

Università di Bologna

Dipartimento di Informatica

LABORATORIO DI MAKING

Progetto “**Sveglia per Sordi**”

di *Giuseppe Spathis*

Prof. Federico Montori

Prof. Renzo Davoli

Anno Accademico: 2024-2025

INTRODUZIONE	3
Obiettivo del progetto	3
Analisi della letteratura	3
Panoramica delle funzionalità	3
Elementi innovativi del dispositivo rispetto alla letteratura	4
Componenti necessari	4
Lista dei componenti hardware	4
Software e servizi	5
Montaggio hardware: passo dopo passo	6
Collegamento Schermo LCD	6
Collegamento dei LED	7
Collegamento del Vibrator Motor	9
Collegamento dei Pulsanti	10
Schema finale completo	11
Software e codice	12
Telegram_bot.py	12
Clock.py	14
Configurazione del Servizio di Avvio di clock.py	17
Interazione tramite Telegram Bot	18
Creazione del bot	18
Connessione al database Firebase	20
Stampa 3D del case	21
Progettazione del case	21
Stampa	22
Esperienza, difficoltà e miglioramenti	23
Sfide incontrate e possibili soluzioni	23
Limiti attuali del progetto	23
Idee per un miglioramento in prospettiva futura	24
Conclusioni	25
Extra: demo	26
Demo schermo Lcd	26
Demo accensione sveglia	27

INTRODUZIONE

Obiettivo del progetto

L'obiettivo di questo progetto è la realizzazione di una sveglia pensata per persone sordi, ipoacusiche, o con difficoltà uditive a vario titolo, che non utilizzi stimoli acustici ma segnali visivi e tattili per il risveglio. In particolare, l'attivazione della sveglia provoca l'accensione di LED e l'attivazione di un motorino vibrante, da posizionare ad esempio sotto il cuscino, in modo da garantire un risveglio efficace anche senza percezione sonora.

L'idea nasce da un problema personale: spesso dormo con i tappi nelle orecchie e mi capita di non sentire le sveglie tradizionali. Da qui la necessità di trovare una soluzione alternativa, affidabile e personalizzabile, che possa essere utile anche ad altri utenti con esigenze simili.

Analisi della letteratura

Le sveglie attualmente disponibili per utenti sordi si basano principalmente sull'uso di un motorino vibrante o, in alternativa, su dispositivi da polso che emettono leggere scosse elettriche; quest'ultima soluzione mi sembra tuttavia troppo invasiva per un utilizzo quotidiano. Per questo motivo ho scelto di adottare il motorino vibrante, integrandolo però con un sistema di segnalazione visiva tramite LED; in tal modo ho cercato di aumentare l'efficacia della sveglia senza compromettere il comfort dell'utente.

Panoramica delle funzionalità

La sveglia può essere controllata da remoto tramite un bot Telegram, chiamato Raspberry Pi Clock Bot, attraverso il quale è possibile impostare, modificare o cancellare le sveglie. Il sistema supporta la multiutenza: ogni sveglia viene associata a un determinato Raspberry Pi, in modo da rendere il progetto facilmente scalabile e adatto a una distribuzione futura del dispositivo su larga scala. Tutti i dati relativi alle sveglie vengono salvati in un realtime database di Firebase, che consente la sincronizzazione tra dispositivi. La gestione del bot Telegram avviene su un server separato, mentre sul Raspberry Pi Zero gira un secondo script Python che monitora continuamente il database e attiva i segnali di sveglia nel momento previsto. Infine, per completare il progetto, è stato realizzato e stampato un case personalizzato in 3D in cui alloggiare tutta la componentistica, rendendo il dispositivo compatto, ordinato e pronto all'uso.

Elementi innovativi del dispositivo rispetto alla letteratura

Rispetto alle soluzioni già presenti in commercio, questo progetto introduce alcune caratteristiche innovative. L'uso combinato di stimoli visivi e tattili rende il risveglio più efficace e meno traumatico; l'interazione tramite Telegram consente all'utente di impostare le sveglie da remoto in modo semplice e intuitivo; il salvataggio dei dati su Firebase permette la sincronizzazione e l'accesso da più dispositivi, mentre l'architettura software, distribuita tra server e dispositivo, garantisce flessibilità e scalabilità. L'aggiunta di un case stampato in 3D, contribuisce infine a rendere il progetto completo anche dal punto di vista pratico e della presentazione estetica.

Componenti necessari

Lista dei componenti hardware

- 1 Raspberry Pi Zero, Stima: €18,00 - €30,00
- 3 LED, Stima: €0,10 - €0,50
- 1 Vibrator Motor, Stima: €5,00 - €10,00
- 3 Pulsanti, Stima: €0,20 - €0,60
- 1 Display LCD, Stima: €10 - €15,00
- 1 batteria da 9 volt, Stima: €1,50 - €3,00
- Diversi jumper maschio-maschio e maschio-femmina, Stima: €2,00 - €5,00
- 2 transistor NPN, Stima: €0,10 - €0,40
- 1 potenziometro, Stima: €0,30 - €1,00
- 1 diodo di flyback, Stima: €0,05 - €0,20
- 3 resistenze 1k Ω, Stima: €0,05 - €0,15
- 3 resistenze 100 Ω, Stima: €0,05 - €0,15
- 1 resistenza 220 Ω, Stima: €0,05 - €0,15

- 1 resistenza 330 Ω, Stima: €0,05 - €0,15
- 1 breadboard, Stima: €3,50 - €6,00

La stima complessiva del costo dei componenti è compresa tra €40,95 e €72,30

Software e servizi

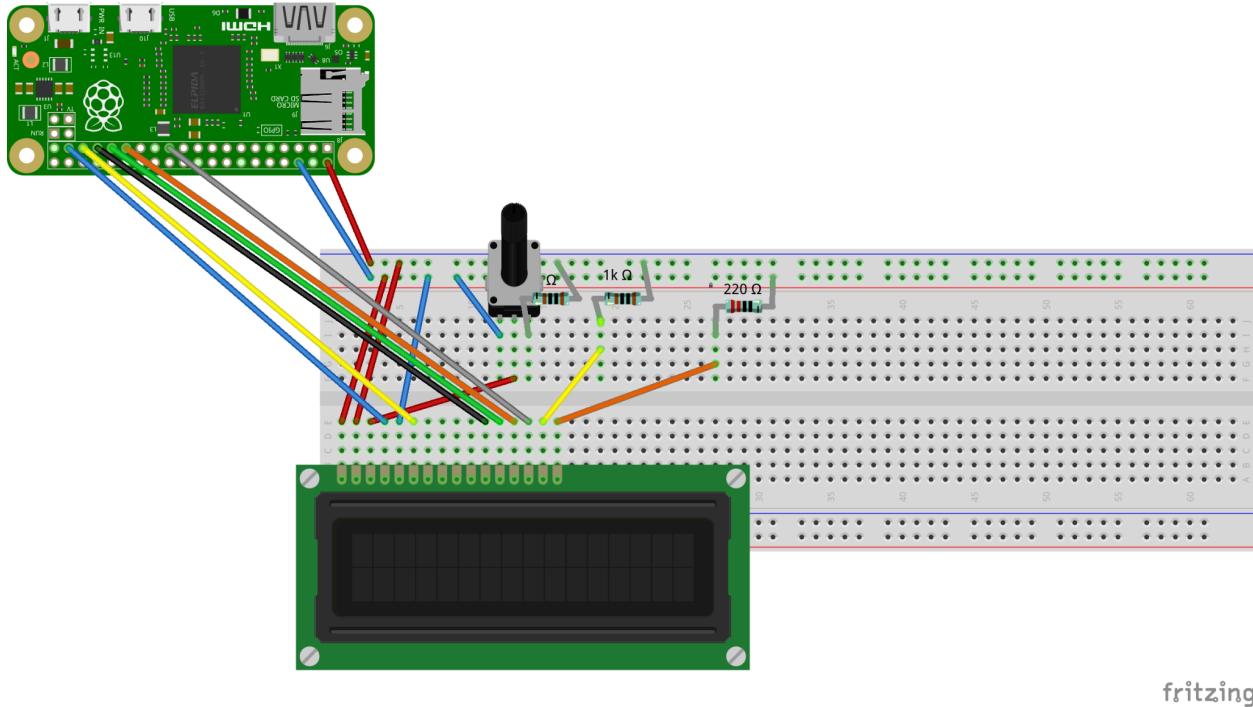
Il sistema è basato su Raspbian OS, installato sul Raspberry Pi Zero, con un'infrastruttura software interamente sviluppata in Python. Sono state utilizzate alcune librerie fondamentali per il funzionamento dell'intero progetto: **Adafruit_CharLCD** per il controllo del display LCD, **telegram** per la gestione del bot Telegram che consente l'impostazione delle sveglie da remoto, e **firebase_admin** per la comunicazione con il database cloud.

Il sistema si appoggia a un progetto Firebase configurato con Realtime Database, che permette la sincronizzazione delle sveglie in tempo reale tra server e dispositivo. Il codice che gestisce il bot e l'inserimento dei dati nel database viene eseguito su un server esterno, mentre il Raspberry Pi legge costantemente dal database per attivare la sveglia al momento giusto, azionando LED e motorino vibrante.

Tutti i componenti software e i servizi cloud sono stati scelti per garantire affidabilità, scalabilità e semplicità di gestione, rendendo il sistema facilmente estendibile e replicabile da altri utenti.

Montaggio hardware: passo dopo passo

Collegamento Schermo LCD



fritzing

Per la visualizzazione delle informazioni cruciali del progetto, come l'ora, la data, lo stato della sveglia e messaggi di sistema, è stato impiegato un display LCD a caratteri standard da 16x2.

