Sistema di Monitoraggio degli Accessi

A Cura di Simone Bertolini e Gianmarco Santini

1. Introduzione

Controllare e conteggiare gli accessi di una determinata area/locale è molto importante per diverse ragioni.
Principalmente, quando si mette a disposizione un luogo ad altra gente, è fondamentale poter salvaguardare gli oggetti al suo interno, proteggendoli quindi dall'usura dovuta al cattivo utilizzo.
Questo si può attuare sicuramente limitando il numero di persone che possono entrare contemporaneamente all'interno del luogo interessato.
Successivamente un controllo degli accessi consente anche di tenere traccia dell'affluenza di visitatori per orari, e quindi di generare statistiche di ogni tipo: sia relative agli orari più o meno affollati sia al grado di piacimento di quanto esposto.

2. Obbiettivi

L'obiettivo di questo progetto è quello di proporre una soluzione semplice, ma funzionale e precisa, per il monitoraggio degli accessi ad un'area ad uso pubblico avente fino a 10 possibilità di accesso.
Il sistema mira a garantire un alto livello di precisione (minimo livello di errore anche su casi ambigui) seguendo però sempre una filosofia di basso consumo energetico; questo per consentire un lunga durata anche utilizzando un'alimentazione a batteria o tramite accumulatore.
Tutti i dati raccolti sono nuovamente consultabili tramite un'interfaccia utente intuitiva ed alla portata di tutti.
Il sistema in questione è stato studiato appositamente per poter funzionare correttamente in ogni condizione termo-climatica, con ogni livello intensità luminosa (luce/buio) e con qualunque persona.

3. Ruoli e funzionalità

Questa è da considerarsi la prima vera e propria fase della parte di progettazione.
Il nostro obiettivo era quello di avere un'unità *Master* ed N unità S*lave*.
Il ruolo dell'unità *master* è quello di accumulatore, unificatore e correttore dei dati ricevuti degli altri dispositivi nel tempo.
Si potrebbe quasi definire l'unità *master* come un'unità di controllo principale in grado di fornire quanto richiesto all'utente, presentandolo in modo a lui comprensibile ed intuitivo.
Le unità *slave*, invece, svolgono un ruolo di acquisizione ed approssimazione dei dati: infatti ad esse sono direttamente collegati i sensori di monitoraggio ingressi/uscite.

Inoltre, queste, sono configurate in modo da poter acquisire dati anche in assenza di comunicazione con il master per poi inviarli non appena la connessione verrà ripristinata.

4. Scelta del Hardware

La scelta dell'*hardware* da utilizzare è stata fatta basandosi su un concetto fondamentale: ciò che conserva i dati deve essere un dispositivo con una memoria verosimilmente illimitata, affidabile ed abbastanza potente da consentirne la consultazione in qualunque momento.
Oltre tutto questo, era necessario tenere altresì in considerazione la questione del basso consumo energetico.

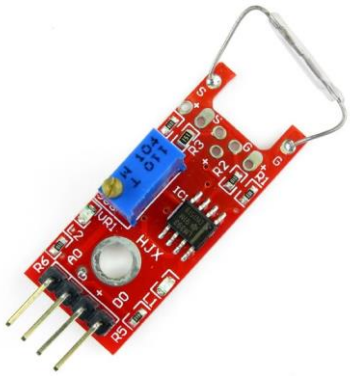
Per il ruolo di *master*, infatti, è stata scelta una scheda STM Nucleo F401RE insieme ad uno *shield* WiFi IDW04A1.
Ci siamo basati su una scheda di sviluppo STM in quanto i prodotti che questa marca rilascia sono spesso sviluppati per essere utilizzati sul professionale.
Questo ci ha quindi dato una garanzia sia per quanto riguarda l'*hardware* stesso che per il supporto in relazione al *firmware*.
La scheda Nucleo F401RE è una *demo board* che monta un F401RET6U come processore le cui caratteristiche sono le seguenti:

Core Size: 32bit
CPU Speed: 168MHz
STM32 F 401 RET 6 / Microcontroller 168MHz, 32bit, CORTEX-M4
Flash: 512KB
RAM: 96KB

Lo *shield* IDW04A1 contiene un modulo WiFi SPWF04SA con un *firmware* di fabbrica aggiornabile tramite seriale o "*On Air*" (passaggio dell'URL).
SPWF04SA è direttamente collegato ad uno *slot* SD-CARD da noi utilizzato come *storage* dei dati.

