set(MQTT_USER, MQTT_PASSWORD)
client.on_connect = on_connect
client.on_message = on_message
client.connect('localhost', 1883, 60)
# Connect to the MQTT server and process messages in a background thread.
client.loop_start()
```

**Figura 35:** Configurazione del MQTT client

Attraverso la libreria Mqtt Phao resa disponibile da Eclipse Foundation [9] è stato possibile definire un MQTT client autenticato con password, che impiega la porta 1883 e ha un “keep-alive” della connessione pari a 60 secondi.

```
def on_message(client, userdata, msg):
    if msg.topic == '/pir_status':
        # Look at the message data and perform the appropriate action.
        update_pir_is_not_down()
        if msg.payload == b'PIR_DETECT':
            logging.info("Alert: Pir Status: Movement Detected")
            realtime_db.child("PirMotion").set(1)
            fcm.motion_sensor_alert()
            sendImage()
            time.sleep(5)
        elif msg.payload == b'PIR_UNDETECT':
            logging.info("Pir Status: ok")
            realtime_db.child("PirMotion").set(0)
        else:
            logging.info("Unknown Message")
            realtime_db.child("PirMotion").set("UNKNOWN")
```

**Figura 36:** Gestione dei messaggi MQTT ricevuti

Nel caso in cui il messaggio sia targato con il topic “/pir\_status” ed il payload contenga il testo “PIR\_DETECT”, allora significa che il sensore PIR ha rilevato un movimento sospetto. In questo caso, il programma in Python prevede che:

1. Si aggiorni il Real Time Database, impostando la voce “PirMotion” ad 1.
2. Venga inviata una push notification allo smartphone tramite il modulo Fcm.
3. La WebCam catturi un’immagine, le attribuisca un nome univoco utilizzando l’ora (*datetime*) ed infine l’immagine venga inviata sul Firebase Storage.

```
#-*- coding: utf-8 -*-
from datetime import datetime

def sendImage():
    image_name = str(subprocess.check_output(['bash', '-c', 'echo $(date +%d-%m-%Y_%H:%M:%S)']))
    import re
    image_name = re.sub("b'", "", image_name)
    image_name = re.sub("n'", "", image_name)
    image_name = image_name[0:len(image_name)-1]
    logging.info('ImageName: ' + image_name)

    dir_file_image = "/home/pi/Desktop/images/" + image_name + ".jpg"
    bashCommand = "cd && fswebcam " + dir_file_image
    output = subprocess.check_output(['bash', '-c', bashCommand])

    #upload image on Firebase Storage
    storage.child(image_name).put(dir_file_image)
    logging.info("Image was uploaded!")
```

**Figura 37: Funzione “sendImage()”**

 25-12-2020_17:22:03	15.58 KB	image/jpeg	25 dic 2020
 28-04-2021_13:11:58	34.07 KB	image/jpeg	28 apr 2021
 28-04-2021_13:14:44	37.81 KB	image/jpeg	28 apr 2021

**Figura 38: Raccolta delle immagini su Firebase Storage**

## 7.2.4 MODULO Adafruit\_DHT

```
api_thingspeak_temp_url = 'https://api.thingspeak.com/update?api_key=RZ' &field1='
api_thingspeak_hum_url = 'https://api.thingspeak.com/update?api_key=RZ' &field2='

# Sensor should be set to Adafruit_DHT.DHT11, Adafruit_DHT.DHT22, or Adafruit_DHT.AM2302.
sensor = Adafruit_DHT.DHT11
# DHT signal sensor pir
pin = 4
# Try to grab a sensor reading.
humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)

if humidity is not None and temperature is not None:
    str_temp = str(round(temperature, 0))
    str_hum = str(round(humidity, 0))
    logging.info('Temperature= ' + str_temp + ' °C')
    logging.info('Humidity= ' + str_hum + ' %')
    realtime_db.child("Diagnosis/DHT11IsDown").set(0)
    data = {"temperature": temperature, "humidity": humidity}
    realtime_db.child("DHT11").set(data)

    res_temp = requests.get(url = (api_thingspeak_temp_url + str_temp))
    logging.info('thingspeak res temperature: ' + str(res_temp))
    time.sleep(15)
    res_hum = requests.get(url = (api_thingspeak_hum_url + str_hum))
    logging.info('thingspeak res humidity: ' + str(res_hum))
    time.sleep(15)
```

Figura 39: Funzione di lettura del sensore DHT11 e gestione dei dati

La funzione di lettura del sensore DHT11 viene eseguita tramite Threading, in loop.

Essa prevede che dopo la lettura e l'assegnamento dei valori alle variabili di temperatura ed umidità, venga aggiornato il Real Time Database di Firebase alle voci “DHT11/temperature”, “DHT11/humidty” e “Diagnosis/DHT11IsDown”. Questo permette la corretta visualizzazione dei dati e dello stato del sensore tramite la mobile app in tempo reale. Di seguito, tramite una chiamata API autenticata (HTTPS di tipo GET), viene inviata la temperatura al “field1” del canale ThingSpeak impostato. Segue poi una pausa di 15 secondi, resa necessaria dal servizio di ThingSpeak API, e poi viene inviato anche il dato relativo all’umidità. A seguito di un’altra pausa di 15 secondi, il processo si ripete.

## 7.2.5 FUNZIONE DI TRIGGER

```
new_garage_light_state = realtime_db.child("GarageLightState").get().val()

if(garage_light_state != new_garage_light_state):

    logging.info("New GarageLightState: " + str(new_garage_light_state))

    if(new_garage_light_state == 1):

        mqtt_turn_on_the_garage_light()

    else:

        mqtt_turn_off_the_garage_light()

    time.sleep(1)
```

**Figura 40:** Funzione di Trigger

La funzione di rilevamento del trigger viene eseguita tramite Threading, in loop.

Essa prevede la lettura del valore relativo alla voce “GarageLightState” del Real Time Database. Nel caso in cui si rilevi che il valore è differente rispetto a quello precedente, allora si richiamano le funzioni per richiedere al Wemos di modificare lo stato del proprio led (che simula altri attuatori, come ad esempio un relè).

```
def mqtt_turn_on_the_garage_light():

    client.publish('/pi_cmds', 'TURN_ON_THE_GARAGE_LIGHT')

def mqtt_turn_off_the_garage_light():

    client.publish('/pi_cmds', 'TURN_OFF_THE_GARAGE_LIGHT')
```

**Figura 41:** Funzioni trigger inviate tramite messaggi MQTT al topic “/pi\_cmds”

## 7.2.6 FUNZIONI DI AUTODIAGNOSI

```
webcam_check = subprocess.run(['ls', '/dev/video0'], capture_output=True, text=True).stdout

if(webcam_check != ''):
    realtime_db.child("Diagnosis/WebcamIsDown").set(0)
    logging.info("Webcam Check: ok, port: " + webcam_check[0:11])
else:
    realtime_db.child("Diagnosis/WebcamIsDown").set(1)
    logging.info("Webcam Check: Error")

time.sleep(15)
```

**Figura 42:** Funzione di WebcamCheck

La funzione di autodiagnosi relativa alla connessione fisica della webcam viene eseguita tramite Threading, in loop. Essa prevede un comando in bash che ne verifichi la connessione ed esegue un update del Real Time Database alla voce “Diagnosis/WebcamIsDown”. Infine, è previsto un delay di 15 secondi.

```
while True:

    if(pir_is_down_countdown <= 0):
        update_pir_is_down()
    if(water_level_is_down_countdown <= 0):
        update_water_level_is_down()

    #5 sec sleep
    time.sleep(5)

    pir_is_down_countdown -= 5
    water_level_is_down_countdown -= 5
```

**Figura 43:** Verifica delle connessioni MQTT relative ai due Wemos

L’ultima parte del programma prevede un loop nel quale viene verificato che i microcontrollori Wemos siano connessi alla rete. Nello specifico, tramite due countdown, viene controllato che la ricezione dei messaggi rientri in un range di 15 secondi. Tale range è stato impostato a seguito dei seguenti test:

- Verifica dell'intervallo di tempo necessario per l'instaurazione della connessione MQTT al broker ed invio del primo messaggio (circa 10 secondi)
- Verifica dell'intervallo di tempo previsto tra la ricezione di due messaggi MQTT (i timer di countdown sono diversificati a seconda del topic MQTT e di conseguenza del Wemos stesso)

Ne segue che nel caso in cui non si ricevano messaggi MQTT ogni 15 secondi, vengono aggiornati i campi “Diagnosis/WaterLevelIsDown” e “Diagnosis/PirlsDown”. In questo caso, l'applicazione mobile manderà a video lo stato “Down” dei sensori e degli attuatori dipendenti dal microcontrollore “in Down”.

### 7.2.7 FUNZIONE DI LOG

Nel caso si verificassero delle problematiche o fosse necessario eseguire degli aggiornamenti a seguito dell'auditing del sistema, è essenziale disporre di un log del programma eseguito in Python, in quanto gestisce il backend e permette di individuare possibili bachi (*o bug*) e altri dati utili.

```
import logging
import sys

logDirFile = '/home/pi/Desktop/log.txt'
formatData= '%(asctime)s; %(message)s'
logging.basicConfig(filename= logDirFile , format= formatData, level=logging.INFO)

root = logging.getLogger()
root.setLevel(logging.DEBUG)

handler = logging.StreamHandler(sys.stdout)
handler.setLevel(logging.DEBUG)
formatter = logging.Formatter('%(asctime)s - %(message)s')
handler.setFormatter(formatter)
root.addHandler(handler)

logging.info('Program Started')
```

**Figura 44:** Configurazione del logger

Il log, è stato implementato mediante i moduli "logging" e "sys" per accedere alle risorse di sistema. Nel merito della configurazione del log sono stati definiti due handler (*o gestori*) dei messaggi:

- Uno viene utilizzato per la scrittura dei dati sul file log.txt, i cui backup sono archiviati su Firebase Storage
- Uno impiegato per l'output a video mediante STDOUT (messaggi della console di terminale), per un riscontro in tempo reale e l'esecuzione di test.

Entrambi gli handler comportano una formattazione del messaggio contenente la data e l'ora come prima stringa dell'evento.

```
2021-04-28 13:06:00,335; Program Started
2021-04-28 13:06:02,580; Converted retries value: 3 -> Retry(total=3, connect=None, read=None, write=None, redirect=None)
2021-04-28 13:06:02,635; Connected with result code 0
2021-04-28 13:06:02,645; Starting new HTTPS connection (1): myiothome-12345.firebaseio.com
2021-04-28 13:06:03,216; Starting new HTTPS connection (2): myiothome-12345.firebaseio.com
2021-04-28 13:06:03,225; Starting new HTTPS connection (3): myiothome-12345.firebaseio.com
2021-04-28 13:06:03,226; Temperature= 22.0 °C
2021-04-28 13:06:03,226; Starting new HTTPS connection (1): firebasestorage.googleapis.com
2021-04-28 13:06:03,230; Humidity= 5.0 %
2021-04-28 13:06:03,243; Starting new HTTPS connection (4): myiothome-12345.firebaseio.com
2021-04-28 13:06:03,841; "POST /v0/b/myiothome-12345.appspot.com/o?name=logfiles/28-04-2021"
2021-04-28 13:06:03,858; Starting new HTTPS connection (5): myiothome-12345.firebaseio.com
2021-04-28 13:06:03,916; "PUT /Diagnosis/HeartbeatIsDown.json HTTP/1.1" 200 1
```

**Figura 45:** Estratto del file "log.txt" e formattazione degli eventi

28-04-2021_13:17:59_log.txt	146.96 KB	text/plain	28 apr 2021
28-04-2021_13:18:40_log.txt	156.66 KB	text/plain	28 apr 2021
28-04-2021_13:19:10_log.txt	160.14 KB	text/plain	28 apr 2021

**Figura 46:** Files presenti nella cartella "logfiles" dello Storage di Firebase

Il codice prevede una funzione eseguita in Threading che invia nella cartella "logfile" del Firebase Storage l'ultima versione del file log.txt, identificata dalla data e l'ora. Oltre a questo è necessario aggiornare la stringa del Real Time Database alla voce "LastLogFileName", in modo che la mobile app sia in grado di individuare l'ultima versione inviata e richiederla.

### 7.3. AUTOMAZIONE DEL SISTEMA

Al fine di rendere il sistema pronto all'uso, si è optato per più soluzioni utili all'automazione del sistema.

In fase di test la connessione ad Internet del Raspberry è stata instaurata aggiungendo il file "wpa-supplicant.conf" alla cartella di "boot". Questa procedura rende possibile l'abilitazione della connessione ssh e in seguito di VNC senza l'utilizzo di un monitor esterno.

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=IT

network={
    ssid="[REDACTED]"
    psk="[REDACTED]"
    key_mgmt=WPA-PSK
}
```