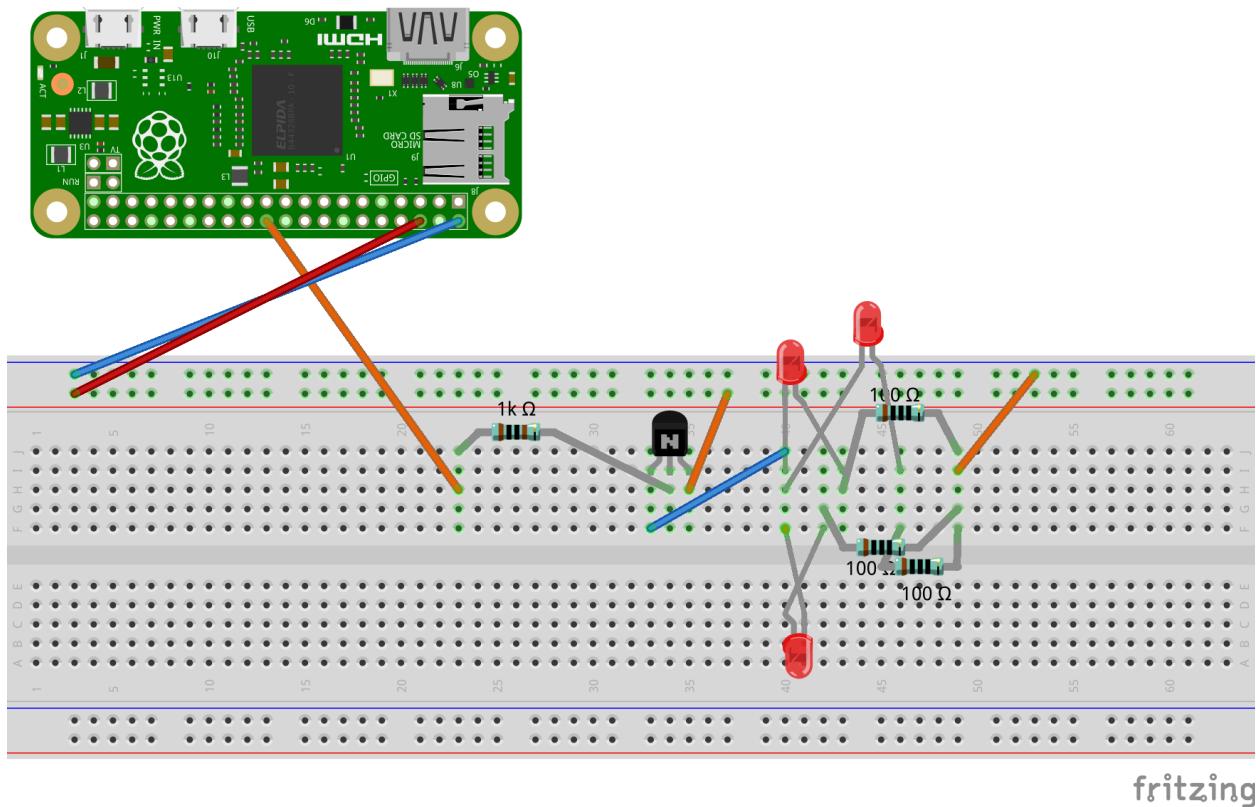
L'alimentazione del display LCD è fornita dal Raspberry Pi Zero: il pin VSS (Pin 1) del display è connesso a una linea di massa (GND) comune, mentre il pin VDD (Pin 2) riceve i +5V. La retroilluminazione, essenziale per la leggibilità in diverse condizioni di luce, è anch'essa alimentata dal Raspberry Pi; il suo anodo (pin A o Pin 15) è collegato ai +5V attraverso una resistenza di limitazione corrente da $1k\Omega$, per proteggere i LED interni, mentre il catodo (pin K o Pin 16) è collegato a massa (GND) attraverso una resistenza da 220Ω .

La nitidezza dei caratteri visualizzati è regolabile grazie a un potenziometro. Il cursore di tale potenziometro è connesso al pin VO (Pin 3) del display. I due terminali esterni del potenziometro sono rispettivamente collegati a +5V e GND, creando così un partitore di tensione che permette una fine regolazione del contrasto.

La comunicazione e il trasferimento dei dati tra il Raspberry Pi Zero e l'LCD avvengono in modalità a 4-bit, non essendo collegati i pin da D0 a D3. Il pin Register Select (RS, Pin 4) dell'LCD è pilotato dal GPIO 26 (BCM) del Raspberry Pi e determina se i dati inviati sono comandi o caratteri da visualizzare. Il pin Read/Write (R/W, Pin 5) è permanentemente collegato a massa (GND), configurando così il display in sola modalità di scrittura, come richiesto dal

progetto. Il segnale di Enable (E, Pin 6) dell'LCD, che convalida i dati sui bus, è connesso al GPIO 19 (BCM). Infine, i quattro pin dati superiori dell'LCD (D4-D7) sono collegati ai seguenti pin GPIO del Raspberry Pi: D4 (Pin 11 dell'LCD) al GPIO 13, D5 (Pin 12 dell'LCD) al GPIO 6, D6 (Pin 13 dell'LCD) al GPIO 5 e D7 (Pin 14 dell'LCD) al GPIO 11. Questa configurazione è gestita a livello software dalla libreria Python Adafruit_CharLCD, che semplifica l'invio di comandi e la visualizzazione del testo sullo schermo.

Collegamento dei LED



Lo schema illustrato mostra un circuito progettato per l'accensione simultanea di tre LED, un sistema di segnalazione visiva che verrà attivato all'innesco di una sveglia. Il cuore di questo circuito di pilotaggio è un transistor NPN, qui impiegato con la funzione di interruttore elettronico, controllato da un segnale digitale proveniente da un pin GPIO del Raspberry Pi Zero.

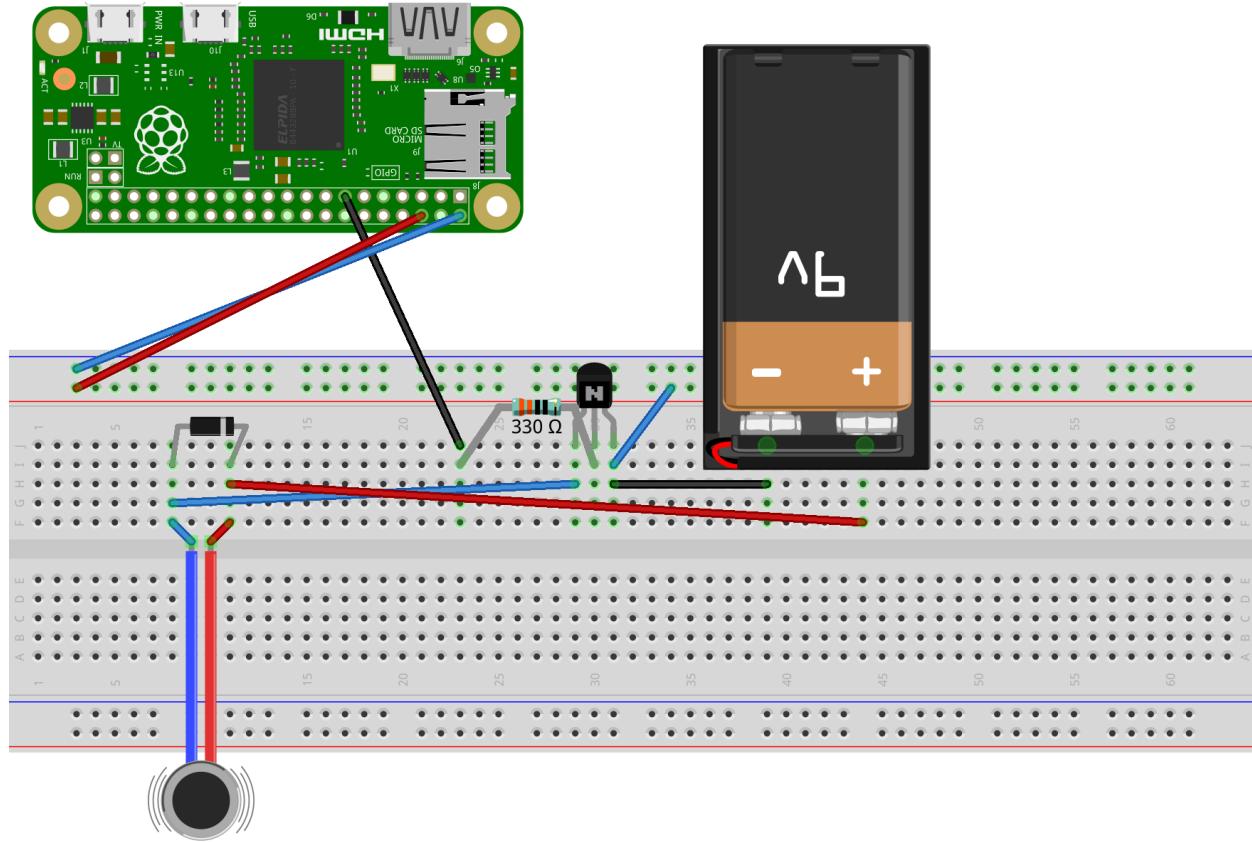
Nel dettaglio, l'alimentazione per i LED e per il circuito di controllo del transistor è fornita dalle linee a +5V e GND (massa) derivate dal Raspberry Pi. Un segnale logico, originato da un pin GPIO del Raspberry Pi e rappresentato da un filo arancione, giunge alla base del transistor NPN. Prima di arrivare alla base, questo segnale attraversa una resistenza da $1\text{k}\Omega$, la cui funzione è cruciale per limitare la corrente assorbita dal pin GPIO e per proteggere il transistor stesso, garantendone un funzionamento corretto e sicuro.

