Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Progettazione e realizzazione di dispositivi low-cost per la raccolta dati tramite Crowdsensing per IoT collaborativo

Relatore:
Chiar.mo Prof.
Luciano Bononi

Co-relatore:
Dott. Luca Bedogni

Presentata da:
Natale Vadalà

Sessione II
Anno Accademico 2015 - 2016

A Mimmo, Lilli e Salvatore.

Abstract

La redazione di questo documento cerca di inquadrare dal punto di vista tecnico e tecnologico il campo dell' *Internet of Things* intersecato con quello del *Crowdsensing* (rilevazione dati) e, in particolare, della progettazione di semplici dispositivi per la misurazione e la raccolta di dati legata a sensori.
L' *Internet of Things* è uno dei campi di ricerca (ed industria) più crescenti e prolifici negli ultimi anni; esso racchiude l' idea di mappare il mondo reale con dispositivi di rilevamento capaci, ad esempio, di fornire un' intelligenza artificiale ad un elettrodomestico, così da consentirgli di relazionarsi con altri elettrodomestici e, in genere, con altri dispositivi *general purpose*.
Il *Crowdsensing*, invece, è un fenomeno che consiste nella raccolta di rilevazioni di varie entità da utilizzare per analisi successive, e si riferisce alla condivisione dei dati raccolti dai singoli dispositivi di rilevamento con l' obiettivo di misurare o fare interagire fenomeni di interesse comune.
Il presente progetto consiste nella realizzazione di una *Weather Station low-cost* collegata ai server Unibo, con integrati sensori di temperatura, umidità e pressione atmosferica, facilmente fruibile dall' utente finale, visto che non è necessaria alcuna conoscenza particolare per il suo utilizzo, se non quella di aprire una pagina web. Essa nasce dalla necessità di fornire all' utente validi e semplici strumenti che consentono di avvicinarsi al *Crowdsensing*, offrendo pure la possibilità di ottenere un qualche benefit da uno *stakeholder* e/o di poter aumentare notevolmente il numero di canali *reliable* (sicuri) analizzabili dai progetti di ricerca attivi nei campi della statistica, dell' ecosostenibilità, dell' intelligenza artificiale, dell' analisi di reti.

La tesi, ovviamente, è solo un punto di partenza, sia per l'Università, che potrà realizzarla e fornirla a degli utenti per ricerche sul territorio, sia per l' utente privato, che seguendo l' Appendice potrà realizzare la sua *Weather Station low-cost* in poco tempo e con un costo relativamente basso.

Keyword: crowdsensing, dati, sensori, iot, collaborativo

Indice

Elenco delle figure

Capitolo 1

Introduzione

In un mondo che vacilla sotto il peso delle nuove sfide economiche e scientifiche, si avverte sempre più l' esigenza di raccogliere grandi quantità di dati (ai fini statistici, analitici o economici) con il dichiarato intento di contenere la spesa massimizzandone i profitti.
È cosi, che partendo da un contesto prettamente finanziario, nasce il *Crowdfunding*, modello di business che permette di raccogliere, attraverso donazioni sul web, una quota monetaria prefissata per uno specifico fine. L' apporto sostanziale di tale tipo di intervento consiste nella traduzione di un sistema in cui qualsiasi aiuto (economico, intellettuale, lavorativo) proveniente dal singolo va ad implementare un sistema in cui la collaborazione è quel punto in più che permette in modo (relativamente) facile, veloce ed economico il raggiungimento di un obiettivo (che può essere la raccolta di fondi, il *fixing* di un'applicazione, lo sviluppo di una piattaforma). Traslando questo concetto - presente nella cultura occidentale già dal '700 grazie allo scrittore Jonathan Swift - in ambito informatico possiamo parlare tranquillamente di *Crowdsourcing*.

