Università degli Studi di Milano
Facoltà di Scienze e Tecnologie
Corso di Laurea in Informatica

Sistema Accessi IoT

Relatore: Andrea TRENTINI

Correlatore: Marco LANZA

TESI DI LAUREA DI:
Andrei CIULPAN
Mat. 872394

Anno Accademico 2018/2019

Ringrazio i miei genitori e la nonna per il sostegno e per la pazienza che hanno avuto.

Ringrazio i miei amici e compagni di università per aver reso l'esperienza più bella e soprattutto più facile.

Ringrazio i miei tutor e colleghi per avermi dato la possibilità di crescere e concludere la mia esperienza universitaria.

Indice

Capitolo 1

Introduzione

Il controllo accessi è un sistema di protezione che permette l'accesso solo a determinate persone per via di qualche procedura di autenticazione: nel mondo fisico si può parlare di una semplice serratura (storicamente il sistema più utilizzato in assoluto) che può essere aperta solo dalle persone in possesso della chiave, mentre nel mondo dell'informatica si può notare l'enorme utilizzo delle procedure di autenticazione (login) che aiutano il sistema, tramite l'inserimento di un nome utente e password, a determinare automaticamente se una persona è autorizzata ad accedere alle risorse del sistema stesso.

Il controllo accessi[1] è un tema molto sviluppato nel campo della sicurezza sia fisica che informatica: è stato segnalato che nel 2017, per il secondo anno consecutivo, il mercato del controllo accessi ha avuto la crescita più rapida nell'industria della sicurezza fisica[2]. È anche un sistema onnipresente, utilizzato in ospedali, fabbriche, supermercati, aziende, sistemi di trasporto pubblico (e.g l'ATM di Milano), case e tanti altri campi.

La tesi si propone di trattare un sistema ibrido in cui il mondo fisico e il mondo informatico lavorano insieme: attraverso sensori ed attuatori è possibile avere la comunicazione tra i due mondi.

Si tratta di un sistema *Internet of Things* (IoT), ovvero un sistema in cui la connessione Internet viene estesa anche al mondo degli oggetti fisici (*smart objects*[3]) di uso comune. Gli oggetti si rendono riconoscibili e " acquisiscono intelligenza " grazie al fatto di possedere una o più delle seguenti funzionalità: identificazione, localizzazione, diagnosi di stato, interazione con l'ambiente circostante, elaborazione dati e ovviamente connessione. Gli oggetti intelligenti di un sistema IoT sono normalmente dotati di un processore embedded, sensori e attuatori e sono in grado di agire sui dati raccolti dall'ambiente e, ancora più importante, mandare questi dati in rete dove possono poi essere analizzati[4]. Un semplice esempio di un sistema IoT si può trovare nella Figura 1.

Figura 1: Esempio di sistema IoT

Le seguenti sezioni di questo capitolo introdurranno il progetto stesso (denominato *SimSim*) che poi verrà dettagliato nei prossimi capitoli.

1.1 Nascita ed evoluzione del progetto all'interno dell'azienda

SimSim è nato all'interno di *Yatta*, il primo makerspace in centro a Milano. Yatta è uno spazio attrezzato con tecnologie e macchinari digitali destinato alla prototipazione e gestito da professionisti del settore 3D, programmazione, elettronica e grafica, in una visione ampliata del concetto di argitiano orientato verso la Manifattura e Industria 4.0. Gli studenti tirocinanti a Yatta sono inseriti nel laboratorio e partecipano allo sviluppo di progetti di physical computing che prevedono automazioni gestite tramite microcontrollori. Inizialmente SimSim aveva un unico scopo: creare un sistema di accessi con la possibilità di salvare i log (su un cloud remoto o su una SD card) per poi leggerli. Quando ho iniziato a lavorare come tirocinante, esisteva già una versione iniziale del progetto (senza database e sito web) incompleta e, secondo me, abbastanza incasinata, perciò ho deciso di iniziare il tutto da capo.

Durante la fase di sviluppo, l'idea del progetto è evoluta fino a diventare un sistema più completo, munito di un database per l'azienda e un sito web per gestirlo. Ho lavorato principalmente in autonomia ma sono stato aiutato anche dal tutor aziendale con alcune scelte progettuali come ad esempio la scelta di alcune componentistiche hardware, creazione degli schemi del database, scelta del template dell'interfaccia web, eccetera. Inoltre mi sono stati forniti tutti i materiali necessari alla creazione del prototipo, compresa la stampante 3D per lo sviluppo della scatola e la porta che si possono vedere nella Figura 3 (un ringraziamento alla mia collega Cecilia per la creazione del modello 3D).

La sequenza temporale delle attività svolte si può vedere nella Figura 2.

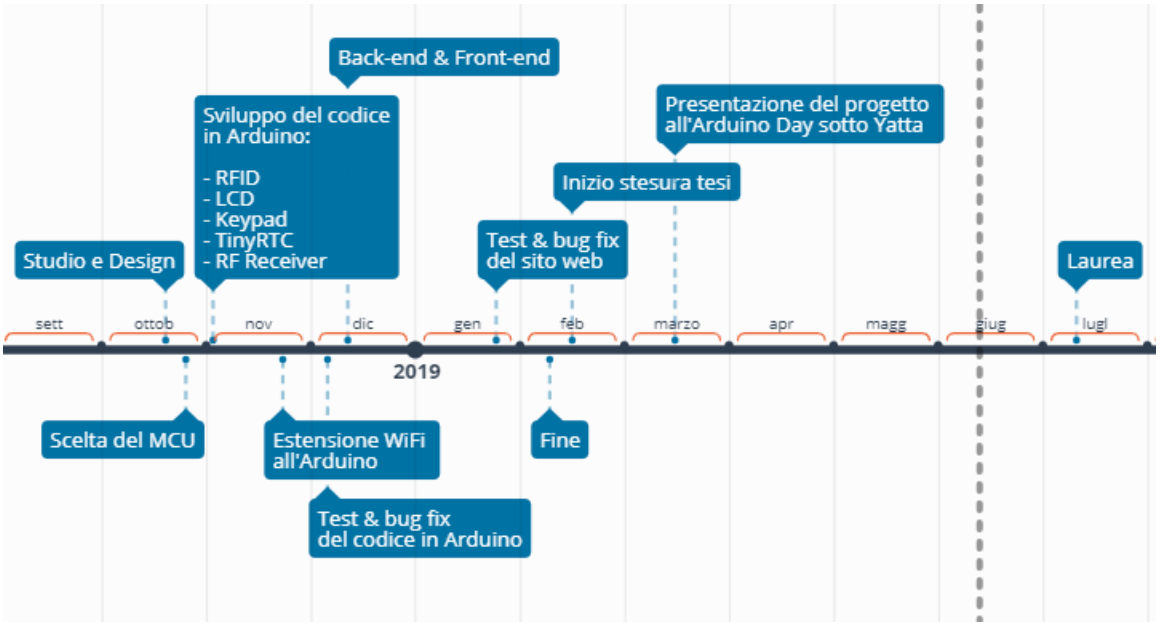


Figura 2: Sequenza temporale delle attività svolte. Creato con il tool https://time.graphics/it/editor

1.2 Requisiti del progetto

L'attività principale del tirocinio è la realizzazione di un sistema di controllo accessi, da applicare presso varchi. Vengono elencati i requisiti del sistema:

1. Deve essere in grado di funzionare con diverse modalità di riconoscimento:
   (a) Tessera RFID
   (b) Tastierino numerico (password)
   (c) Telecomando
2. Deve dare la possibilità di visualizzare su un display LCD ciò che sta succedendo (accesso consentito/negato ecc...). Inoltre deve essere implementata una funzione che permetta di visualizzare i codici delle tessere RFID sul display LCD (operazione protetta da una password).
3. A seguito di ogni accesso avvenuto con successo deve aprire una porta (usando un servomotore) e salvare il log in un database remoto (in LAN) tramite WiFi.
4. Deve avere un database per:

   (a) Tessere RFID
   (b) Log degli accessi
   (c) Soci

5. Deve avere un'interfaccia grafica (in questo caso un sito web) tramite la quale l'utente amministratore possa gestire il database con le seguenti funzioni:
   (a) Aggiungere/modificare/cancellare soci
   (b) Aggiungere/modificare/cancellare tessere
   (c) Visualizzare log

I dispositivi scelti per soddisfare i requisiti sono stati un Arduino UNO (per la parte hardware) e un Raspberry Pi 3B+ su cui ospitare il server.

Nella Figura 3 si può vedere una foto con il prototipo finito. Nelle Figure 4 e 5 si possono trovare rispettivamente un diagramma dei casi d'uso e un diagramma di flusso del sistema.

Figura 3: Prototipo di SimSim

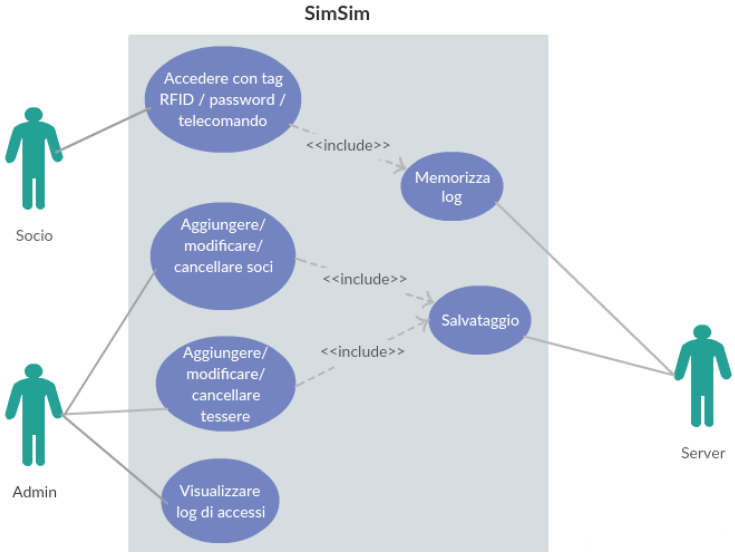


Figura 4: Casi d'uso del sistema

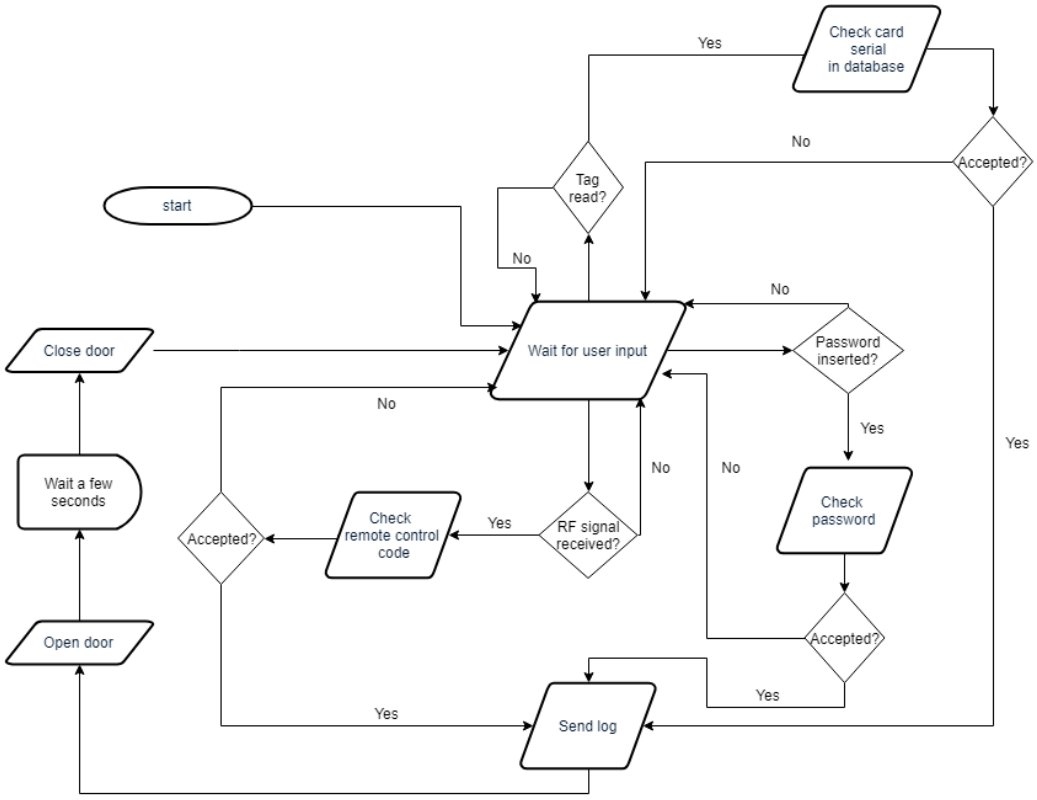


Figura 5: Diagramma di flusso del sistema

Capitolo 2

Sviluppo del sistema embedded

In questo capitolo vengono dettagliati i dispositivi hardware utilizzati all'interno del sistema embedded. Uno schema completo del circuito elettrico si trova nella Figura 6.

2.1 Raspberry Pi

Raspberry Pi (Figura 7) è una serie di piccoli computer (system-on-chip) dalle dimensioni simili a quelle di una carta di credito. La scheda è stata sviluppata nel Regno Unito (Fondazione Raspberry Pi). Le prime versioni avevano poca potenza computazionale e sono state progettate con l'intenzione di offrire a insegnanti e studenti uno strumento semplice e poco costoso con cui poter insegnare e imparare a programmare. La scheda ha avuto un enorme successo anche in altri campi (e.g. robotica) e con ogni nuova versione è stata migliorata. Le ultime versioni sono assai potenti e possono essere utilizzate per tante altre cose. Le schede sono basate su sistemi operativi di tipo GNU/Linux, in particolare Raspbian che può essere scaricato dal sito della fondazione (https://www.raspberrypi.org/downloads/).