Quando il pin GPIO del Raspberry Pi viene impostato a un livello logico alto (HIGH, corrispondente a 3.3V), una piccola corrente fluisce attraverso la resistenza da $1\text{k}\Omega$ e raggiunge la base del transistor NPN. Questa corrente di base "attiva" il transistor, portandolo in uno stato di conduzione. In questo stato, il transistor permette a una corrente significativamente maggiore di fluire dal suo terminale Collettore (C) al suo terminale Emettitore (E). L'Emettitore del transistor è collegato direttamente a massa (GND).

Il Collettore del transistor è a sua volta connesso al lato negativo (catodo) dei tre LED rossi, i quali sono disposti in parallelo tra loro. Ciascun LED ha il proprio anodo (lato positivo) collegato alla linea di alimentazione positiva (+5V o +3.3V) attraverso una resistenza di limitazione di corrente individuale da 100Ω . Queste resistenze sono essenziali per proteggere ciascun LED da una corrente eccessiva che potrebbe danneggiarlo, e per garantire che ogni LED riceva una corrente appropriata per una luminosità ottimale e uniforme.

Pertanto, quando il transistor NPN è attivato dal segnale del Raspberry Pi, chiude il circuito per i tre LED, permettendo alla corrente di fluire dall'alimentazione positiva, attraverso le rispettive resistenze da 100Ω e i LED, per poi passare attraverso il transistor (dal Collettore all'Emettitore) e infine verso massa. Questo provoca l'accensione simultanea dei tre LED. Al contrario, quando il pin GPIO del Raspberry Pi è a livello logico basso (LOW, 0V), non c'è corrente di base sufficiente per attivare il transistor. Il transistor rimane quindi "spento" (interdetto), interrompendo il flusso di corrente verso i LED, che di conseguenza restano a loro volta spenti. Questo meccanismo permette al Raspberry Pi di controllare con precisione l'accensione dei LED come indicatore visivo dell'attivazione della sveglia.

Collegamento del Vibrator Motor



fritzing

Nello schema presente, il motorino a vibrazione è controllato dal Raspberry Pi Zero attraverso un circuito che impiega un transistor NPN come interruttore, in modo simile a come venivano pilotati i LED nel circuito precedente. Questo meccanismo è progettato affinché, al segnale di una sveglia, il motorino si attivi fornendo una notifica tattile.

Il circuito di controllo prevede che un pin GPIO del Raspberry Pi (collegato con un filo nero nello schema) invii il segnale di attivazione. Questo segnale passa attraverso una resistenza da 330Ω , che serve a limitare la corrente verso la base del transistor NPN, proteggendo sia il pin GPIO del Raspberry Pi che il transistor stesso.

Quando il pin GPIO del Raspberry Pi viene portato a livello logico alto (HIGH), la corrente fluisce nella base del transistor; ciò "accende" il transistor, permettendo alla corrente di passare tra il collettore e l'emettitore. L'emettitore del transistor è collegato direttamente al terminale negativo (GND) della fonte di alimentazione esterna (la batteria da 9V).

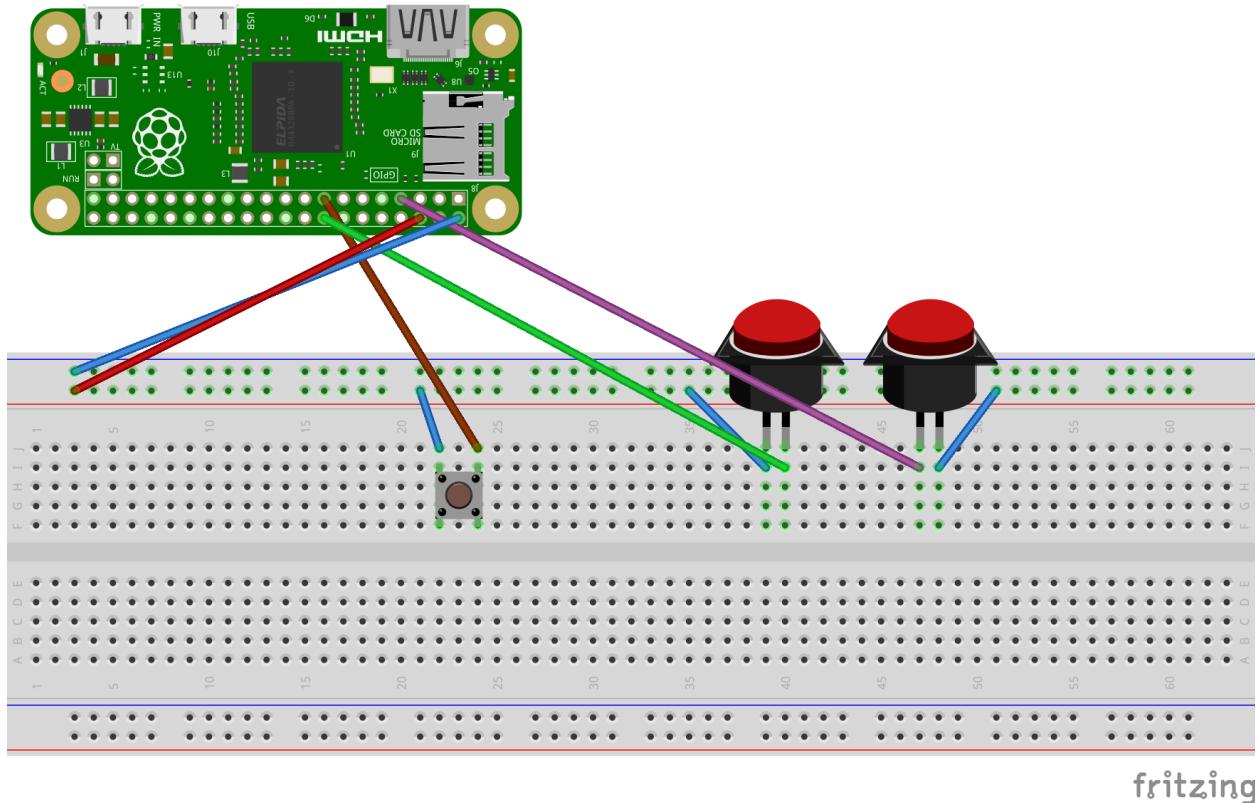
Il motorino a vibrazione è collegato tra il terminale positivo della batteria da 9V e il collettore del transistor. È importante notare che il motorino è alimentato da una fonte esterna (la batteria da

9V) e non direttamente dal Raspberry Pi. Il Raspberry Pi, tramite il transistor, controlla semplicemente se il circuito del motorino verso massa è chiuso o aperto.

In parallelo ai terminali del motorino è presente un diodo, con il catodo (la striscia sul corpo del diodo) rivolto verso il positivo dell'alimentazione del motore (collegato al positivo della batteria da 9V) e l'anodo rivolto verso il collettore del transistor. Questo diodo è un diodo di flyback. I motori sono carichi induttivi e, quando la corrente che li attraversa viene interrotta bruscamente, ovvero quando il transistor si spegne, generano picchi di tensione inversa che potrebbero danneggiare il transistor. Il diodo di flyback fornisce un percorso sicuro per questa energia induttiva, sopprimendo il picco di tensione e proteggendo il transistor.

Pertanto, nel momento in cui la sveglia si attiva, il Raspberry Pi imposta il pin GPIO dedicato a livello alto, ciò attiva il transistor NPN, che a sua volta chiude il circuito di alimentazione per il motorino a vibrazione, facendolo funzionare. Quando il segnale di sveglia cessa, il GPIO va a livello LOW, il transistor si spegne, interrompendo l'alimentazione al motorino, che smette di vibrare. La batteria da 9V fornisce l'energia necessaria al motorino, mentre il diodo di flyback assicura la protezione del componente di commutazione.

Collegamento dei Pulsanti



Nello schema si osservano tre pulsanti interfacciati con il Raspberry Pi Zero, ognuno destinato a una specifica interazione utente. Tutti e tre i pulsanti sono configurati in modalità “pull-up interno”, una tecnica comune che semplifica il cablaggio. Ciò significa che ogni pin GPIO del

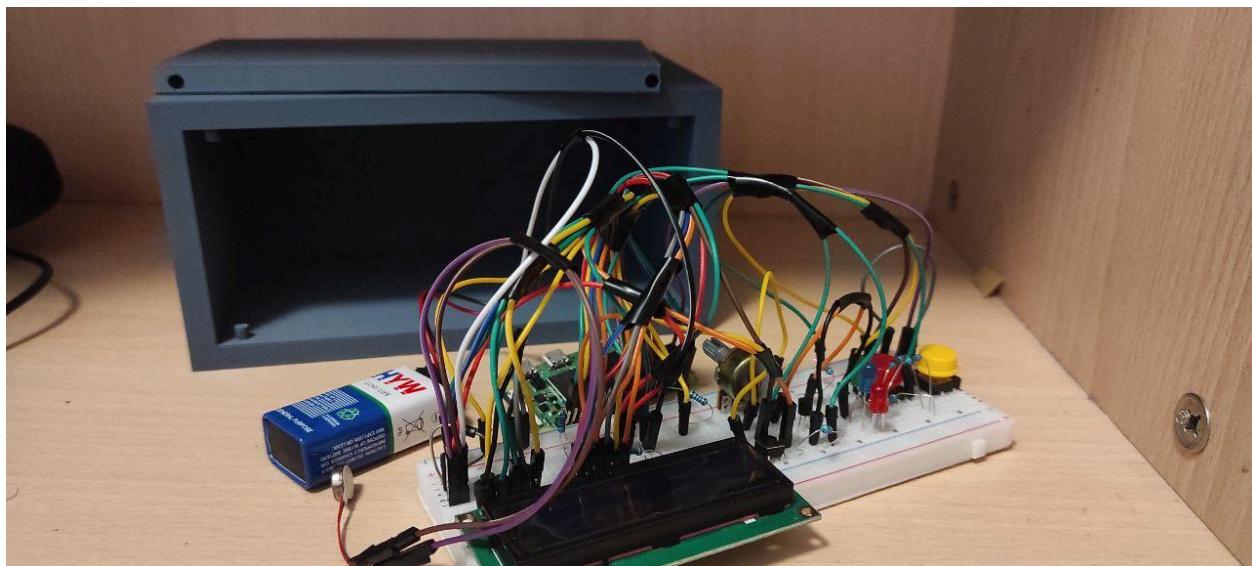
Raspberry Pi a cui è collegato un pulsante è configurato via software per avere una resistenza interna che lo mantiene a un livello logico alto (HIGH) quando il pulsante non è premuto. Quando un pulsante viene premuto, esso collega il rispettivo pin GPIO direttamente a massa (GND), portando così il livello logico del pin a basso (LOW).