Figura 47: File wpa-supplicant.conf

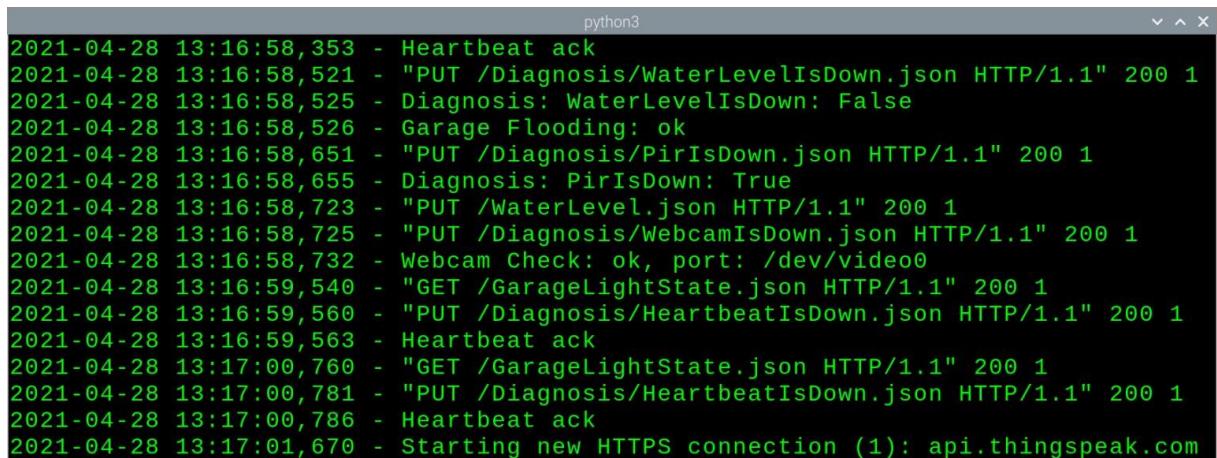
Una seconda automazione consiste nell'avvio automatico del programma in Python subito dopo la fase di boot del sistema e l'avvenuta connessione alla rete. Quanto descritto, è regolato dal file "boot\_my\_iot\_home.desktop" contenuto nella directory

“home/pi/.config/autostart”. Il file in questione schedula l’esecuzione del programma principale in Python attraverso l’applicazione Xterm, simile al terminale.



```
[Desktop Entry]
Version=1:0
Type=Application
Terminal=true
Exec=/usr/bin/xterm -fa monaco -fs 24 -bg black -fg green -hold
-e python3 /home/pi/Desktop/my_iot_home.py
Name=avvia_programma_py
Name[it_IT]=boot_myiothome
```

Figura 48: File di autostart del programma principale in Python



```
python3
2021-04-28 13:16:58,353 - Heartbeat ack
2021-04-28 13:16:58,521 - "PUT /Diagnosis/WaterLevelIsDown.json HTTP/1.1" 200 1
2021-04-28 13:16:58,525 - Diagnosis: WaterLevelIsDown: False
2021-04-28 13:16:58,526 - Garage Flooding: ok
2021-04-28 13:16:58,651 - "PUT /Diagnosis/PirIsDown.json HTTP/1.1" 200 1
2021-04-28 13:16:58,655 - Diagnosis: PirIsDown: True
2021-04-28 13:16:58,723 - "PUT /WaterLevel.json HTTP/1.1" 200 1
2021-04-28 13:16:58,725 - "PUT /Diagnosis/WebcamIsDown.json HTTP/1.1" 200 1
2021-04-28 13:16:58,732 - Webcam Check: ok, port: /dev/video0
2021-04-28 13:16:59,540 - "GET /GarageLightState.json HTTP/1.1" 200 1
2021-04-28 13:16:59,560 - "PUT /Diagnosis/HeartbeatIsDown.json HTTP/1.1" 200 1
2021-04-28 13:16:59,563 - Heartbeat ack
2021-04-28 13:17:00,760 - "GET /GarageLightState.json HTTP/1.1" 200 1
2021-04-28 13:17:00,781 - "PUT /Diagnosis/HeartbeatIsDown.json HTTP/1.1" 200 1
2021-04-28 13:17:00,786 - Heartbeat ack
2021-04-28 13:17:01,670 - Starting new HTTPS connection (1): api.thingspeak.com
```

Figura 49: Esecuzione automatica di Xterm ed informazioni di log in STDOUT

## 7.4. FUNZIONE DI HEARTBEAT

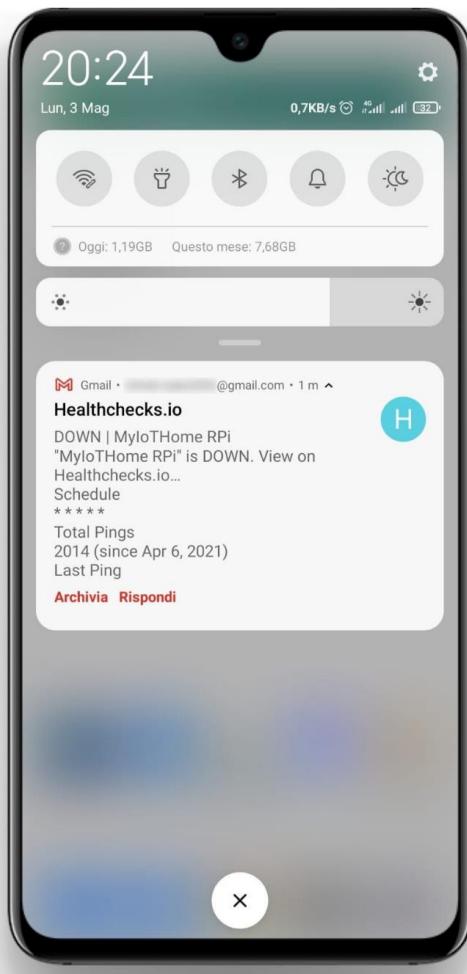
La funzione di heartbeat verifica che il Raspberry sia alimentato e connesso ad Internet tramite il servizio offerto da Healthchecks.io. Nello specifico è stato impostato un crontab job, che viene eseguito ogni minuto e che prevede una chiamata HTTPS di tipo GET all’endpoint specificato da Healthchecks.io per il progetto in questione.

```
GNU nano 3.2                               /tmp/crontab.FhiVuS/crontab

# Crontab jobs
# m h dom mon dow   command
* * * * * /usr/bin/curl --silent https://hc-ping.com/b0ff
&>/dev/null
```

**Figura 50:** Cron job impostato ogni minuto per la funzione di Heartbeat

Nel caso in cui dovessero passare oltre 2 minuti senza l'avvenuta chiamata HTTPS, allora Healthchecks.io provvede ad inviare una notifica via mail (sono configurabili anche altri canali di comunicazione, utili nel caso in cui il sistema venga gestito da più persone) come riportato nella seguente immagine:



**Figura 51:** Notifica di Healthchecks.io

## 7.5. MOBILE APP

La mobile app sviluppata in Flutter permette il controllo remoto in tempo reale della smart home. È anche possibile installare l'applicazione su tablet avente sistema operativo Android. Inoltre, Flutter permette di eseguire il deployment anche per iOS a partire dallo stesso codice sorgente.

Fondamentalmente l'app è costituita da un insieme di widgets (componenti) stateless, vale a dire statici, oppure stateful, cioè dinamici.

### 7.5.1 SCHERMATA DELLA DASHBOARD

La prima schermata a cui si viene indirizzati quando l'applicazione si avvia è la dashboard.

Questa schermata prevede l'instaurazione della comunicazione al Firebase Realtime Database e di conseguenza la creazione di un oggetto Sensor che recupera ed elabora tutti i campi presenti nel Real Time Database.

Ne consegue poi il rendering di 6 widgets SensorCard contenenti un'icona, la descrizione, lo stato del sensore o attuatore, ed infine un bottone che richiede il reindirizzamento ad un'altra schermata (ad esempio per visualizzare il grafico della temperatura oppure per osservare le foto scattate dalla webcam) o l'esecuzione di una funzione (come ad esempio il trigger dello stato relativo alla luce presente nel garage).



Figura 52: Dashboard della Mobile App

```

Sensor(Map<dynamic, dynamic> json) {
    DHT11IsDown = json['Diagnosis']['DHT11IsDown'].round();
    PirIsDown = json['Diagnosis']['PirIsDown'].round();
    WaterLevelIsDown = json['Diagnosis']['WaterLevelIsDown'].round();
    WebcamIsDown = json['Diagnosis']['WebcamIsDown'].round();
    temperature = json['DHT11']['temperature'].round();
    humidity = json['DHT11']['humidity'].round();
    waterLevel = json['WaterLevel'];
    pirMotion = json['PirMotion'];
    GarageLightState = json['GarageLightState'];
    String risk = (waterLevel == 0) ? 'LOW' : 'HIGH';
    print('WaterLevel is: ' + risk);
    String motion = (pirMotion == 0) ? 'NOT DETECTED' : 'DETECTED';
    print('Motion: ' + motion);
}

```

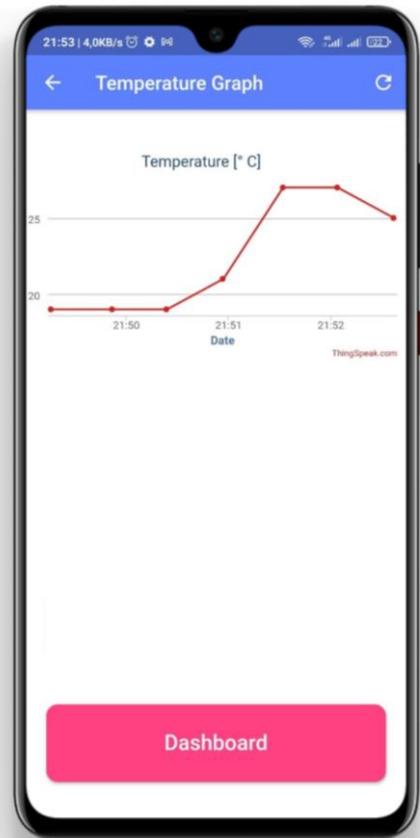
**Figura 53:** Costruttore dell'oggetto Sensor

### 7.5.2 SCHERMATE DEI GRAFICI DI THINGSPEAK

Come precedentemente anticipato, se premuto il bottone relativo ai widgets `SensorCard` della temperatura o dell'umidità, si viene reindirizzati ad una schermata che mediante richiesta HTTPS autenticata al servizio di Thingspeak permette la visualizzazione del grafico in-app.

Un alternativa a Thingspeak precedentemente testata, è stata l'implementazione dei grafici attraverso la dipendenza `charts_flutter` e lo storage dei dati nel Database di Flutter; ma i grafici con essa prodotti non erano in grado impostare al meglio l'asse X, del tempo, come fa Thingspeak, attraverso MATLAB.

Nello specifico è stato utilizzato il componente `WebView` e si decisio di prevenire l'esecuzione di chiamate HTTP



**Figura 54:** Schermata del grafico della temperatura con Thingspeak

o HTTPS all'interno dell'app attraverso i link presenti (ad esempio quello presente sulla scritta “Thingspeak.com” in basso a destra).