La scheda utilizzata nel progetto (Raspberry Pi 3B+, https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/) possiede un'architettura su 64 bit, essenziale per poter utilizzare in maniera efficiente MongoDB (che è limitato a una memoria di 2 GB su un sistema a 32bit). Il Raspberry Pi 3B+ possiede inoltre una scheda WiFi, il che la rende un buon candidato per lo scopo principale del progetto, ovvero quello di ospitare il server.

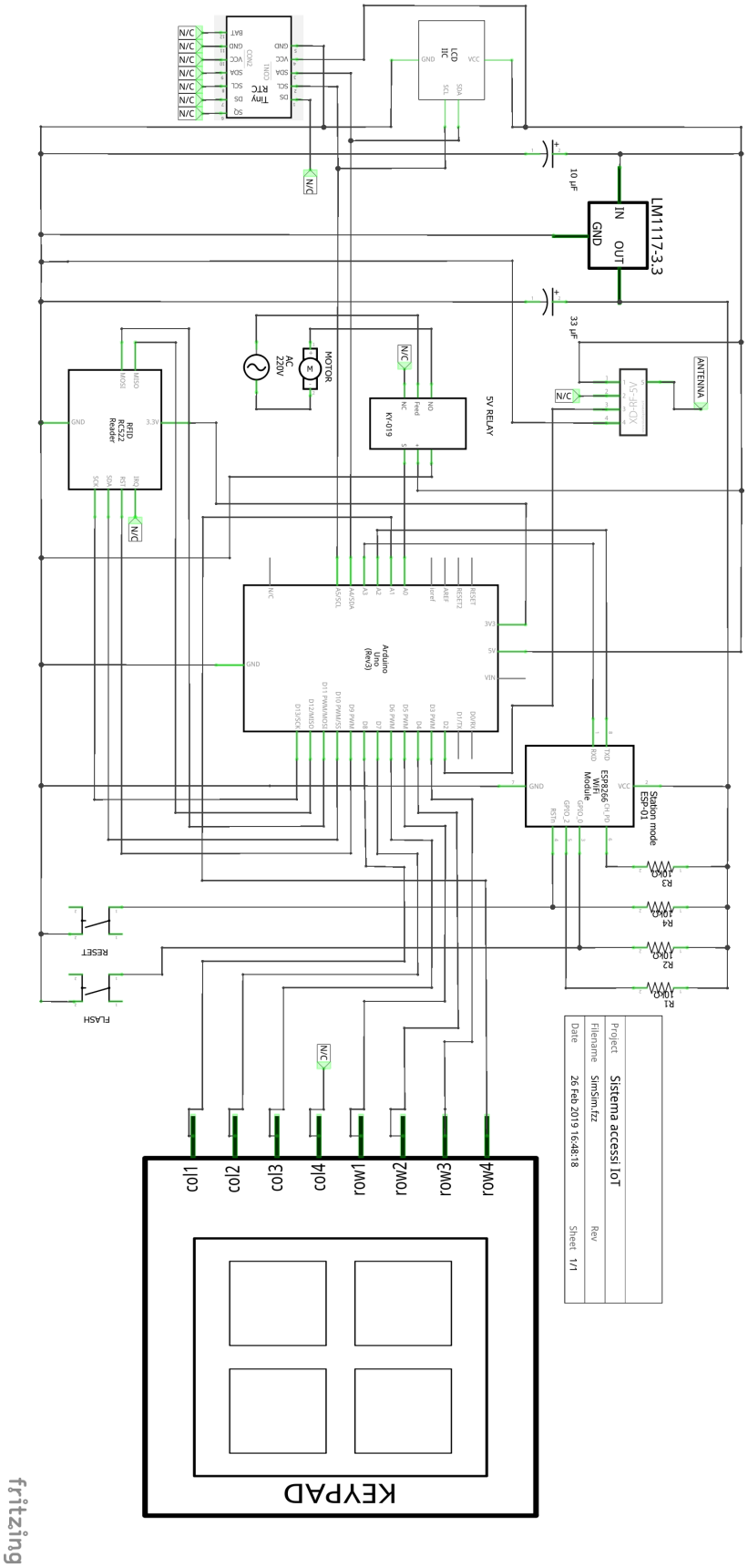


Figura 6: Schema del circuito

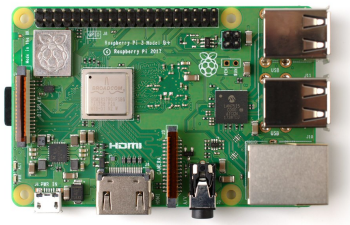


Figura 7: Scheda Raspberry Pi

2.2 Arduino UNO

Arduino[5] è nata ad Ivrea nel 2005 come piattaforma di prototipazione elettronica di basso costo e open-source (dettaglio molto importante che ha portato al successo che ha avuto) che si basa su hardware e software flessibili e facili da usare. Il nome della scheda deriva di quello del bar di Ivrea frequentato dai fondatori del progetto ed la scheda è stata creata per artisti, designer e hobbisti. Essendo open-source, ha avuto un grande successo nel mondo degli hobbisti, anche a livello mondiale (tale da diventare una delle invenzioni più considerevoli nella storia dell'elettronica), grazie alla facilità di programmazione, basso costo e soprattutto disponibilità in rete delle informazioni necessarie per poter permettere a persone con grandi idee e poche capacità informatiche ed elettroniche di realizzare i propri progetti.

Al giorno d'oggi Arduino potrebbe anche non essere la scelta immediata, anzi, ormai ci sono schede più potenti, più veloci e con migliori rapporti prestazioni/prezzo, ma non scordiamoci che è tutto grazie agli sviluppatori della scheda. Perchè scegliere, quindi, un Arduino? Essendo la scheda che ha fatto partire tutto, è anche la scheda più studiata e ciò significa che gira intorno a più informazioni, più librerie già pronte, più dispositivi compatibili e ha anche un sito di appassionati che si aiutano a vicenda dove si possono trovare tutorial, libri e tant'altro.

La scheda utilizzata nel progetto, Arduino UNO (Figura 8), è assolutamente la scheda più conosciuta al mondo e si può ormai trovare a dei prezzi molto bassi (ci sono tanti clone diversi in rete grazie al fatto di essere OpenHardware). Tra le varie caratteristiche vengono elencate le più interessanti:

- Microcontrollore: ATmega328 (https://www.sparkfun.com/datasheets/Components/SMD/ATMega328.pdf)
- Digital I/O Pins: 14 tra cui 6 offrono la funzione PWM (si veda la sezione 2.5 " Servomotore ") e 4 implementano il protocollo SPI (si veda la sezione 2.10.2 " SPI ")

- Analog I/O Pins: 6 tra cui 2 implementano il protocollo I^2C (si veda la sezione 2.10.1 " I^2C ")
- Flash Memory: 32 KB
- EEPROM: 1 KB
- Clock Speed: 16 MHz
- DC Current per i pin I/O: max 40 mA
- DC Current per il pin 3V3: max 50 mA
- Operating Voltage: 5V
- Input Voltage (recommended): 7-12V

L'Arduino UNO è quindi un dispositivo ideale per il progetto; l'unica cosa che manca è la possibilità di accedere al WiFi, indispensabile per l'IoT, ma questo problema è stato risolto grazie ad un chip chiamato ESP8266 (si veda la sezione 2.9 " ESP8266 ").

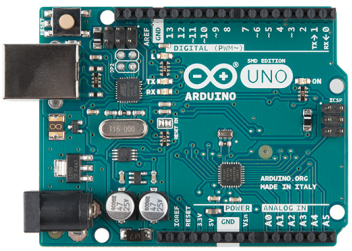


Figura 8: Scheda Arduino UNO

2.2.1 Arduino Programming Language

La scheda Arduino UNO (e qualsiasi altra board compatibile con Arduino) può essere facilmente programmata utilizzando, ma non solo, l'*Arduino IDE* (Arduino Integrated Development Software), un software sviluppato da Arduino.cc e funzionante sulle 3 piattaforme dominanti (Windows, Linux e MacOS). Il linguaggio di programmazione si chiama " Arduino Programming Language "[6] ed è un derivato di Wiring, una

piattaforma di sviluppo open-source composta da un linguaggio di programmazione, un IDE ed un circuito stampato basato su un microcontrollore. La sintassi è molto simile a quella del linguaggio C/C++; infatti il linguaggio Arduino è un insieme di funzioni e librerie C/C++ che possono essere chiamate dal codice. Forse la cosa diversa e che si nota di più rispetto ad altri linguaggi di programmazione è la diversa strutturazione dello sketch (sinonimo di programma): per poter funzionare vengono sempre definite due funzioni iniziali, setup() e loop(); la prima viene eseguita soltanto una volta all'avvio del programma e serve per dichiarare variabili, modalità dei pin (e.g input o output) ecc., mentre nella seconda viene messo il programma vero e proprio che viene eseguito ripetutamente... all'infinito.

2.3 Real-time Clock

I requisiti del progetto richiedono il salvataggio del log dopo ogni accesso avvenuto con successo, e per fare ciò è necessario essere in grado di tenere traccia del tempo. Il Raspberry Pi, diversamente dai PC a cui siamo abituati, può tenere traccia del tempo solo se ha accesso alla connessione Internet per accedere ad un server NTP (Network Time Protocol); nel progetto in questione, il Raspberry Pi, nonostante sia connesso alla rete LAN, è stato privato della connessione Internet per questioni di sicurezza (nessun utente esterno alla rete può accedere al server in locale perciò elimina gran parte degli attacchi possibili). Ciò significa che per poter avere il cosiddetto timestamp nei log bisogna trovare un modo di tenere traccia della data e dell'ora. Fortunatamente esistono dei piccoli chip chiamati RTC che fanno questo lavoro.

Nella Figura 9 si può vedere il dispositivo utilizzato nel progetto, chiamato TinyRTC, un modulo basato sul clock chip DS1307 (https://datasheets.maximintegrated.com/en/ds/DS1307.pdf) che supporta il protocollo I^2C (si veda la sezione 2.10.1 " I^2C ") per cui è facilmente utilizzabile dall'Arduino UNO. L'orologio può essere programmato con la data e l'ora corrente direttamente con il programma dell'Arduino e, grazie alla batteria al litio (CR1225), continua a tenere traccia del tempo anche senza alimentazione esterna. Il dispositivo è in grado di fornire, con abbastanza precisione, data e ora in termini di secondi, minuti, ora, giorno, mese e anno. Grazie alla popolarità di Arduino ci sono tante librerie per che permettono il facile utilizzo del dispositivo; nel progetto è stata usata una libreria chiamata RTClib che si può trovare nel seguente repository: https://github.com/adafruit/RTClib. Nella Figura 10 si può vedere il collegamento con l'UNO.

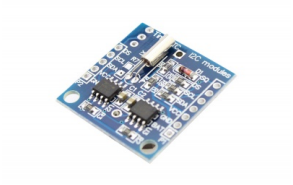


Figura 9: Tiny RTC

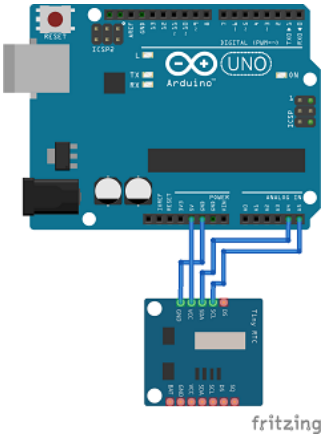


Figura 10: Tiny RTC e Arduino UNO

2.4 Trasmettitore-ricevitore RF

Ci sono due principali tipi di tecnologie per i telecomandi: a raggi infrarossi o a radio-frequenza (RF). La tecnologia a raggi infrarossi è quella più utilizzata ed è usata da tanti dispositivi elettronici (un esempio molto comune è il telecomando della TV). In questo caso trasmettitore e ricevitore devono essere messi faccia-a-faccia e non troppo distanti tra di loro, perciò non è il caso di usare questa tecnologia nel progetto (non si potrebbe, ad esempio, aprire la porta stando dietro ad un muro).

La tecnologia scelta è quindi quella a radio-frequenza, in cui trasmettitore e ricevitore possono comunicare anche se sono messi in camere diverse. Il dispositivo scelto (MX-05V/XD-RF-5V, Figura 11) è un modulo composto da trasmettitore e ricevitore in radio-frequenza a 433 MHz (https://sites.google.com/site/summerfuelrobots/arduino-sensor-tutorials/rf-wireless-transmitter-receiver-module-433mhz-for-arduino). Il trasmettitore è stato utilizzato soltanto per programmare i telecomandi dell'azienda con i codici voluti grazie alla loro funzione di clonazione (codice nel Listato 2.1).

Figura 11: Modulo transceiver MX-05V/XD-RF-5V

```
#include <RCSwitch.h>

int i = 0;
const char code[24] = "011100011011000110011010";
RCSwitch mySwitch = RCSwitch();

void setup() {
  Serial.begin(9600);
  mySwitch.enableTransmit(A0);
  mySwitch.setPulseLength(600);
  mySwitch.setProtocol(1);
}

void loop() {
  i = 0;
    while (i < 4) {
    mySwitch.send(code);
    i++;
    delay(400);
  }
  delay(3000);
}
```

Listato 2.1: L'Arduino utilizza il trasmettitore per trasmettere i codici clonati dai telecomandi.