Il primo pulsante, quello di dimensioni più piccole, è dedicato alla gestione del motorino a vibrazione. La pressione di questo pulsante invierà un segnale al Raspberry Pi che, tramite software, attiverà o disattiverà la funzionalità del motorino a vibrazione, permettendo all'utente di scegliere se ricevere o meno la notifica tattile. Nello schema, questo pulsante è collegato con un filo marrone a un pin GPIO del Raspberry Pi e l'altro suo terminale è connesso alla linea di massa (GND) della breadboard.

Il secondo pulsante, il primo dei due più grandi partendo da sinistra, ha la funzione cruciale di "spegnimento sveglia". Quando una sveglia è attiva e quindi i LED di segnalazione sono accesi e il motorino a vibrazione è in funzione, la pressione di questo pulsante comunicherà al Raspberry Pi l'intenzione di disattivare l'allarme. Il software provvederà quindi a spegnere i LED e a fermare il motorino a vibrazione. Il secondo pulsante è collegato con un filo verde a un altro pin GPIO del Raspberry Pi e, come gli altri, a massa.

Infine il terzo pulsante, di dimensioni maggiori rispetto agli altri e posizionato più a destra, è designato per la visualizzazione del codice univoco del Raspberry Pi sul display LCD. Tale codice è essenziale per l'accoppiamento (pairing) del dispositivo con un bot Telegram, permettendo così la comunicazione e il controllo remoto. Alla pressione di questo pulsante, il Raspberry Pi mostrerà l'ID del dispositivo sull'LCD per un tempo determinato, facilitando la procedura di configurazione con il servizio esterno. Il collegamento di questo pulsante al Raspberry Pi è realizzato tramite un filo viola connesso a un pin GPIO dedicato, con l'altro terminale del pulsante sempre collegato a massa.

Schema finale completo



L'immagine presente offre una visione d'insieme dell'assemblaggio finale del prototipo, dove tutti i singoli circuiti discussi in precedenza sono stati integrati e interconnessi sulla breadboard con il Raspberry Pi Zero. Come si può osservare, la densità dei collegamenti rende l'insieme piuttosto articolato e, per tentare una minima organizzazione dei numerosi cavi e migliorare la gestione dello spazio, è stato fatto un uso di nastro adesivo come soluzione pratica di cable management.

Software e codice

Per un'analisi completa del codice si rimanda al repository GitHub al seguente link:

<https://github.com/GiuseppeSpathis/SvegliaSordi/tree/main>

Telegram_bot.py

Lo script `telegram_bot.py` agisce come un server centrale che gestisce un bot Telegram interattivo. La sua funzione principale è permettere agli utenti di associare i propri dispositivi Raspberry Pi, impostare, visualizzare e cancellare sveglie personalizzate per ciascun dispositivo. Utilizza Firebase Realtime Database come backend per archiviare le associazioni utente-Pi, le liste di sveglie e, crucialmente, per inviare i segnali di "trigger" ai Raspberry Pi quando una sveglia deve suonare.

Il cuore della logica di attivazione delle sveglie risiede nella funzione `check_and_trigger_alarms_runner`, eseguita in un thread separato per non bloccare l'interazione del bot con gli utenti. Questa funzione è responsabile del monitoraggio costante di tutte le sveglie impostate per tutti i dispositivi Raspberry Pi registrati nel sistema.

Quando la funzione rileva che l'ora e la data correnti corrispondono a quelle di una o più sveglie memorizzate in Firebase, essa interviene modificando lo stato del trigger associato al Raspberry Pi specifico. In pratica, imposta il valore nel percorso `/triggers/ID_DEL_PI` su true nel database Firebase.

Il Raspberry Pi, attraverso il suo script locale (`clock.py`), legge periodicamente questo valore dal database. Se rileva che il suo trigger è stato impostato su true, interpreta questo come un comando per attivare fisicamente la sveglia, accendendo i LED e il motorino a vibrazione. Una volta che la sveglia è stata attivata e l'allarme gestito, questo stesso meccanismo di trigger viene utilizzato per resettare lo stato, impostando il valore su false, il che segnala al Raspberry Pi di spegnere gli indicatori. Le sveglie attivate vengono poi rimosse dal database per evitare che suonino di nuovo.

```

# (all'interno di check_and_trigger_alarms_runner)
# ...
now_aware = datetime.now(tz_info)
current_date_str = now_aware.strftime("%Y-%m-%d")
current_time_str = now_aware.strftime("%H:%M") # Confronto HH:MM

triggered_pi_ids_this_minute = set()
alarms_to_delete_this_minute = []

try:
    logger.debug(f"Controllo allarmi per {current_date_str} {current_time_str}")
    all_pi_alarms = load_all_pi_alarms() # Carica {"pi_id": [alarms...]}]

    if not isinstance(all_pi_alarms, dict):
        logger.warning("load_all_pi_alarms non ha restituito un dizionario.")
        all_pi_alarms = {}

    for pi_id, pi_alarms_list in all_pi_alarms.items():
        if isinstance(pi_alarms_list, list):
            for alarm in pi_alarms_list:
                if isinstance(alarm, dict) and alarm.get("date") == current_date_str and alarm.get("time") == current_time_str:
                    logger.info(f"MATCH! Allarme per PI `{pi_id}` alle {current_date_str} {current_time_str}")
                    triggered_pi_ids_this_minute.add(pi_id)
                    alarms_to_delete_this_minute.append({"pi_id": pi_id, "alarm_data": alarm})

```

Questo frammento di codice illustra come il bot recupera tutte le sveglie da Firebase e le confronta con l'ora e la data correnti. Se viene trovata una corrispondenza, l'ID del Raspberry Pi associato a quella sveglia viene aggiunto a un set (`triggered_pi_ids_this_minute`) e l'allarme stesso viene marcato per la successiva cancellazione.

```

# (all'interno di check_and_trigger_alarms_runner, dopo il controllo allarmi)
# ...
# Leggi tutti i trigger attuali per sapere quali resettare
current_triggers = active_triggers_ref.get()
if not isinstance(current_triggers, dict): current_triggers = {}
# ...
# Imposta i trigger per i Pi attivi questo minuto
for pi_id_to_trigger in triggered_pi_ids_this_minute:
    try:
        if current_triggers.get(pi_id_to_trigger) is not True:
            logger.info(f"Invio trigger=True a Firebase per PI `{pi_id_to_trigger}`")
            db.reference(f'/triggers/{pi_id_to_trigger}').set(True)
        else:
            logger.debug(f"Trigger per PI `{pi_id_to_trigger}` già True.")
    except Exception as e_set:
        logger.error(f"Errore impostando trigger per {pi_id_to_trigger}: {e_set}")

```

Una volta identificati i Raspberry Pi le cui sveglie devono suonare, il bot imposta il loro rispettivo flag di trigger su true in Firebase. Viene fatto un controllo per evitare scritture ridondanti se il trigger è già true.

```
# Resetta i trigger per i Pi che erano attivi ma non lo sono questo minuto
for pi_id_was_active, trigger_value in current_triggers.items():
    if trigger_value is True and pi_id_was_active not in triggered_pi_ids_this_minute:
        try:
            logger.info(f"Reset trigger=False a Firebase per PI `{pi_id_was_active}`")
            db.reference(f'/triggers/{pi_id_was_active}').set(False)
        except Exception as e_reset:
            logger.error(f"Errore resettando trigger per {pi_id_was_active}: {e_reset}")
# ...|
```

Per i Raspberry Pi che avevano un trigger attivo nel ciclo precedente ma non hanno sveglie attive nel minuto corrente, il trigger viene resettato a false. Questo assicura che la sveglia sul Pi si spenga se non ci sono più allarmi attivi per quel dispositivo in quel momento.