Per quanto riguarda lo slave, la scelta è ricaduta sul microcontrollore NodeMCU. Questo è una piattaforma IoT open source che include un firmware che gira su ESP8266 con l'SDK Espressif Non-OS e l'hardware basato sul modulo ESP-12. Il dispositivo offre 4MB di memoria flash, 80MHz di clock di sistema, circa 50k di RAM utilizzabile e un ricetrasmettitore Wifi su chip. Inoltre espone la porta seriale del modulo ESP8266 tramite un bridge USB CP2102 che è anche collegato ai pin di avvio del modulo, consentendo una virtualizzazione continua del dispositivo.
Abbiamo scelto NodeMCU in quanto ha un basso consumo, un basso costo e anche se la memoria è limitata e sicuramente sufficiente per la raccolta ed approssimazione dei dati dai sensori da noi utilizzati.

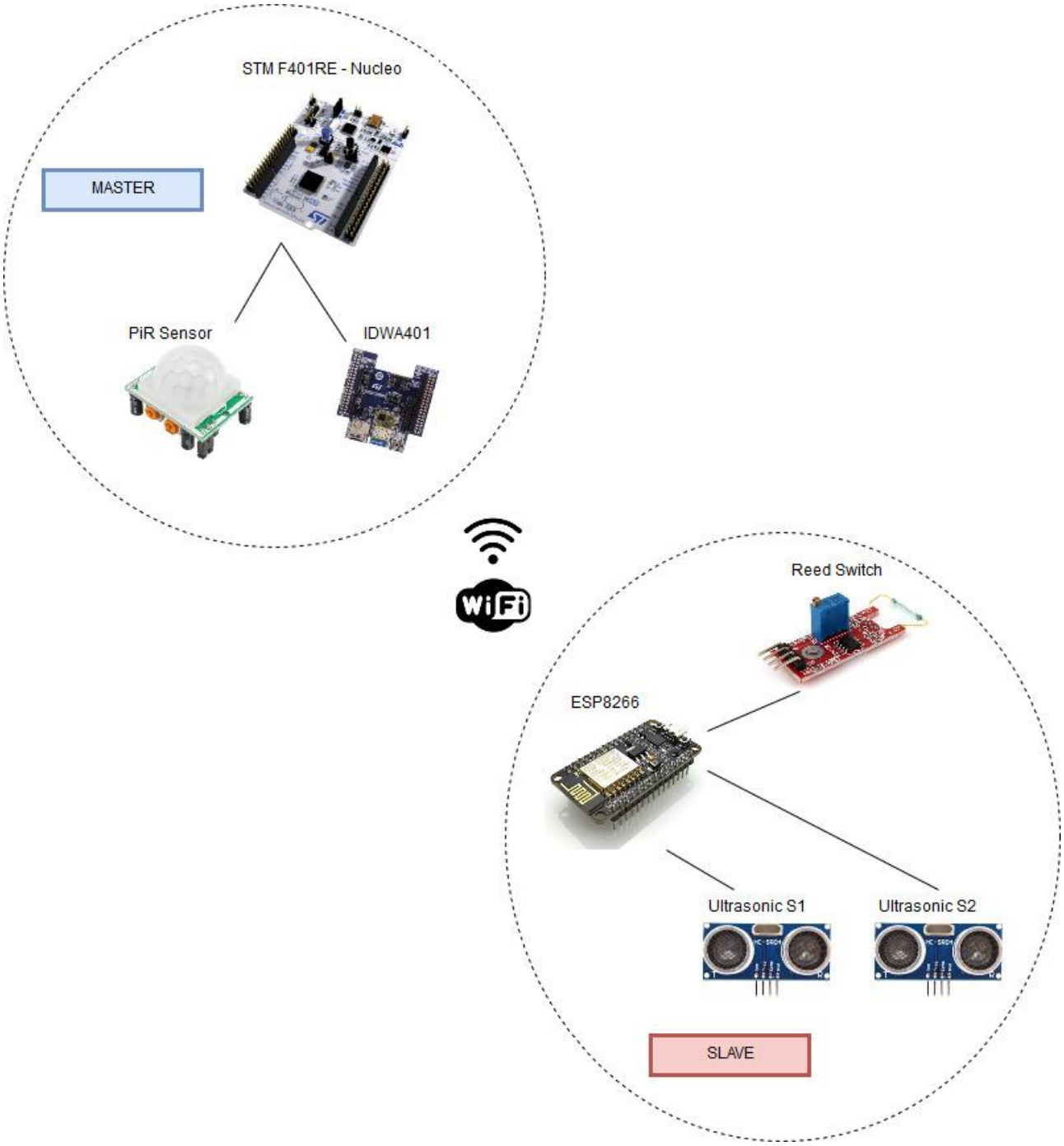
Come sensori esterni sono stati utilizzati moduli ad ultrasuoni HC-SR04, un *reed switch* magnetico ed un sensore PIR classico.