Crowdsourcing

"Richiesta di idee, suggerimenti, opinioni, rivolta agli utenti di Internet da un'azienda o da un privato in vista della realizzazione di un progetto o della soluzione di un problema.".[1]
Con "*Crowdsourcing*" ci si riferisce generalmente alla collaborazione in un progetto di un (numeroso) gruppo di persone, esterne agli ideatori/iniziatori del progetto. Il termine è stato coniato e utilizzato per la prima volta da John Howe sulla rivista *Wired*. Le strategie più famose basate sul *Crowdsourcing* sono quattro:

1. **Crowdfunding**
   Raccolta di fondi attraverso piccoli contributi da molteplici sostenitori della stessa idea/progetto, una sorta di microfinanziamento.
   StartsUp[2], o i ben più famosi *KickStarter*[3] e *Indiegogo*[4] sono le piattaforme online per dare visibilità al proprio progetto e ricevere finanziamenti, o finanziare progetti altrui;

2. **Crowdcreation**
   Collaborazione democratica nelle scelte di un progetto o alla soluzione di un problema.
   Un esempio di piattaforma che ne dà la possibilità è *Starteed*[5], (che è anche una piattaforma per *Crowdfunding*);

3. **Crowdvoting**
   Espressione di un'idea, sensazione, parere riguardo a qualcosa.
   Esempio lampante: *Facebook*[6];

4. **Crowdwisdom**
   Ci riferiamo a crowdwisdom come teoria sociologica e relativa applica-

[1] https://www.google.it/#q=crowdsourcing
[2] http://www.starsup.it/
[3] https://www.kickstarter.com/
[4] https://www.indiegogo.com/
[5] https://starteed.com/crowdcreation
[6] https://www.facebook.com

zione secondo cui la massa sia più precisa nella valutazione di quanto non lo sia un team di esperti.
Esempio di applicazione è il sistema di *PageRanking* di *Google*.

A fronte di una definizione "malformata" e poco confacente alla reale definizione e filosofia che ci sta dietro, Estellés y González fornisce nel 2012 una definizione ben più calzante:
"*Il Crowdsourcing è una tipologia di attività online partecipativa nella quale una persona, istituzione, organizzazione non a scopo di lucro o azienda propone ad un gruppo di individui, mediante un annuncio aperto e flessibile, la realizzazione libera e volontaria di un compito specifico. La realizzazione di tale compito, di complessità e modularità variabile, e nella quale il gruppo di riferimento deve partecipare apportando lavoro, denaro, conoscenze e/o esperienza, implica sempre un beneficio per ambe le parti. L'utente otterrà, a cambio della sua partecipazione, il soddisfacimento di una concreta necessità, economica, di riconoscimento sociale, di autostima, o di sviluppo di capacità personali, il crowdsourcer d'altro canto, otterrà e utilizzerà a proprio beneficio il contributo offerto dall'utente, la cui forma dipenderà dal tipo di attività realizzata.*". [1]
Degli esempi noti di progetti che hanno fatto del *Crowdsourcing* il proprio modello di business:

- *Wikipedia*[7], enciclopedia web scritta dalla collaborazione della comunità;
- *Linux*[8], sistema operativo Unix *open-source*;
- *MIT Climate CoLab*[9], piattaforma che coinvolge più di 10.000 utenti per ideare soluzioni al cambiamento climatico, presso il *MIT Center for Collective Intelligence*[10];

[7] https://it.wikipedia.org
[8] http://www.linux.it/
[9] http://climatecolab.org/
[10] http://cci.mit.edu/

- Il mondo dell' *open-source* in generale, del *software* libero e delle piattaforme per gestire progetti *software* (*GitHub*[11]).