Clock.py

Lo script `clock.py` è il software eseguito direttamente sul Raspberry Pi Zero e costituisce il cuore pulsante della sveglia. È responsabile dell'inizializzazione e del controllo di tutti i componenti hardware collegati, come il display LCD, i LED di segnalazione, il motorino a vibrazione e i pulsanti di interazione. Inoltre, gestisce la comunicazione con Firebase per leggere lo stato del "trigger" della sveglia impostato dal `telegram_bot.py` e per ottenere un ID univoco che identifichi il dispositivo nella rete. Lo script si occupa anche di visualizzare informazioni utili sull'LCD, come l'ora corrente, lo stato della sveglia e messaggi di sistema, e di rispondere agli input dell'utente provenienti dai pulsanti fisici.

```

# Variabile per tenere traccia del minuto corrente (per il reset del flag)
current_minute = None
print("Inizializzazione completata. In attesa di eventi...")

while True:
    now = datetime.now(tz_info)
    if current_minute is None or now.minute != current_minute:
        current_minute = now.minute
        alarm_manually_disabled = False # Reset del flag

    if display_mode == 'showing_id' and id_display_start_time is not None:
        if time.monotonic() - id_display_start_time > ID_DISPLAY_DURATION:
            display_mode = 'clock' # Torna alla modalità orologio
            id_display_start_time = None
            last_lcd_message = "" # Forza l'aggiornamento dell'LCD
    if display_mode == 'vibrator_message' and vibrator_message_start_time is not None:
        if time.monotonic() - vibrator_message_start_time > VIBRATOR_MESSAGE_DURATION:
            display_mode = 'clock' # Torna alla modalità orologio
            vibrator_message_start_time = None
            last_lcd_message = "" # Forza l'aggiornamento dell'LCD

    if display_mode == 'clock':
        current_trigger_state = False
        firebase_error = False
        try:
            trigger_value = db.reference(f"/triggers/{MY_PI_ID}").get()
            current_trigger_state = (trigger_value is True) # Converte in booleano
            firebase_error = False
        except Exception as e:
            print("Firebase Admin error:", e)
            current_trigger_state = last_trigger_state
            firebase_error = True

        if not firebase_error: # Esegui solo se la lettura da Firebase è andata a buon fine
            if current_trigger_state and not last_trigger_state:
                alarm_manually_disabled = False # Resetta la disabilitazione manuale
                time_button_pressed = None
                light_lds() # Accende LED e (se abilitato) motore
            elif not current_trigger_state and last_trigger_state:
                alarm_manually_disabled = False # Resetta la disabilitazione manuale
                time_button_pressed = None
                turn_off_lds() # Spegne LED e motore

        if not current_trigger_state or alarm_manually_disabled:
            turn_off_lds()
        elif current_trigger_state and not alarm_manually_disabled:
            light_lds()

        if not firebase_error:
            last_trigger_state = current_trigger_state

    if lcd: # Controlla se l'oggetto LCD è stato inizializzato
        now_lcd = datetime.now(tz_info)
        data_lcd = now_lcd.strftime("%Y-%m-%d")
        ora_lcd = now_lcd.strftime("%H:%M:%S")
        new_message = ""
        message_set = False # Flag per indicare se un messaggio prioritario è stato impostato

        if alarm_manually_disabled and time_button_pressed is not None:
            if time.monotonic() - time_button_pressed <= DISABLED_MESSAGE_DURATION:
                new_message = f"Sveglia disab.\n{ora_lcd}"
                message_set = True

        if not message_set:
            effective_trigger_state = last_trigger_state if firebase_error else current_trigger_state
            if effective_trigger_state and not alarm_manually_disabled:
                new_message = f"SVEGLIA ATTIVATA!\n{ora_lcd}"
                message_set = True

        if not message_set:
            if firebase_error:
                new_message = f"Errore Rete FB\n{ora_lcd}"
            else:
                new_message = f"{data_lcd}\n{ora_lcd}" # Messaggio standard: data e ora

        if new_message != last_lcd_message:
            lcd.clear()
            lcd.message(new_message)
            last_lcd_message = new_message

    sleep(POLLING_INTERVAL)
else:
    # In modalità differenti (showing_id o vibrator_message),
    # il polling è più frequente per controllare rapidamente il timeout del messaggio.
    sleep(0.5)

```

Il ciclo while True all'interno dello script `clock.py` è il vero e proprio motore che anima la sveglia, mantenendola costantemente attiva e reattiva agli eventi esterni e agli input dell'utente. All'inizio di ogni ciclo, lo script si preoccupa di alcune operazioni preliminari di gestione dello stato: una di queste è il reset automatico della variabile `alarm_manually_disabled` allo scoccare di ogni nuovo minuto. Questa logica assicura che se un utente disabilita manualmente una sveglia attiva, tale disabilitazione sia temporanea e specifica per quell'istanza dell'allarme; al minuto successivo, il sistema sarà pronto a reagire a un nuovo eventuale segnale di sveglia senza rimanere bloccato in uno stato di disabilitazione indefinito.

Successivamente, il ciclo gestisce la visualizzazione temporanea di messaggi informativi sull'LCD. Il sistema prevede infatti diverse "modalità di visualizzazione" (`display_mode`). Se, ad esempio, l'utente ha premuto il pulsante per visualizzare l'ID del Raspberry Pi (modalità `'showing_id'`) o quello per mostrare lo stato del motorino a vibrazione (modalità `'vibrator_message'`), lo script controlla se il tempo predefinito per la visualizzazione di questi messaggi è scaduto. Se lo è, riporta automaticamente il `display_mode` alla modalità principale, quella denominata `'clock'`, e resetta il contenuto dell'LCD per forzare un aggiornamento con le informazioni dell'orologio.

Quando il dispositivo si trova nella sua modalità operativa standard, ovvero `display_mode == 'clock'`, il ciclo entra nel vivo del suo funzionamento. La prima azione cruciale è la lettura dello stato del trigger da Firebase; si tratta di un semplice segnale booleano (vero o falso) che viene impostato dal `telegram_bot.py` per indicare se una sveglia debba essere attiva o meno per quel specifico Raspberry Pi. Lo script tenta di recuperare questo valore e gestisce anche eventuali errori di comunicazione con Firebase: in caso di problemi di rete, per prudenza, si affida all'ultimo stato del trigger noto per evitare comportamenti anomali.

Una volta ottenuto lo stato del trigger, lo script ne analizza le variazioni rispetto al ciclo precedente. Se il trigger è appena passato da falso a vero, significa che una sveglia è appena scattata: in questo caso, la disabilitazione manuale viene resettata e si attivano i LED e, se abilitato, viene attivato pure il motorino a vibrazione. Al contrario, se il trigger passa da vero a falso, la sveglia è terminata o è stata cancellata dal bot, quindi i LED e il motore vengono spenti. Indipendentemente dalle transizioni, il sistema prevede un controllo continuo: se il trigger è false o se l'utente ha premuto il pulsante per disabilitare manualmente la sveglia attiva, tutti gli indicatori (LED e motore) vengono spenti. Se invece il trigger è true e non c'è una disabilitazione manuale in corso, gli indicatori vengono accesi.

Parallelamente alla gestione del trigger, lo script si occupa di aggiornare costantemente le informazioni visualizzate sull'LCD, costruendo dinamicamente il messaggio da mostrare: se la sveglia è stata disabilitata manualmente da poco, mostrerà "Sveglia disab."; se la sveglia è attiva (trigger true e non disabilitata manualmente), visualizzerà "SVEGLIA ATTIVA!"; in caso di problemi di connessione a Firebase, informerà l'utente con "Errore Rete FB". In assenza di queste condizioni particolari, l'LCD mostrerà la data e l'ora correnti, formattate secondo il fuso orario impostato. Per ottimizzare le prestazioni l'LCD viene effettivamente aggiornato solo se il nuovo messaggio da visualizzare è diverso da quello precedente. Al termine di queste operazioni in modalità `'clock'`, lo script attende per un breve intervallo (`POLLING_INTERVAL`) prima di iniziare un nuovo ciclo.

Se, invece, il `display_mode` non è '`clock`' (perché si sta mostrando l'ID del Pi o lo stato del vibratore), la logica principale di gestione della sveglia e dell'aggiornamento dell'orologio viene temporaneamente sospesa. In queste modalità alternative, lo script si limita a una pausa più breve, controllando con maggiore frequenza se il tempo per il messaggio temporaneo è scaduto, per poi ritornare il prima possibile alla modalità '`clock`' e riprendere le sue funzioni primarie. Questo assicura che i messaggi informativi siano visibili ma non blocchino indefinitamente il funzionamento principale del dispositivo come sveglia e orologio.

Configurazione del Servizio di Avvio di `clock.py`

```
GNU nano 5.4                                     svegliasordi.service
[Unit]
Description=SvegliaSordi Clock Script
After=multi-user.target network-online.target
Wants=network-online.target

[Service]
ExecStart=/usr/bin/python3 /home/giuse/Desktop/SvegliaSordi/clock.py
WorkingDirectory=/home/giuse/Desktop/SvegliaSordi
StandardOutput=journal
StandardError=journal
Restart=always
User=giuse
[Install]
WantedBy=multi-user.target
```

Per garantire che la sveglia sia operativa immediatamente dopo l'accensione del Raspberry Pi Zero, senza la necessità di un intervento manuale per lanciare lo script, è stato configurato un servizio systemd. Questo sistema di init, standard sulle moderne distribuzioni Linux come Raspberry Pi OS, permette di gestire l'avvio, l'arresto e il monitoraggio di applicazioni e script come servizi di background.

È stato creato un file systemd personalizzato, denominato `svegliasordi.service`, come si può osservare dallo screenshot, e collocato in `/etc/systemd/system/`. Questo file di configurazione specifica i dettagli per l'esecuzione dello script `clock.py`, includendo il percorso completo allo script, la directory di lavoro (`/home/giuse/Desktop/SvegliaSordi/` per assicurare il corretto reperimento dei file `pi_id.txt` e `firebaseKey.json`), e l'utente con cui eseguire il processo.

La configurazione del servizio prevede anche che lo script si avvii solo dopo che la connettività di rete è stabilita (`After=network-online.target`), requisito fondamentale per la comunicazione con Firebase. Inoltre, è stata impostata la direttiva `Restart=always` per far sì che, in caso di chiusura inaspettata dello script, systemd tenti automaticamente di riavviarlo; in tal modo si aumenta l'affidabilità del dispositivo. L'output standard e gli errori generati dallo script `clock.py` vengono reindirizzati al journal di sistema, facilitando il monitoraggio e il debugging tramite il comando `journalctl`.