Il costo di questi sensori è pressoché nullo, ma nonostante ciò la loro precisione è risultata sufficiente per raggiungere gli scopi prefissati.

5. Architettura della comunicazione, protocollo

La struttura del sistema risulta quindi essere quella di seguito illustrata.



Come è stato già descritto in precedenza, il sistema è composto da due ruoli principali, un'unità *Master* ed N unità *Slave*.
L'unità *Master* comprende la scheda STM Nucleo, lo *shield* IDW04A1 ed il sensore PIR.
Un'unità Slave invece è composta da una scheda su cui è' stato installato un ESP8266 (Node MCU nel nostro caso).

La comunicazione tra i diversi *slave* e l'unità *master* avviene attraverso una connessione TCP su WiFi 802.11.
Entrambe le unità si aspettano una rete provvista di DHCP e viene utilizzato il *multicast* DNS per il *discovery* tra le *board*: *i*nfatti l'unità *Master,* attraverso pacchetti di *multicast* DNS, fornisce alle unità S*lave* tutte le proprie informazioni di connessione.
Ogni comando di un unità *slave* ha la seguente struttura:

0xFE	1 byte
0xFF	1 byte
0x02	1 byte (* Identificatore Notifica*)
Node_ID	1 byte (* 0 per il Nodo 1 e 1 per il Nodo 2 …*)
Code	1 byte (* Indica quale proprietà è stata modificata *) [0x01 = Ingresso, 0x02 Uscita]
Valore	1 byte (* Nuovo valore *)

In questo pacchetto viene indicato che tipo di comando si sta inviando (notifica == 0x02), l'identificativo del nodo mittente, il codice della proprietà modificata (ingresso o uscita) ed infine la quantità aggiunta o sottratta al valore.
Il fatto che la comunicazione avvenga tramite TCP garantisce la presenza dell'unità Master sull'indirizzo e la porta specificati, ed inoltra che il comando sia stato ricevuto con successo (numero di ritrasmissioni di default ed ACK di conferma).
Nel caso in cui per qualche motivo una modifica al valore di una proprietà non venga trasmessa con successo, la trasmissione della modifica successiva includerà anche quella precedente.
Un'altra comunicazione molto importante è quella tra la scheda Nucleo o lo shield WiFi. Il protocollo con il quale questi due componenti comunicano è un set di comandi chiamato "AT".
Questa tipologia di comandi viene spesso utilizzata in via seriale, ma nel nostro caso vi sarebbe stata la possibilità di impiegarlo anche su SPI (funzionalità che non è però stata utilizzata).
I comandi AT hanno la seguente sintassi:

AT+ [Comando] = [Propietà],[Valore]