[11] https://github.com/

Capitolo 2

Stato dell'arte

2.1 Crowdsensing

Nato come ramo del *Crowdsourcing*, il *Crowdsensing* è un modello di business nato negli ultimi anni, in cui *devices* diversi sono sfruttati, oltre che ai fini per cui sono stati pensati (telefonare, collegarsi ad Internet, gestire file multimediali), anche per comunicare e condividere valori rilevati da sensori su una qualche piattaforma di raccolta dati.
In analogia, le "idee, suggerimenti, opinioni" che, nella fase di "*outsourcing*" erano il soggetto dello scambio in un modello di *Crowdsourcing*, nel Crowdsensensing vengono sostituite con valori di sensori e/o informazioni legate a misurazioni ambientali o di processi.
Distinguiamo due tipi di *Crowdsensing*, in base allo "sforzo" che l'utente compie per attivare la procedura di comunicazione dei dati:

- *Crowdsensing* partecipativo, che è quello in cui l'utente deve attivare la rilevazione di un certo sensore, o la sua inizializzazione, ed implica il massimo sforzo tra i casi possibili;
  Es. *SenSquare* [2], un servizio che raccoglie valori da sensori (*Crowdsensing* ambientale) inviati da stazioni meteo minimali di utenti e offre un'interfaccia ai possibili *stakeholder* per accedere a queste rilevazioni

- *Crowdsensing* opportunistico, che è quello in cui l'utente manda automaticamente i valori, probabilmente perché sono sensori inizializzati senza input utente (fuorché l'alimentazione), e comporta un minimo sforzo.

  Es. *Google* o *Facebook* o gli operatori telefonici, ed in generale i dispositivi che si prestano al *Mobile Crowdsensing*, che tengono, cioè, traccia di sensori come luce, rumore (disturbo), posizione (GPS) e movimento (accelerometro) automaticamente senza una esplicita azione dell'utente se non l'installazione dell'applicazione (e la concessione dei permessi necessari, come nel caso delle apps di Google) e/o alla sottoscrizione del servizio (come per gli operatori, che possono leggere i valori riferiti alla qualità della nostra linea dati o alla nostra posizione cellulare).

2.2 Internet of Things

Risale all'ormai lontano 1999, la nascita dell'estensione naturale di quella che era già allora la rete più grande mai creata fra persone sul globo: Internet. Il neologismo *Internet of Things* si riferisce all'integrazione della rete con gli oggetti di uso comune, alla connessione fra strumenti e all'accesso reciproco ad informazioni acquisite da altri dispositivi.
Naturalmente, per spiegare un così grosso impatto sociologico di un fenomeno del quale parliamo ancora dopo quasi 20 anni, è evidente che esso contenga in sé potenzialità, e quindi possibilità di sviluppo e/o campi d'applicazione, ben più interessanti della mera definizione o della storia dell'IoT.

Oggi l'IoT è pesantemente presente in:

- Domotica;
- Industria;
- Telematica;
- WSN (*Wireless Sensor Network*);

- *Security*/Sorveglianza;
- Robotica;
- Monitoraggio ambientale (Fini geologici, sismologici, meteorologici, ecologici).

Mentre i settori in cui il mondo dell'*IoT* sta convogliando le proprie attenzioni sono:

- Ingegneria Biomedica;
- Sicurezza (intrinseca) dei dispositivi adattati al mondo dell'*IoT*;
- Ricerca scientifica.

Partendo dall'idea di *IoT*, passando per un business model come il *Crowdsourcing*, approdiamo proprio in uno dei settori in cui l'*IoT* si sta focalizzando: l'*IoT* collaborativo, un sistema che permetta di raccogliere misurazioni da *devices* **eterogenei** in un'unica locazione, che le gestisca e che fornisca a **molteplici** utenti una metodologia di scrittura/lettura di ciò che può essere ritenuto interessante: in poche parole, l'*IoT* collaborativo è la nuova frontiera della condivisione di informazioni [3].

2.3 Lavori correlati

In uno scenario ormai attivo da quasi vent'anni, la letteratura prodotta è innumerevole. Molteplici sono gli articoli scientifici che trattano la standardizzazione dei protocolli per l'*IoT*, come tante sono le pubblicazioni che presentano ciò che viene definito come "*Smart Cities*"; addirittura in letteratura, troviamo progetti simili a quello che stiamo trattando (con qualche differenza implementativa), prodotti da grandi aziende/fornitrici di servizi online.

