LIBRERIE REALIZZATE E UTILIZZATE

KEYPAD.PY

Abbiamo definito una libreria per la gestione degli input provenienti dal keypad.
Nello specifico:

• max_value_column( ) e get_key( ) effettuano la traduzione dei valori letti sui 3 pin analog collegati alle tre colonne del keypad. Ogni analog è collegato a una colonna del keypad. Quando si clicca un tasto si legge la colonna cliccata e il relativo valore, che attraverso dei range viene tradotto nel tasto premuto. I valori dei range in questi sono stati ottenuti dopo svariati test per ottenere la maggior precisione possibile.

• change_pass( ) serve per permettere la modifica della password.

• check_pass( ) controlla se la combinazione inserita corrisponde con la password.

• pass_correct( ) e pass_wrong( ) implementano i rispettivi comportamenti all'inserimento di una password corretta o una password sbagliata.

```
f = False

streams.serial()

matrix = [['1','4','7','CANC'], ['2','5','8','0'], ['3','6','9','ENTER']]

def max_value_column(V1,V2,V3):
    if V1!=0 and V2==0 and V3==0:
        return 0, V1
    elif V1==0 and V2!=0 and V3==0:
        return 1, V2
    elif V1==0 and V2==0 and V3!=0:
        return 2, V3
    else:
        return -1,-1


def get_key(V1,V2,V3):
    col, value = max_value_column(V1,V2,V3)
    if value >= 4090:       #il max risultato dai testo è 4095
        return matrix[col][0]
    elif value>=1550 and value<=1900:
        return matrix[col][1]
    elif value>=600 and value<=900:
        return matrix[col][2]
    elif value>=190 and value<=530:
        return matrix[col][3]
    else:
        return -1
def change_pass(V1,V2,V3):
    #se clicco entrami i valori risultano più piccoli per entrambi gli ADC
    if (V1>=5 and V1<=290) and (V3>=5 and V3<=290) and V2==0:
        return True
    else:
        return False

def check_pass(passw, comb):
    return passw==comb

#================================================#
#funzione per leggere gli input analaogici delle colonne
def read_col(C1, C2, C3):
    return int(analogRead(C1)), int(analogRead(C2)), int(analogRead(C3))

#funzione per l'inserimento della password corretta
def pass_correct(flag_attempt, comb, mode):
    flag_attempt=0
    mode = 2
    comb.clear()
    return flag_attempt, comb, mode

#funzione per l'inserimento della password sbagliata
def pass_wrong(flag_attempt, comb):
    flag_attempt+=1
    comb.clear()
    return flag_attempt, comb
```

• manager_input( ) ci permette di realizzare la gestione degli input dal keypad interpretandoli a seconda della modalità e della situazione in cui si trova il sistema. Implementa inoltra la gestione della combinazione inserita.

```
def manager_input(mode, flag, comb, password, COL1, COL2, COL3, R, G, B):
    global f
    if mode == 1:
        f = False
        V1,V2,V3 = read_col(COL1,COL2,COL3)
        key = get_key(V1,V2,V3)
        if key!=-1:
            RGB_Led.blue(R,G,B,500)
            if len(comb)==4:
                if key=='ENTER':
                    if check_pass(password, comb):
                        flag, comb, mode = pass_correct(flag, comb, mode)
                        RGB_Led.green(R,G,B,500)
                    else:
                        flag, comb = pass_wrong(flag, comb)
                        RGB_Led.yellow(R,G,B,500)
                elif key=='CANC' and len(comb)!=0:
                    comb.pop()
            else:
                if key == 'ENTER':
                    flag, comb = pass_wrong(flag, comb)
                    RGB_Led.yellow(R,G,B,500)
                elif key=='CANC' and len(comb)!=0:
                    comb.pop()
                else:
                    comb.append(key)
        while V1+V2+V3 != 0:
            V1,V2,V3 = read_col(COL1,COL2,COL3)
            sleep(100)
    elif mode==2:
        V1,V2,V3 = read_col(COL1,COL2,COL3)
        if change_pass(V1,V2,V3) or f:
            f = True
            V1,V2,V3 = read_col(COL1,COL2,COL3)
            key = get_key(V1,V2,V3)
            if key!=-1:
                RGB_Led.blue(R,G,B,500)
                if len(comb)==4:
                    if key=='ENTER':
                        password.clear()
                        password = comb
                    elif key =='CANC' and len(comb)!=0:
                        comb.pop()
                else:
                    if key=='CANC' and len(comb)!=0:
                        comb.pop()
                    else:
                        comb.append(key)
                while V1+V2+V3 != 0:
                    V1,V2,V3 = read_col(COL1,COL2,COL3)
                    sleep(100)
            else:
                comb.clear()
            sleep(50)
    else:
        f = False
        sleep(3000)
    if flag==3:
        mode=3
    return mode, flag, comb, password
```