6. Ambienti di sviluppo e librerie

Nucleo

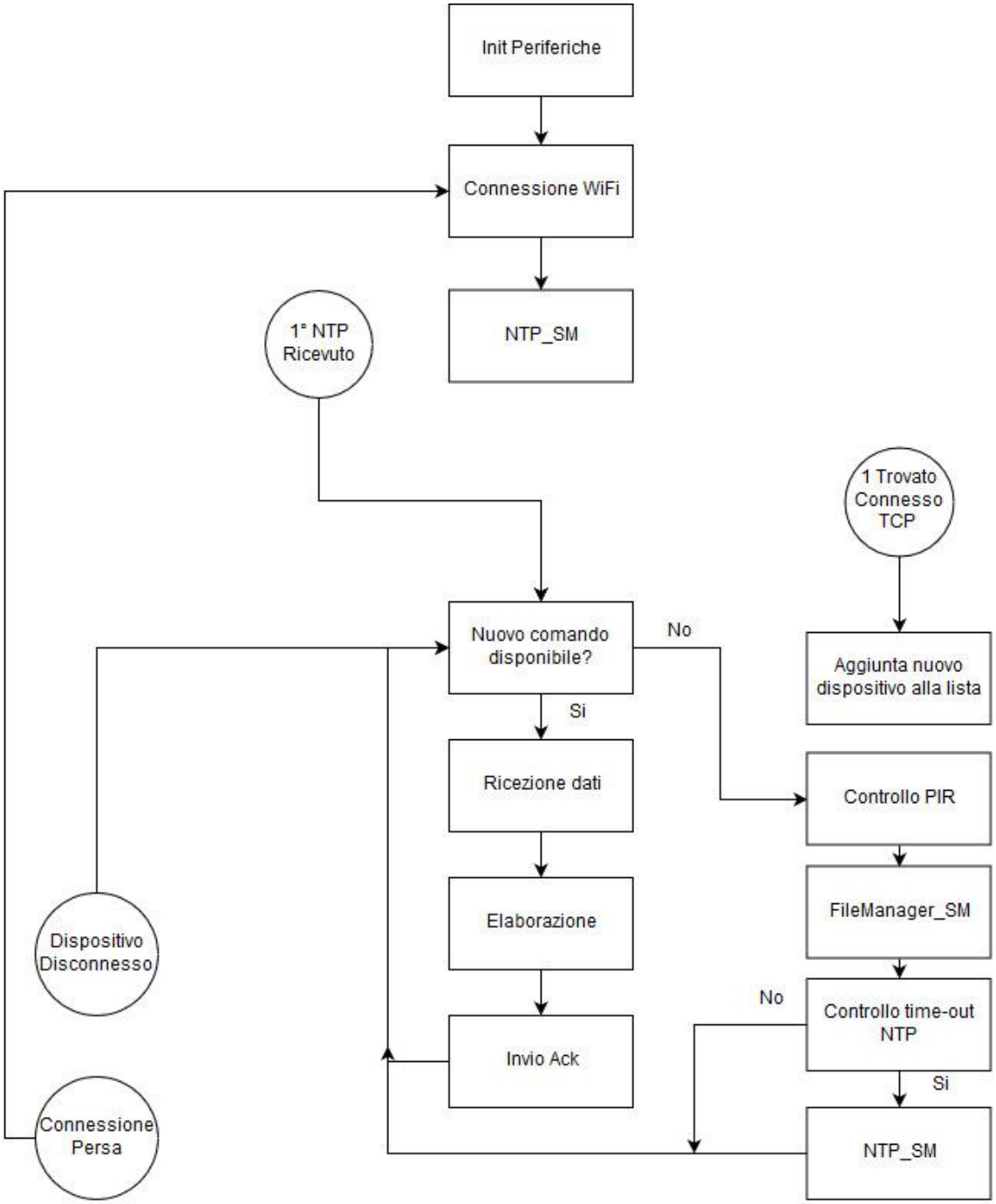
Per sviluppare il *firmware* della *board* nucleo vi erano due strade possibili: l'utilizzo di Mbed, oppure l'impiego di strumenti un po' più a basso livello come uVision Keil oppure Atollic. La scelta

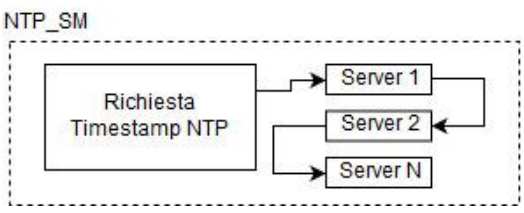
è stata però costretta dal fatto che per MBed non vi era nulla da cui partire per lo sviluppo della comunicazione con lo *shield* IDW.
Le uniche librerie e/o esempi per questo *shield* erano progetti prodotti per Keil ed Atollic.
E' stato dunque scelto di utilizzare Keil come IDE. Questo è un prodotto Arm che consente di sviluppare, compilare e caricare il codice su microcontrollori in linguaggio C.
Le librerie contenevano già un parte di protocollo AT di comunicazione con la *board* IDW, la quale è stata da noi espansa ed integrata con le funzionalità delle quali avevamo bisogno.
La gestione dello *shield* IDW è un po' simile a quello di un *router,* ovvero esso va configurato impostando parametri tramite i comandi AT, cosí che possa poi inviare comandi in caso si verifichino eventi (esempio: connessione stabilita oppure ricevuto pacchetto).
Per lo sviluppo del *firmware* del microcontrollore non è stato utilizzato nessun RTOS, infatti l'applicativo è single thread e gestisce le priorità attraverso interrupt e timer.