2.3.1 Standard

Sebbene uno standard non sia stato ancora raggiunto, su più fronti la ricerca sta affrontando questa tematica, su più versanti e da più angoli visuali. Il primo punto da affrontare è quello del protocollo di rete (a livello d'applicazione) da utilizzare, diverso da HTTP (possibilmente più leggero), ma comunque standardizzato da una qualche specifica. È qui che ci vengono in aiuto CoAP[1] (Constrained Application Protocol) e MQTT[2](Message Queuing Telemetry Transport), due protocolli lightweight facilmente interfacciabili con HTTP e, soprattutto, che supportano bene la comunicazione Machine2Machine.

Già nel 2012 Zack Shelby e altri ricercatori presentavano la potenza di CoAP per attività di Crowdsensing e per dispositivi con risorse limitate in un articolo pubblicato sull'*IEEE Internet Computing*[4], lo stesso Shelby che l'anno dopo ha rilasciato la specifica per il protocollo[5]. Nel 2016 viene pubblicato un modello di integrazione fra dispositivi per IoT tramite CoAP, che presenta, oltre ad un confronto fra i vari protocolli di rete, il protocollo CoAP come il migliore fra tutti quelli analizzati (MQTT, DDS, XMPP, RestFull HTTP). "CoAP è un protocollo applicativo per *Internet of Things*, che utilizza le funzionalità simili a HTTP, con basso *overhead* e *multicasting* per la comunicazione di gruppo all'interno dell'*IoT*. A differenza dei protocolli basati su HTTP , CoAP opera su UDP in modo che il sovraccarico del protocollo TCP possa essere ridotto al minimo. Così, CoAP ottimizza la lunghezza del *datagram* e fornisce una comunicazione affidabile sulla sommità di un protocollo UDP inaffidabile. Inoltre, si è basato su architettura REST con metodi di base, come ad esempio GET, POST, PUT e DELETE, per l'accesso delle diverse risorse Internet tramite *Web Services*. Il modello client / server viene applicata con l'utilizzo di comunicazione *Request / Response*. Il protocollo CoAP offre le seguenti caratteristiche: protocollo specializzato

[1]https://tools.ietf.org/html/rfc7252
[2]http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

per *IoT*, scambio di messaggi asincroni, trasporto UDP con livello di applicazione affidabile, spese generali di *header* ridotte, mappatura stateless HTTP a COAP, supporto proxy HTTP/CoAP con caching."[6] Inoltre, nel modello presentato precedentemente, all'apice della piramide (il Web Service), viene arricchito il "vecchio" MES (Manufacturing Execution System) con lo stesso MES ospitato, questa volta, in un cloud, l'hosting del XXI secolo. Ciò viene confermato da moltissimi progetti e articoli che considerano il *cloud* come la base di partenza per un servizio dedicate alle interazioni *IoT*, come [7] dell'Università di Bologna, dimostrando quanto la scalabilità di questo tipo di reti possa essere mantenuta solo usando i mezzi adatti (*cloud* e protocolli MQTT/CoAP).
Uno dei progetti attivi basati su CoAP è descritto nell'articolo[8], che tratta della progettazione e realizzazione di un sistema di *Crowdsensing* sulla salute umana (usando il set di standard ISO/IEEE 11073 Personal Health Data), e dimostrando ancora quanto CoAP sia più adatto al *Crowdsensing* e all'*IoT*.
Il secondo punto riguarda gli standard di lettura di sensori, trattati egregiamente in un *paper* del 2016 che descrive lo standard ISO/IEC/IEEE P21451-001, riguardante la corretta misurazione di *smart transductor* (sensori)[9] e l'algoritmo RTSAL (Real Time Segmentation And Labeling). L'IEEE, dal canto suo, allo stato attuale ha in stato di lavorazione l'"IEEE Draft Recommended Practice for Signal Treatment Applied to Smart Transducers", sempre riguardante la famiglia di standard P21451-001.