RGB_LED.py

```
import pwm

def red(R, G, B, time=0):
    digitalWrite(R, HIGH)
    digitalWrite(G, LOW)
    digitalWrite(B, LOW)
    if time!=0:
        sleep(time)
        off(R,G,B)

def green(R, G, B, time=0):
    digitalWrite(R, LOW)
    digitalWrite(G, HIGH)
    digitalWrite(B, LOW)
    if time!=0:
        sleep(time)
        off(R,G,B)

def blue(R, G, B, time=0):
    digitalWrite(R, LOW)
    digitalWrite(G, LOW)
    digitalWrite(B, HIGH)
    if time!=0:
        sleep(time)
        off(R,G,B)

def yellow(R, G, B, time=0)
    digitalWrite(R, HIGH)
    digitalWrite(G, HIGH)
    digitalWrite(B, LOW)
    if time!=0:
        sleep(time)
        off(R,G,B)

def off(R, G, B):
    digitalWrite(R, LOW)
    digitalWrite(G, LOW)
    digitalWrite(B, LOW)
```

← Ogni funzione permette di accendere il led RGB con una colorazione diversa. È inoltre possibile definire un tempo (in millisecondi) per il quale il led deve restare acceso per poi essere spento. I pin R, G e B da passare alle funzioni sono i rispettivi pin digitali collegati ai pin Red, Green e Blue del led. Il led RGB richiede una scrittura digitale sui 3 pin presenti. Le funzioni chiamate in inglese red ,green , blu impostano con una digital write gli altri pin a low e il colore che si desidera ad high. Il colore giallo è stato ottenuto dalla combinazione di rosso e giallo, infatti rosso e giallo sono porti come high mentre blu su low.

BUZZER.PY

La libreria buzzer serve per permettere al buzzer passive buzzer piezo di suonare. La melodia emessa dal buzzer è quella di una sirena a segnalare la situazione di allarme. Tale libreria è implementata attraverso l'ausilio di un thread avviato dal main.
La funzione alarm richiede come parametro il pin digitale del buzzer ed esegue un primo for per innalzare la tonalità ed uno in seguito per riabbassare la tonalità, all'interno dei rispettivi for è implementata una pwm.write e un'operazione aritmetica ovvero una divisione che restituisce solo la parte intera (//) sul period che consiste nel ridurre il periodo ad ogni esecuzione del for al fine di renderlo multiplo di x.

```
import pwm

def alarm(buzzerpin):
    for x in range(1200, 1800, 10):
        period=1000000//x
        pwm.write(buzzerpin,period,period//2,MICROS)
        sleep(10)
    for x in range(1800, 1200, -10):
        period=1000000//x
        pwm.write(buzzerpin,period,period//2,MICROS)
        sleep(10)
    pwm.write(buzzerpin, 0, 0, MICROS)
```

SERVO.PY

La libreria servo.py permette la gestione del micro servo motore. Tale libreria richiede l'import della pwm in quanto la rotazione del servo è permessa univocamente mediante questa funzione. In seguito a vari test, sono stati scelti i seguenti parametri inseriti nella pwm.write. Come è possibile notare sono presenti 2 funzioni, quali close e l'open che richiedono come paremetro il pin digitale sul quale sarà eseguita la pwm.

```
import pwm

def close(servo):
    pwm.write(servo, 20000, 1400, MICROS)
    return True

def open(servo):
    pwm.write(servo, 20000, 400, MICROS)
    return True
```

DRIVER_LCD_DISPLAY.PY

Il driver per il funzionamento del display LCD è stato recuperato dal web, tuttavia, abbiamo aggiunto la funzione print_on_display( ) che ci permette di stampare una stringa sul display per un determinato intervallo di tempo dopo il quale viene cancellata.