Il ricevitore, invece, è sempre collegato all'Arduino (Figura 12) e riesce a ricevere i segnali mandati dai telecomandi anche da qualche decine di metri di distanza. Se il codice che viene mandato dal telecomando è corretto allora l'Arduino lo risconosce e apre la porta, altrimenti fa finta di niente.

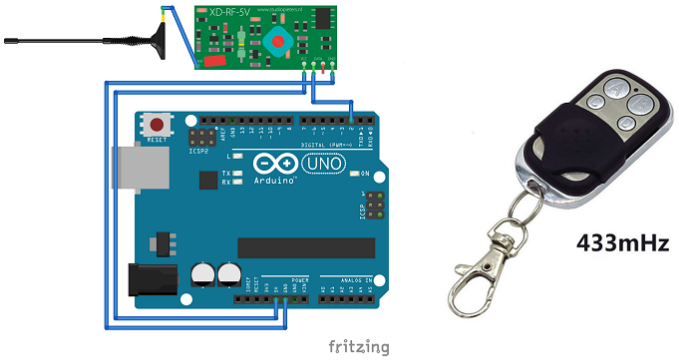


Figura 12: Ricevitore RF con telecomando e Arduino UNO

Il ricevitore, per aumentare il range, è stato accoppiato con un'antenna lunga 17,3 cm. Il motivo di questa scelta è che la lunghezza dell'antenna deve essere 1/4 di quella della lunghezza d'onda. Bisogna quindi calcolare la lunghezza d'onda λ:

$$\lambda(m) = \frac{v(m/s)}{f(Hz)} = \frac{299,792,458}{433,000,000} = 0,692m$$

dove v è la velocità della luce e f è la frequenza (433 MHz). Ora si può trovare la lunghezza dell'antenna:

$$\frac{\lambda}{4} = \frac{0,692}{4} = 0,173m = 17,3cm$$

2.5 Servomotore

Figura 13: Servomotore

Esistono tanti tipi diversi di motori Arduino-compatibili, ognuno con i propri punti forti e deboli, perciò bisogna valutare bene quale scegliere in base alle necessità del progetto. In questo caso il lavoro del motore è semplice: deve girare la serratura di una porta da una parte e dall'altra, quindi senza particolare precisione, velocità o potenza. Siccome non c'è nemmeno il bisogno di avere una rotazione completa (bastano 180°), è immediato pensare al servomotore che è anche facilmente controllabile da un Arduino grazie ai suoi pin PWM e alla libreria Servo (https://www.arduino.cc/en/reference/servo).

Il servomotore (Figura 13) è un dispositivo elettromeccanico a basso consumo energetico ed è costituito da un motore DC, un gruppo di ingranaggi (per ridurre le rotazioni al minuto e aumentare il momento torcente), un circuito di controllo (un ponte-H integrato), un potenziometro interno e un contenitore di plastica che lo racchiude. La particolarità del servomotore è che può ruotare con una precisione di alcuni gradi (normalmente da 0° a 180°) in base alla larghezza dell'impulso che riceve. Questa è una tecnica che si chiama PWM.

Il motore è collegato al potenziometro attraverso gli ingranaggi: quando il motore ruota, la resistenza del potenziometro cambia e quindi il circuito di controllo può misurare esattamente in quale direzione punta l'albero del motore. Quando l'albero del motore è arrivato alla posizione desiderata, esso si ferma perchè il circuito di controllo

toglie l'alimentazione. Questo succede perchè vengono comparati due segnali: quello in ingresso e quello del potenziometro (che dipende dalla posizione dell'albero del motore). La differenza tra i due segnali produce un segnale chiamato *segnale d'errore* che viene usato come input (dopo essere stato amplificato) al motore per farlo girare. Questo errore può essere sia positivo che negativo facendolo quindi girare sia da una direzione che dall'altra. Siccome il motore continua a girare, il segnale del potenziometro cambia, fino a essere uguale a quello in ingresso: quando si arriva a questo punto il segnale d'errore è pari a zero e ciò significa che non c'è più alimentazione e il motore si ferma. Come si può notare, il servomotore funziona sulla base della retroazione.

2.5.1 PWM

Pulse Width Modulation (PWM) è una tecnica che consiste in una serie di impulsi di larghezza variabile che determinano la posizione del servo. È quindi un modo di ricavare un risultato analogico a partire da un segnale digitale.

Un segnale PWM ha una forma d'onda quadra, con periodo fisso, ma spesso non simmetrica: la durata della semionda alta può variare (si veda la Figura 14). La durata *T[ON]* del segnale alto viene misurata in percentuale e si chiama duty cycle: ad esempio un duty cycle del 25% significa che *T[ON]* dura per 1/4 della durata del periodo fisso *T*. Questo tipo di segnale non è un vero e proprio segnale analogico, però può essere interpretato tale: quello che succede è che la tensione viene applicata e levata nel giro di frazioni di secondo perciò quello che si chiama tensione efficace è in realtà minore della tensione reale applicata. Per calcolare la tensione efficace *Veff* basta sapere la tensione applicata *Vcc* e il duty cycle *DC* (che si trova facendo il rapporto tra *T[ON]* e *T*):

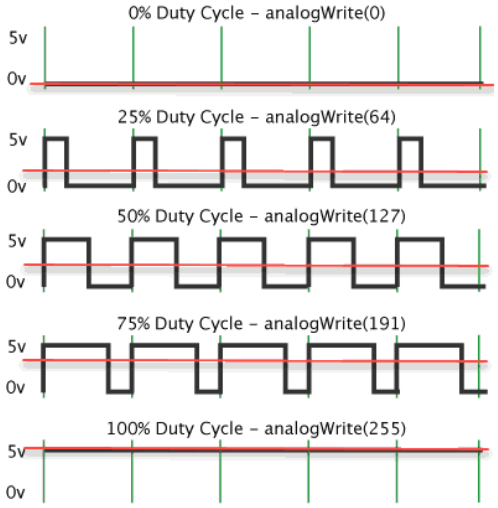


Figura 14: Segnali PWM e comando analogWrite(int value) per generarli con l'Arduino. La *Veff* è rappresentata dalla linea orizzontale che copre il segnale digitale

$$Veff = Vcc \cdot DC$$

Generalmente, nel caso dei servomotori, si ha un periodo totale $T = 20$ ms, perciò ogni 20 ms il servo riceve un nuovo impulso. La posizione del motore dipende quindi

dalla durata del tempo *T[ON]* che questo impulso rimane ad un livello logico HIGH. Nel grafico della Figura 15 si nota che la durata dell'impulso alto è un valore compreso tra 1ms (Duty Cycle = 5%) e 2ms (Duty Cycle = 10%). Fissata la posizione centrale del servo a 90°, allora abbiamo che:

- se la durata del livello logico HIGH dell'impulso sta nell'intervallo [1ms, 1.5ms] allora il servo ruota verso sinistra a partire dalla posizione centrale (la sua posizione varia tra 0° e 90°)
- se la durata del livello logico HIGH dell'impulso è uguale a 1.5ms allora il servo resta fermo a 90°
- se la durata del livello logico HIGH dell'impulso sta nell'intervallo [1.5ms, 2ms] allora il servo ruota verso destra a partire dalla posizione centrale (la sua posizione varia tra 90° e 180°)

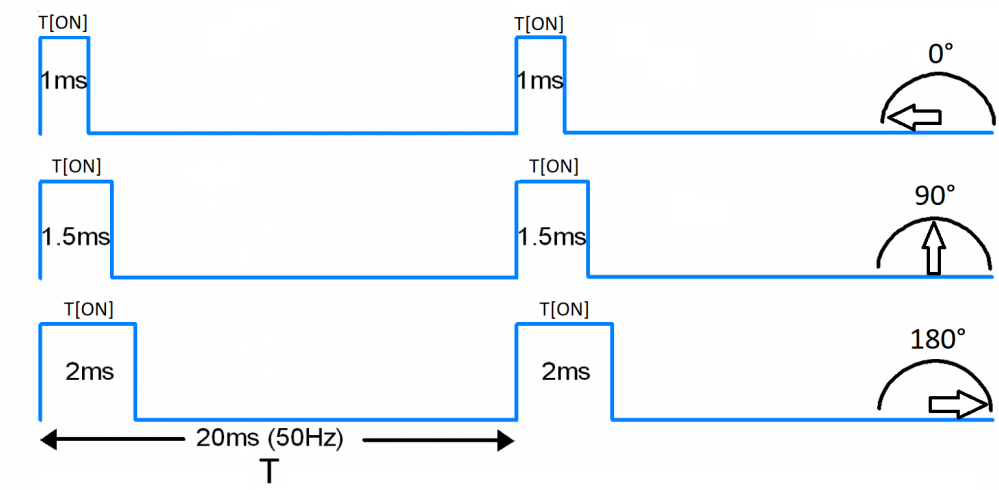


Figura 15: Segnali PWM nel servomotore

Controllare un servomotore con l'Arduino è in realtà semplice perché esiste una libreria (https://www.arduino.cc/en/reference/servo) che ne facilita il lavoro: basta pensare in gradi. Nella Figura 16 viene fatto vedere un esempio di collegamento tra il servo e l'Arduino e un esempio di codice per controllarlo si trova nel Listato 2.2.

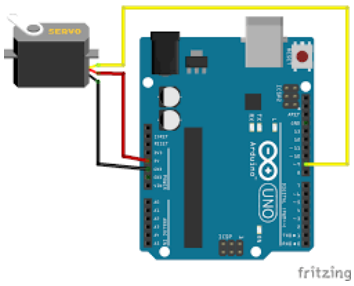


Figura 16: Servomotore e Arduino UNO

```
#include <Servo.h>

Servo myservo;
int pos = 0;

void setup() {
  myservo.attach(9);
}

void open() {
  for (pos = 0; pos <= 180; pos += 1) {
    myservo.write(pos);
    delay(15);
  }
  delay(5000);
  for (pos = 180; pos >= 0; pos -= 1) {
    myservo.write(pos);
    delay(15);
  }
}
```

Listato 2.2: Esempio di codice per controllare il servomotore: innanzittutto il servo viene collegato al pin 9 dell'Arduino e viene fatto ruotare di 180 gradi da una parte (aprendo la porta); dopo 5 secondi viene fatto ruotare di 180 gradi dalla parte opposta

2.6 RFID Reader

Radio Frequency Identification (RFID) è un termine utilizzato per descrivere tecnologie contactless che fanno uso delle onde radio per identificare oggetti, persone e animali sia da vicino (tag passivi) che da lontano (tag attivi). L'identificazione è resa possibile grazie ad un lettore RFID in grado di interrogare i cosidetti tag, ovvero dei

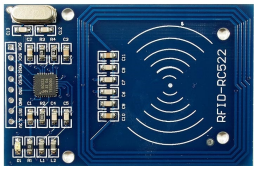


Figura 17: Lettore RFID MFRC522

dispositivi elettronici di varie forme e dimensioni che possiedono i dati necessari per l'identificazione e che principalmente si dividono in due categorie:

- I tag passivi sono poco costosi e contengono un'antenna e un piccolo chip con identificativo univoco e privo di alimentazione. Questi tag vengono utilizzati per l'identificazione a breve distanza (pochi centimetri) e vengono " attivati " dalle onde radio del lettore.

- I tag attivi sono più costosi e permettono, grazie all'alimentazione interna (emettono loro stessi i segnali radio), l'identificazione a lunga distanza (fino a 200 metri).

La tecnologia RFID si diversifica anche nella frequenza utilizzata e trova utilizzo in tanti campi diversi quali gestione accessi, tracciabilità degli animali, gestione dei biglietti elettronici, ecc. (per le basse e medie frequenze), ma anche in Telepass, identificazione di auto in movimento, tracciabilità degli oggetti in movimento ecc. (per le frequenze alte e altissime).

Il lettore RFID utilizzato nel progetto è il MFRC522 illustrato nella Figura 17 (https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf). Esso opera su una frequenza di 13.56 MHz ed è in grado di comunicare con i tag che si conformano allo standard ISO/IEC 14443 A/MIFARE. Implementa vari protocolli di comunicazione (SPI, Serial UART, I^2C) ma è stata sviluppata solo una libreria di Arduino (https://github.com/miguelbalboa/rfid) basata sul protocollo SPI (si veda la sezione 2.10.2 " SPI ").

Nella Figura 18 si trova il circuito per collegare il lettore RFID con l'UNO e un esempio di codice di Arduino per leggere i tag si trova nel Listato 2.3.

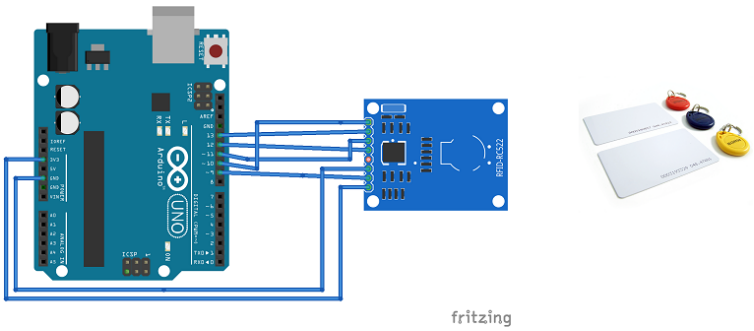


Figura 18: RFID con dei tag e Arduino UNO

```
boolean rfidCardRead(void) {
  if ( !rfid.PICC_IsNewCardPresent() ) return false;
  if ( !rfid.PICC_ReadCardSerial() ) return false;
  currentCardNumber = "";
  for ( uint8_t i=0; i<5; i++ ) {
    currentCardNumber.concat(String(rfid.uid.uidByte[i], HEX));
  }
  currentCardNumber.toUpperCase();
  rfid.PICC_HaltA();
  return true;
}
```

Listato 2.3: Procedura per leggere i tag. I tag vengono letti un byte alla volta e poi vengono concatenati in una stringa.