### 7.5.3 SCHERMATA DELLE FOTO CATTURATE DALLA WEBCAM

Premendo sul bottone relativo alla Webcam all'interno della dashboard, si viene reindirizzati alla lista di immagini scattate dalla Webcam partendo dalla più recente e arrivando alla più datata.

Nello specifico la schermata prevede la renderizzazione di un indicatore di loading durante il caricamento delle immagini archiviate nella directory principale del Firebase Storage.

Dopo il recupero dei dati relativi ai file contenuti nello Storage, attraverso l'elaborazione eseguita dal widget *RenderImages*, ogni immagine è posta all'interno di un componente *Card*, che rende nota la data e l'ora in cui è stata scattata l'immagine: due dati recuperati del costruttore di oggetti *CloudImage* a partire dal nome di archiviazione del file.

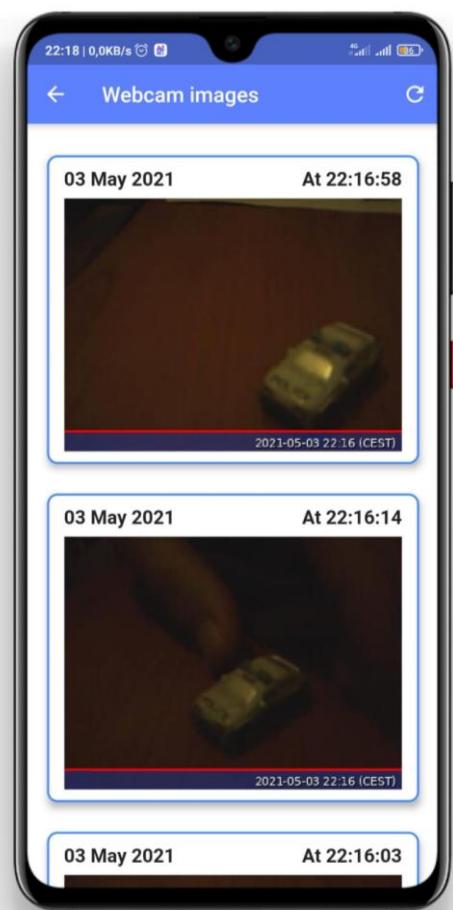


Figura 55: Schermata delle immagini della Webcam

#### 7.5.4 MENU' LATERALE

Il menù laterale, richiamato attraverso l'icona a forma di hamburger in alto a sinistra, prevede il logo dell'applicazione in alto e poi un insieme di componenti che permettono l'indirizzamento ad altre pagine:

- Dashboard → La schermata principale
- Log file → Accesso all'ultimo file di log
- Settings → Impostazione dell'app

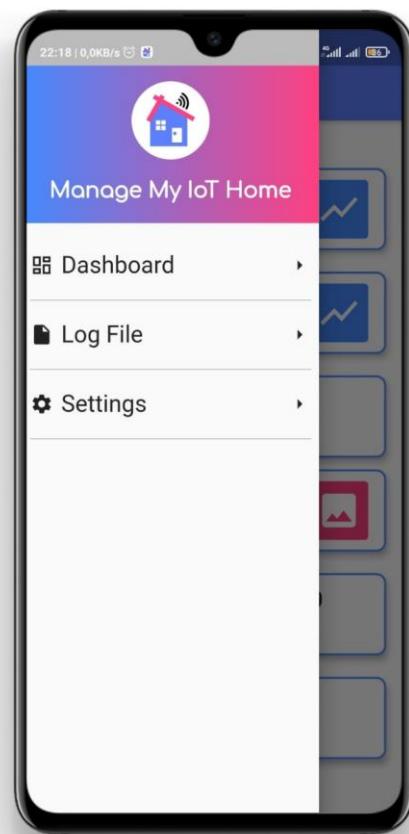


Figura 56: Menù laterale dell'app

#### 7.5.5 SCHERMATA DEI LOG

La schermata di Log dei file prevede il recupero del nome dell'ultimo file di log alla voce "LastLogFileName" del Real Time Database e di conseguenza il recupero stesso del file.

Dopodichè viene renderizzato un bottone che attraverso la dipendenza *flutter\_downloader* abilita a scaricare l'ultimo file di log all'interno della memoria di archiviazione del dispositivo. Segue una lista che rende noti gli eventi di log a partire dai più recenti. Infine appare un bottone "Add Lines!" che prevede l'aggiunta di 10 eventi di log alla lista superiore.

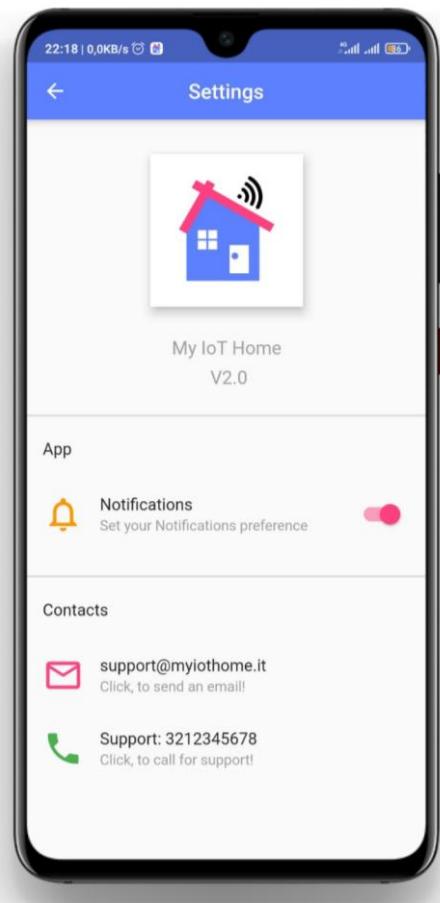


Figura 57: Schermata di Log dell'app

## 7.5.6 SCHERMATA DELLE IMPOSTAZIONI

La schermata delle impostazioni implica tre sezioni principali:

1. La prima prevede l'immagine del progetto My IoT Home e la scritta della versione
2. La seconda consiste in un toggle della preferenza delle notifiche FCM, che richiede il reindirizzamento alla dashboard, in modo che il programma principale possa essere riavviato ed eseguire la modifica richiesta di *subscribe* o *unsubscribe* al canale FCM.
3. La terza prevede il link diretto ai contatti, mail o numero di telefono, relativi all'assistenza cliente, in modo che l'esperienza utente sia resa più gradevole nel caso di problematiche o necessità di comunicazione.



**Figura 58:** Schermata delle impostazioni dell'app

```
// Start FCM
startFirebase() async {
  print('verify not');
  final prefs = await SharedPreferences.getInstance();
  final key = 'notification';
  final value = prefs.getInt(key) ?? 0;
  if (value == 1) {
    print('Notification ON');
    PushNotificationService().initialise(context);
    PushNotificationService().subscribeAlertNotification(context);
  } else {
    PushNotificationService().unsubscribeAlertNotification(context);
  }
}
```

**Figura 59:** Codice relativo alla preferenza delle notifiche

### 7.5.7 SERVIZIO DI NOTIFICHE PUSH MEDIANTE FCM

Infine, come illustrato precedentemente, in base alle preferenze di notifica si ricevono o meno le push notification relative al topic “alert”.

Nello specifico, si possono ricevere push notification che avvisano dei seguenti eventi:

- È stato rilevato un livello alto dell'acqua in garage, perciò il rischio di allagamento è imminente
- È stato rilevato un movimento attraverso il sensore di PIR ed è stata scattata un'istantanea attraverso la Webcam

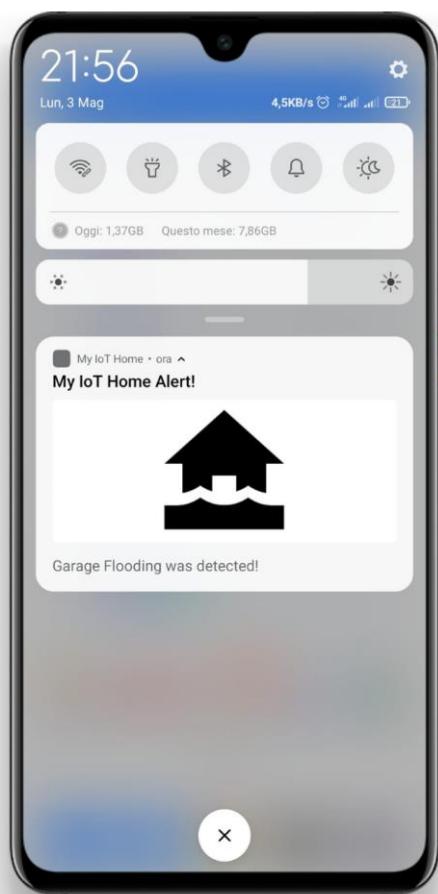


Figura 60: Garage Flooding Alert,  
FCM Push Notification



Figura 61: Motion Alert,  
FCM Push Notification

## **8. PENETRATION TEST E MESSA IN SICUREZZA**

### **8.1. SCENARIO**

Per garantire la sicurezza In ambito IoT è necessario porre l'attenzione su alcuni aspetti:

- Cose (Thing/sensori)
- Connessioni
- Configurazioni

Dunque nel caso specifico si verificherà la sicurezza dei seguenti aspetti:

- La comunicazione via protocollo MQTT tra i microcontrollori Wemos e il Raspberry (nella LAN)
- La configurazione del broker MQTT esposto dal Raspberry
- La comunicazione instaurata con i servizi web via protocollo HTTPS (Firebase, Thigspeak, Healthchecks.io, RealVNC Viewer)
- Configurazione del nome utente e password del Raspberry

### **8.2. VULNERABILITÀ DELLA CONFIGURAZIONE MQTT**

Il protocollo MQTT ed il software server Mosquitto di per sè non presentano vulnerabilità note, ma richiedono una configurazione che garantisca confidenzialità, autenticità e autenticazione, come spiegato in un articolo di Avast, società celebre nel campo della cyber security e dei software di antivirus [5].

A tal proposito, subito dopo la fase iniziale di recap del sistema della versione 1.0, si è deciso di verificare la sicurezza del sistema attraverso un penetration test e di eseguire la messa in sicurezza delle fallo riscontrate.

Nello specifico sono stati eseguiti dei passaggi a ritroso per verificare prima quali dati sono sensibili e poi quali sono le azioni che un hacker dovrebbe compiere per accedere a questi dati:

1. Analisi con Wireshark su Raspberry
2. Man in the middle con Kali OS
3. WiFi Password Cracking

### 8.3. ANALISI WIRESHARK SU RASPBERRY

Il primo passaggio eseguito prevede un'analisi di rete attraverso l'applicativo Wireshark e la selezione del flusso relativo al WiFi.

Eseguendo la cattura di rete, appaiono pacchetti contraddistinti dal protocollo MQTT relativi alla connessione del Raspberry e di un ESP8266.

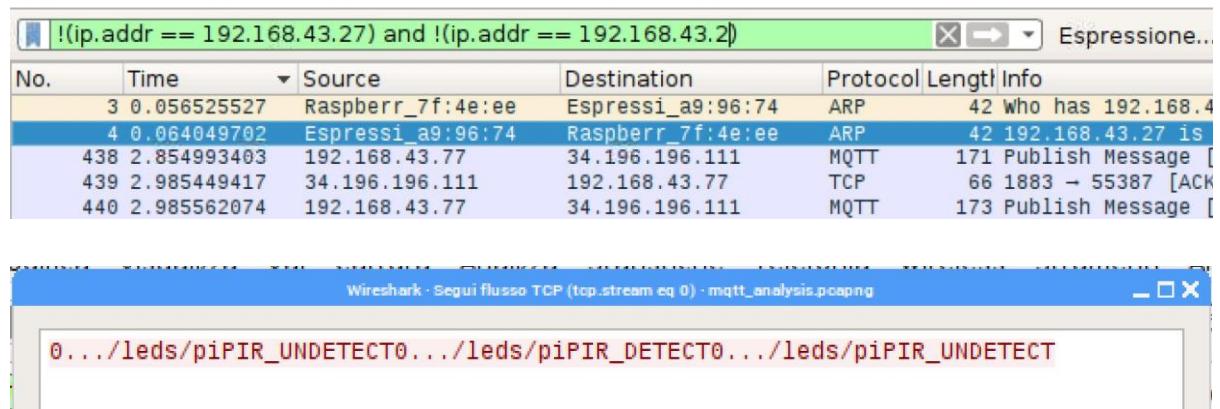


Figura 62: Payload di un pacchetto MQTT inviato dal ESP8266 al Raspberry

Emerge che un'analisi di rete eseguita da qualsiasi dispositivo connesso alla rete permette di conoscere i topics MQTT ed il payload del messaggio inviato in chiaro.