Utilizziamo principalmente le funzioni lcd_display_string( ) per visualizzare e lcd_clear( ) per cancellare un messaggio sul display.

```
# put string function with optional char positioning
def lcd_display_string(string, line=1, pos=0):
    global Rs
    if line == 1:
        pos_new = pos
    elif line == 2:
        pos_new = 0x40 + pos
    elif line == 3:
        pos_new = 0x14 + pos
    elif line == 4:
        pos_new = 0x54 + pos

    lcd_write(0x80 + pos_new)

    for char in (string):
        lcd_write(ord(char), Rs)

# clear lcd and set to home
def lcd_clear():
    lcd_write(LCD_CLEARDISPLAY)
    lcd_write(LCD_RETURNHOME)
```

```
def print_on_display(str, time):
    lcd_clear()
    lcd_display_string(str)
    sleep(time)
    lcd_clear()
```

COMMUNICATION.PY

La libreria serve a implementare la comunicazione in MQTT con l'applicazione di interfaccia.

• La funzione mqtt_init( ) inizializza la connessione al server MQTT. Abbiamo optato per un broker locale sul nostro pc. Vengono inoltre inizializzate le on_subscribe di due topic che ci serviranno per rispondere a messaggi proventi dall'app, ovvero la richiesta dello stato del sistema e la notifica per disattivare l'allarme.

• notify_alarm( ) permette di pubblicare un messaggio sul topic sul quale è in ascolto il client MQTT dello smartphone, per ricevere la notifica di allarme.

• send_state( ) invia il link di un immagine al client MQTT sullo smartphone. L'immagine viene visualizzata dall'interfaccia e ovviamente è diversa per ogni modalità in cui si trova il sistema.

• deactivate_alarm( ) viene eseguita quando si riceve un messaggio sul topic "/idrawer/disattiva_allarme" e disattiva l'allarme se nel payload è contenuta la password corretta.

• request_status( ) viene eseguita quando si riceve un messaggio sul topic "/idrawer/request_status" e richiama la funzione send_state( ).

```
import streams

from lwmqtt import mqtt
mode_changed=0
streams.serial()
local_password = ''

def update_mode(new_mode):
    global mode_changed
    mode_changed=new_mode

def update_pass(new_pass):
    global local_password
    s=''
    for x in new_pass:
        s+=x
    local_password=s


def mqtt_init(password):
    s=''
    for x in password:
        s+=x
    local_password=s
    global client
    client=mqtt.Client("ESP32-",True)
    for retry in range(10):
        try:
            client.connect("192.168.1.236",60,1883)
            break
        except Exception as e:
            print("connecting..." + str(e))
    client.set_username_pw('mqtt', 'admin')
    client.subscribe("/idrawer/disattiva_allarme", deactivate_alarm, 2)
    client.subscribe("/idrawer/richiesta_stato", request_status, 2)
    client.loop()
```