2.7 Display LCD 16x2

Figura 19: Display LCD 16x2

I moduli LCD trovano ampio utilizzo nei sistemi embedded grazie al basso costo, disponibilità e utilità. Il display utilizzato si chiama “ 16x2 LCD ” (Figura 19) proprio

perchè possiede uno schermo su cui si possono scrivere 2 linee di 16 caratteri ciascuna. Questi dispositivi sono abbastanza complessi (ci sono 1280 pixel da indirizzare) e per poterli utilizzare in maniera semplice ed efficiente è necessario avere un'interfaccia. Infatti i display LCD spesso vengono accoppiati con un controller che rende la vita facile al programmatore. Il controller più utilizzato (infatti è considerato uno standard *de facto*) si chiama HD44780 (https://www.sparkfun.com/datasheets/LCD/HD44780.pdf), costruito dalla Hitachi, ed è in grado di controllare display LCD capaci di visualizzare solo caratteri. L'HD44780 supporta il trasferimento parallelo a 8 bit (possiede 8 bit di dati) e il suo compito principale è di ricevere i comandi e i dati in arrivo dall'MCU per poi processarli e mandarli all'LCD.

Il problema principale dell'LCD 16x2, come si può vedere nella Tabella 1, è il numero di pin necessari da collegare (e la pazienza per farlo) all'Arduino per poter farlo funzionare correttamente. Questo significa che, siccome l'Arduino ha un numero limitato di pin (13 digitali e 6 analogici), essi potrebbero non bastare per un progetto di dimensioni più grandi. Fortunatamente c'è una soluzione: I^2C (si veda la sezione 2.10.1 " I^2C "). Esiste un chip che rende possibile la riduzione del numero di pin utilizzati a due, SDA e SCL, cambiando l'interfaccia parallela del controller in una seriale. Questo chip si chiama PCF8574 I/O Expander ed è definito come un backpack (è letteralmente uno zaino che si salda sopra l'LCD) e permette appunto all'LCD e all'Arduino di comunicare utilizzando il protocollo I^2C. In Arduino si può utilizzare una libreria già esistente chiamata LiquidCrystal_PCF8574 (https://platformio.org/lib/show/1165/LiquidCrystal_PCF8574/installation) che sfrutta la libreria Wire (https://www.arduino.cc/en/Reference/Wire) per implementare alcune funzioni utili al programmatore. Esempio di codice nel Listato 2.4:

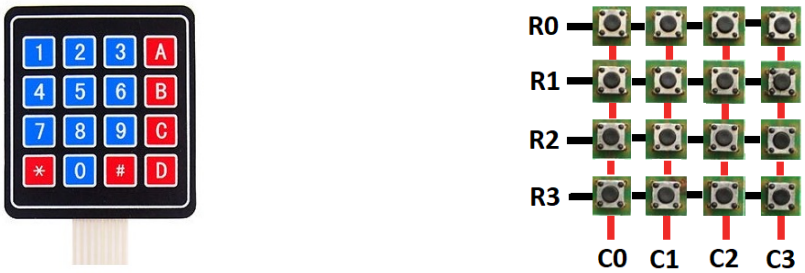
```
LiquidCrystal_PCF8574 lcd(lcdAddr); // lcdAddr = 0x27

void lcdSetup(void) {
  Wire.beginTransmission(lcdAddr);
  if ( Wire.endTransmission() == 0 ) Serial.println(F("LCD OK"));
  else Serial.println(F("LCD MISSING"));
  lcd.begin(16,2);
  lcd.noBlink();
  lcdHomeMode();
}

void lcdHomeMode(void) {
  lcd.setBacklight(0);
  lcd.clear();
}
```

Listato 2.4: L'Arduino inizializza la comunicazione con l'LCD tramite il protocollo I^2C e chiama la funzione lcdHomeMode() che semplicemente spegne la retroilluminazione e pulisce il testo.

Pin	Funzione del pin
1	Vss (GND)
2	Vcc (5V)
3	Vee (Contrasto)
4	RS (Register Select: 0 per selezionare l'invio di un comando, 1 per i dati)
5	R/W (Read/Write: 0 per la scrittura di dati/comandi, 1 per la lettura dati/stato)
6	E (Enable bit: inizia il ciclo di scrittura o lettura)
7	D0 (Data 0)
8	D1 (Data 1)
9	D2 (Data 2)
10	D3 (Data 3)
11	D4 (Data 4)
12	D5 (Data 5)
13	D6 (Data 6)
14	D7 (Data 7)
15	A (Anodo (+): retroilluminazione)
16	K (Catodo (-): retroilluminazione)

Tabella 1: Pinout di un display LCD 16x2 con il controller HD44780

2.8 Keypad



(a) Tastierino 4x4 **(b)** Rappresentazione a matrice dei pulsanti

Figura 20: Keypad

I tastierini sono dei dispositivi di input ampiamente utilizzati per dare la possibilità alle persone di interagire con un sistema (e.g. inserire una password, controllare un robot ecc.). Normalmente i tasti sono disposti in un formato a matrice (in questo caso 4x4, ovvero 16 pulsanti) e permettono ad un microcontrollore di capire facilmente

quale tasto è stato premuto in base alle linee che si attivano: nella Figura 20 si può facilmente vedere che premendo il pulsante “ 2 ” vengono attivate le linee “ R0 ” e “ C1 ” e il microcontrollore, grazie a come è stato configurato, è in grado di dire che quel pulsante specifico è stato premuto. Un esempio di configurazione in Arduino del tastierino si può trovare nel Listato 2.5.

```
const byte keypadRows = 4;    // quattro righe
const byte keypadCols = 3;   // tre colonne
const byte keypadRowPins[keypadRows] = {5, 4, 3, 15}; // 15 = A1
const byte keypadColPins[keypadCols] = {8, 7, 6};
const char keypadConfig[keypadRows][keypadCols] =
{
{'1','2','3'},
{'4','5','6'},
{'7','8','9'},
{'*','0','#'}
};
```

Listato 2.5: Configurazione dei pin del tastierino. In questo caso non viene utilizzata l’ultima colonna con le lettere.

Le funzioni per utilizzare il tastierino sono implementate nella libreria Keypad di Arduino (https://playground.arduino.cc/Code/Keypad/). Nella Figura 21 si può vedere un esempio di collegamento tra il tastierino e l’Arduino UNO.

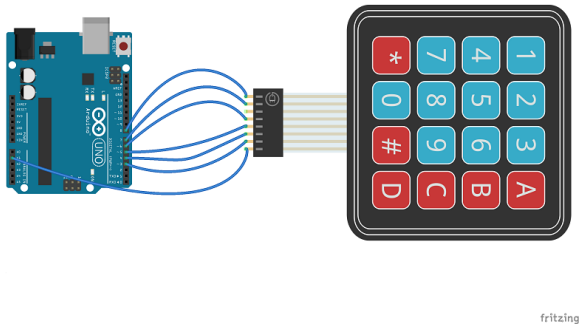


Figura 21: Keypad e Arduino UNO

2.9 ESP8266

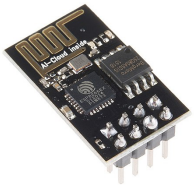


Figura 22: L'ESP-01 di Ai-Thinker

L'Arduino UNO, a differenza di altre board più recenti, non ha il WiFi incorporato perciò bisogna accoppiarlo con uno shield che gli permette di avere quella funzionalità necessaria per un progetto IoT. Per questo progetto è stata scelta l'ESP8266, una famiglia di microchip WiFi a basso costo prodotta da Espressif Systems. Il dispositivo usato in realtà si chiama ESP-01 (Figura 22), è basato sul modulo ESP8266 ed è prodotto da terze parti (Ai-Thinker). Questo piccolo modulo permette a microcontrollori di connettersi alla rete WiFi per fare semplici chiamate HTTP. Tra le varie caratteristiche vengono elencate le più interessanti:

- basso costo e molto compatto;
- antenna incorporata nel PCB;
- 512 KB flash memory su cui caricare il programma;
- 80 KB user data RAM;
- può essere usato sia come Station (il caso in questione) che come Access Point;
- implementa la funzione Deep Sleep per evitare il consumo eccessivo durante l'inattività;
- può essere programmato attraverso l'IDE di Arduino;
- implementa il protocollo IEEE 802.11 b/g/n (ovvero WiFi).

Il pinout e le funzioni dei pin si trovano rispettivamente nella Figura 23 e nella Tabella 2.

L'ESP-01 va alimentato a una tensione di 3.3V ma secondo il datasheet (http://www.microchip.ua/wireless/esp01.pdf) potrebbe richiedere anche fino a 170mA di corrente perciò non si può alimentare direttamente dal pin 3V3 dell'Arduino perchè non eroga abbastanza corrente (max 50 mA secondo https://store.arduino.cc/arduino-uno-rev3). Per

questo motivo è stato usato il pin 5V dell'Arduino collegato ad un regolatore di tensione chiamato LM1117 (datasheet: http://www.ti.com/lit/ds/symlink/lm1117.pdf). Il circuito si può trovare nella Figura 24.

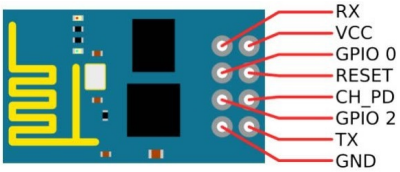


Figura 23: ESP-01 Pinout

Pin	Funzione del pin
1	RX (Receive data bit)
2	GPIO0 (General Purpose Input/Output Pin)
3	GPIO2 (General Purpose Input/Output Pin)
4	GND
5	VCC (+3.3V)
6	RST (Reset chip)
7	CH_PD (Chip power-down)
8	TX (Transmit data bit)

Tabella 2: Funzioni dei pin dell'ESP-01

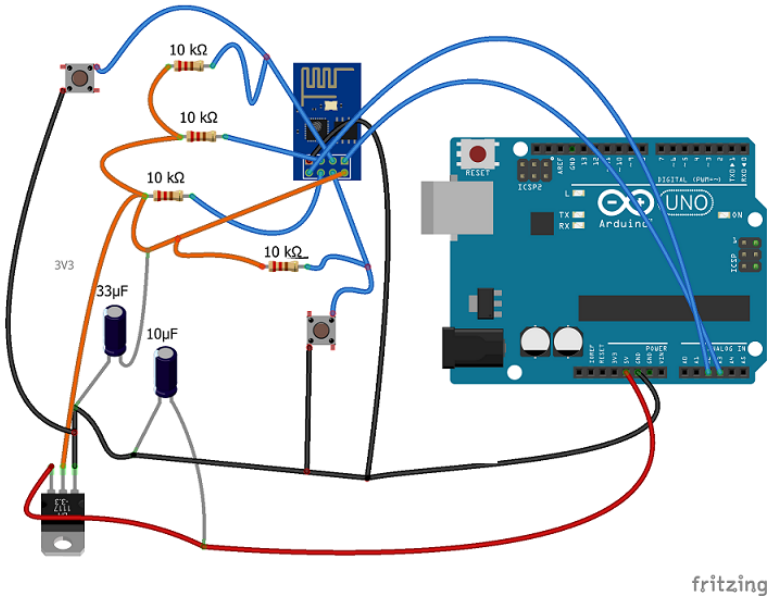


Figura 24: ESP-01 e Arduino UNO

Il dispositivo viene pre-programmato con il firmware AT (https://www.electrodragon.com/w/ESP8266_AT-Command_firmware) ma è possibile programmarlo anche con il codice proprio (vedere la Tabella 3 per le modalità di funzionamento) utilizzando l'IDE di Arduino (Listato 2.6).

GPIO-0	Modalità
GND	Flash mode (modalità programmazione)
VCC	Normal mode (il chip esegue il codice caricato)

Tabella 3: Modalità di funzionamento dell'ESP-01

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include "wifi_secrets.h"

const char* ssid = SECRET_SSID;
const char* password = SECRET_PASS;

void setup() {

  Serial.begin(9600);
  WiFi.begin(ssid, password);
  Serial.print("Connecting");
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
  Serial.println("connected");
}
```

Listato 2.6: Riprogrammazione dell'ESP-01 con il codice proprio. All'accensione, il dispositivo non fa altro che connettersi alla rete WiFi locale. Per poter fare ciò, è stata utilizzata una libreria molto bene documentata (https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/readme.html)

Dopo aver effettuato la connessione, l'ESP-01 si mette in ascolto sulla porta seriale per dati in arrivo dall'Arduino per poi eseguire varie funzioni in base alla richiesta (Listato 2.7).