## 8.4. MAN IN THE MIDDLE CON KALI OS

Il secondo passo prevede l'esecuzione di un man in the middle eseguito con Kali OS.

L'operazione prevede che un attaccante con accesso alla rete si interponga tra la comunicazione che avviene tra il Raspberry (che ha funzione di server) ed un ESP8266 (con funzione di client) mediante IP spoofing [19].

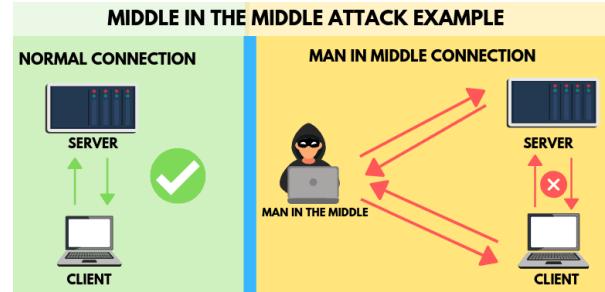


Figura 63: Attacco Man in the middle (MITM)

L'attacco passerà pressoché inosservato alle vittime, specie in questo caso in cui i due dispositivi in questione vengono controllati solo remotamente e dunque la rete non subirà particolari rallentamenti.

```
# Enable port forwarding
sysctl -w net.ipv4.ip_forward=1

# Spoof connection between Victim and Router
# Note: Run this command in a new terminal and let it running
arpspoof -i [Network Interface Name] -t [Victim IP] [Router IP]

# Same step but inverted (nope, it's not the same ...)
# Note: Run this command in a new terminal and let it running
arpspoof -i [Network Interface Name] -t [Router IP] [Victim IP]

# Network dump with Wireshark
tshark -r network.pcap

# Disable port forwarding once you're done with the attack
sysctl -w net.ipv4.ip_forward=0
```

Figura 64: Comandi eseguiti da terminale

## 8.5. WIFI PASSWORD CRACKING

Mediante i comandi precedentemente illustrati e performando una scansione di pacchetti mediante Wireshark si possono ottenere i dati sensibili trasmessi mediante il protocollo MQTT. Comunque per eseguire il Man In The Middle è necessario essere connessi alla rete e dunque si rende necessaria l'esecuzione del password cracking relativo al WiFi.

Le operazioni in questione sono state svolte nell'arco temporale di circa 30 minuti, si ricorda inoltre che è possibile accelerare il processo nel caso si disponga "Jammer", che attraverso il disturbo delle connessioni WiFi permette di disconnettere i dispositivi, forzando così la riconnessione (solitamente automatica) e di conseguenza le autenticazioni 4-way-handshake che sfruttano più volte la stessa chiave crittografica (inviata nel terzo messaggio dell'autenticazione 4-way-handshake). Questo permette la crittoanalisi della password di accesso attraverso l'utilizzo di un dizionario o mediante un attacco di forza bruta.

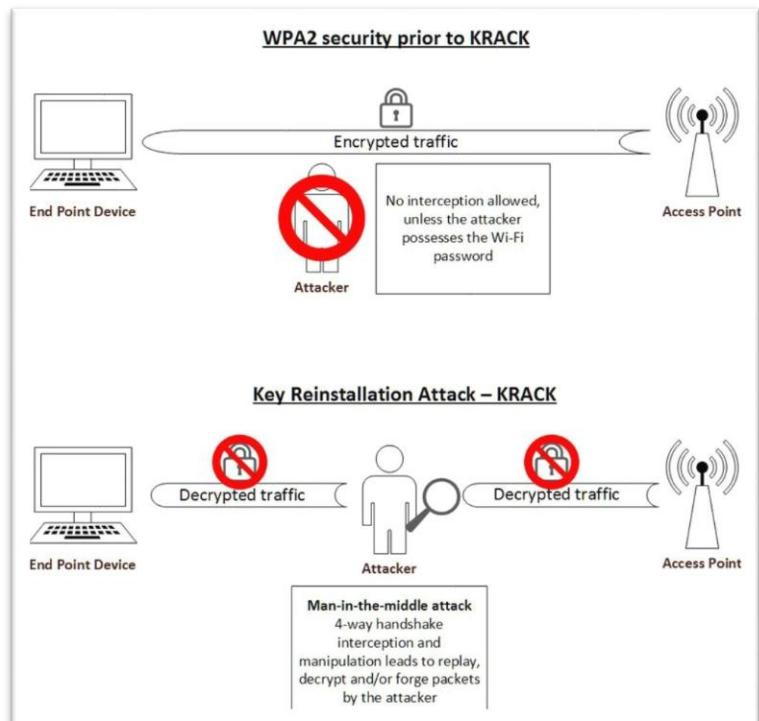


Figura 65: Vulnerabilità del 4-way-handshake

A tal riguardo, l'ENSIA (*European Union Agency For Cybersecurity*) pone l'attenzione sul fatto che questo tipo di attacco necessiti della presenza fisica dell'attaccante nella zona circostante all'AP (*Access Point*) e che solitamente i dati vengono crittografati mediante ulteriori layer di sicurezza, come ad esempio l'SSL/TLS [11].

Per completare il penetration test è stato utilizzato il tool Aircrack-ng, ampiamente documentato sul sito ufficiale di Kali OS [20].

In seguito all'avvenuta cattura degli oltre 30 mila pacchetti necessari per il cracking della password, è possibile testare il dizionario scelto e individuare la password, qualora fosse contenuta nel dizionario.

```
Aircrack-ng 1.6

[00:00:00] 1/3 keys tested (8.48 k/s)

Time left: 0 seconds          33.33%

KEY FOUND! [REDACTED]

Master Key      : EF 07 D9 8E 5B 0B 02 5F AA F8 F8 C5 F8 67 D3 1D
[REDACTED]

Transient Key   : 55 90 D8 AE 00 C7 CF 9E CF E7 B6 E8 73 26 22 03
FB 0A 4D F7 20 86 D3 5C F4 55 45 73 82 BC 52 BE
[REDACTED]

EAPOL HMAC     : AA 6A 33 3B 6E 81 F2 EF B3
[REDACTED]
```

**Figura 66: Aircrack WiFi Password**

## 8.6. PROBLEMATICHE RISCONTRATE

Per velocizzare la ricerca in questa fase di test, la password utilizzata è stata inserita in un file di testo, impiegato come dizionario. Comunque è da notare che la stessa password è stata individuata anche all'interno di un dizionario online del peso di circa 9 GB, quindi è stato verificato che essa non garantisce un buon livello di sicurezza.

Ne consegue che un utente malintenzionato può realmente venire in possesso dei dati sensibili trasmessi via protocollo MQTT senza autenticazione ed operare altri crimini informatici.

## 8.7. SOLUZIONI ADOTTATE E ACCORGIMENTI FUTURI

Per risolvere la problematica relativa alla comunicazione MQTT non autenticata esistono almeno due soluzioni:

1. Configurare il broker MQTT in modo che esegua un'autenticazione mediante password dei clients
2. Impostare al broker MQTT un layer SSL/TLS e utilizzare certificati di autenticazione

Momentaneamente si è optato per la prima opzione, in quanto la configurazione richiesta è rapida da attuare, e allo stesso tempo prevede che un malintenzionato deciso a connettersi al broker debba conoscere username e password. Nonostante ciò, se il malintenzionato dovesse riuscire ad eseguire il WiFi password cracking ed il man-in-the-middle attack, allora con l'utilizzo di un jammer potrebbe determinare la disconnessione dell'ESP8266 per un tempo superiore al keepalive di 60 secondi e dunque generare la trasmissione in chiaro delle credenziali e venirne in possesso tramite sniffing.

L'opzione 2 è la più sicura, in quanto anche su un malintenzionato dovesse accedere alla LAN e catturare i pacchetti, non avrebbe modo di leggere il payload in mancanza del certificato di autenticazione. Sicuramente in una versione distribuita al pubblico sarà necessario investire ulteriore tempo e risorse per implementare il layer TLS (sono già stati eseguiti alcuni test). Benché il rischio sia ridotto alla LAN, è essenziale porre rimedio ad eventuali falle nella rete per non comprometterla.

Un altro accorgimento futuro richiederà la modifica delle Firebase rules, che al momento sono impostate in modalità di test, permettendo lettura e scrittura a chiunque abbia accesso al link del progetto e delle credenziali (che già di per sé offrono un buon grado di autenticazione). Bisognerà quindi implementare l'autenticazione lato server autorizzando solo il Raspberry e la Mobile App.

## 8.8. SCELTA DI PASSWORD AD ALTO VALORE DI ENTROPIA

Oltre al metodo di cracking precedentemente illustrato attraverso l'utilizzo di dizionari, un malintenzionato potrebbe eseguire anche attacchi di brute force con password autogenerate. A fronte di grandi potenze di calcolo, l'unico modo per difendersene è sfruttare il principio dell'entropia delle password. Nello specifico il grado di entropia di una qualsiasi stringa si calcola secondo la formula:

➤  $H = \log_2(N^L) = L \times \log_2(N)$

Dove:

$H$  = Entropia

$\log_2$  = logaritmo in base 2

$N$  = Numero di simboli possibili

$L$  = Lunghezza della chiave

Ne segue che creando una password in codice ASCII (spazi compresi) il numero di simboli  $N$  è pari a 95, mentre ( $L = x$ ) è variabile, come mostrato nel grafico:

La funzione presenta un andamento direttamente proporzionale alla lunghezza della password.

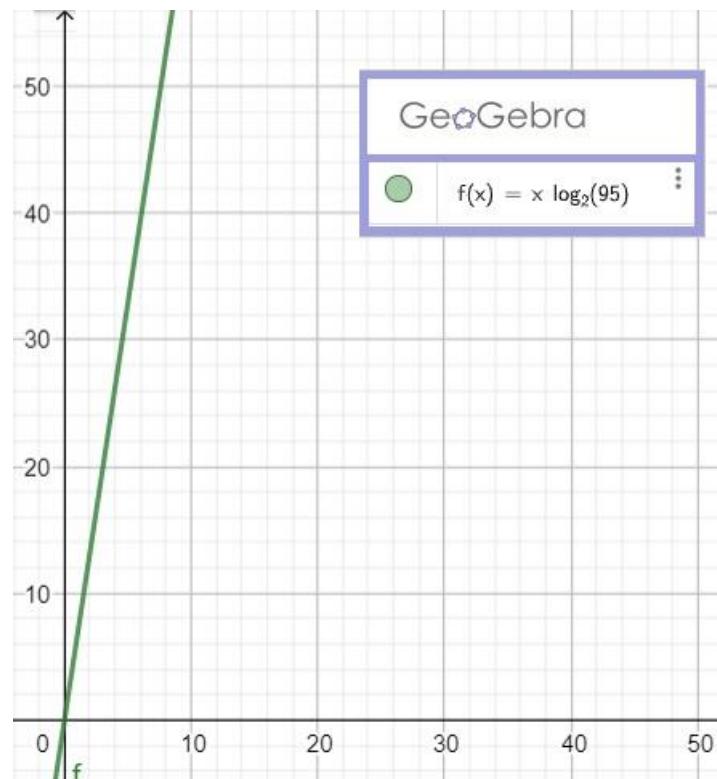


Grafico 3: Funzione relativa all'entropia di una password in codice ASCII (Geogebra.org)

Assumendo che la legge di Moore (il raddoppio della capacità di calcolo ogni due anni) sia rimasta valida fino al 2020, si presuppone che una macchina possa valutare  $10^{12}$  possibilità al secondo.