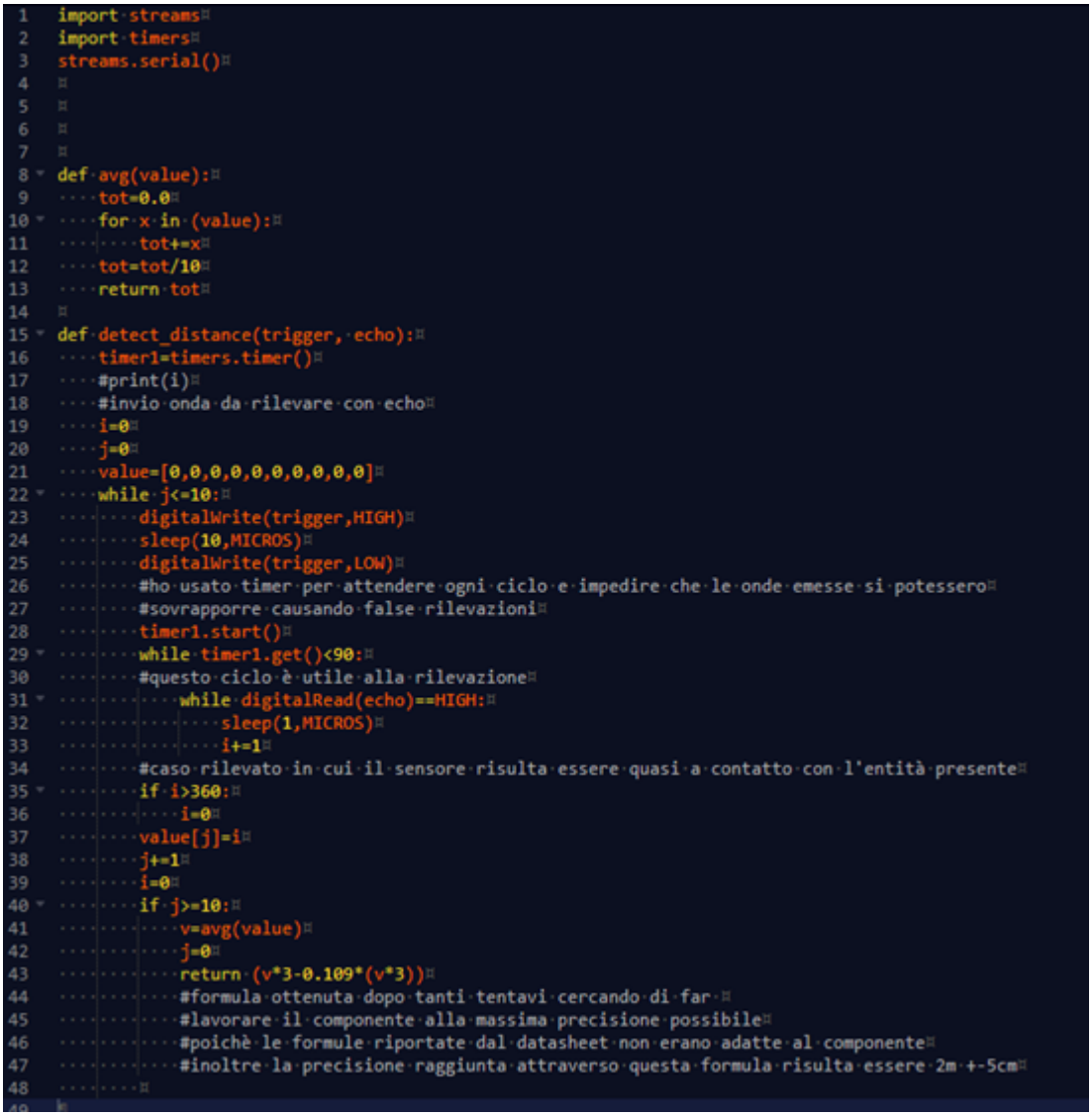
```
def notify_alarm():
    client.publish("/idrawer/notifica_allarme","ALARM",2,True)
    sleep(50)

def send_state(mode_changed):
    global mode_changed
    if(mode_changed==1):
        url="https://imagizer.imageshack.com/v2/1600x1200q90/924/0ngsGw.jpg"
        client.publish("/idrawer/invio_stato",url,2,True)
        #closed status
    if (mode_changed==2):
        url="https://imagizer.imageshack.com/v2/1600x1200q90/922/KRi7uf.jpg"
        client.publish("/idrawer/invio_stato",url,2,True)
        #open status
    if (mode_changed==3):
        url="https://imagizer.imageshack.com/v2/1600x1200q90/922/Gl8Pth.png"
        client.publish("/idrawer/invio_stato",url,2,True)
        #alarm status
    sleep(200)

def deactivate_alarm(mqtt_client, payload, topic):
    global local_password, mode_changed
    if(payload==local_password):
        mode_changed=2

def request_status(mqtt_client, payload, topic):
    global local_password, mode_changed
    send_state(mode_changed)
```

ULTRASONIC_SENSOR.PY

La libreria è composta da 2 funzioni avg e detect_distance. La funzione detect_distance richiede come parametri trigger ed echo, il funzionamento di questo sensore è originato da 2 cilindri di cui uno emette delle onde e l'altro le riceve rispettivamente triggere ed echo

Il funzionamento illustrato dal datasheet non era coerente con i dati rilevati, di conseguenza dopo uno studio e test è stato possibile definire una funzione che facesse le misurazioni in modo più preciso.