Dopo aver creato il file di servizio, sono stati eseguiti i comandi `sudo systemctl daemon-reload` per informare systemd della nuova unità, e `sudo systemctl enable svegliasordi.service` per abilitarne l'avvio automatico a ogni boot del sistema. In tal modo, all'accensione del Raspberry

Pi, lo script `clock.py` viene lanciato autonomamente in background, rendendo la sveglia immediatamente funzionale.

Interazione tramite Telegram Bot

Creazione del bot

Per consentire un controllo remoto e una gestione flessibile della sveglia, è stata implementata un'interfaccia utente basata su un bot Telegram. La creazione di questo intermediario software è un processo standardizzato all'interno dell'ecosistema Telegram, reso possibile grazie a "BotFather", il bot ufficiale fornito da Telegram per la registrazione e la gestione di altri bot.

La procedura per creare un nuovo bot inizia avviando una conversazione con BotFather direttamente dall'applicazione Telegram. Una volta aperta la chat, si invia il comando `/newbot`. A questo punto, BotFather guida l'utente attraverso i passaggi successivi: richiede dapprima un nome descrittivo per il bot, che sarà quello visualizzato agli utenti nelle conversazioni. Successivamente, è necessario scegliere un username univoco per il bot; questo username deve terminare con la parola "bot" e BotFather ne verificherà l'unicità a livello globale. Se l'username scelto è valido e disponibile, BotFather confermerà la creazione del bot e, cosa fondamentale, fornirà un token di accesso HTTP API. Si tratta di una stringa alfanumerica che agisce come una chiave segreta, indispensabile per permettere allo script Python (`telegram_bot.py`) di comunicare con l'API di Telegram e quindi di controllare il comportamento del bot appena creato. È di fondamentale rilevanza conservare questo token in modo sicuro, poiché chiunque ne sia in possesso potrebbe assumere il controllo del bot. Una volta ottenuto il token e configurato nello script di backend, il bot diventa operativo e può iniziare a ricevere ed elaborare comandi specifici definiti per questo progetto.



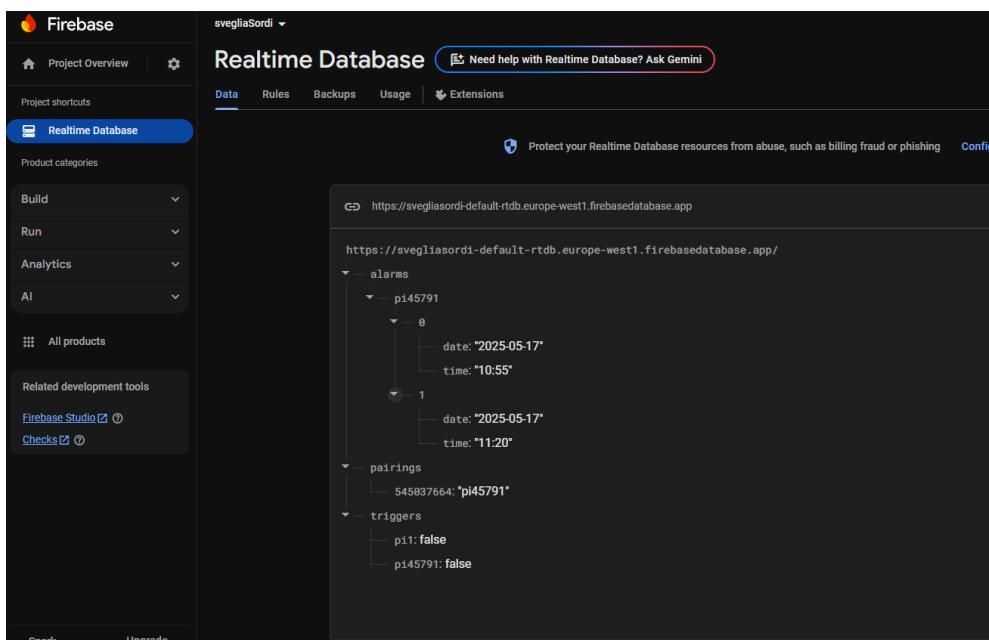
Gli utenti possono interagire con la sveglia intelligente utilizzando i seguenti comandi principali:
`/pair`, per associare il proprio account Telegram a un specifico dispositivo Raspberry Pi;
`/unpair`, per rimuovere tale associazione; `/add`, per impostare una nuova sveglia specificando data e ora; `/list`, per visualizzare tutte le sveglie attive per il dispositivo associato; e infine `/delete`, per cancellare una sveglia precedentemente impostata. Questa interfaccia testuale offre un metodo pratico e accessibile da qualsiasi dispositivo con Telegram per la gestione completa del sistema di allarme.

Connessione al database Firebase

Per orchestrare la comunicazione e la persistenza dei dati tra il bot Telegram e i dispositivi Raspberry Pi, il progetto si affida a Firebase Realtime Database, una piattaforma NoSQL cloud di Google che offre sincronizzazione in tempo reale. L'adozione di Firebase inizia con la creazione di un nuovo progetto, seguita dall'attivazione del servizio Realtime Database, specificando la regione del server in questo caso specifico, `europe-west1`, come si evince dall'URL del database

<https://svegliasordi-default-rtbd.firebaseioapp.com>.

L'interazione programmatica da parte degli script di backend (`telegram_bot.py` e `clock.py`) con Firebase avviene in modo sicuro attraverso il Firebase Admin SDK. Questo SDK richiede un file di credenziali di servizio, tipicamente denominato `firebaseKey.json`, per autenticare le richieste con privilegi amministrativi. Tale file, contenente chiavi private, viene generato dalla sezione "Account di servizio" nelle impostazioni del progetto Firebase e deve essere trattato con la massima riservatezza, evitando la sua inclusione in repository pubblici, per garantire la sicurezza dell'infrastruttura.



The screenshot shows the Firebase Realtime Database interface. On the left, there's a sidebar with project settings like 'Project Overview', 'Realtime Database' (which is selected), 'Build', 'Run', 'Analytics', 'AI', and 'All products'. Below that are 'Related development tools' with links to 'Firebase Studio' and 'Checks'. At the bottom are 'Spark' and 'Upgrade' buttons. The main area is titled 'Realtime Database' with tabs for 'Data', 'Rules', 'Backups', 'Usage', and 'Extensions'. It shows a URL 'https://svegliasordi-default-rtbd.firebaseioapp.com/' and a warning about protecting resources from abuse. The data structure is displayed as a tree view:

```
https://svegliasordi-default-rtbd.firebaseioapp.com/
  alarms
    pi45791
      0
        date: "2025-05-17"
        time: "10:55"
      1
        date: "2025-05-17"
        time: "11:20"
    pairings
      545037664: "pi45791"
    triggers
      p1: false
      pi45791: false
```

La struttura dei dati all'interno del Realtime Database, come illustrato nello screenshot della console Firebase, è organizzata gerarchicamente in formato JSON per facilitare la gestione delle informazioni. Al primo livello troviamo tre nodi principali: `/alarms`, `/pairings` e `/triggers`. Il nodo `/alarms` contiene, per ogni ID univoco di Raspberry Pi, un elenco di sveglie programmate, ciascuna definita da una data e un'ora. Il nodo `/pairings` gestisce le associazioni tra gli ID degli utenti Telegram e gli ID dei Raspberry Pi a cui sono collegati,

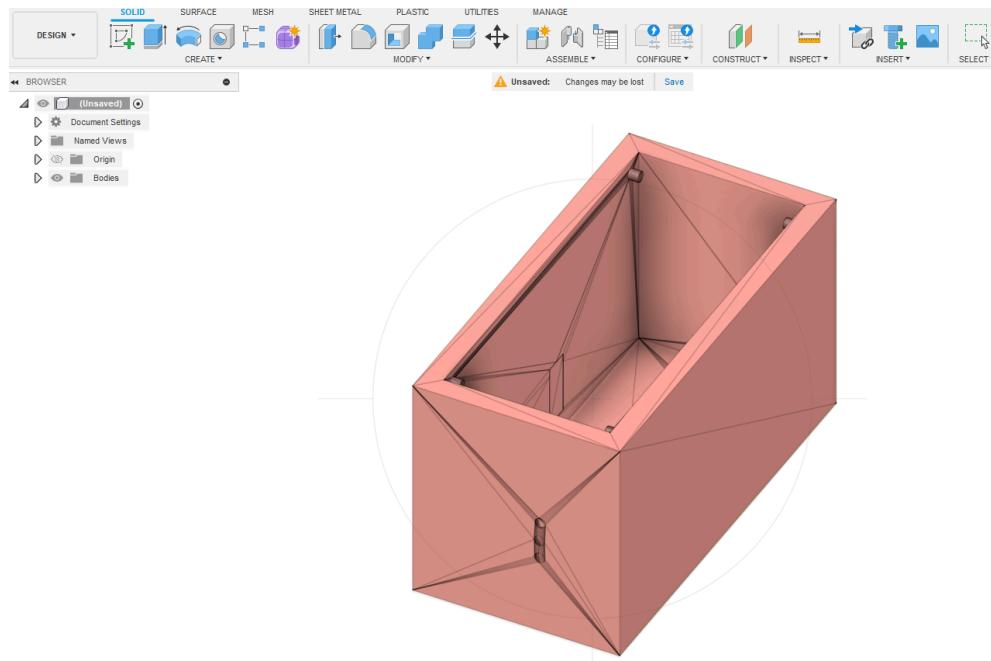
permettendo al bot di indirizzare correttamente comandi e informazioni. Infine, il nodo `/triggers` è fondamentale per l'attivazione delle sveglie: per ogni Raspberry Pi, memorizza un semplice valore booleano che segnala al dispositivo fisico se deve attivare o disattivare la sveglia. Questa architettura dati consente una gestione flessibile e una comunicazione efficiente per il sistema di allarme distribuito.