Nello specifico si calcola il tempo necessario per decrittare una password codificata in ASCII lunga 8 caratteri e una lunga 20 caratteri.

Per valutare il numero di disposizioni possibili (con ripetizione) attraverso codifica ASCII ( $N = 95$ ) si fa riferimento alla seguente formula:

$$\triangleright D_{N,L} = N^L = 95^L$$

**Dove:**

$D$  = Numero di distribuzioni

$N$  = Numero di simboli possibili

$L$  = Lunghezza della chiave

Mentre per valutare il tempo di decrittazione si dividerà  $D_{N,L}$  per il numero di operazioni eseguite in un intervallo temporale (con ordini di grandezza proporzionati al risultato).

	PW da 8 caratteri	PW da 20 caratteri
$H = \text{Entropia} = \log_2(N^L)$	$\log_2(95^8) = 53,56$ [bit]	$\log_2(95^{20}) = 131,40$ [bit]
$\text{Distribuzioni}^R = D_{N,L} = N^L = 95^L$	$95^8 = 6,63 \cdot 10^{15}$ [casi]	$95^{20} = 3,58 \cdot 10^{39}$ [casi]
$\text{Tempo di decrittazione} = T$	$D_{N,L} / (10^{12} \cdot 60) = 6,63 \cdot 10^{15} / (10^{12} \cdot 60) = 100,57$ [minuti]	$D_{N,L} / (10^{12} \cdot 3600 \cdot 24 \cdot 365,25) = 3,58 \cdot 10^{39} / (10^{12} \cdot 3600 \cdot 24 \cdot 365,25) = 1,14 \cdot 10^{20}$ [anni]

Tabella 4: Valutazione del tempo di decrittazione di una password codificata in ASCII

Dai calcoli effettuati, ne segue che una password da 8 caratteri codificata in ASCII è facilmente crackabile tramite un attacco di forza bruta a generazione casuale.

Diversamente, una password da 20 caratteri codificata in ASCII richiederebbe un tempo di elaborazione superiore di 10 ordini di grandezza rispetto a quello dell'età dell'universo (circa  $1,4 \times 10^{10} = 14$  miliardi di anni [18]).

In linea con i dati riscontrati, oggi gli esperti di sicurezza consigliano di utilizzare password con almeno 128 bit (almeno 20 caratteri) o 256 bit di entropia.

Seguono gli elementi del sistema My IoT Home V2.0 che dovrebbero rispettare questo principio:

- Password di accesso al Raspberry (e modifica del nome utente)
- Password di accesso al servizio RealVNC Viewer
- Password di autenticazione del broker MQTT
- Certificato di crittografia per una comunicazione MQTT criptata (quando verrà implementato)

Nonostante questo, secondo alcune ricerche i moderni computer quantistici, che hanno infranto la legge di Moore, riescono a decrittare in relativamente poco tempo anche password con livello di entropia pari a 256 bit. La soluzione finale quindi consiste nell'utilizzo di:

- Password ad altissima entropia, archiviabili tramite gestori di password
- Metodi di autenticazione a 2 o più fattori

Ne deriva che il servizio più protetto da questo tipo di attacco, nel progetto My IoT Home V2.0, consiste in RealVNC Viewer, in quanto richiede un'autenticazione a due fattori.

## 8.9. IL FATTORE UMANO ED IL SISTEMA

Un fattore che incide profondamente sulla sicurezza di ogni sistema informatico è quello umano. Di fatto anche il sistema più protetto, sia fisicamente che lato software, potrebbe essere messo a dura prova dal “social engineering”. Come spiega l'ex hacker più famoso del mondo Kevin Mitnick nel suo libro “L'arte dell'inganno”, l'anello

più debole di tutta la catena è l'uomo. Da queste considerazioni nasce la necessità di formare il proprio cliente e i collaboratori riguardo alle modalità con cui difendersene [15].

## 9. PIANO DI INDIRIZZAMENTO

### 9.1. CABLAGGIO STRUTTURATO

Siccome si presuppone che il sistema si andrà a configurare su una rete (è possibile che sia già esistente) è necessario accertarsi che il cablaggio rispetti le normative vigenti in Europa rispettando lo standard EN50175 [fare riferimento alla sezione “NORMATIVE”].

### 9.2. PROPOSTA DI INDIRIZZAMENTO

Nonostante il sistema prototipale abbia utilizzato l'hotspot del telefono per garantire la connessione a Internet, è bene progettare un piano di indirizzamento statico per la realizzazione di una nuova rete o per la modifica dello stato di una rete già presente.

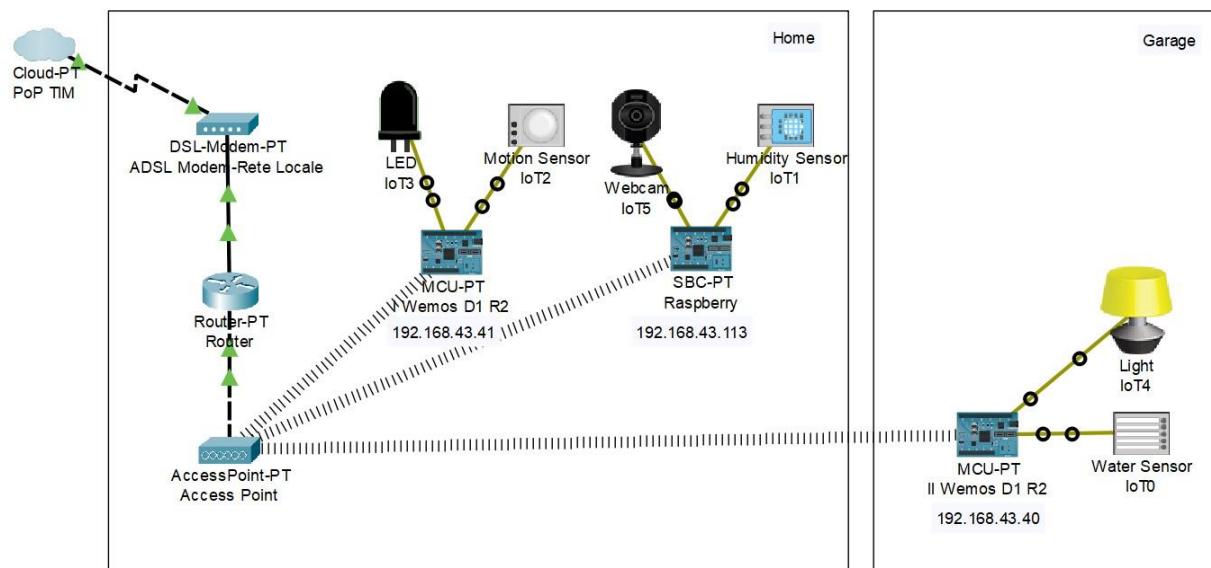


Figura 67: Cablaggio su Cisco Packet Tracer

Su Packet Tracer è stata rappresentata una rete connessa ad Internet mediante connessione ADSL ed è stato impiegato un AP con servizio DHCP (simulando quanto accade con l'hotspot allo stato attuale) con tecnologia IEEE 802.11n, in modo da garantire frequenze di lavoro a 2.4GHz e 5GHz (non viene impiegato lo standard IEEE 802.11ac perchè prevede solo la frequenza di lavoro a 5GHz). Si noti comunque che nel caso del progetto in questione si rende necessaria una configurazione statica da eseguire in loco a seconda della rete già presente.

Si propone dunque il seguente piano di indirizzamento:

DISPOSITIVO	IP	SUBNET	GATEWAY
Router F0/0	192.168.0.1	255.255.255.0	//
Router F0/1	192.168.43.254	255.255.255.0	//
I Wemos D1 R2	192.168.43.41	255.255.255.0	192.168.43.254
Raspberry	192.168.43.113	255.255.255.0	192.168.43.254
II Wemos D1 R2	192.168.43.40	255.255.255.0	192.168.43.254

RANGE IP LIBERI	SUBNET
192.168.43.1 - 192.168.43.39	255.255.255.0
192.168.43.42 - 192.168.43.112	255.255.255.0
192.168.43.114 - 192.168.43.153	255.255.255.0

**Tabelle 5-6: Tabelle di indirizzamento**

DESTINAZIONE	CIDR	NEXT HOP	INTERFACCIA
192.168.43.1 - 192.168.43.253	/24	*	S
default	/0	//	N (Verso Internet)

**Tabella 7: Tabella di routing**

## 10. MESSA IN FUNZIONE SUL CAMPO

Sono state valutate diverse tecnologie per la comunicazione wireless, ed una delle motivazioni per cui si è optato per la comunicazione WiFi su un canale a 2.4GHz è certamente la possibilità di espandibilità del sistema garantita dallo standard IEEE 802.11n (tale standard permette anche la comunicazione a 5GHz).

Lo standard in questione prevede che la comunicazione si svolga nella banda ISM (*Industrial, Scientific, Medical*) che va da 2,4 a 2,4835 GHz, per un totale di 83,5 MHz o anche in banda unlicensed, ad un data rate massimo di 600 Mbit/s (con tecnologia MIMO). La banda totale a disposizione è suddivisa in 13 canali radio con banda a 20 MHz, per cui  $f_c = 2407 + 5 * \text{Num canale [MHz]}$ . [16]

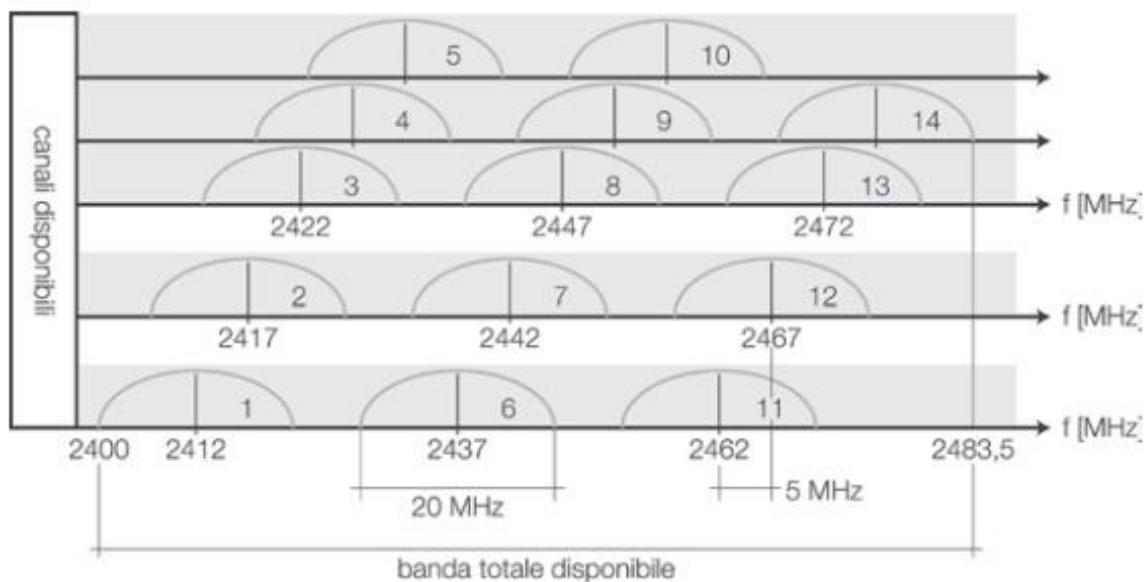


Figura 68: Canali radio disponibili in Europa per il Wi-Fi nella banda ISM 2,4 GHz

Come mostrato in figura, in Europa sono disponibili 13 canali per eseguire la trasmissione. In base alla presenza di altri AP nella zona circostante e dell'intensità del loro segnale è buona norma selezionare il canale meno soggetto ad interferenze [21].