All'interno della funzione detect distance trigger vi è una lista di valori inizializzata a 0 , la creazione di un timer che sarà utilizzato per calcolare quanti ms restano a completare la rilevazione e un counter denominato come j per rilevare quante rilevazioni sono state fatte ed una variabile i che sarà usata per calcolare quanti ms l'echo resta high ovvero quanti ms è colpito dall'onda emanata dal trigger. Il primo while è per contare quante misurazioni sono state fatte infatti a 10 termina mentre all'interno del while vi è il trigger che è impostato come alto per 10 ms e poi impostato a low ,ciò gli permette di emanare le onde e di interrompersi dopo 10 ms. Le onde emanate risultano essere ancora nello spazio e quindi potenzialmente assorbite da parte dell' echo entro 90ms al massimo, di conseguenza è stato implementato un ciclo while per valutare questa condizione. L'altro while annidato invece è per calcolare quanti ms risulta essere colpito dalle onde l'echo e quindi avere alla fine di entrambi i while la variabile i come counter dei ms in cui si rileva l'onda. Se l'i risulta essere maggiore di 360 è implementato un if che azzeri i poiché la vicinanza all'oggetto da parte del sensore è pari a 0. Il valore rilevato viene riposto nella lista value e incrementato. Una volta raggiunte le 10 rilevazioni vi è un if che richiama la funzione avg in grado di calcolare la media della lista caricata precedente dalle rilevazioni e ritornare il valore calcolato secondo una formula ottenuta da più test.



```
import streams
import timers
streams.serial()



def avg(value):
    tot=0.0
    for x in (value):
        tot+=x
    tot=tot/10
    return tot

def detect_distance(trigger, echo):
    timer1=timers.timer()
    #print(i)
    #invio onda da rilevare con echo
    i=0
    j=0
    value=[0,0,0,0,0,0,0,0,0,0]
    while j<=10:
        digitalWrite(trigger,HIGH)
        sleep(10,MICROS)
        digitalWrite(trigger,LOW)
        #ho usato timer per attendere ogni ciclo e impedire che le onde emesse si potessero
        #sovrapporre causando false rilevazioni
        timer1.start()
        while timer1.get()<90:
        #questo ciclo è utile alla rilevazione
            while digitalRead(echo)==HIGH:
                sleep(1,MICROS)
                i+=1
        #caso rilevato in cui il sensore risulta essere quasi a contatto con l'entità presente
        if i>360:
            i=0
        value[j]=i
        j+=1
        i=0
        if j>=10:
            v=avg(value)
            j=0
            return (v*3-0.109*(v*3))
            #formula ottenuta dopo tanti tentavi cercando di far
            #lavorare il componente alla massima precisione possibile
            #poichè le formule riportate dal datasheet non erano adatte al componente
            #inoltre la precisione raggiunta attraverso questa formula risulta essere 2m +-5cm
        
```

MAIN

```
import streams
streams.serial()

import pwm
import servo
from mqtt import mqtt
import adc
import keypad

import RGB_Led

import buzzer

import threading

import communication

import ultrasonic_sensor

import driver_LCD_display

from wireless import wifi
from espressif.esp32net import esp32wifi as wifi_driver
wifi_driver.auto_init()

#====== INIZIALIZZAZIONE =====#
if True:

    LED_R = D22
    pinMode(LED_R, OUTPUT)
    LED_G = D21
    pinMode(LED_G, OUTPUT)
    LED_B = D19
    pinMode(LED_B, OUTPUT)

    BUZZER = D2.PWM

    LOCK = D23.PWM
    pinMode(LOCK, OUTPUT)

    ECHO = D5
    pinMode(ECHO,INPUT)
    TRIGGER = D4
    pinMode(TRIGGER,OUTPUT)

    LCD = driver_LCD_display.LCD_init(I2C0)

    COL1 = A4
    pinMode(COL1, INPUT_ANALOG)
    COL2 = A7
    pinMode(COL2, INPUT_ANALOG)
    COL3 = A2
    pinMode(COL3, INPUT_ANALOG)

    mode = 1
    #1: cassetto chiuso
    #2: cassetto aperto
    #3: allarme

    #passord di default, da cambiare volontariamente
    password = ['1','2','3','4']
    comb = []
    flag_mode2=False
    flag_attempt = 0
lock_buzzer = threading.Lock()
lock_buzzer.acquire()
def th_buzzer():
    while True:
        lock_buzzer.acquire()
        buzzer.alarm(BUZZER)
        lock_buzzer.release()
thread(th_buzzer)

distance = 150

def modify_distance(new_distance):
    global distance
    distance = new_distance

lock_ultrasonic = threading.Lock()
lock_ultrasonic.acquire()
def th_ultrasonic():
    while True:
        lock_ultrasonic.acquire()
        modify_distance(ultrasonic_sensor.detect_distance(TRIGGER,ECHO))
        lock_ultrasonic.release()
        sleep(500)
thread(th_ultrasonic)


connected = False
```