```
void loop() {
  if ( WiFi.status() == WL_CONNECTED ) {
    if ( Serial.available() ) {
    String from_arduinoJSON = Serial.readString();
      if (from_arduinoJSON.startsWith("check")) {
        checkTessera(from_arduinoJSON.substring(5));
      }
      else if (from_arduinoJSON.startsWith("log")) {
        sendLog(from_arduinoJSON.substring(3));
      }
    }
  }
}
```

Listato 2.7: L'ESP-01 si mette in ascolto sulla porta seriale collegata all'Arduino. Esso chiama la funzione corretta in base al formato della stringa ricevuta.

L'Arduino è programmato per mandare 2 tipi di richieste:

- " check " per quando deve inoltrare all'ESP-01 il codice del tag per la verifica (Listato 2.8)
- " log " per quando deve inoltrare all'ESP-01 un log (Listato 2.9)

```
boolean checkCardSerial(String serial) {
boolean allowed = false;
String bodyJSON = "check{\"seriale\":\"" + serial + "\"}";
int statusCode = printBodyToESP(bodyJSON);
switch (statusCode) {
  case 200:
    sendLogTessera(serial);
    allowed = true;
    break;
  ... // altri casi
  }
return allowed;
}
```

Listato 2.8: L'Arduino manda all'ESP-01 il seriale della tessera in formato JSON e aspetta la risposta con lo status code. Se statusCode = 200 allora mette la variabile allowed a " true ". La funzione printBodyToESP si può vedere nel Listato 2.13

```
void sendLog(String tipo) {
  String date = getCurrentFormattedDate();
  String bodyJSON = "log{\"data\":\"";
  bodyJSON.concat(date);
  bodyJSON.concat("Z\",\"tipo\":\"" + tipo + "\"}\0");
  printBodyToESP(bodyJSON);
}
```

Listato 2.9: L'Arduino manda all'ESP-01 il log in formato JSON e aspetta la risposta con lo status code. Se statusCode = 200 allora scrive un messagio di successo, altrimenti stampa un messaggio d'errore. La funzione printBodyToESP si può vedere nel Listato 2.13.

In base alla richiesta vengono attivate 2 funzioni diverse:

- una per mandare la verifica del tag al server tramite una chiamata HTTP (Listato 2.10);
- una per mandare il log al server tramite una chiamata HTTP (Listato 2.11).

```
void checkTessera(String tesseraJSON) {
  HTTPClient http;
  http.begin("http://192.168.1.71:3000/api/tessere/check");
  http.addHeader("Content-Type", "text/plain");
  int statusCode = http.POST(tesseraJSON);
  Serial.print(String(statusCode)); // invia all'Arduino;
  http.end();
}
```

Listato 2.10: L'ESP-01 manda una richiesta POST all'endpoint specificato. Il corpo della richiesta è in formato JSON e include il seriale e il tipo della tessera che si vuole verificare.

```
void sendLog(String logJSON) {
  HTTPClient http;
  http.begin("http://192.168.1.71:3000/api/logs/add");
  http.addHeader("Content-Type", "text/plain");
  int statusCode = http.POST(logJSON);
  Serial.print(String(statusCode)); // send code to Arduino;
  http.end();
}
```

Listato 2.11: L'ESP-01 manda una richiesta POST all'endpoint specificato. Il corpo della richiesta è in formato JSON e rappresenta il log.

2.9.1 Comunicazione tra l'Arduino UNO e l'ESP-01

L'Arduino UNO è programmato per comunicare con l'ESP-01 tramite SoftwareSerial (https://www.arduino.cc/en/Reference/SoftwareSerial), una libreria che replica, per via software, le funzionalità della comunicazione seriale hardware (sui pin 0 e 1) anche su altri pin digitali. Un esempio di codice per il setup della comunicazione seriale tra l'ESP-01 e l'Arduino UNO tramite SoftwareSerial si trova nel Listato 2.12

```
SoftwareSerial ESP(16, 17);

void arduinoEspSerialSetup() {
  ESP.begin(9600);
}
```

Listato 2.12: Inizializzazione della comunicazione seriale tra Arduino UNO e ESP-01

Un esempio di utilizzo di SoftwareSerial si può trovare nel Listato (2.13).

```
int printBodyToESP(String bodyJSON) {
int statusCode;
ESP.print(bodyJSON);
delay(100);
uint32_t lastMillis = millis();
while (!ESP.available()) {
// se passano 5 secondi senza risposta, break (per evitare loop)
if (!checkTimeout(lastMillis)) break;
}
// adesso abbiamo lo status code
statusCode = uint32_t(ESP.parseInt());
return statusCode;
}

boolean checkTimeout(uint32_t lastMillis) {
// espTimeoutInterval = 5000
if ( (millis() - lastMillis) < espTimeoutInterval ) return true;
else return false;
}
```

Listato 2.13: La funzione printBodyToESP. Essa manda all'ESP-01 (tramite l'oggetto SoftwareSerial) la stringa ricevuta tra gli argomenti e poi aspetta per la risposta. Dopo aver ricevuto la risposta restituisce statusCode.

Come si è visto precedentemente, l'Arduino inoltra all'ESP-01 delle stringhe che sono quasi in formato JSON: prima dell'oggetto JSON c'è un'altra stringa usata per identificare la funzione voluta (e.g. " check{object} " e " log{object} "). L'ESP-01 poi filtra questi dettagli non più necessari e chiama la funzione richiesta. Viene presentata la sequenza completa degli eventi per la verifica del tag:

(I) L'utente avvicina il tag al lettore RFID;

(II) L'Arduino legge il seriale del tag e chiama " checkCardSerial(serial) " (Listato 2.8). Questa funzione manda una stringa in formato " quasi-JSON " all'ESP-01;

(III) L'ESP-01 riceve questa stringa. Esso filtra l'informazione non più necessaria e chiama " checkTessera(tessera) " (Listato 2.7);

(IV) La funzione eseguita manda l'oggetto JSON al server e aspetta una risposta con lo status code. Dopo aver ricevuto la risposta, esso inoltra lo status code all'Arduino tramite la porta seriale (Listato 2.10);

(V) L'Arduino prende una decisione in base allo status code: se 200 allora il tag è accettato e apre la porta, altrimenti viene rifiutato.

Come si può vedere dalla sequenza degli eventi, l'ESP-01 non è altro che un intermediario tra l'Arduino e il server.

2.10 Protocolli di comunicazione tra dispositivi

2.10.1 I^2C

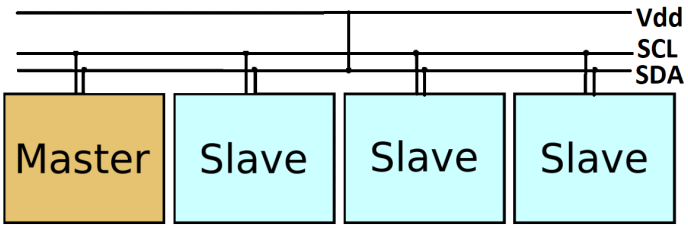


Figura 25: I^2C Bus

Inter-Integrated Circuit (I^2C) è un protocollo seriale a due vie inventato dalla NXP per far comunicare tra di loro dispositivi come microcontrollori, memorie EEPROM, dispositivi di input/output ecc. È un protocollo creato per la comunicazione sincrona e a breve distanza tra dispositivi chiamati dispositivi master e slave (Figura 25):

- Lo slave è un dispositivo che riceve i comandi dal master. Ogni slave ha un indirizzo (deve essere unico all'interno del sistema) normalmente specificato nel datasheet (e.g. per il display LCD 16x2 con il backpack PCF8574 utilizzato nel progetto l'indirizzo è 0x27) attraverso il quale viene indirizzato dal master.

- Il master non ha bisogno di un indirizzo ed è quello che genera il clock (per via di SCL). Il suo compito è di fare partire la comunicazione con i slave e inviare comandi di lettura/scrittura. Un sistema può avere più dispositivi master.

I^2C è un protocollo a due vie perchè utilizza due linee bidirezionali:

- *Serial Data* (SDA) attraverso il quale viaggiano i dati. I dati vengono trasferiti in *messaggi* che vengono divisi in blocchi di dati a 8 bit. Ogni messaggio parte con il " Start Condition " e finisce con " Stop Condition " e ha al suo interno, oltre ai blocchi di dati, l'indirizzo dello slave con cui il master desidera comunicare. Per gestire i comandi di lettura/scrittura è presente un Read/Write bit. Nel messaggio vengono inclusi anche i cosidetti ACK/NACK (acknowledge/no-acknowledge) bit che vengono mandati dal destinatario (lo slave in modalità " write ", il master in modalità " read ") dei blocchi di dati inviati. Un esempio di messaggio si trova nella Figura 26.
- *Serial Clock* (SCL) utilizzato dal master per la sincronizzazione.

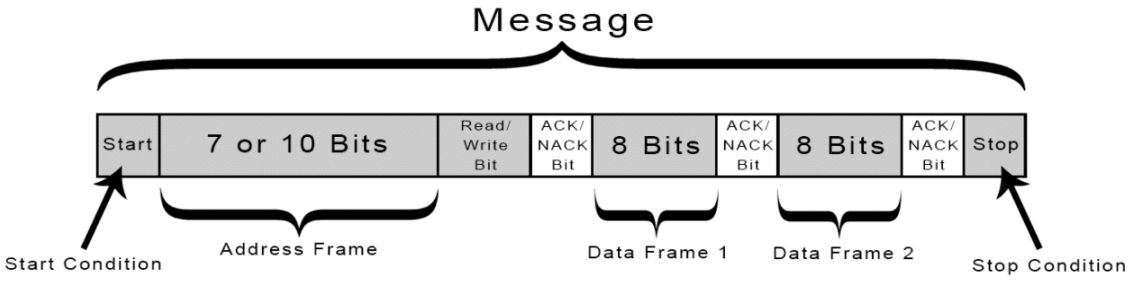


Figura 26: Esempio di messaggio inviato sulla linea SDA con il protocollo I^2C

2.10.2 SPI

Serial Peripheral Interface (SPI) è un protocollo di comunicazione sincrona inventato dalla Motorola e ampiamente utilizzato nei sistemi embedded. Similmente al protocollo I^2C, è stato creato per la comunicazione a breve distanza tra un dispositivo master (in questo caso può essere uno solo) e uno o più slave. SPI è basato su 4 segnali (4-wire bus):

- *Serial Clock* (SCLK) utilizzato dal master per la sincronizzazione.
- *Master Input Slave Output* (MISO) collegato all'input del master e all'output degli slave. Viene utilizzata dagli slave per inviare dati al master.

- *Master Output Slave Input* (MOSI) collegato all'output del master e all'input degli slave. Viene utilizzata dal master per inviare dati allo slave.
- *Chip Select* (CS) utilizzato per selezionare lo slave che deve essere abilitato alla comunicazione con il master. Il master può selezionare lo slave con cui vuole parlare semplicemente mettendo a LOW il segnale CS dello slave.

Anche se sembrano molto simili, ci sono alcune differenze importanti tra i due protocolli presentati che li rendono adatti per varie applicazioni. Le differenze vengono descritte nella Tabella 4.

I^2C	**SPI**
Può avere più master	Può avere un singolo master
Protocollo a 2 vie	Protocollo a 4 vie (+1 per ogni slave addizionale)
Velocità fino a 3.4 Mbps in modalità High Speed	Velocità illimitata (normalmente implementata tra 10 e 100 Mbps)
Massimo 112 dispositivi con indirizzamento a 7-bit	Limitato dal numero di pin CS del master
Half-duplex	Full-duplex

Tabella 4: I^2C vs SPI[7]

Capitolo 3

Sviluppo del sito web

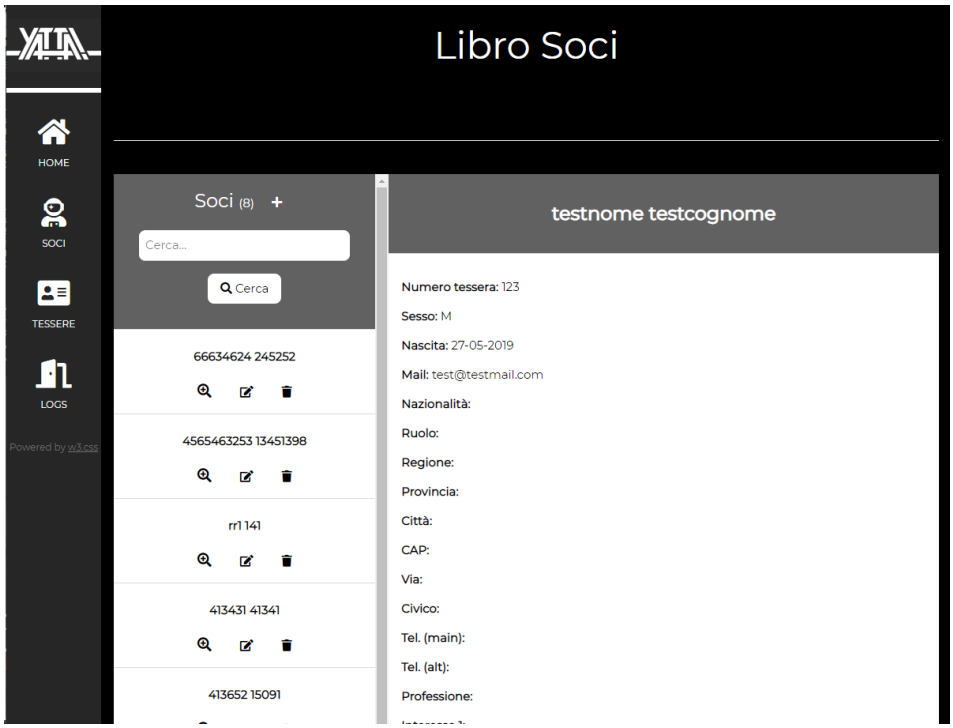


Figura 27: Screenshot fatto durante la fase di test del sito web

In questo capitolo vengono descritte le tecnologie e le scelte progettuali che hanno portato allo sviluppo dell'interfaccia grafica richiesta dai requisiti per la gestione del database dell'azienda (si veda la sezione 1.2 " Requisiti del progetto "). Nella Figura 27 si può vedere un esempio della grafica creata per la gestione del libro soci.

3.1 Back-End

In ingegneria del software, il termine back-end rappresenta il lato server di un'applicazione, ovvero un programma che si occupa dei dati dell'applicazione con i quali un utente interagisce indirettamente tramite il front-end. Generalmente il back-end è implementato utilizzando Javascript (Node.js), Python, PHP o Ruby.

3.1.1 Tecnologie utilizzate

Node.js

Node.js è un framework open-source che esegue codice JavaScript al di fuori da un browser. È basato sul motore JavaScript V8 di Google ed è stato creato per rendere possibile la creazione di un server utilizzando un linguaggio di programmazione con la stessa sintassi di JavaScript (si parla del paradigma " JavaScript-everywhere "). Vengono elencati alcuni vantaggi principali di Node.js

- rende possibile la programmazione ad eventi (un approccio asincrono): un'azione viene lanciata solo quando succede qualcosa (un evento) e non perforza nell'ordine del codice scritto, aumentando l'efficienza soprattutto nell'ambito di networking dove capita spesso dover rimanere in attesa di una risposta;
- scalabilità del software;
- NPM (Node Package Manager): è il più grande ecosistema di librerie open source al mondo. Infatti Node.js ha un enorme quantità disponibile di librerie scritte da altri sviluppatori e possono essere installati facilmente utilizzando NPM;
- utilizza lo stesso linguaggio di programmazione degli sviluppatori front-end, perciò è ancora più semplice diventare un full-stack developer grazie al fatto di non dover perforza conoscere linguaggi come Ruby o PHP.

L'applicazione utilizza vari moduli di Node.js:

- Path (https://nodejs.org/docs/latest/api/path.html) fornisce alcune utilità nel lavorare con i file e con i percorsi delle directory. L'applicazione usa il modulo per definire una directory statica assoluta per i file dell'applicazione senza dover quindi utilizzare l'intero indirizzo della directory " public " (Listato 3.1);