## 10.1. CORRETTA CONFIGURAZIONE DELL'AP

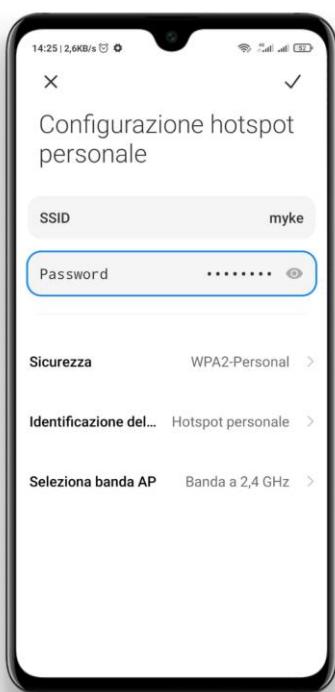


Figura 69: Configurazione Hotspot con standard IEEE 802.11n

In fase di test del sistema è stato utilizzato l'hotspot del telefono, paragonabile alle funzionalità offerte da una "saponetta" con funzionalità di AP. In alternativa, a seconda del contesto in cui opererà il sistema, si potrebbe eseguire l'accesso tramite una WLAN esposta da un router dotato di funzionalità di AP.

Qualunque sia il caso tra i precedenti, è buona norma verificare che la comunicazione WiFi sia configurata correttamente, allo scopo di evitare interferenze o di comunicare su canali già utilizzati da altri AP.

Per eseguire la verifica della copertura della rete WLAN si è optato per l'utilizzo dell'applicativo inSSIDer sviluppato da MetaGeek, LLC.

Dall'esecuzione della scansione radio alla ricerca di AP WiFi, emerge il seguente grafico nel dominio della frequenza:

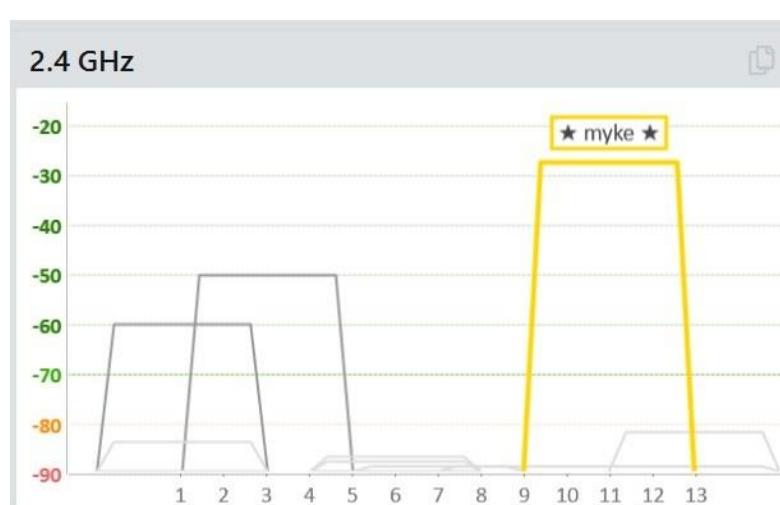


Figura 70: Scansione radio alla ricerca di AP WiFi

Si comprende dunque che il canale su cui opera la rete con SSID "myke" è l'undicesimo (11°), dunque  $fc = 2407 + 5 * 11 = 2462$  [MHz]. Dal grafico emerge inoltre che la rete opera sulla stessa banda di altri 2 canali, ma l'interferenza che determinano è minima. Nello specifico, il livello di potenza ricevuta , secondo il parametro RSSI (*Received Signal Strength Indicator*) espresso in dB differisce di oltre 50 [dB], un parametro che è sufficiente a garantire la corretta ricezione dei pacchetti inviati nella WLAN.

Oltre al grafico nel dominio della frequenza, inSSIDer permette di verificare la potenza del segnale nel dominio del tempo, e il grafico sotto riportato riscontra la stabilità del segnale intorno ai -25[dB]:



**Figura 71:** Ulteriori dettagli sul segnale WiFi

InSSIDer rende noto anche il tipo di protezione impiegato dall'AP, vale a dire la WPA2-Personal ed il Massimo Data Rate registrato, pari a 192,6 [Mbps] (si tenga conto che non è stata impiegata la tecnologia MIMO). In definitiva, tutti i parametri riscontrati sono da considerarsi idonei per l'impiego che My IoT Home V2.0 fa della WLAN.

## **10.2. POSSIBILE PROBLEMATICA: COPERTURA DELLA WLAN**

Siccome la copertura della WLAN varia in base allo scenario di progetto, è possibile che in determinate circostanze la connessione tra il Raspberry e i microcontrollori dotati di ESP8266 non possa instaurarsi per debolezza del segnale WiFi o per la schermatura di un luogo (ad esempio il microcontrollore Wemos potrebbe trovarsi in un garage schermato).

## **10.3. POSSIBILI SOLUZIONI**

A seconda del problema riscontrato e dello scenario stesso, si possono percorrere le seguenti soluzioni:

- La prima opzione, nel caso in cui la problematica fosse dovuta alla debolezza del segnale, consiste nell'installare un repeater WiFi, che agendo sul livello 1 della pila ISO/OSI (livello fisico) rigenera il segnale ampliando l'area di copertura della WLAN.
- La seconda opzione, nel caso in cui la schermatura impedisse ugualmente la comunicazione WiFi, consiste nell'aggiungere al microcontrollore un modulo GSM, in modo che possa direttamente connettersi all'API di Thingspeak e comunicare i dati con protocollo MQTT o HTTPS. Conseguentemente, il Raspberry avrà modo di recuperare i dati mediante lo stesso API e di reindirizzarli al Firebase Realtime Database mediante codice in Python.
- La terza opzione potrebbe consistere nell'instaurazione di una connessione seriale via cavo tra i due dispositivi, mediante cavo USB – USB-micro. Di conseguenza, nel programma in Python si utilizzerebbero i moduli relativi all'input seriale al posto dell'MQTT.

## 11. IMPLEMENTAZIONI FUTURE

### 11.1. SENSORI E ATTUATORI FISICI

Nelle future versioni del progetto My IoT Home si è pensato di inserire nuovi sensori, attuatori o microcontrollori, tra questi:

- Il Sonoff, è un microcontrollore dotato di ESP8266 che costa circa 5€. È un dispositivo di ridotte dimensioni e mediante protocollo MQTT agisce da interruttore per dispositivi che di norma non sono smart (ad esempio un lampadario) [28].
- Il sensore DHT22, è un sensore di temperatura e umidità dotato di una maggiore sensibilità rispetto al DHT11. Nello specifico permetterebbe letture di temperatura da -40 a 80 [°C] con  $\pm 0,5$  [°C] di sensibilità, e lettura di umidità nel range 0-100% con un'accuratezza del 2-5%. il suo costo si aggira intorno ai 10€.
- Una webcam ad una risoluzione minima di 1280x720 pixel. Una funzionalità essenziale, è che disponga della visione notturna. Si potrebbe decidere di impiegare anche una webcam progettata unicamente per Raspberry, il costo si aggira intorno ai 25€. L'utilità di questa webcam potrebbe essere sfruttata appieno mediante le librerie in Python relative all'intelligenza artificiale, permettendo di avvisare l'utente mediante notifica push solo quando è realmente necessario e non ad ogni movimento captato (esempio tenda mossa dal vento).

## **11.2. SEGNALE GPS COME TRIGGER**

Un'implementazione futura realizzabile attraverso le dipendenze di Flutter consiste nel determinare costantemente la geolocalizzazione dello smartphone dell'utente in background, sfruttando questi dati come trigger per un evento. Questo permetterebbe ad esempio di accendere le luci di casa e di inviare un comando al termostato non appena l'utente torna a casa. In una parola: comodità!

## **11.3. TEXT TO SPEECH E ASSISTENTI VOCALI**

Un'altra implementazione realizzabile tramite le dipendenze open-source di Flutter consiste nella possibilità di integrare un assistente vocale in-app. Dunque premendo un bottone l'utente potrebbe ordinare il trigger di un evento o ancora potrebbe chiedere a voce i dati relativi della propria smart home.

Una volta sviluppato questo codice in Python sul Raspberry, sarebbe possibile implementare tale funzione anche per gli assistenti vocali Google Home e Amazon Alexa: dispositivi che garantiscono un buon livello di sicurezza.

## **11.4. AUTENTICAZIONE TRAMITE FIREBASE AUTH**

Nel caso di distribuzione su grande scala e l'eventuale modularizzazione del sistema, sarà necessario eseguire un'autenticazione tramite Firebase Auth, implementabile sull'app in Flutter. Ne conseguirebbe che oltre ad implementare un'autenticazione i dati di ogni sistema verrebbero associati ad un determinato profilo.

Nel caso in cui il sistema fosse distribuito solo a privati, l'implementazione sarebbe ugualmente necessaria. In quanto determina l'autenticazione dell'utente stesso. In questo modo l'apk dell'applicazione sarebbe installabile da tutti coloro che possaggono il file, ma soltanto l'utente finale potrà accedere alla dashboard personale.

## **11.5. HEARTBEAT CON FCM E LINUX VM**

Allo stato attuale il sistema prevede l'impiego del servizio di Healthchecks.io per la funzione di heartbeat. Dunque se il Raspberry andasse in Down, arriverebbe una e-mail. Un servizio più uniforme ed user friendly prevederebbe invece l'emulazione dei checks svolti da Healthchecks.io attraverso un programma eseguito su una macchina virtuale Linux remota (ad esempio utilizzando Google Cloud Platform) ed il conseguente invio di push notification mediante Firebase Cloud Messaging (FCM) gestite dall'app in Flutter.

## **11.6. WEB APP**

Dal 2021, Flutter 2 permette la build del codice sorgente anche con JavaScript, abilitando la creazione di una WebApp simile alla versione per Android e iOS. Momentaneamente non tutte le dipendenze utilizzate supportano questa funzione, ma in futuro potrà essere un'implementazione che aggiungerà valore al sistema, rendendolo più flessibile alle esigenze del cliente.

## **12. NORMATIVE, STANDARD E LICENZE**

### **12.1. NORMATIVE SULLA PRIVACY: GDPR UE 2016/679 E DLGS 196/2003 30 GIUGNO 2003 n.196**

I'UE (*Unione Europea*) protegge il diritto alla privacy dei cittadini degli stati membri mediante il GDPR (*General Data Protection Regulation*) UE 2016/679 [6]. Similmente, il parlamento italiano l'8 maggio 2014 ha eseguito un'aggiunta riguardante il Dlgs 196/2003 del 30 giugno 2003, prevedendo che i siti e di conseguenza anche le mobile app che fanno utilizzo dei cookies, devono esplicitamente fornire un link alla privacy policy spiegando l'eventuale utilizzo di cookies tecnici, analytics (analisi del sito) e se le informazioni vengono fornite a "terze parti", come reso noto dal Garante per la protezione dei dati personali [12].

Ne consegue che la commercializzazione della mobile app richiederebbe l'adesione a queste normative indicando che i contenuti sono (al momento) analizzati dalla piattaforma Firebase, mediante Google Analytics e che i dati estratti vengono utilizzati per migliorare il servizio offerto al cliente e che non sono distribuiti a terzi.

### **12.2. DISTRIBUZIONE SU PLAY STORE E APP STORE**

In osservanza delle normative citate precedentemente, la distribuzione dell'app su Google Play Store e App Store richiede, il link alla privacy policy pubblicamente consultabile.