Stampa 3D del case

Progettazione del case

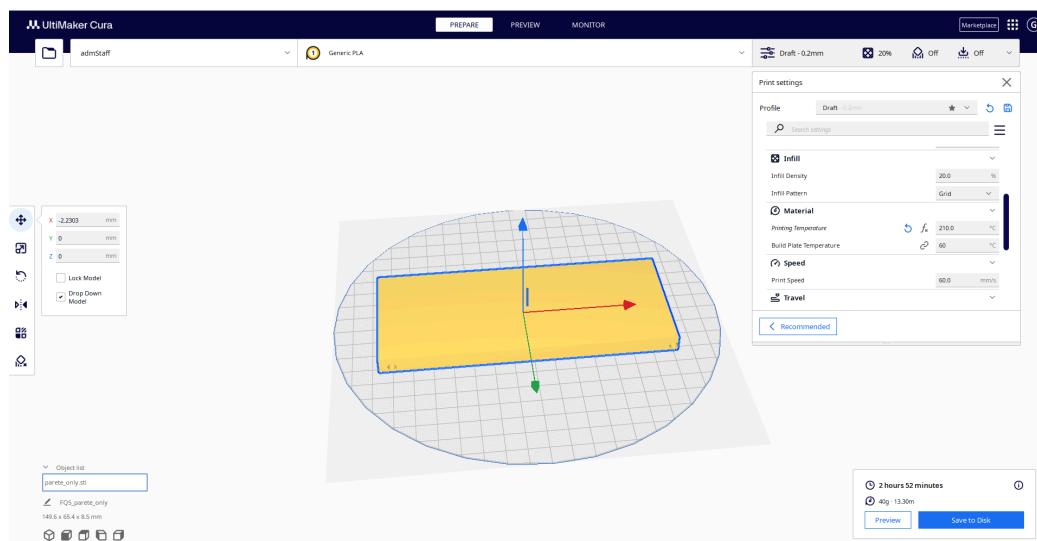
Per conferire al progetto un aspetto finito e per proteggere i componenti elettronici interni, è stato progettato e realizzato un case su misura mediante stampa 3D. La modellazione tridimensionale dell'involucro è stata interamente eseguita utilizzando il software Fusion 360. Le dimensioni esterne del case sono state fissate a 20 cm in lunghezza, 11 cm in larghezza e 8 cm in altezza, offrendo così uno spazio adeguato per alloggiare tutti i componenti, inclusi il Raspberry Pi Zero, la breadboard, il display LCD, la batteria e gli altri elementi circuituali.

Un aspetto chiave della progettazione del case è stata l'implementazione di un pratico meccanismo di incastro per una delle pareti. Questa soluzione è stata concepita per consentire un facile accesso ai componenti interni per eventuali manutenzioni, modifiche o per l'inserimento della batteria, senza la necessità di viti o altri elementi di fissaggio esterni. La parete rimovibile si integra con il corpo principale del case attraverso guide e geometrie complementari che ne assicurano una chiusura stabile e al contempo una semplice apertura quando necessario.



Il risultato di questa fase di progettazione sono due file distinti in formato **.stl**, pronti per essere processati da un software di slicing e inviati a una stampante 3D. Il primo file, denominato **case_only_no.stl**, definisce la struttura principale della scatola, concepita già priva della parete destinata all'apertura, come si può notare nello screenshot. Il secondo file, **parete_only_no_holes.stl**, rappresenta la parete separata, dotata del meccanismo complementare di incastro che le permette di essere montata e smontata agevolmente dal corpo principale del case. Questa suddivisione in due parti stampabili separatamente ha facilitato il processo di stampa e ha permesso di ottimizzare l'orientamento dei pezzi per una migliore qualità e resistenza meccanica.

Stampa



Una volta finalizzati i modelli 3D si è proceduto alla preparazione necessaria per la stampa 3D. Tale fase, nota come slicing, è stata effettuata utilizzando il software Ultimaker Cura, un'applicazione ampiamente diffusa per la sua interfaccia intuitiva e le numerose opzioni di personalizzazione; un esempio dell'ambiente di lavoro è visibile nello screenshot appena presentato.

Per garantire un buon compromesso tra resistenza meccanica dei componenti, qualità superficiale e tempi di realizzazione, sono stati definiti specifici parametri di stampa. È stato scelto un valore di riempimento (infill) del 20%, con un pattern a griglia per l'interno dei pezzi. La temperatura di estrusione per il filamento è stata impostata a 210°C, mentre la temperatura del piatto di stampa è stata mantenuta a 60°C, valori ottimali per il materiale utilizzato. Come materiale di stampa è stato impiegato filamento PLA (Acido Polilattico) di colore grigio, apprezzato per la sua facilità di stampa, la bassa tendenza al warping e la buona resa estetica per prototipi.

La realizzazione fisica dei due componenti del case è stata affidata a una stampante 3D modello Longer LK4 Pro. L'intero processo di stampa, comprensivo di entrambi i file, ha richiesto un tempo complessivo di circa 1 giorno e 13 ore per essere completato.

Esperienza, difficoltà e miglioramenti

Sfide incontrate e possibili soluzioni

Durante lo sviluppo del progetto, sono emerse alcune sfide tecniche che hanno richiesto modifiche al piano originale e l'adozione di soluzioni alternative.

Inizialmente, era stata prevista l'integrazione di un sensore di movimento (motion sensor) come metodo per disattivare automaticamente l'allarme della sveglia, una volta rilevato il movimento dell'utente vicino alla sveglia. Tuttavia, durante la fase di test, si è riscontrato che il sensore tendeva a generare un numero eccessivo di falsi positivi, attivandosi anche in assenza di un reale movimento intenzionale dell'utente. Questa inaffidabilità avrebbe compromesso l'esperienza d'uso, portando potenzialmente alla disattivazione involontaria della sveglia. Di fronte a tale problematica, e per garantire un funzionamento più robusto e prevedibile, si è deciso di rimuovere il sensore di movimento dal design finale, affidando la disattivazione della sveglia esclusivamente all'interazione manuale tramite il pulsante dedicato.

Un'altra difficoltà si è presentata durante la fase di realizzazione fisica del case. Il primo tentativo di stampa 3D è stato effettuato utilizzando una stampante differente da quella poi impiegata per i pezzi finali. Purtroppo, questa prima stampante presentava un difetto sul piatto di stampa, specificamente un'area danneggiata o "tagliata", che causava una scarsa adesione del materiale e la deformazione delle stampe in corrispondenza di quel punto. Questo rendeva impossibile ottenere pezzi precisi e dimensionalmente corretti, specialmente per un case con meccanismi di incastro. Per ovviare a questo inconveniente, è stato necessario cambiare stampante, optando per la Longer LK4 Pro, che ha permesso di ottenere stampe di qualità adeguata e di realizzare con successo i componenti del case come da progetto.

Limiti attuali del progetto

Nonostante il raggiungimento degli obiettivi prefissati, il progetto presenta alcuni limiti intrinseci e aree suscettibili di futuri miglioramenti.

Una prima considerazione riguarda la dipendenza dalla connettività a Internet per la gestione delle sveglie. Attualmente, il sistema si basa interamente su Firebase per la memorizzazione e il recupero degli orari delle sveglie. In caso di disconnessione del Raspberry Pi da Firebase o dal servizio Internet in generale, non è previsto un meccanismo di backup locale delle sveglie. Questo significa che, in assenza di connessione al momento previsto per una sveglia, questa

potrebbe non attivarsi, e nuove sveglie non potrebbero essere sincronizzate dal bot Telegram. L'implementazione di una cache locale delle sveglie sul Raspberry Pi rappresenterebbe un significativo passo avanti per aumentarne l'affidabilità.

Dal punto di vista degli attuatori fisici, l'intensità della notifica tattile fornita dal motorino a vibrazione è risultata piuttosto contenuta; il motorino selezionato produce infatti una vibrazione leggera che potrebbe non essere sufficientemente percettibile in tutte le situazioni o da tutti gli utenti. Analogamente, i LED utilizzati per la segnalazione visiva, pur essendo funzionali, sono di dimensioni standard e la loro intensità luminosa potrebbe essere considerata non ottimale per un impatto visivo forte, specialmente in ambienti luminosi. La selezione di un motorino a vibrazione più potente o di LED ad alta luminosità, o l'utilizzo di più LED, potrebbe migliorare l'efficacia di queste notifiche.

Infine, per quanto riguarda l'involucro stampato in 3D, sebbene funzionale, il meccanismo di incastro della parete laterale del case, pur permettendo l'accesso ai componenti interni, potrebbe beneficiare di ulteriori iterazioni di progettazione. Una revisione del design potrebbe portare a un sistema di chiusura più robusto, più semplice da manovrare o con una finitura estetica superiore, migliorando l'esperienza d'uso complessiva e la durabilità del case stesso.

I limiti individuati non sminuiscono la funzionalità del prototipo attuale, ma offrono spunti concreti per future evoluzioni e perfezionamenti del progetto.

Idee per un miglioramento in prospettiva futura

Il prototipo attuale della sveglia intelligente rappresenta una solida base funzionale, ma offre numerosi spunti per evoluzioni e miglioramenti successivi che potrebbero arricchirne significativamente le capacità e l'esperienza d'uso.