```
1 app.use(express.static(path.join(__dirname, 'public')));
```

Listato 3.1: Uso del modulo *path*.

- Express (https://expressjs.com/) è un framework robusto e flessibile che fornisce un insieme di funzioni che facilitano e velocizzano lo sviluppo di applicazioni web con Node.js. L'applicazione fa uso di express per:
  - definire delle middleware per rispondere a richieste HTTP e per gestire gli errori. Un esempio di middleware si può trovare nel Listato 3.2;

```
1  // Catch 404 and forward to error handler
2
3  app.use(function(req, res, next) {
4    var err = new Error('Not Found');
5    err.status = 404;
6    next(err);
7  });
8
9  app.use(function(err, req, res, next) {
10
11 // render the error page
12   if (err.status === 404) res.render('404');
13   else { // other types of errors
14     res.status(err.status || 500);
15     res.render('error');
16   }
17 });
```

Listato 3.2: Middleware per caricare la pagina " 404.ejs " nel caso in cui un utente provi ad accedere ad una pagina non esistente

  - definire una tabella di routing (lista di endpoint accessibili) che viene usata per eseguire diverse azioni in base all'URL e al tipo di richiesta HTTP (esempio nel Listato 3.3);

```
1  app.get('/', (req, res) => res.render('index'));
```

Listato 3.3: Implementazione di un endpoint. Quando viene fatta una richiesta GET all'endpoint " / " allora, come risposta, l'applicazione invia all'utente un file chiamato " index.ejs " (che rappresenta la homepage) che viene poi caricato dal suo browser

  - caricare pagine HTML in maniera dinamica passando degli argomenti ai template (e.g. usando un linguaggio di templating chiamato EJS);
  - si possono creare in maniera semplice e veloce delle API RESTful (si veda la sezione 3.1.3);
  - interfacciarsi con vari database (e.g. MongoDB, MySQL ecc.);
  - creare un server che ascolta su una porta preimpostata. Per far ciò, express fornisce un'astrazione del modulo *http* di Node.js (esempio nel Listato 3.4).

```
var server = app.listen(port, () => {
  console.log("Listening on port " + server.address().port);
});
```

Listato 3.4: Creazione di un server con il modulo *express*

- Body-Parser (https://www.npmjs.com/package/body-parser) ispeziona i contenuti delle richieste HTTP. Utile per poter leggere i dati in formato JSON (e non solo). Esempio nel Listato 3.5;

```
// for parsing
app.use(bodyParser.json());
app.use(bodyParser.urlencoded( { extended: true } ));
app.use(bodyParser.text());
```

Listato 3.5: Uso del modulo body-parser. Vengono definiti dei middleware che analizzano i contenuti delle richieste HTTP.

- EJS (https://ejs.co/) è un linguaggio di templating che permette di avere dati provenienti dal server direttamente all'interno di documenti HTML. Il codice nel lato server si può vedere nel Listato 3.6, mentre un esempio di utilizzo di EJS nel lato client si può vedere nel Listato 3.7;

```
app.set('view engine', 'ejs');
```

Listato 3.6: Impostazione del " view engine "

```
<% for(var i=0; i<tessere.length; i++) { %>
  <tr class="w3-black tessere-table-mobile">
    <td><%= tessere[i].numero_tessera %></td>
    <td><%= tessere[i].seriale %></td>
    ...
    <% if ( tessere[i].stato === "ATTIVATA" ) { %>
      <td><div class="green-circle"></div></td>
    <% } else { %>
      <td><div class="red-circle"></div></td>
    <% } %>
  </tr>
<% } %>
```

Listato 3.7: Uso di *EJS* nel lato client. Il codice JavaScript all'interno della pagina HTML è racchiuso tra " <% %> " e l'oggetto " tessere " è stato passato dal server in formato JSON. Gli oggetti passati dal server sono racchiusi tra " <%= %> "

- Mongoose (https://mongoosejs.com/) è una libreria di *Object Data Modeling* (ODM) che fornisce un ambiente per la modelizzazione dei dati di MongoDB permettendo la definizione degli oggetti sotto uno schema fortemente tipizzato (si veda la sezione 3.1.2 " Database ").

MVC

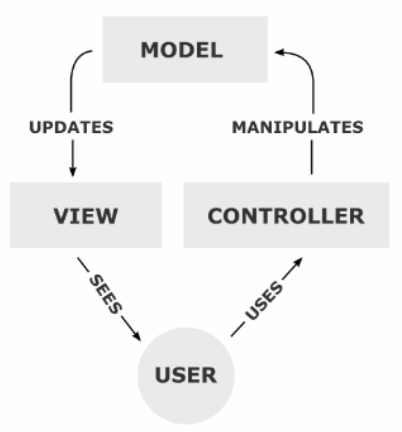


Figura 28: Esempio delle interazioni tra i componenti del pattern MVC

Per garantire l'indipendenza tra i dati e l'interfaccia, si può modularizzare il codice: il sistema viene diviso in moduli, in modo che esso risulti più semplice da comprendere e da manipolare. La modularizzazione è necessaria soprattutto quando aumenta la complessità di un progetto (che deve essere scalabile) poichè garantisce che ogni parte sia modificabile, estendibile e riutilizzabile. Il sito web è stato creato sulla base di MVC (Model-View-Controller), un pattern architetturale (Figura 28) che separa un'applicazione in tre principali componenti logici: il modello, la vista e il controller. Ognuno di questi componenti è progettato per gestire diversi aspetti specifici dello sviluppo di un'applicazione:

- Il modello corrisponde alla struttura e alla logica dei dati dell'applicazione (esempio nel Listato 3.8);
- La vista corrisponde alla logica dell'*User Interface* (UI) dell'applicazione. Rappresenta il modo in cui vengono presentati i dati all'utente (esempio nel Listato 3.9);
- Il controller accetta l'input dell'utente (e.g. visitare il sito web, cliccare su un pulsante, inviare un modulo ecc.) e lo trasforma in comandi per il modello o per la vista (esempio nel Listato 3.10).

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var LogSchema = new Schema (
{
tipo: { type: String, enum: ["telecomando", "tessera", "keypad"],
   required: true, lowercase: true, trim: true },
seriale: { type: String, default: "N/A", uppercase: true, trim: true
    },
nome: { type: String, default: "N/A", lowercase: true, trim: true },
cognome:  { type: String, default: "N/A", lowercase: true, trim:
   true },
numero_tessera: { type: String, default: ""},
data: { type: Date, required: true }
});

LogSchema.index( { '$**': 'text'}); // full-table search index

/*  Tutte le stringhe vengono incluse nell'indice.
 *  Permette cercare l'intero database in questo modo:
 */ Log.find( { $text: { $search: searchTerm }})

module.exports = mongoose.model("Log", LogSchema);
```

Listato 3.8: Creazione del modello dei log con *mongoose*

```
<table class="w3-table w3-left w3-border w3-centered">
  <thead>
    <tr class="w3-white tessere-table-mobile w3-large">
      <th>Numero tessera</th>
      <th>Seriale</th>
      ... <!-- altre colonne -->
    </tr>
  </thead>
  <tbody>
    <% for(var i=0; i<tessere.length; i++) { %>
      <tr class="w3-black tessere-table-mobile">
        <td><%= tessere[i].numero_tessera %></td>
        <td><%= tessere[i].seriale %></td>
        ...
      </tr>
  </tbody>
</table>
```

Listato 3.9: Parte del codice HTML della vista creata per presentare le tessere all'utente

```
exports.fullTableSearch = (req, res) => {
  let searchTerm = req.params.term;
  Socio.
  find( { $text: { $search: searchTerm } } ).
  exec(function(err, soci) {
    if (err) {
      console.error(err);
      return res.status(500).send("Internal Server Error");
    } else return res.render("../views/soci/soci_index", { searched:
        "true", soci: soci, sociString: JSON.stringify(soci) });
  });
}
```

Listato 3.10: Parte del codice del controller creato per gestire i comandi relativi ai soci. Questa funzione, quando viene chiamata, cerca nel database i soci in base al termine inviato tra i parametri e poi carica una pagina con i risultati.

3.1.2 Database

Il Database Management System (DBMS) scelto per l'applicazione si chiama MongoDB (https://www.mongodb.com/). MongoDB è un DBMS open-source e non relazionale (è classificato come un database di tipo NoSQL) ed è orientato ai documenti: al posto delle tipiche tabelle dei database relazionali vengono utilizzati dei documenti in formato JSON, facilitando quindi l'integrazione dei dati con quasi tutte le applicazioni web (infatti JSON è il formato di dati più utilizzato per lo scambio dei dati in rete). MongoDB si integra facilmente con Node.js grazie a dei driver creati dagli sviluppatori stessi (https://mongodb.github.io/node-mongodb-native/3.2/). Questi driver, purtroppo, non portano tanta astrazione e sollevano dei problemi per chi non è abituato a un database NoSQL:

- la logica dei dati è diversa rispetto a quella che viene insegnata nei corsi tradizionali di basi di dati perciò potrebbe essere difficile abituarsi (ma, come si vedrà, anche non necessario);
- non c'è la possibilità di validare i dati in maniera semplice ma bisogna scrivere del codice che potrebbe essere soggetto a errori;
- le query con i driver nativi di MongoDB non sono tanto leggibili e possono diventare molto complesse perchè usano un insieme di operatori (e.g. quello di uguaglianza " $eq "). Esempio:

```
Account.find({ nome: { $eq: 'Andrea' } });
```

- la manutenzione è resa più complicata per la mancanza di una struttura fissa dei dati.

È possibile però avere più astrazione (ad un basso costo di performance) utilizzando Mongoose, una libreria basata sui driver di MongoDB ideata per la modellazione degli oggetti di MongoDB e progettata per funzionare in un ambiente asincrono (come Node.js). Mongoose rende più semplice e immediato l'utilizzo di MongoDB per alcuni motivi:

- facilità di apprendimento;
- query più semplificate e leggibili grazie al " method chaining " di JavaScript (soprattutto quando diventano complesse). Esempio nel Listato 3.11;

```
1  exports.updateSocio = (req, res) => {
2    Socio.
3    findOne( { _id: req.params.id } ).
4    exec( (err, socio) => {
5      if (err) {
6        console.error(err);
7        return res.status(500).send("Internal Server Error");
8      } else return res.render("../views/soci/soci_update", { socio:
           socio });
9    });
10 };
```

Listato 3.11: Esempio di query con *mongoose*. Questa funzione cerca nel database dei soci per l'identificatore passato tra i parametri e poi carica la pagina per aggiornare il socio trovato

- fornisce un modo per dare una struttura ai dati grazie alla creazione degli schemi; ogni schema può essere compilato in un modello, ovvero una classe con cui vengono costruiti i documenti con i comportamenti (metodi) e le proprietà (attributi) dello schema dato;
- permette di validare i dati all'interno degli schemi (e.g. attributo: { type: String, lowercase: true });
- libreria progettata per un ambiente asincrono: è possibile definire delle middleware (pre, post) da chiamare prima o dopo una certa operazione (e.g. di salvataggio).

Come esempio dimostrativo per accedere al database, viene presentata la sequenza degli eventi per aggiungere un log (dopo un accesso con la tessera) all'interno del database e poi visualizzarlo sul sito:

(I) L'ESP-01 manda una richiesta di aggiunta del log all'endpoint */api/logs/add* del server (Listato 2.11);

(II) L'applicazione, dopo aver ricevuto una richiesta POST all'endpoint specificato precedentemente, chiama una funzione per gestire la richiesta (Listato 3.12);

```
app.post('/api/logs/add', logs.addLog);
```

Listato 3.12: Endpoint per aggiungere i log

(III) La funzione gestisce la richiesta e salva il log nel database (Listato 3.13);

```
exports.addLog = (req, res) => {
  var body = JSON.parse(req.body);
  ... // alcune righe di codice per validare i dati
  if (body.tipo === "tessera") { // log con tessera
    Tessera.
    findOne( { seriale: body.seriale } ). // trova la tessera
    populate('socio').
    exec( (err, tessera) => {
      if (err) return console.error(err);
      if (tessera) {
        let log = new Log ({
          nome: tessera.socio.nome,
          cognome: tessera.socio.cognome,
          numero_tessera: tessera.numero_tessera,
          seriale: tessera.seriale,
          data: date,
          tipo: body.tipo
        });
        log.save( (err) => {
          if (err) console.error(err);
          else {
            console.log("Log aggiunto");
            return res.status(200).send();
          }
        });
      } else return res.status(404).send("Not Found");
    });
  } else { ... } // codice per il caso senza tessera
}
```

Listato 3.13: Questa funzione aggiunge il log all'interno del database. Il corpo della richiesta POST contiene il log in formato JSON

(IV) Il log è visualizzabile sul sito accedendo all'endpoint */api/logs/view* (la pagina caricata si può vedere nella Figura 29). Il codice eseguito dopo una richiesta GET all'endpoint specificato si può trovare nel Listato 3.14.