Un altro elemento tenuto in considerazione per lo sviluppo dell'app riguarda l'elenco dei protocolli utilizzati. Siccome il file necessario per scaricare l'applicazione sull'App Store risiederà su server stabiliti negli USA, è necessario rispettare anche le

normative vigenti in tale luogo fisico. Ne consegue che l'unico protocollo per cui non è necessario fare dichiarazioni, velocizzando dunque le pratiche, è l'HTTP(S), perchè riconosciuto come standard mondiale. Questo è stato il motivo principale per cui a priori si è deciso che l'applicazione mobile si sarebbe servita esclusivamente di tale protocollo [4].

L'ultimo documento necessario per la distribuzione su App Store riguarda la dichiarazione della proprietà intellettuale del logo utilizzato nell'app.

### **12.3. MODIFICA DEL PROGRAMMA “ADAFRUIT MQTT”**

Il codice scritto in Arduino per i due microcontrollori Wemos, è stato distribuito da Adafruit Industries con licenza MIT [22], con la limitazione di dover inserire un commento nell'header contenente i crediti. Tale licenza prevede anche che nel caso in cui il codice modificato dovesse diventare open-source, esso sarà distribuito con licenza MIT o compatibile.

### **12.4. FLUTTER E PACKAGES**

Lo scopo principale dei packages distribuiti su Flutter Pub è la condivisione open di codice, librerie, file e immagini secondo la privacy policy di Flutter Pub stesso [24].

Ne segue che ogni dipendenza ha associata una certa licenza resa nota in genere sul repository pubblico ed hostato su GitHub. Dunque, qualora l'app venisse distribuita su Google Play Store e App Store, sarebbe necessario includere i riconoscimenti ed il tipo di licenza utilizzata all'interno di una schermata in-app.

## **12.5. NORMATIVA EUROPEA RELATIVA AL CABLAGGIO STRUTTURATO: CABLAGGIO EN50175**

Siccome si presuppone che il sistema si andrà a configurare su una rete (è possibile che essa sia già esistente e configurata) è necessario accertarsi che il cablaggio rispetti le normative vigenti in Europa secondo lo standard EN50175, derivato da quello internazionale TIA/EIA 568A-568B (standard statunitense) [17].

## **13. ASSISTENZA CLIENTI E AUDITING**

### **13.1. COME RISALIRE AL PROBLEMA**

Nel caso dovessero sorgere delle problematiche per risalire al problema e risolverlo, al fine di rendere la risoluzione il più immediata possibile, sono state stabilite le seguenti pratiche di troubleshooting:

1. Download dei log attraverso l'accesso alla console di Firebase e analisi dei log
2. Nel caso il problema sia risolvibile da remoto, allora si rende necessario l'accesso mediante RealVNC Viewer
3. Osservazione in loco del sistema, alla ricerca della problematica (problemi di alimentazione, danneggiamento dei dispositivi, corruzione SD card, ecc.)
4. Nel caso in cui si fosse corrosa l'SD card (problematica comune ai Raspberry) allora sarebbe necessario sostituirla. A tal proposito, la versione commerciale del sistema prevede la creazione una SD di backup da provvedere al cliente, incidendo notevolmente sull'abbassamento dei tempi di risoluzione.
5. Sostituzione dell'intero sistema

### **13.2. AGGIORNAMENTI PERIODICI DEL SISTEMA**

Per garantire la sicurezza del sistema è necessario aggiornare i codici scritti (ogni 6 mesi circa) in Python e Flutter implementando le ultime versioni delle dipendenze sul Flutter Pub o l'ultima versione dei moduli utilizzati con Python. È un'azione volta sia a garantire le ultime patch disponibili, ma anche ad aggiornare il codice stesso.

Oltre a questi aggiornamenti periodici se ne prevedono altri “straordinari” dovuti alle richieste del cliente o all'audit (valutazione tecnica) relativo ai dati analitici raccolti, come ad esempio i rapporti raccolti dalla piattaforma di Firebase.

## 14. MARKETING, INVESTIMENTI E PREVENTIVI

### 14.1. LOGO, SITO WEB E VIDEO ILLUSTRATIVO

Al fine di pubblicizzare e rendere note le caratteristiche del sistema si è deciso di realizzare un logo che possa identificare il progetto My IoT Home. Lo stesso logo è stato poi impiegato per l'applicazione:

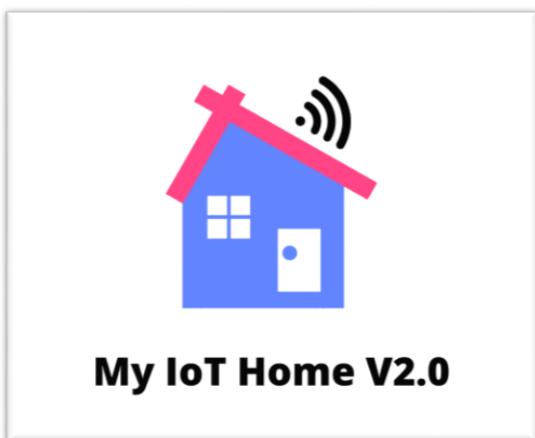


Figura 72: Logo del sistema My IoT Home V2.0

Si è anche pensato, in futuro, di acquistare un dominio che comprenda il servizio di posta elettronica. In questo modo si potrà, sia avere un sito online in cui i clienti interessati possono ricevere informazioni, sia un recapito utile per l'assistenza.

Si propongono più tecniche volte a pubblicizzare il prodotto:

- La realizzazione di un video illustrativo, simile a quello realizzato per la prima versione del prodotto (<https://youtu.be/B1fcfAcfR64>)
- Campagne pubblicitarie a basso costo, mirate a determinate fasce di clienti (profilati dalle piattaforme social)

## **14.2. OPEN SOURCE E CODECANYON**

Attualmente si sta valutando se rendere open-source il codice sorgente tramite una licenza permissiva quale la MIT (permette che il codice sorgente possa essere riutilizzato e ridistribuito solo mediante la stessa licenza) [22], in modo che il progetto possa crescere ed essere supportato da una community in futuro. Farlo permetterebbe anche di identificare eventuali falle di sicurezza.

Si è inoltre pensato di distribuire una versione a pagamento su Codecanyon.net, avente funzionalità aggiuntive, in modo da ottenere delle risorse per finanziare lo sviluppo del progetto.

## **14.3. VALUTAZIONE DEL PREVENTIVO**

Per rendere il progetto commercializzabile è stato necessario redarre un preventivo (allegato denominato “DIMITRIU\_MIHAIL\_ELABORATO\_ALLEGATO\_04.zip”). Al momento ne è stato stilato uno a scopo didattico che tenga conto del costo componenti (includendo aggiunte, quali la stampa 3D di tre box per i dispositivi) e del tempo totale dedicato alle varie fasi (dalla ricerca alla sviluppo finito). Il costo del sistema attualmente realizzato è pari a 961,40€.

Si tenga conto del fatto che la fase di progettazione del “sistema base” avviene una sola volta, perciò il costo commerciale del progetto probabilmente verrà suddiviso per il numero di sistemi commissionati, o che si suppone verranno commissionati.

Oltre a questo “zoccoletto duro”, ogni sistema richiederà delle personalizzazioni che si sommeranno al costo totale. In ogni caso il prezzo deve essere commisurato in modo da essere soddisfacente per il cliente e finanziare lo sviluppo del progetto stesso. Si tenga inoltre conto del fatto che l’assistenza avrà un costo e che essa rappresenta un’altra fonte di guadagno.

## **CONCLUSIONI**

Come illustrato nell'elaborato, è stato realizzato il prototipo preposto come obiettivo. Il sistema My IoT Home dispone ora di una mobile app connessa attraverso i servizi web offerti dalle piattaforme Firebase, Healthchecks.io, Thingspeak.com e VNC Viewer. Sono stati rispettati i requisiti preposti di customizzazione, sicurezza, real-time control, affidabilità e gradevolezza dell'user experience relativi al sistema. Il progetto implementa versioning e backup ed è conforme agli standard istituzionali. Sono state valutate le implementazioni future necessarie per rendere il sistema del tutto commercializzabile ed è stato intrapreso un percorso atto alla pubblicizzazione del prodotto My IoT Home.

## **ALLEGATI**

- DIMITRIU\_MIHAIL\_ELABORATO\_ALLEGATO\_01.zip:  
Comprime la presentazione del sistema in formato PDF. Il file è denominato "Presentazione\_sistema\_My\_IoT\_Home\_V2.0.pdf"
- DIMITRIU\_MIHAIL\_ELABORATO\_ALLEGATO\_02.zip:  
Include il file dell'Android App denominato "My\_IoT\_Home\_V2.0\_Android\_App.apk"
- DIMITRIU\_MIHAIL\_ELABORATO\_ALLEGATO\_03.zip:  
Contiene il file dello schema a blocchi del sistema, denominato "Schema\_a\_blocchi\_sistema\_My\_Iot\_Home\_V2.0.drawio" realizzato con l'applicativo Draw.io
- DIMITRIU\_MIHAIL\_ELABORATO\_ALLEGATO\_04.zip:  
Contiene un file denominato "Rete\_LAN\_su\_Packet\_Tracer.pkt" che rappresenta la rete LAN del prototipo mediante l'applicativo Packet Tracer
- DIMITRIU\_MIHAIL\_ELABORATO\_ALLEGATO\_05.zip:  
Tratta i calcoli relativi al preventivo resi noti nel file "Preventivo\_sistema\_My\_IoT\_Home\_V2.0.pdf"
- DIMITRIU\_MIHAIL\_ELABORATO\_ALLEGATO\_06.zip:  
Contiene i files relativi al progetto sviluppato in flutter con Android Studio, nella cartella "flutter\_project\_my\_iot\_home\_v2.0.zip"
- DIMITRIU\_MIHAIL\_ELABORATO\_ALLEGATO\_07.zip:  
Include il file principale in Python eseguito su Raspberry, denominato "my\_iot\_home.py"

- DIMITRIU\_MIHAIL\_ELABORATO\_ALLEGATO\_08.zip:  
Contiene il file “fcm.py” relativo al modulo in Python utilizzato per il servizio di notifiche push FCM
- DIMITRIU\_MIHAIL\_ELABORATO\_ALLEGATO\_09.zip:  
Comprime il file “Log\_file\_exaple.txt” relativo al Log generato dall'esecuzione del programma principale in Python
- DIMITRIU\_MIHAIL\_ELABORATO\_ALLEGATO\_10.zip:  
Contiene lo schema a blocchi del programma principale in Python all'interno del file “Schema\_programma\_principale\_Python.drawio” realizzato con Draw.io
- DIMITRIU\_MIHAIL\_ELABORATO\_ALLEGATO\_11.zip:  
Comprime il file “autostart\_my\_iot\_home.desktop” utile all'autoesecuzione del programma principale in Python dopo la fase di boot del Raspberry
- DIMITRIU\_MIHAIL\_ELABORATO\_ALLEGATO\_12.zip:  
Contiene l'analisi Wireshark, svolta durante il funzionamento a regime del sistema.  
Il file in questione è denominato “Analisi\_Wireshark.pcap”
- DIMITRIU\_MIHAIL\_ELABORATO\_ALLEGATO\_13.zip:  
Include il file “Collegamenti\_circuitali\_Raspberry.fzz” relativo ai collegamenti circuitali del raspberry con il sensore DHT11
- DIMITRIU\_MIHAIL\_ELABORATO\_ALLEGATO\_14.zip:  
Comprime il file “Programma\_Wemos\_PIR.ino” contenente il programma per il microcontrollore Wemos dotato di sensore PIR
- DIMITRIU\_MIHAIL\_ELABORATO\_ALLEGATO\_15.zip:  
Contiene il file “Schema\_programmaWemos\_PIR.drawio” relativo allo schema del programma Arduino riguardante il microcontrollore Wemos dotato di sensore PIR