2.3.2 Smart Cities

Molteplici sono i progetti che riguardano il campo delle "*Smart Cities*", che consiste nell' integrare servizi in modo da poter risolvere, o almeno tenere sotto controllo, alcuni fenomeni problematici come il cambiamento del clima, l'offerta e la domanda di energia, la scarsità di carburante e risorse naturali, la crescita della popolazione, l'assistenza sanitaria, l'urbanizzazione, il cibo, la tracciabilità e sicurezza, il declino dell'ecosistema naturale. Questa sfida è stata colta in pieno nel 2015 dall' ALMANAC Project[10], fondato dalla

Commissione Europea per creare reti capillari integrando le reti EDGE e quelle metropolitane in una *Smart City Platform*.
Altri famosi progetti di "*Smart Cities*" sono:

- **SmartSantader**[3], nato a Praga nel 2009, è ciò che incarna lo spirito di ALMANAC, è una rete sperimentale di dispositivi e servizi che tuttora è integrata con il progetto ALMANAC;
- **UrbanWater**[4] project, nato per una maggiore efficienza nella gestione idrica di Almerìa (Spagna);
- **Open IoT**[5] definisce una soluzione *cloud open-source* per l'*IoT*, e può essere visto come pioniere tecnologico nel campo delle piattaforme *Smart City*;
- **Mobosens US** fornisce ai cittadini una piattaforma per la raccolta e la condivisione ambientale di dati.

Gaurav Sarin, indiano, ha prodotto nel 2016 un *paper* intitolato "*Developing Smart Cities using Internet of Things: An Empirical Study*"[11], molto interessante poiché ci mostra un dato per nulla banale: nel 2011 il numero dei dispositivi connessi ad Internet era quasi il doppio del numero di abitanti del nostro pianeta (12.5 miliardi contro 7 circa).
Partendo da ciò, e considerata l'importanza (per cittadini, aziende, governi) di poter sfruttare tutta questa corrente di dati e di potenza di calcolo a disposizione, Sarin valuta come fattori da tenere in considerazione 5 aree:

1. **Congestione del traffico**, installando un GPS in ogni nuovo veicolo;
2. **Qualità dell'aria**, con delle sorta di "*backdoors*" sulle app per l'allenamento fisico che comunicano con delle stazioni il livello di qualità dell'aria;

[3]http://www.smartsantander.eu/
[4]http://urbanwater-ict.eu/
[5]http://www.openiot.eu/

3. **Salute**, monitorando i cambiamenti del battito cardiaco, della temperatura, della pulsazione e della respirazione dei pazienti critici;

4. **Energia**, dando un riscontro alla cittadinanza di quanta energia sta consumando un edificio pubblico, ad esempio;

5. **Infrastrutture**, tenendo sotto controllo con sensori le vibrazioni, il rumore e lo stress dell'edificio, oltre che la possibilità di installazione di qualsiasi stazione, utile alle categorie precedenti, proprio su infrastrutture importanti.

2.3.3 Mobile Crowdsensing

Parliamo di *Mobile Crowdsensing* (*MCS*) quando, banalmente, il dispositivo che raccoglie informazioni non è destinato a rimanere fisso in una data posizione geografica.
I dispositivi che sono considerati adatti per il MCS, in linea di massima, sono gli *smartphone*, i *tablet*, i *laptop*, e tutti i *device wearables* ("vestibili"), come orologi, occhiali, pacemaker, braccialetti con computer integrati (Smartwatch, FitBit[6] e i Google Glasses[7] sono gli esempi più lampanti).
Tra i progetti più famosi italiani di MCS troviamo:

- **SecondNose** [12], progetto italiano per misurare la qualità dell'aria con un dispositivo simile ad un portachiavi dato a 80 cittadini di Trento, che attualmente conta circa 30.000 rilevazioni al giorno;

- **ParkHere!** [13], sistema di comunicazione fra utenti per segnalare un parcheggio in una città, basato su giroscopi e accelerometri.