Il main racchiude le chiamate a tutte le librerie implementate e soprattutto la logica di funzionamento. Gli import sono import che riguardano le librerie scritte e quindi utilizzate oppure come nel caso della libreria esp32wifi e mqtt solamente utilizzate. Unicamente per semplicità visiva nella stesura del codice è stato scritto un if True che racchiude tutte le inizializzazioni dei pin e delle eventuali variabili di sistema. Si precisa per comprensione del codice la variabile “mode” è la variabile che indica lo stato del sistema creato che nel caso sia 1 indica che il cassetto è chiuso, nel caso 2 aperto e nel caso 3 invece in allarme.

La password del sistema è inizializzata ad 1 2 3 4. E’ stata eseguita la creazione del lock di un thread definito come th_buzzer da eseguire ed è stato subito dopo acquisito. Inoltre è stata creata una funzione (th_buzzer) che il thread dovesse eseguire che consiste nel far suonare il buzzer.

Al di sotto è riportata un’altra funzione che dovrà essere eseguita da un’altro thread ovvero modify_distance, che verrà eseguita per la rilevazione della distanza. Anche in questo caso è stato creato il lock ed eseguito l’acquire del lock.

```
while not(connected):
    try:
        print("Establishing Link...")
        wifi.link("TIM-29030027",wifi.WIFI_WPA2,"LWGXGbIbsppKZIGgls84wWZF")
        connected=True
    except Exception as e:
        driver_LCD_display.print_on_display("LINK ERROR", 2000)
        sleep(3000)

communication.mqtt_init(password)
driver_LCD_display.print_on_display("Sistem connected", 2000)
sleep(1000)

communication.update_pass(password)

print("connesso")

while True:

    if communication.mode_changed==2:
        mode=2
        RGB_Led.off(LED_R,LED_G,LED_B)
        flag_attempt=0
        communication.update_mode(mode)

    if mode==1:
        flag_mode2=False
        communication.update_mode(mode)
        servo.close(LOCK)
        flag_v=flag_attempt
        mode, flag_attempt, comb, password = keypad.manager_input(mode, flag_attempt, comb, password, COL1, COL2, COL3, LED_R, LED_G, LED_B)
        if flag_v!=flag_attempt and flag_attempt!=0:
            driver_LCD_display.print_on_display("WRONG PASS", 3000)
        driver_LCD_display.lcd_display_string('ATTEMPT #'+str(flag_attempt),1)
        driver_LCD_display.lcd_display_string('INSERT PASS:', 2)
        driver_LCD_display.lcd_display_string('....', 2, 12)
        driver_LCD_display.lcd_display_string(comb,2,12)
        lock_ultrasonic.release()
        while(not(lock_ultrasonic.acquire())):
            sleep(50)
        if distance>10:
            mode = 3
            driver_LCD_display.print_on_display('---BREACH---', 1500)
            communication.update_mode(mode)
        sleep(100)
```

La connessione viene stabilita nel main dove nel while not viene passata come parametro una variabile connected inizializzata alla fine della foto precedente a false. Dopo la connessione viene inizializzata la connessione MQTT e quindi avviata.

Nel while True viene eseguito il controllo del sistema, dove viene controllato attravero il primo if se l'allarme è stato disattivato dallo smartphone in tal caso provvede a cambiare la modalità e ad aprire il cassetto.