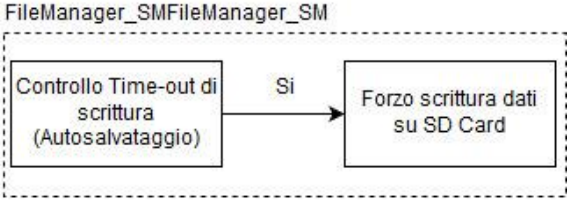
Per lo sviluppo del firmware del microcontrollore su ESP 8266, è stato utilizzato l'ambiente di sviluppo Arduino IDE, che fornisce il supporto alla board una volta aggiunta dal Board Manager.
In modo analogo le librerie utilizzate sono state aggiunte in parte dal Library Manager e in parte aggiunte direttamente nella cartella librerie di Arduino IDE.
Le librerie importate sono "ESP8266Ping" che fornisce le primitive per fare ping di rete, "ESP8266WiFi" che fornisce le primitive per la gestione della connessione WiFi e delle connessioni UDP e TCP tramite WiFi , "ESP8266mDNS" che fornisce le primitive per la gestione di comunicazioni multicast DNS.

7. Descrizione del funzionamento dei singoli dispositivi

Macchina stati unità Master





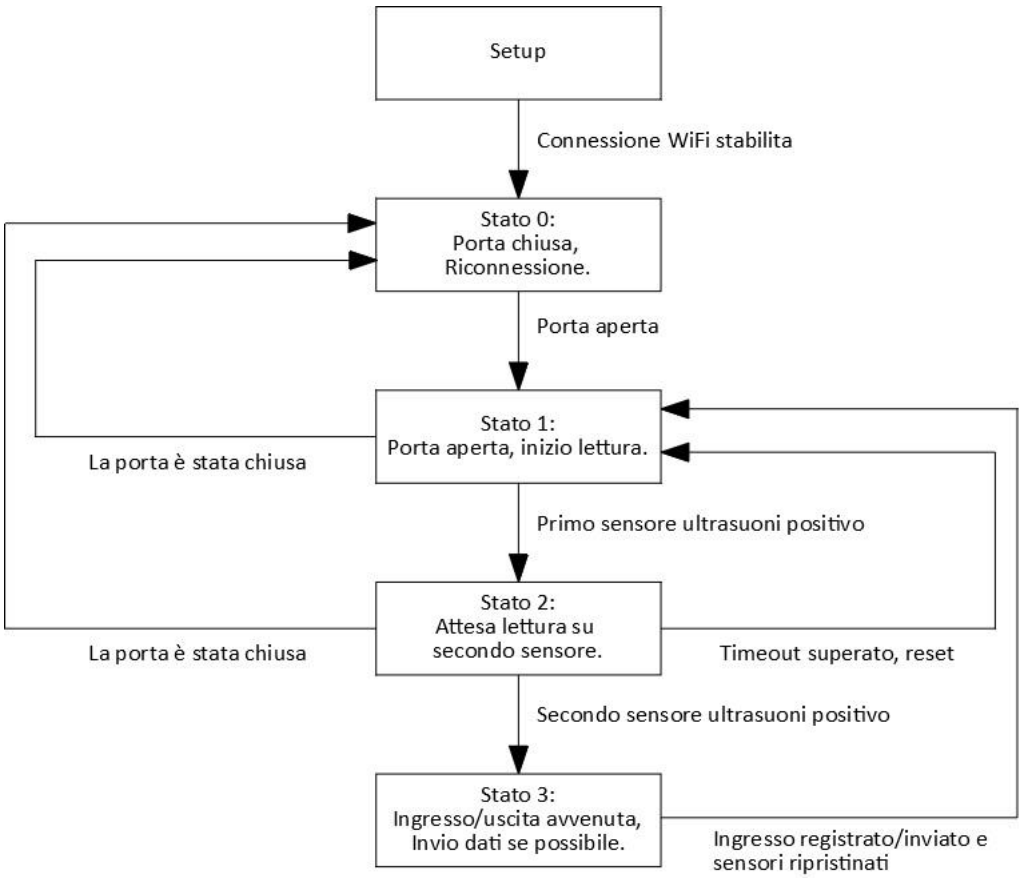


La macchina a stati scheda Master ha come primo *step* l'inizializzazione di tutte le periferiche ovvero led, pir, porta seriale, modulo WiFi ed i vari *timer*.
Successivamente, essa tenta di connettersi alla rete impostata di default; questo processo prosegue ad oltranza fino a quando o viene stabilita una connessione, oppure viene riavviata la scheda.
Una volta ottenuto un indirizzo IP, viene configurato anche il servizio di *multicast* DNS inserendo le informazioni di cui le unità *Slave* hanno bisogno per connettersi ed inviare i dati al *master* (IP e porta).
A seguire viene inviato un pacchetto NTP per la configurazione dell'orologio interno. Anche questo è uno *step* bloccante in quanto in mancanza di un orologio sincronizzato non è possibile elaborare i dati e salvarli in modo corretto.
Le richieste NTP hanno 3 server sui quali fare tentativi, quindi se per qualche motivo uno di essi non fosse disponibile, il firmware proverebbe con i restanti 2.
Le richieste NTP vengono effettuate ogni 2 minuti circa, questo perché il *timer* di auto aggiornamento dell'orologio interno ha uno sfasamento di circa 12 ms ogni secondo calcolato.
Una volta che almeno una risposta NTP è stata ricevuta, viene sincronizzato l'orologio interno e viene aperto il *socket* tcp sul quale dovranno poi connettersi le unità *Slave*.
D'ora in poi tutti i dispositivi che si collegheranno al *socket* prima citato potranno inviare informazioni con la sintassi indicata nel paragrafo precedente.
La macchina a stati prosegue ora iterando sul controllo di ricezione di nuovi comandi.
Nel caso non vi siano comandi da processare, il sistema controlla lo stato del PIR, verifica se è il momento di effettuare un auto-salvataggio dei dati in memoria e se il timer NTP indica che è ora di effettuare nuovamente una richiesta.
Il PIR serve per azzerare gli errori quando il luogo da monitorare è vuoto. Infatti se il PIR non rileva movimenti entro un tempo determinato a priori, il contatore di persone nel locale viene azzerato.