2.3.4 Back End

Sebbene nel progetto in discussione ci si appoggi su server proprietari, esistono alcuni servizi *online* che forniscono spazi web gratuiti per scrivere le

[6] https://www.fitbit.com/
[7] https://www.google.com/glass/start/

proprie rilevazioni, analizzarne i dati e scaricarli su un terzo dispositivo.
Queste piattaforme per l'*Iot*, oltre a fornire un'indubbia comodità, grazie alla grandissima comunità che le utilizzano sono diventati *de facto* dei database mondiali di misurazioni.

- **Thingspeak**[8], lanciato nel 2010, è una piattaforma open-source che fornisce spazio web e API su protocollo HTTP. Importante poichè, fra l'altro, integra MATLAB della MathWorks, Inc.[9]. È stata utilizzata nella fase iniziale del nostro progetto, come anche in passato per lo sviluppo di Thingspeak4Android[10], applicazione per crowdsensing dell'Università di Bologna. Thingspeak, inoltre, nel 2016 presenta un progetto di Weather Station basata su Arduino[11] e/o Raspberry Pi[12].

- **SparkFun Electronics**[13], azienda nel campo dell'elettronica, produce e vende microcontrollori ed accessori utili per l'*IoT*, tutto con *hardware open-source*. Lanciata nel 2003 come azienda, più tardi come servizio API-REST per *Crowdsensing*, oltre ad un funzionamento simile a Thingspeak [14], presenta annualmente progetti/sfide nel campo dell'automazione, tra cui la costruzione di un' automobile autonoma.

[8] https://thingspeak.com/
[9] http://www.mathworks.com
[10] https://bitbucket.org/eldiablo100000/thingspeak4android
[11] https://www.arduino.cc/
[12] https://www.raspberrypi.org/
[13] https://www.sparkfun.com/
[14] https://data.sparkfun.com/