- DIMITRIU\_MIHAIL\_ELABORATO\_ALLEGATO\_16.zip:  
Include il file realizzato con l'applicativo Fritzing  
“Collegamenti\_circuituali\_Wemos\_PIR.fzz” relativo allo schema circuitale del Wemos dotato di sensore PIR
- DIMITRIU\_MIHAIL\_ELABORATO\_ALLEGATO\_17.zip:  
Comprime il file “Programma\_Wemos\_Water\_Level.ino” contenente il programma per il microcontrollore Wemos dotato di sensore Water Level
- DIMITRIU\_MIHAIL\_ELABORATO\_ALLEGATO\_18.zip:  
Contiene il file “Schema\_programma\_Wemos\_Water\_Level.drawio” relativo allo schema del programma Arduino riguardante il microcontrollore Wemos dotato di sensore Water Level

DIMITRIU\_MIHAIL\_ELABORATO\_ALLEGATO\_19.zip:  
Include il file realizzato con l'applicativo Fritzing  
“Collegamenti\_circuituali\_Wemos\_Water\_Level.fzz” relativo allo schema circuitale del Wemos dotato di sensore Water Level

# INDICE DELLE FIGURE

Figura 1: Progetto My IoT Home V1.0 .....	1
Figura 2: Bot Telegram di My IoT Home V1.0 .....	2
Figura 3: Attività principali del Gantt Chart .....	4
Figura 4: Sottoattività e Milestones del Gantt Chart .....	5
Figura 5: Repository privato su GitHub.com e relativi commits .....	6
Figura 6: SD Backup attraverso Win32DiskImager .....	7
Figura 7: Sviluppo iterativo e incrementale secondo il principio dell'agile development .....	7
Figura 8: Schema a blocchi di My IoT Home V2.0.....	10
Figura 9: Wemos D1 R2 WiFi .....	12
Figura 10: Raspberry Pi 3B+ .....	12
Figura 11: Water Level Sensor .....	14
Figura 12: PIR Sensor HC-SR50.....	15
Figura 13: Sensore DHT11.....	15
Figura 14: Webcam Logitech C120 .....	16
Figura 15: Diodi led e relativi resistori da 220 [ohm].....	17
Figura 16: Wemos dotato di sensore Water Level.....	18
Figura 17: Wemos dotato di sensore PIR.....	19
Figura 18: Raspberry dotato del sensore DHT11 e Webcam .....	20
Figura 19: Stack MQTT .....	22
Figura 20: Modello di pubblicazione e sottoscrizione MQTT .....	23
Figura 21: Stack HTTPS (HTTP + SSL/TLS).....	24
Figura 22: Configurazione di base si Healthchecks.io .....	26
Figura 23: Controllo remoto e assistenza tramite RealVNC .....	26
Figura 24: Flowchart a blocchi del programma in Arduino del Wemos dotato di sensore PIR.....	28

Figura 25: Definizione delle variabili utili alla sottoscrizione con autenticazione del Broker MQTT .....	29
Figura 26: Definizione di publisher e subscriber MQTT del Wemos dotato di sensore PIR .....	29
Figura 27: Funzione void loop() del Wemos dotato di sensore PIR.....	30
Figura 28: Flowchart a blocchi del programma in Arduino del Wemos dotato di sensore Water Level e LED.....	30
Figura 29: Codice di gestione dei trigger rilevati dal subscriber MQTT .....	31
Figura 30: Flowchart a blocchi del programma principale in Python.....	32
Figura 31: Pyrebase config e definizione dei servizi utilizzati .....	33
Figura 32: Real Time Database.....	33
Figura 33: Definizione delle funzioni relative al modulo Fcm .....	34
Figura 34: Servizio di analitica relativo al servizio FCM.....	34
Figura 35: Configurazione del MQTT client .....	35
Figura 36: Gestione dei messaggi MQTT ricevuti.....	35
Figura 37: Funzione "sendImage()" .....	36
Figura 38: Raccolta delle immagini su Firebase Storage .....	36
Figura 39: Funzione di lettura del sensore DHT11 e gestione dei dati .....	37
Figura 40: Funzione di Trigger.....	38
Figura 41: Funzioni trigger inviate tramite messaggi MQTT al topic “/pi_cmnds” .....	38
Figura 42: Funzione di WebcamCheck.....	39
Figura 43: Verifica delle connessioni MQTT relative ai due Wemos.....	39
Figura 44: Configurazione del logger.....	40
Figura 45: Estratto del file "log.txt" e formattazione degli eventi .....	41
Figura 46: Files presenti nella cartella "logfiles" dello Storage di Firebase.....	41
Figura 47: File wpa-supplicant.conf .....	42
Figura 48: File di autostart del programma principale in Python.....	43
Figura 49: Esecuzione automatica di Xterm ed informazioni di log in STDOUT .....	43
Figura 50: Cron job impostato ogni minuto per la funzione di Heartbeat .....	44
Figura 51: Notifica di Healthchecks.io .....	44
Figura 52: Dashboard della Mobile App .....	45
Figura 53: Costruttore dell'oggetto Sensor .....	46

Figura 54: Schermata del grafico della temperatura con Thingspeak.....	46
Figura 55: Schermata delle immagini della Webcam.....	47
Figura 56: Menù laterale dell'app.....	48
Figura 57: Schermata di Log dell'app .....	48
Figura 58: Schermata delle impostazioni dell'app.....	49
Figura 59: Codice relativo alla preferenza delle notifiche .....	49
Figura 60: Garage Flooding Alert, FCM Push Notification .....	50
Figura 61: Motion Alert, FCM Push Notification.....	50
Figura 62: Payload di un pacchetto MQTT inviato dal ESP8266 al Raspberry .....	52
Figura 63: Attacco Man in the middle (MITM).....	53
Figura 64: Comandi eseguiti da terminale .....	53
Figura 65: Vulnerabilità del 4-way-handshake .....	54
Figura 66: Aircrack WiFi Password.....	55
Figura 67: Cablaggio su Cisco Packet Tracer .....	61
Figura 68: Canali radio disponibili in Europa per il Wi-Fi nella banda ISM 2,4 GHz ..	63
Figura 69: Configurazione Hotspot con standard IEEE 802.11n .....	64
Figura 70: Scansione radio alla ricerca di AP WiFi .....	64
Figura 71: Ulteriori dettagli sul segnale WiFi .....	65
Figura 72: Logo del sistema My IoT Home V2.0.....	74

## **INDICE DEI GRAFICI**

Grafico 1: Dispositivi IoT connessi e previsioni future .....	3
Grafico 2: Visualizzazione della Temperatura mediante ThingSpeak.com e MATLAB .....	25
Grafico 3: Funzione relativa all'entropia di una password in codice ASCII (Geogebra.org) .....	57

## **INDICE DELLE TABELLE**

Tabella 1: Collegamenti fisici del Wemos dotato del sensore Water Level.....	18
Tabella 2: Collegamenti fisici del Wemos dotato del sensore PIR.....	19
Tabella 3: Collegamenti fisici del Raspberry e del sensore DHT11 .....	20
Tabella 4: Valutazione del tempo di decrittazione di una password codificata in ASCII .....	58
Tabelle 5-6: Tabelle di indirizzamento .....	62
Tabella 7: Tabella di routing .....	62

## BIBLIOGRAFIA E SITOGRADIA

- [1] Adafruit Industries. Datasheet PIR Sensor: <https://cdn-learn.adafruit.com/downloads/pdf/pir-passive-infrared-proximity-motion-sensor.pdf>
- [2] Adafruit Industries. Datasheet DHT11 Sensor: <https://cdn-learn.adafruit.com/downloads/pdf/dht.pdf>
- [3] Adafruit Industrie. Datasheet Diodo Led: <https://cdn-shop.adafruit.com/datasheets/WP7113SRD-D.pdf>
- [4] Apple Inc. App store connect help: <https://help.apple.com/app-store-connect/en.lproj/static.html>
- [5] Avast Software sro. Are smart homes vulnerable to hacking?:  
<https://www.kaspersky.it/resource-center/definitions/what-is-cyber-security>
- [6] Commissione europea. La protezione dei dati nell'UE:  
[https://ec.europa.eu/info/law/law-topic/data-protection/data-protection-eu\\_it](https://ec.europa.eu/info/law/law-topic/data-protection/data-protection-eu_it)
- [7] Dashbird (AWS partner). Amazon AWS vs Firebase:  
<https://dashbird.io/blog/aws-lambda-vs-firebase/>
- [8] Eclipse Foundation. Eclipse Mosquitto Technology: <https://mosquitto.org/>
- [9] Eclipse Foundation. Mqtt Phao Lib: <https://www.eclipse.org/paho/>

- [10] Emartee. Datasheet Water Level Sensor:  
<https://www.emartee.com/Attachment.php?name=42240.pdf>
- [11] ENSIA (European Union Agency For Cybersecurity). An overview of the Wi-Fi WPA2 vulnerability: <https://www.enisa.europa.eu/publications/info-notes/an-overview-of-the-wi-fi-wpa2-vulnerability>
- [12] Garante per la protezione dei dati personali. Individuazione delle modalità semplificate per l'informativa e l'acquisizione del consenso per l'uso dei cookie – 8 maggio 2014: <https://www.garanteprivacy.it/home/docweb/-/docweb-display/docweb/3118884>
- [13] IBM. Getting to know MQTT:  
<https://developer.ibm.com/technologies/messaging/articles/iot-mqtt-why-good-for-iot/>
- [14] IoT Analytics. Dispositivi IoT connessi e previsioni future: <https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/>
- [15] Kevin Mitnick, Ottobre 2003, L'arte dell'inganno, Feltrinelli.
- [16] Manuale cremonese – Informatica e telecomunicazioni. Tecnologia WiFi IEEE802.11, pagg.1927,1928, Zanichelli
- [17] Manuale cremonese – Standard del cablaggio strutturato, pagg.1906, Zanichelli
- [18] Nasa. How old is the universe?:  
<https://starchild.gsfc.nasa.gov/docs/StarChild/questions/question28.html>
- [19] NortonLifeLock Inc. What is a man in the middle attack?:  
<https://us.norton.com/internetsecurity-wifi-what-is-a-man-in-the-middle->

[attack.html](#)

- [20] OffSec Services (KALI). Aircrack-ng description: <https://tools.kali.org/wireless-attacks/aircrack-ng>
- [21] Onelio Bertazioli. Corso di telecomunicazioni volume III. Installazione, configurazione e verifica della copertura radio di un access point, pag. 168, Zanichelli
- [22] Open Source Initiative. MIT Licence: <https://opensource.org/licenses/MIT>
- [23] Oracle. What is IoT?: <https://www.oracle.com/internet-of-things/what-is-iot/>
- [24] Pub Dev by Google. Pub.dev policy: <https://pub.dev/policy>
- [25] Python.org. Documentazione ufficiale del modulo “Threading”:  
<https://docs.python.org/3/library/threading.html>
- [26] Raspberry Pi Foundation. Datasheet e documentazione Raspberry Pi 3B+:  
<https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/>
- [27] SIA Monkey See Monkey Do. About Healthchecks.io:  
<https://healthchecks.io/about/>
- [28] Sonoff. Sonoff products: <https://sonoff.tech/products/>
- [29] The MathWorks Inc. Learn more about ThingSpeak:  
[https://thingspeak.com/pages/learn\\_more](https://thingspeak.com/pages/learn_more)
- [30] Wemos. Datasheet Wemos D1 R2 WiFi con ESP8266:  
<https://web.archive.org/web/20170904154405/https://wiki.wemos.cc/products:d1>

[:d1#documentation](#)

[31] Wemos. Schema Wemos D1 R2 WiFi con ESP8266:

[https://web.archive.org/web/20190218234323/https://wiki.wemos.cc/\\_media/pro  
ducts:d1:d1\\_v2.0.0.pdf](https://web.archive.org/web/20190218234323/https://wiki.wemos.cc/_media/products:d1:d1_v2.0.0.pdf)

## **RINGRAZIAMENTI**

*Mi è doveroso dedicare questo spazio dell'elaborato a tutti i professori che nel corso degli ultimi cinque anni hanno condiviso il loro tempo e le loro conoscenze contribuendo alla mia crescita personale e professionale.*

*Un grazie di cuore ai miei genitori che mi hanno sempre sostenuto.*