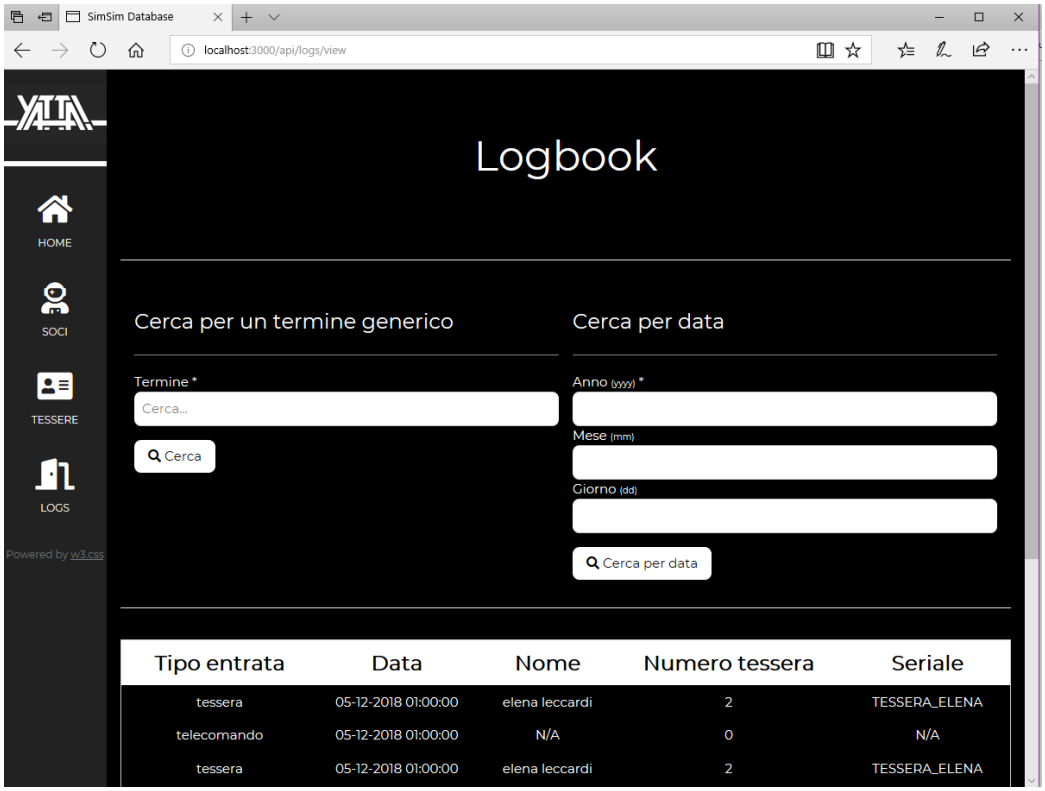


Figura 29: Il logbook dell'azienda

```
exports.viewAllLogs = (req, res) => {
  Log.find( (err, logs) => {
    if (err) {
      console.error(err);
      return res.status(500).send("Internal Server Error");
    }
    else return res.render("../views/logs/logs_index", { logs: logs
        });
  });
}
```

Listato 3.14: Questa funzione viene eseguita quando un browser accede (richiesta GET) all'endpoint */api/logs/view*

Il ragionamento per aggiungere i soci e le tessere al database è simile, solo che si può fare direttamente dal sito: il browser manda i form compilati tramite una richiesta POST al server. Per ogni tipo di richiesta diversa, verrà chiamata una funzione specifica per gestirla.

3.1.3 RESTful API

In ambito web, il termine *Application Programming Interface* (API) è utilizzato per rappresentare un insieme di specificazioni per far comunicare tra di loro client (generalmente il browser) e server tramite richieste e risposte HTTP. Vengono specificati e descritti gli endpoint, i tipi di richiesta (e.g. GET/POST/UPDATE/DELETE) e i tipi di risposta (struttura dei dati inviati, messaggi di errore ecc.). Le API RESTful sono basate su tecnologia *Representational State Transfer* (REST), uno stile architetturale spesso utilizzata nello sviluppo delle applicazioni web. Il concetto più importante in REST è l'esistenza di risorse a cui si possono accedere tramite identificatori univoci chiamati *Uniform Resource Identifier* (URI), ovvero gli endpoint. Tipicamente le API RESTful sfruttano le chiamate HTTP per dare la possibilità a utenti di leggere (GET), aggiornare (PUT), creare (POST) e rimuovere (DELETE) risorse.

Un elenco delle API dell'applicazione si può trovare nella Tabella 5. Le API sono state testate con Postman (https://www.getpostman.com/), un software progettato per lo sviluppo e il testing delle API web. Postman dà la possibilità allo sviluppatore di creare diversi tipi di richieste HTTP da inviare ad un URL. Un esempio di utilizzo di Postman si può trovare nella Figura 30.

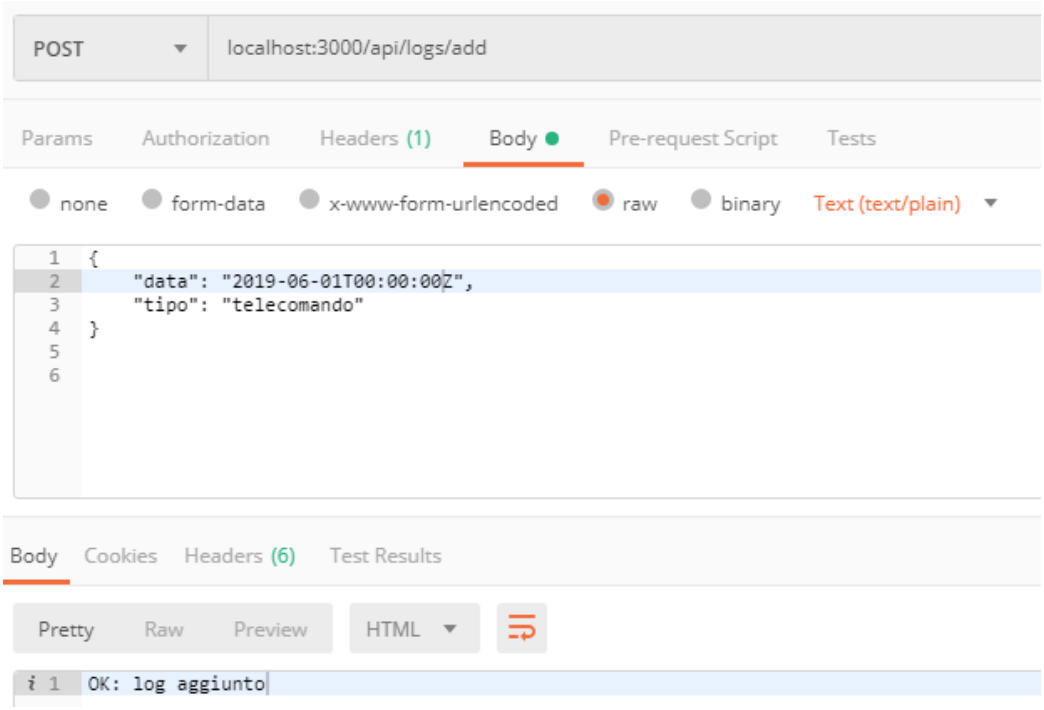


Figura 30: Esempio di testing con Postman dell'endpoint relativo all'aggiunta dei log

Metodo + URI	**Parametro**	**Body**	**Descrizione**
GET /	N/A	N/A	Carica la homepage
GET /api/logs/search/:key	Chiave per la ricerca	N/A	Cerca nel database dei log in base al parametro inviato e carica la pagina con i risultati
GET /api/logs/view	N/A	N/A	Carica la pagina con tutti i log
POST /api/logs/add	N/A	Oggetto in formato JSON secondo lo schema Log	Aggiunge un log nel database
GET /api/soci/search/:key	Chiave per la ricerca	N/A	Cerca nel database dei soci in base al parametro inviato e carica la pagina con i risultati
GET /api/soci/view	N/A	N/A	Carica la pagina con tutti i log
GET /api/soci/create	N/A	N/A	Carica la pagina per creare i soci
GET /api/soci/update/:id	Chiave identificativa del socio	N/A	Carica la pagina per aggiornare un socio
POST /api/soci/add	N/A	Oggetto in formato JSON secondo lo schema Socio	Aggiunge un socio nel database
PUT /api/soci/update/:id	Chiave identificativa del socio	Oggetto in formato JSON secondo lo schema Socio	Aggiorna un socio
DELETE /api/soci/delete/:id	Chiave identificativa del socio	N/A	Cancella un socio dal database
GET /api/tessere/view	N/A	N/A	Carica la pagina con tutte le tessere
POST /api/tessere/check	N/A	Oggetto in formato JSON secondo lo schema Tessera	Verifica se un tag esiste nel database
POST /api/tessere/add	N/A	Oggetto in formato JSON secondo lo schema Tessera	Aggiunge una tessera
PUT /api/tessere/update/:id	Chiave identificativa della tessera	N/A	Cambia lo stato di una tessera (attiva/disattiva)
DELETE /api/tessere/delete/:id	Chiave identificativa della tessera	N/A	Cancella una tessera dal database

Tabella 5: Specificazione delle API dell'applicazione

3.2 Front-End

In ingegneria del software, il termine front-end rappresenta l'interfaccia dell'applicazione, ovvero la parte visibile agli utenti e quella con cui essi interagiscono.

3.2.1 Tecnologie utilizzate

Per creare il front-end sono stati utilizzati i tipici linguaggi come HTML (per la struttura e contenuto), CSS (per lo stile) e client-side JavaScript per aggiungere dinamicità al sito (pulsanti, chiamate AJAX ecc.).

HTML5

Hyper Text Markup Language (HTML) è il linguaggio di markup standard nato all'inizio degli anni novanta per la formattazione e impaginazione di documenti progettati per essere visualizzati in un browser. È stato sviluppato assieme al protocollo HTTP dedicato al trasferimento di documenti di tale formato. Viene spesso utilizzato insieme ad un linguaggio di stile (CSS) e uno di scripting (JavaScript). L'ultima versione standardizzata dal W3C è HTML 5.1. I file HTML generati nel progetto hanno l'estensione EJS perchè vengono usati assieme al linguaggio di templating per portare i dati provenienti dal server direttamente nel documento stesso.

CSS

I documenti HTML sono stilizzati con il linguaggio di stile più diffuso al mondo chiamato *Cascading Style Sheets* (CSS): esso descrive, attraverso delle regole, i modi in cui gli elementi di un documento devono essere presentati e animati nell'interfaccia grafica. Le regole possono essere scritte direttamente nel file HTML, ma ciò crea solo confusione ed è generalmente buona norma avere i file CSS separati e poi importati nel documento (come nel Listato 3.15)

```
1 <link rel="stylesheet" href="/css/w3.css">
2 <link rel="stylesheet" href="/css/custom.css">
3 <link rel="stylesheet" href="/css/font-awesome.min.css">
```

Listato 3.15: Importazione dei file di stile nel documento HTML

Normalmente gli sviluppatori non scrivono le proprie regole (è un lavoro molto lungo e per fare una cosa bella serve tanta esperienza in web design) ma trovano dei template (esistono sia gratis che a pagamento) da utilizzare. In questo caso è stato utilizzato un template (https://www.w3schools.com/w3css/tryw3css_templates_dark_portfolio.htm) di W3.CSS scelto dal tutor aziendale. W3.CSS è un framework moderno, semplice, veloce e gratis che supporta un design mobile-first che aiuta lo sviluppatore

a creare siti web moderni e responsivi. Il vantaggio di un sito responsivo è il fatto che è stato progettato per rispondere automaticamente a tutti i tipi di dispositivi utilizzati per visualizzarlo (e.g. desktop, mobile, tablet). Questo approccio viene implementato sfruttando le caratteristiche delle *media queries* (Listato 3.16).

```
@media (max-width:768px)
{
  .red-circle
  {
    width: 25px!important;
    height: 25px!important;
  }

  .green-circle
  {
    width: 25px!important;
    height: 25px!important;
  }
}
```

Listato 3.16: Esempio di *media query*

Inoltre è stato utilizzato *Font Awesome*, un toolkit di icone distribuito sotto licenze libere (CC BY 4.0 e MIT).