Nel secondo if ovvero in mode=1 si ha invece lo stato chiuso del cassetto. In questa fase vengono eseguiti 2 controlli attivamente ovvero: la distanza attraverso il thread modify_distance per rilevare effrazioni (eseguito a partire dal lock_ultrasonic release())e allo stesso tempo vengono rilevati i tasti premuti dall'utente e mostrati a schermo i numeri inseriti, tenendo conto dei tentativi restanti e mostrandoli anche a display. In questa fase viene anche notificato l'inserimento di effrazione e di password sbagliata attraverso il display.

```
elif mode==2:
    servo.open(LOCK)
    if flag_mode2==False:
        driver_LCD_display.print_on_display("8\" TO OPEN", 8000)
        flag_mode2=True
    communication.update_mode(mode)
    driver_LCD_display.lcd_display_string('WELCOME  #&* TO')
    driver_LCD_display.lcd_display_string('CHANGE PASS:',2)
    old_password=password
    mode, flag_attempt, comb, password = keypad.manager_input(mode, flag_attempt, comb, password, COL1, COL2, COL3, LED_R, LED_G, LED_B)
    if password!=old_password:
        driver_LCD_display.print_on_display("PASS CHANGED", 3000)
        old_password=password
        comb.clear()
        communication.update_pass(password)
    driver_LCD_display.lcd_display_string('....', 2, 12)
    driver_LCD_display.lcd_display_string(comb,2, 12)
    lock_ultrasonic.release()
    while(not(lock_ultrasonic.acquire())):
        sleep(50)
    if distance<5:
        communication.update_mode(1)
        mode = 1
        driver_LCD_display.print_on_display('CLOSE', 1500)
    sleep(100)
```

Se viene rilevata la mode=2 il cassetto si apre, con servo.open(LOCK), dopodiché si concedono 8 secondi per aprire il cassetto, avviso che viene visualizzato sul display, nel caso in cui non si apra il cassetto, esso viene richiuso.
Una volta aperto il cassetto il sistema si mette in attesa di chiusura o di cambiamento di password.
Per cambiare la password basta cliccare contemporaneamente * + # e poi inserire la nuova password.
Nel caso in cui il sensore ad ultrasuoni rilevi che il cassetto si è chiuso, riporta il sistema nella mode 1.

```
else:
    communication.update_mode(mode)
    lock_buzzer.release()
    flag_mode2=False
    servo.close(LOCK)
    communication.notify_alarm()
    driver_LCD_display.lcd_display_string("-----ALARM------", 1)
    driver_LCD_display.lcd_display_string("-----ALARM------", 2)
    RGB_Led.red(LED_R,LED_G,LED_B)
    while(not(lock_buzzer.acquire())):
        sleep(50)
    sleep(1000)
```

Se il sistema si trova invece nella mode tre allora entra nell'ultime else.
Viene rilasciato il lock del thread del buzzer così da far scattare l'allarme acustico, viene stampato sul display il messaggio di allarme e viene acceso il led con colorazione rossa. Viene inoltre notificato l'allarme allo smartphone per tutto il tempo per il quale ci si trova in questa modalità. Viene infine riacquisito il lock del thread del buzzer, per evitare che suoni anche dopo che l'allarme viene disattivato.

Comunicazione smartphone – idrawer

La comunicazione tra smartphone e idrawer è stata implementata attraverso la libreria communication.py per quanto riguarda l'esp32. Come è possibile notare è stato creato un broker locale in esecuzione sul pc con Windows10 attraverso l'utilizzo del software Mosquitto. Il broker è stato configurato in modo tale da permettere l'accesso solo con nome utente e password.

In aggiunta per testare l'efficienza del broker è stato utilizzato un software aggiuntivo "MQTT explorer" che permette in sostanza di visualizzare i messaggi inviati sia lato publisher che subscriber in arrivo al broker.