Un'area di sicuro interesse è l'ottimizzazione del case stampato in 3D. Si potrebbe esplorare l'integrazione di un sistema di chiusura magnetico per la parete rimovibile, offrendo una soluzione elegante e pratica rispetto all'attuale meccanismo ad incastro. Inoltre, la progettazione potrebbe essere rivista per includere un alloggiamento dedicato per una batteria ricaricabile, rendendo il dispositivo più portatile e autonomo dall'alimentazione esterna continua, e magari un interruttore di accensione generale.

Parallelamente, si potrebbe intervenire per potenziare l'efficacia delle notifiche fisiche. Come discusso in relazione ai limiti attuali, la sostituzione del motorino a vibrazione con un modello più potente e l'adozione di LED ad alta luminosità, o l'aumento del loro numero, renderebbero le segnalazioni tattili e visive decisamente più incisive e percepibili.

Un'aggiunta particolarmente significativa in termini di accessibilità sarebbe l'integrazione di un buzzer o di un piccolo altoparlante per fornire una notifica sonora. Questo renderebbe la sveglia fruibile anche da utenti ipoidenti o ipoacustici o in situazioni in cui le sole vibrazioni o luci potrebbero non essere sufficienti. Il segnale sonoro potrebbe essere personalizzabile o variare in intensità.

Guardando oltre, si potrebbero considerare funzionalità software più avanzate, come la possibilità di impostare sveglie ricorrenti direttamente dal bot Telegram, o l'integrazione con servizi smart home. Anche l'implementazione di un meccanismo di backup locale delle sveglie sul Raspberry Pi, per garantire il funzionamento anche in assenza temporanea di connettività Internet, rappresenterebbe un importante passo avanti in termini di affidabilità.

Queste idee rappresentano solo alcune delle direzioni che il progetto potrebbe intraprendere, trasformando il prototipo attuale in un dispositivo ancora più completo, versatile e personalizzabile.

Conclusioni

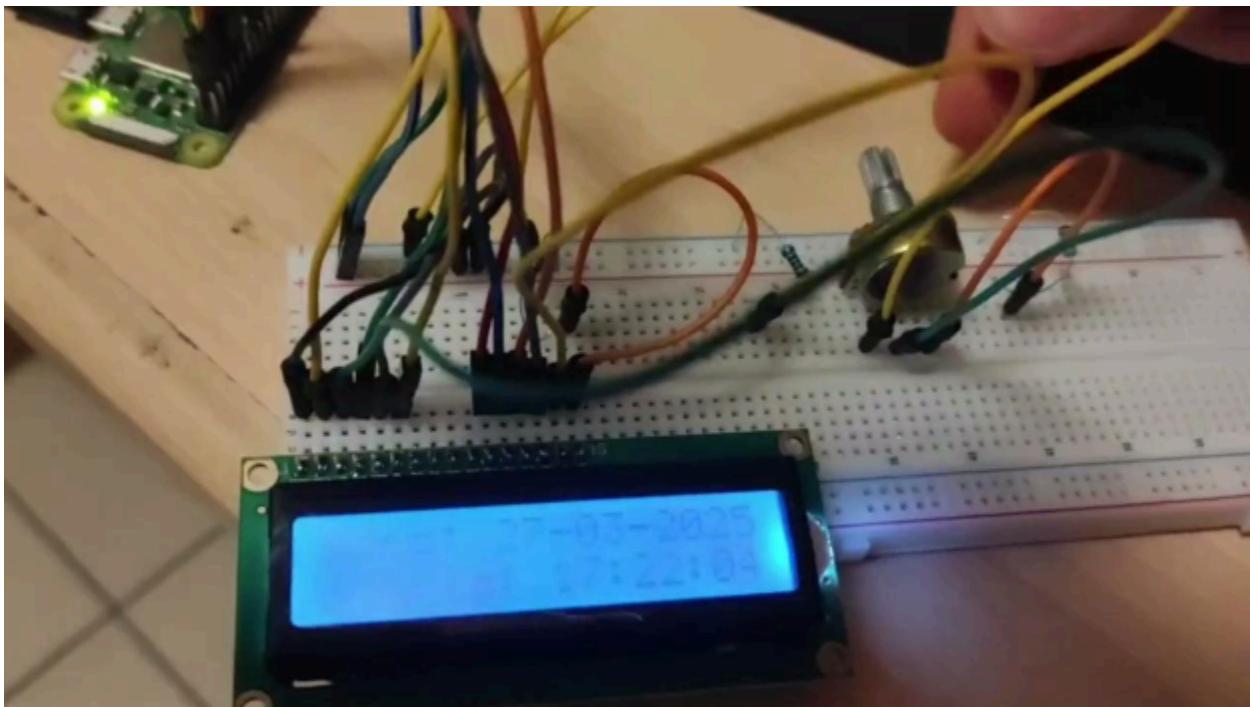
Il progetto "Sveglia per Sordi" ha dimostrato con successo la fattibilità di un dispositivo di allarme alternativo, basato su stimoli visivi e tattili, per rispondere a esigenze specifiche come quella di chi non percepisce le sveglie sonore tradizionali. Partendo da una necessità personale, si è sviluppato un prototipo completo che integra hardware dedicato, un'interfaccia di controllo remoto tramite bot Telegram, e una gestione dei dati basata su cloud Firebase, il tutto racchiuso in un case personalizzato stampato in 3D.

L'iterazione progettuale, che ha incluso la selezione dei componenti, l'assemblaggio dei circuiti, lo sviluppo del software per il Raspberry Pi e per il server del bot, fino alla modellazione e stampa dell'involucro, ha permesso di affrontare e risolvere diverse sfide tecniche, consolidando l'apprendimento pratico. Sebbene il prototipo attuale presenti margini di miglioramento, come l'ottimizzazione delle notifiche fisiche e una maggiore resilienza alla disconnessione da Internet, esso raggiunge l'obiettivo primario di fornire un sistema di sveglia efficace e personalizzabile.

In definitiva, questo progetto non solo offre una soluzione concreta a un problema specifico, ma rappresenta anche un'esperienza formativa significativa nel campo del making, evidenziando l'importanza dell'integrazione tra hardware, software e design fisico per creare prodotti funzionali e mirati alle esigenze dell'utente. Le direzioni future delineate aprono la strada a ulteriori sviluppi che potrebbero trasformare questo prototipo in un prodotto ancora più robusto, accessibile e versatile.

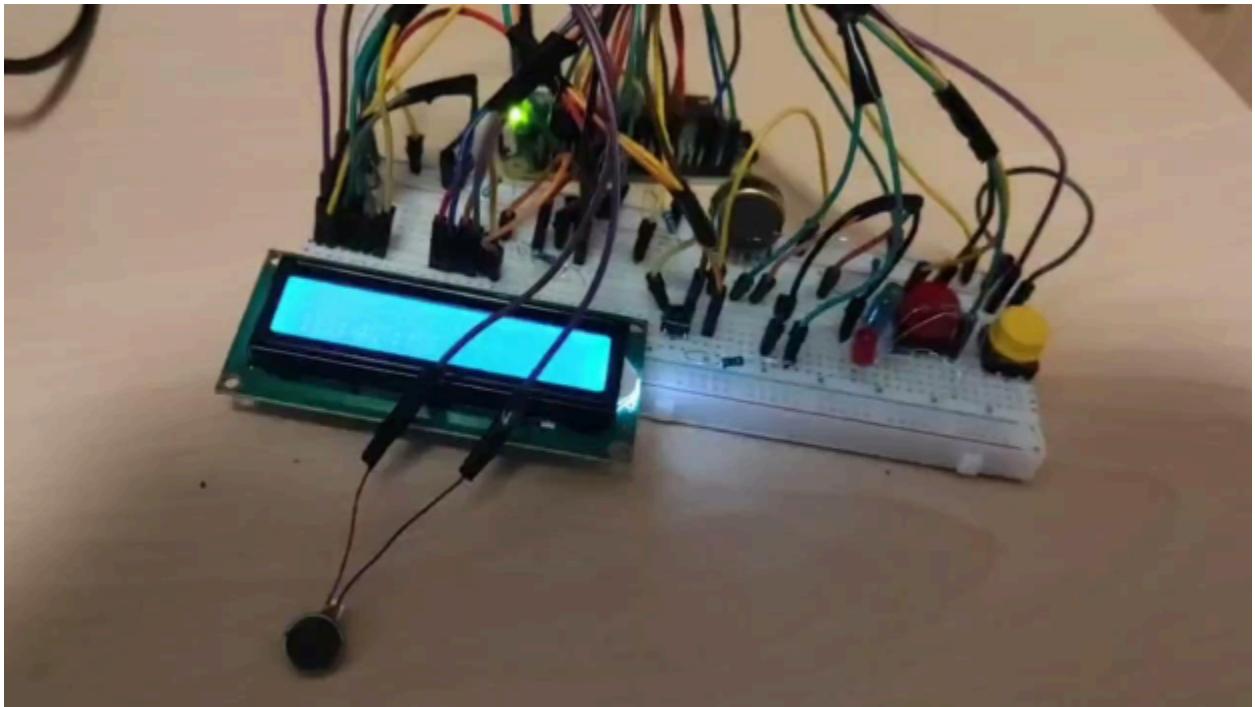
Extra: demo

Demo schermo Lcd



https://drive.google.com/file/d/1Q7jtf0SxnXCgkoWfFYSSuRgkZMN_O1kM/view?usp=sharing

Demo accensione sveglia



https://drive.google.com/file/d/1F-ltq_8ihC_W1T_zRhQAKA3INDmSW-Vr/view?usp=sharing