L'auto salvataggio garantisce una minor perdita di dati in caso di caduta di tensione sull'alimentazione.
La scrittura su SD Card prevede l'utilizzo di un *buffer* non più grande di 1 KB quindi tutti i salvataggi vengono eseguiti ogni volta che il *buffer* temporaneo raggiunge quella dimensione oppure quando scatta l'auto salvataggio.
I dati vengono salvati in formato JSON su file di testo denominati [Timestamp in seconds].txt ed aventi la seguente struttura, in cui "time" indica l'istante esatto di tempo in cui il contatore di presenze all'interno della stanza ha assunto il valore "value".

```
[
        {
                “time”:000000000,
                “value”:00
        },
        .
        .
        .
        {
                “time”:000000000,
                “value”:00
        }
]
```

Tutti i *file* vengono poi indicizzati in un ulteriore *file* chiamato db_index.txt che contiene, separati da una “,” tutti i *file* salvati nella memoria. Questo, come vedremo successivamente, serve per evitare l’utilizzo di *script* lato server come PHP o Python.
Nel caso, invece, in cui venga ricevuto un comando, questo viene controllato ed elaborato aggiornando il relativo contatore degli ingressi, infine salvato in memoria per poi essere successivamente scritto, quando sarà il momento, su SD Card.
Agli *Slave* è lasciata la possibilità di connettersi e disconnettersi dal TCP in qualunque momento; se invece si verifica una disconnessione dall’*access point* WiFi, la *board* si preoccuperà di effettuare nuovamente la connessione quando questa tornerà ad essere disponibile.

Macchina stati unità Slave



Nella prima fase di setup il microprocessore si occupa di inizializzare tutte le variabili e i pin di input/output, stabilire la connessione WiFi con la rete specificata, far partire un mDNS responder per la gestione di multicast DNS e prova a recuperare l'indirizzo IP e la porta del master e iniziare una connessione TCP.

Nel caso sia riuscita la connessione WiFi il sistema passa allo stato 0, questo è lo stato in cui la porta e ancora chiusa e quindi il sensore magnetico (reed switch) restituisce un output negativo. Se la porta viene aperta e il valore del sensore cambia allora si passa allo stato 1, questo è lo stato in cui la porta è aperta e si continua a leggere dai sensori finché non avviene una lettura positiva, nel caso il sensore magnetico cambi valore allora si ritorna allo stato 0.

Quando avviene una lettura positiva su uno dei due sensori a ultrasuoni allora si passa allo stato 2, in cui si continua a leggere il valore dell'altro sensore a ultrasuoni finché non restituisce un valore positivo o si raggiunge un valore massimo di iterazioni (timeout), come prima nel caso il sensore magnetico cambi valore si torna allo stato 0.