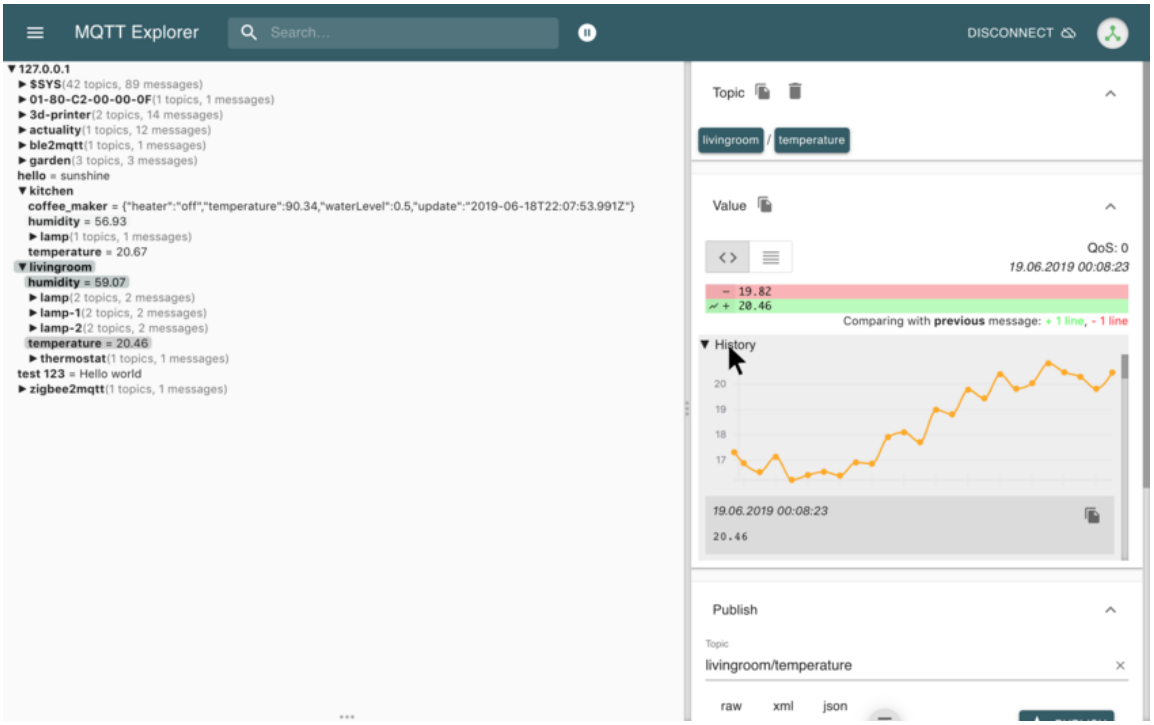
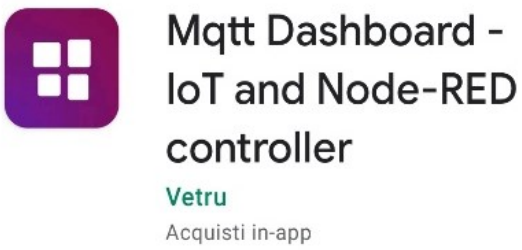


Immagine semplificativa della funzionalità di MQTT explorer.

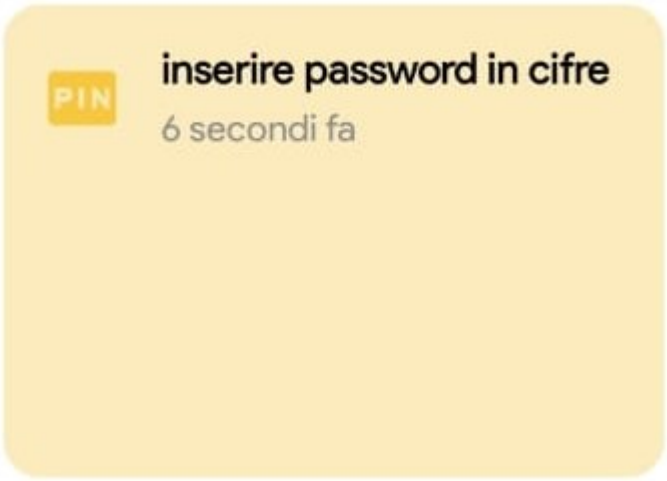
Lato smartphone è stata utilizzata l'app MQTT dashboard su dispositivo android facilmente scaricabile dal play store.

L'app permette la connessione ad un broker mqtt gratuitamente.

La dashboard consente la creazione di widget o tasti, modificabili in colore ed icona. Ogni widget creato deve essere sottoscritto al topic come client o subscriber corrispettivo secondo le funzionalità da implementare.



Per la richiesta dello stato del sistema abbiamo inserito un bottone (Richiedere stato) mentre per visualizzare lo stato in cui si trova il sistema in quel momento abbiamo utilizzato delle immagini significative, da sinistra verso destra, cassetto chiuso, cassetto aperto e allarme.



Per la disattivazione dell'allarme abbiamo utilizzato un widget che ci permette di inviare un messaggio contenente, come payload, un testo. Se si inserisce la password giusta il sistema disattiva l'allarme.

Per notificare l'allarme allo smartphone abbiamo utilizzato un riquadro che ci permette di inviare una notifica push per ogni messaggio ricevuto sul topi d'iscrizione.