JavaScript

JavaScript è un linguaggio di programmazione introdotto nel 1995 con lo scopo di aggiungere programmi all'interno di siti web sul browser Netscape Navigator. Il linguaggio è stato adottato da quasi tutti gli altri browser esistenti (è stato anche standardizzato sotto il nome di ECMAScript) e ha reso possibile avere applicazioni web moderne con cui un utente può interagire direttamente senza dover ricaricare la pagina[9]. È un linguaggio molto evoluto e studiato nel tempo e ora si definisce come un linguaggio di scripting client-side (ma non solo, come si è visto con Node.js) eseguito dai browser. Il codice JavaScript può essere scritto all'interno di qualsiasi documento, ma è buona norma avere i file con estensione .js separati e poi inclusi alla fine del documento (per far sì che tutti gli elementi vengano caricati dal browser), come nel Listato 3.17.

```
<script type="text/javascript" src=/js/jquery-3.3.1.min.js></script>
<script type="text/javascript" src=/js/forms.js></script>
<script type="text/javascript" src=/js/buttons.js></script>
<script type="text/javascript" src=/js/dettagliSoci.js></script>
```

Listato 3.17: Importazione dei file .js nel documento HTML

JavaScript è un linguaggio che utilizza un singolo thread (organizza l'esecuzione in un'unica pila chiamata *call stack*) e segue il paradigma della programmazione a eventi, perciò è possibile costruire un comportamento non bloccante. Il motore JavaScript crea una coda di eventi (che possono essere operazioni di I/O, richieste di rete, chiamate di WebWorker, funzioni inserite in timer ecc.) che, al verificarsi di un evento, possono modificare l'ordine di esecuzione sequenziale del codice. Si prenda come esempio la funzione nel Listato 3.18.

```
1 function myFunction() {
2   setInterval(function() {
3     alert("Hello World");
4   }, 3000);
5 }
```

Listato 3.18: Programmazione a eventi: funzione che scrive " Hello World " ogni 3 secondi (in questo caso l'evento è rappresentato dal timer)

JavaScript interagisce con il browser attraverso il *Document Object Model* (DOM) della pagina HTML. Il DOM è stato standardizzato dalla W3C e viene creato dal browser dopo che una pagina è stata caricata. Esso segue un modello gerarchico di oggetti (Figura 31) che possono essere manipolati dagli script dando la possibilità di aggiornare dinamicamente il contenuto, la struttura e lo stile dei documenti. Grazie al DOM è possibile accedere agli elementi (e loro proprietà, metodi ed eventi) delle pagine HTML.

Nel progetto è stato utilizzato *jQuery* (https://jquery.com/), una libreria molto diffusa (ormai di meno rispetto a qualche anno fa dato che ora ci sono altre scelte più moderne come React.js, Vue.js e Angular.js) di JavaScript proposta a ridurre e semplificare il codice per lo sviluppo del front-end. Esistono tanti test di benchmark per quanto riguarda la prestazione di jQuery rispetto al " vanilla JavaScript ". Secondo Marco Trombino, uno sviluppatore di front-end che ha testato una funzione che inserisce 10,000 elementi nuovi con una classe all'interno di un altro elemento, è stato rilevato che jQuery è molto meno performante rispetto al JavaScript normale: *" As we could have expected Vanilla performed the task in 18,99 ms, whereas jQuery did it in 195,89 ms. Ten times faster. "*[10]. Nonostante ciò, la scelta di usare jQuery è stata fatta poichè il progetto non è di grosse dimensioni e il costo da pagare per la prestazione non è notevole; in cambio il codice scritto è più corto e semplificato.

Un elemento molto importante di JavaScript di cui ne vale la pena parlare è la tecnica *Asynchronous JavaScript and XML* (AJAX). AJAX è una tecnica molto utilizzata in ambito web che consente l'aggiornamento dinamico di una pagina web senza dover ricaricarla: lo scambio dei dati tra client e server (e.g. invio di un form) avviene nel background.

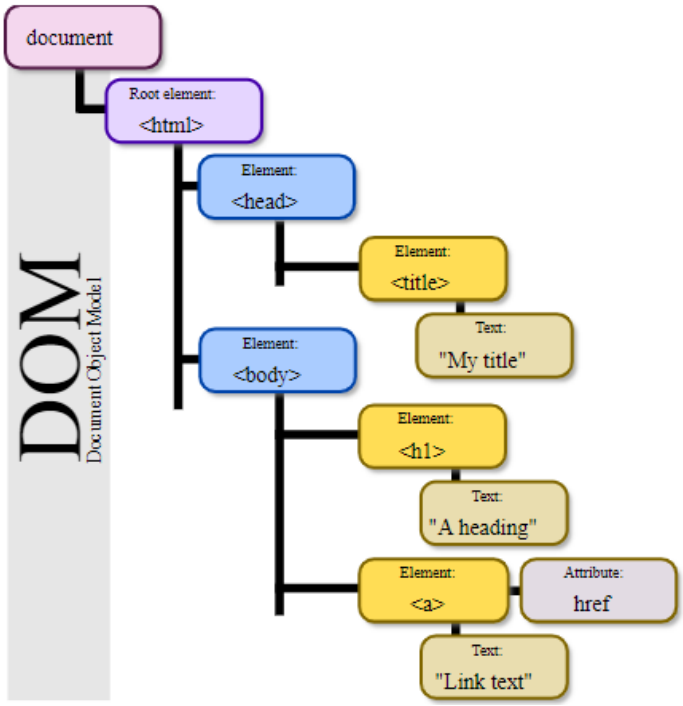


Figura 31: Esempio di gerarchia del DOM in un documento HTML. Immagine sotto licenza CC BY-SA 3.0. Autore: Birger Eriksson. Link: https://commons.wikimedia.org/wiki/File:DOM-model.svg

Alcuni vantaggi dovuti al fatto di non dover ricaricare la pagina sono:

- velocità ed efficienza: il traffico sul server viene ridotto grazie al fatto di dover scambiare solo i dati necessari anzichè l'intera pagina
- sito più responsivo: è possibile aggiornare solo una parte della pagina
- utilizzo minore della larghezza di banda sia dal lato del client che dal lato del server
- validazione istantanea dei form

Tutto ciò è possibile grazie all'utilizzo di un oggetto chiamato XMLHttpRequest che può comunicare con il server per mandare e ricevere dati in vari formati quali JSON, XML, HTML e file di testo in maniera asincrona. Un esempio che descrive l'utilizzo di AJAX si può vedere nel Listato 3.19.

```
1  const rimuoviSocioButton = $(".w3-btn.w3-round.rimuoviSocioButton");
2
3  rimuoviSocioButton.on('click', function(e) {
4    e.preventDefault();
5    if (!confirm("Rimuovere il socio?")) return false;
6    let xHttp = new XMLHttpRequest();
7    xHttp.onreadystatechange = function() {
8      if (xHttp.readyState == 4) {
9        if (xHttp.status === 200) alert("Socio rimosso dal database");
10       else alert("Problema: Status Code = " + xHttp.status);
11     }
12   };
13   let id = this.value;
14   xHttp.open("DELETE", "/api/soci/delete/" + id, true);
15   xHttp.send();
16 });
```

Listato 3.19: Esempio di utilizzo della tecnica AJAX. La funzione, dopo aver premuto il pulsante per rimuovere un socio, manda una richiesta (nel background) del tipo " DELETE " all'endpoint */api/soci/delete/* con l'ID del socio che si desidera cancellare

Capitolo 4

Analisi e Conclusioni

4.1 Problemi affrontati

In seguito vengono descritti alcuni problemi affrontati durante lo sviluppo del progetto e le soluzioni adottate per superarli.

4.1.1 Incompatibilità tra Raspbian e le ultime versioni di MongoDB

Durante lo sviluppo del progetto ho scoperto che Raspbian, il sistema operativo progettato per il Raspberry Pi, in realtà non è stato ancora aggiornato per essere un sistema operativo a 64 bit, perciò non poteva sfruttare al meglio l'architettura a 64 bit del Raspberry Pi 3B+. L'ultima versione di MongoDB che può essere installata su Raspbian è v2.4 (rilasciato nel 2013) e ha alcuni problemi a connettersi con le ultime versioni di Node.js. Inoltre la versione di MongoDB installata su Raspbian è stata progettata per un'architettura a 32 bit, limitata, per motivi tecnici, a 2GB di dati[11]. Come soluzione a questo problema, è stato installato un sistema operativo chiamato openSUSE Tumbleweed (https://software.opensuse.org/distributions/tumbleweed) che funziona su architetture ARMv8 a 64 bit. Ci sono alcune inconvenienze tra cui la più notevole è la minore prestazione offerta dalla Raspberry Pi, però non ha creato grossi problemi e il server è abbastanza responsivo ai comandi.

4.1.2 Problema di sicurezza all'interno della rete locale

Siccome non è stato sviluppato un sistema di login per il sito web (per questioni di tempo), qualsiasi persona connessa alla rete locale dell'azienda poteva accedere al sito e vedere/modificare le informazioni del database, perciò è stato deciso di utilizzare un router dedicato per creare una rete LAN privata (e priva di connessione Internet

dato che il server risiede in locale) all'interno del progetto. All'ESP-01 e Raspberry Pi sono stati assegnati indirizzi IP statici per far sì che, una volta configurati, riescano a comunicare tra di loro. L'amministratore, per poter accedere al sito web, deve semplicemente connettersi alla rete LAN (solo lui conosce SSID e password).

4.1.3 Disattivazione delle tessere dei soci non più iscritti

L'ingresso nello spazio di Yatta è regolato da un tesseramento. Ogni anno, a fine Gennaio, tutte le tessere che non sono state rinnovate vengono disattivate: questo significa che devono essere disattivate automaticamente anche nel database. Per fare questo lavoro è stato creato uno scheduler dal lato del server (Listato 4.1).

```
1  var Tessera = require('../models/tessera.model.js');
2  var schedule = require('node-schedule');
3
4  var rule = new schedule.RecurrenceRule();
5  rule.month = 0; // gennaio - 0
6  rule.date = 30; // giorno 30
7  rule.hour = 18; // ore 18
8  rule.minute = 30; // minuti 30
9  rule.second = 0; // secondi 0
10
11 // e.g. a Gennaio 2019 disattiva le tessere con ultimo_rinnovo<=2018
12 var j = schedule.scheduleJob(rule, function(){
13   nowYear = new Date().getFullYear();
14   Tessera.find( (err, tessere) => {
15     if (err) console.error(err);
16     else {
17       for (i=0; i<tessere.length; i++) {
18         tessera = tessere[i];
19         if ( (nowYear !== tessera.ultimo_rinnovo.getFullYear())
20              && (tessera.stato == "ATTIVATA") ) {
21           tessera.stato="DISATTIVATA";
22           tessera.save( (err, tessera) => {
23             if (err) console.error(err);
24             else console.log(tessera.numero_tessera + " bloccata");
25           });
26         }
27       }
28     }
29   });
30 });
```

Listato 4.1: Implementazione di uno scheduler che disattiva automaticamente le tessere non rinnovate. Questa funzione viene eseguita una volta all'anno e disattiva tutte le tessere ancora attivate e non iscritte all'anno corrente.

4.2 Possibili miglioramenti

Ci sono alcuni possibili miglioramenti che sono stati tralasciati per questioni di tempo:

- Sistema di login sul sito web per l'amministratore: con questo sistema si può eliminare il router dedicato alla rete LAN dato che solo l'amministratore può fare il login.
- Orario di lavoro: è possibile far sì che l'Arduino sia attivo soltanto durante la fascia oraria 9-18 in cui c'è qualcuno in sede. Questo serve per evitare che qualcuno riesca ad aprire la porta in qualsiasi modo (e.g. telecomando o tessera rubata) al di fuori della fascia oraria.
- Sensore di rilevamento: l'Arduino è stato programmato per chiudere la porta dopo un certo intervallo di tempo. È possibile aggiungere un sensore per il rilevamento delle persone per fare in modo che la porta venga chiusa solo quando non ci sono più persone nella vicinanza.

4.3 Conclusioni

TODO

Sitografia

[1] J. Allen, Opening new doors with IP access control, 16 Marzo, 2018. https://www.axis.com/blog/secure-insights/opening-new-doors-with-ip-access-control/

[2] R. Alalouff, Access control leads growth in physical security market but video surveillance still dominates, 25 Gennaio, 2018. https://www.ifsecglobal.com/access-control/access-control-leads-growth-physical-security-market-video-surveillance-still-dominates/

[3] A. Tumino, Internet of Things: gli oggetti intelligenti prima di ogni " cosa ", 24 Gennaio, 2018. https://blog.osservatori.net/it_it/internet-of-things-oggetti-intelligenti-prima-di-ogni-cosa

[4] M. Rouse, Internet of Things (IoT), ultimo aggiornamento Marzo 2019. https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT

[5] Arduino, Un po' di storia. https://playground.arduino.cc/Italiano/StoriaDiArduino/

[6] A. Carraturo, A. Trentini, Sistemi Embedded: Teoria e Pratica, prima edizione: Settembre 2017. http://www.ledizioni.it/prodotto/a-carraturo-a-trentini-sistemi-embedded-teoria-pratica/

[7] S. Hymel, SPI vs I2C Protocol Differences and Things to Consider, ultimo aggiornamento Dicembre 2018. https://articles.saleae.com/logic-analyzers/spi-vs-i2c-protocol-differences-and-things-to-consider

[8] Mozilla, JavaScript, ultimo aggiornamento Maggio 2019. https://developer.mozilla.org/en-US/docs/Web/JavaScript

[9] M. Haverbeke, ELOQUENT JAVASCRIPT: A modern Introduction to Programming, terza edizione. https://eloquentjavascript.net/

[10] M. Trombino, You might not need jquery: A 2018 performance case study. Maggio 2018. https://medium.com/@trombino.marco/you-might-not-need-jquery-a-2018-performance-case-study-aa6531d0b0c3

[11] MongoDB Blog, 32-bit limitations, 8 Luglio, 2009. https://www.mongodb.com/blog/post/32-bit-limitations