Se si raggiunge il numero massimo di iterazioni allora il sistema torna allo stato 1, se invece si ottiene un valore positivo dal secondo sensore allora vuol dire che è avvenuto un ingresso o un'uscita e si passa allo stato 3. In questo stato si registra l'avvenuto ingresso/uscita, lo si comunica al master se possibile, si resettano i valori delle variabili e si riporta il sistema allo stato 1.

8. Strumenti di Debug

Unità Master

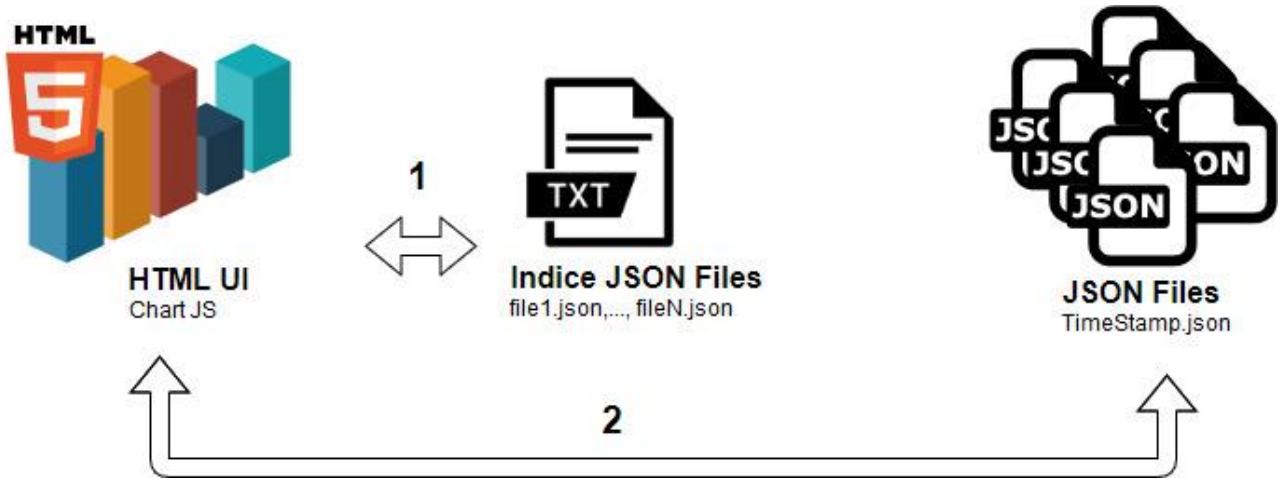
Un primo strumento di *debug* è appunto l'ST-LINKG *debugger* integrato nella *board*, il quale consente di utilizzare i *break point* nel codice sorgente ed avere una visione completa della memoria del microcontrollore in tempo reale.
Questo è probabilmente lo strumento di *debug* più comodo ed affidabile in assoluto perché consente di bloccare il processo in qualunque momento e verificare lo stato delle variabili del programma.
In supporto molto utile usufruire delle funzionalità di un oscilloscopio attraverso il quale abbiamo misurato con precisione la discrepanza tra il nostro *timer* locale ed il reale scorrere del tempo.

Unità Slave

Per il debug riguardante il modulo NodeMCU è stata utilizzata la seriale come metodo di analisi della memoria e del comportamento del firmware.

9. Interfaccia utente e tracciamento dei dati.

L'interfaccia è stata sviluppata in formato html in modo da poterla inserire all'interno dell'SD Card dello *shield* IDW. Questo infatti svolge un ulteriore funzione, ovvero quella di *web server.*
Nonostante il supporto integrato di micro-python, si è scelto di non utilizzare nessuno *script* lato server per non appesantire il microprocessore dello *shield*.
Al fine di consentire di eseguire *query* sui dati si è scelto di implementare un mini database basato su file JSON.
La struttura dell'applicativo è la seguente.



Inizialmente all'interno della UI viene richiesto di selezionare un intervallo di tempo che ha come limite superiore il giorno e l'ora corrente. Successivamente viene analizzato il file degli indici ed effettuata una prima query di selezione sul nome dei file.
Dato un intervallo chiuso [A,B] vengono tenuti in considerazione tutti i file il cui nome, ovvero timestamp, è incluso nell'intervallo più il primo file disponibile antecedente all'intervallo prescritto.
Tutti i file JSON hanno come dimensione 1 KB, il che vuol dire circa 40 coppie timestamp-numero di persone: questa scelta è stata fatta per alleggerire le query e filtrare quindi molti dati semplicemente dal nome del file stesso.
La seconda selezione dei dati consiste nell'apertura di tutti i file prima selezionati ed a seguire l'eliminazione di tutti i dati al di fuori dell'intervallo.
L'intervallo viene poi diviso in N (variabile a seconda delle preferenze) sotto-intervalli e vengono calcolate le medie pesate per ognuno di essi. Esattamente a metà di ogni sotto-intervallo di tempo viene disegnato il dato medio relativo alle persone presenti all'interno della stanza.
Per il plot dei dati è stata utilizzata la libreria Chart JS ed è scegliendo la visualizzazione ad istogramma verticale. Le letture dei file vengono effettuate utilizzando JQuery.

10. Test effettuati

I test iniziali sono stati effettuati utilizzando un modulo Fishino Piranha e due Infrared Obstacle Avoidance Sensors che comprendono un trasmettitore e ricevitore infrarossi, il trasmettitore invia il segnale infrarossi e quando esso viene rifratto da un oggetto e colpisce il ricevitore viene segnalato l'evento.
L'idea era di utilizzare questi sensori ai lati della porta in modo che quando una persona fosse passata davanti avrebbe fatto scattare l'evento dei sensori, questa soluzione però non è risultata possibile in quanto la lunghezza del raggio dei sensori arriva al massimo a 30 cm, troppo corto in certi casi, e soprattutto i due segnali dei sensori provocavano interferenza a vicenda.
Scartata questa opzione abbiamo preso in considerazione l'uso dei sensori a ultrasuoni, questi sensori riescono a rilevare la presenza di un oggetto fino a 4 metri di distanza, più che abbastanza per il nostro sistema.
Rimane comunque un problema di interferenza reciproca tra i due sensori il quale è stato risolto procedendo prima con la lettura di un sensore e poi, dopo aver aspettato il tempo necessario al suono per tornare indietro, con la lettura dell'altro.
Successivamente sono stati effettuati test riguardanti la connessione con il master, in particolare con l'emissione e la lettura del segnale di multicast DNS, questo ci ha permesso di scoprire che il modulo che stavamo utilizzando non forniva il supporto a questa tecnologia e di conseguenza siamo passati all'utilizzo di un modulo NodeMCU.

11. Consumi energetici

Unità Master

STM Nucleo F401RE + Shield IDW04A1 = 110 mA (5V) ≈ 550 mW

PIR Sensor = 9.3 mA (5V) ≈ 46.5 mW

Totale = 596.5 mW

Unità Slave

Esp8266 = 90 mA (5V) ≈ 450 mW

2 * HC-SR04 (ultrasonic) = 2 *15 mA (5V) ≈ 2 * 75 mW = 150 mW

Reed Switch = pochi uA ≈ 0 mW

Totale = 600 mW

Come si può notare, i valori calcolati di consumo energetico risultano perfettamente in linea con la politica a basso consumo adottata. Infatti alimentando tutto con una batteria del cellulare di qualche anno fa (2500 mAh), si dovrebbe avere una autonomia teorica di 10 ore.

12. Problemi riscontrati

Uno dei principali problemi riscontrati lato ST, è stato il fatto che tutti i componenti utilizzati ad eccezione del microcontrollore fossero da aggiornare (ST-Link e IDW). Infatti dato il loro malfunzionamento non vi era la possibilità di far funzionare la board come avrebbe dovuto.

ST-Link cancellava la Flash ogni volta che veniva alimentato per la prima volta mentre l'IDW non si riusciva a resettare correttamente per avviare una nuova connessione WiFi.

Uno secondo problema, invece, è stato quello dell'interferenza tra i segnali, sia dei sensori a infrarossi, inizialmente utilizzati per poi essere scartati dopo svariati test, che quelli a ultrasuoni. Nel primo caso non era risolvibile in quanto i sensori a infrarossi emanavano di continuo il segnale e quindi non è stato possibile alternarli come per i sensori a ultrasuoni.

Infine un terzo problema riguardava il protocollo multicast DNS, in particolare con il modulo piranha non siamo riusciti a far funzionare questa tecnologia ed abbiamo dovuto virare su un altro dispositivo con funzionalità di rete adeguate.

13. Risultati e Conclusioni

Il sistema registra ingressi e uscite da una stanza con una buona precisione, ci sono casi particolari in cui fa più fatica come quello in cui una persona esce contemporaneamente a una che entra, in tal caso verrà riconosciuta solo una delle due azioni. Inoltre il sistema funziona anche se disconnesso dalla rete e riproverà a riconnettersi solo quando la porta sarà chiusa in modo da perdere il minor numero possibile di accessi.