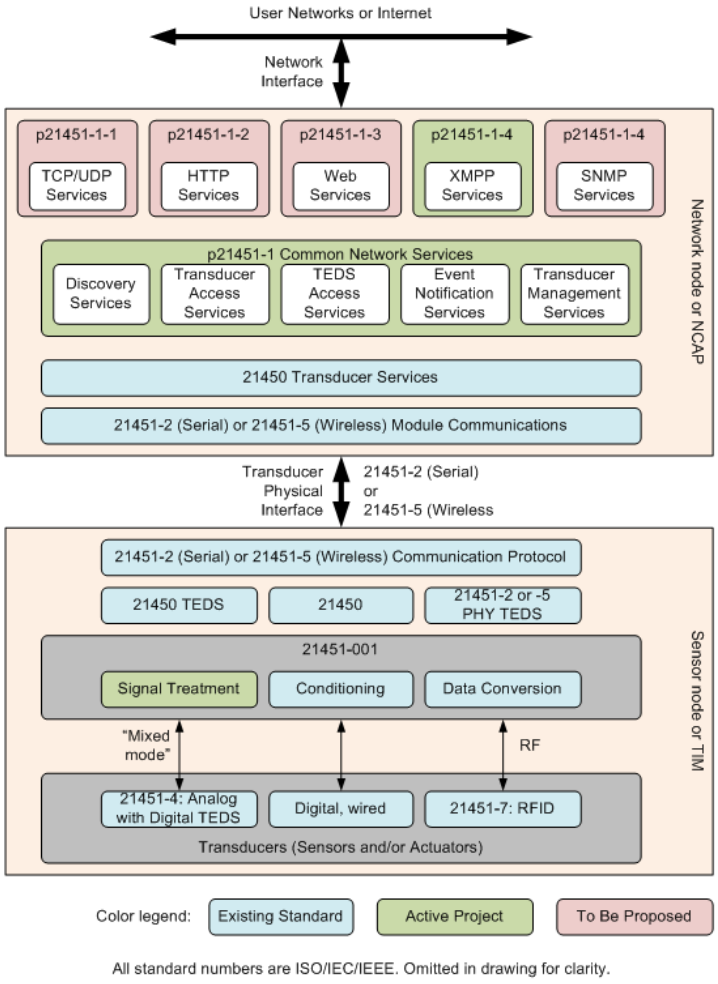


Figura 2.1: Famiglia di standard P21451-01

Fonte: http://www.eejournal.com/files/7814/3044/0385/21450_family_reduced.png

Capitolo 3

Hardware

Abbiamo considerato la fase di progettazione e sviluppo *hardware* come la migliore da cui partire, sia nella creazione effettiva che nella documentazione, per poter fornire una visione *BOTTOM-UP* del progetto, e quindi rendendolo abbastanza intuitivo da capire.

3.1 Progetto

La fase di progettazione, iniziata nel mese di Ottobre 2016, è stata preceduta da una fase embrionale, in cui è stata avviata un'analisi delle componenti e dei microcontrollori al fine di sviluppare una libreria per Arduino per fare *Crowdsensing*, appoggiandosi su un *Web Service online* (Thingspeak) per salvare i monitoraggi, gestirli e ottenerne statistiche. Questa fase è durata circa un mese, e ha contribuito ad individuare le problematiche intrinseche di alcune *boards* (schede) e, quindi, di affinare quegli strumenti valutativi che, in un momento successivo, possono consentire di operare la scelta più funzionale, economica ed espandibile fra le alternative.

Il presente progetto consiste nell' aggregazione di un modulo microcontrollore, una scheda di rete, dei sensori e una predisposizione per l'alimenta-

zione da pile.

La scelta operata è stata quella di creare una *Weather Station* non più basata su Arduino (come in fase embrionale), bensì direttamente su un modulo ESP8266, un modulo *WiFi low-cost* con un MCU (*Micro Controller Unit*) integrato, con ovvi ed evidenti benefici:

1. Abbattere i costi.
   No Arduino, no *shields*/schede di rete/moduli XBEE, solamente un modulo ESP acquistabile a 9$;

2. Essere svincolati da boards "preconfezionate".
   Poter ottimizzare il tutto, eliminando ciò che non è necessario che potrebbe solamente consumare energia inutilmente alla *Weather Station* e costruendo la *board ad hoc* per la nostra esigenza;

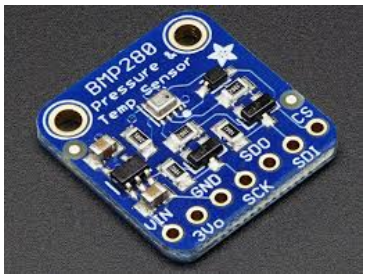
3. Velocizzare la fase di sviluppo.
   Esiste un'intera *community* su GitHub dedicata al "chip più cheap" del mondo *network* [1].

3.1.1 Componenti

Le componenti utilizzate sono:

DHT22 (AM2302)
Sensore di temperatura ed umidità.

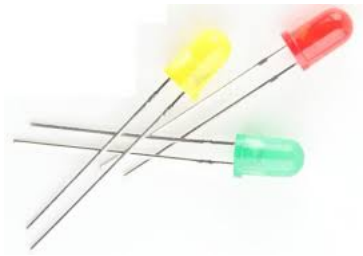
[1] https://github.com/esp8266/



BMP280,
Sensore di temperatura e pressione atmosferica.

ESP8266
Chip WiFi con MCU integrata.

Bottoni
Uno per registrare la *Weather Station* sulla rete, attraverso un'interfaccia web **B1**; l'altro per il *reboot/wake-up* del dispositivo **B2**.



Led
Uno giallo **L1** e uno rosso **L2**, per dare un riscontro all'utente dell'effettivo funzionamento del dispositivo senza errori.

Batteria
Una sola batteria ricaricabile al litio per ottenere un ottimo *trade-off* tra una potenziale richiesta superiore di energia dalla *Station* e la durabilità di queste batterie.

3.1.2 Casi d'uso

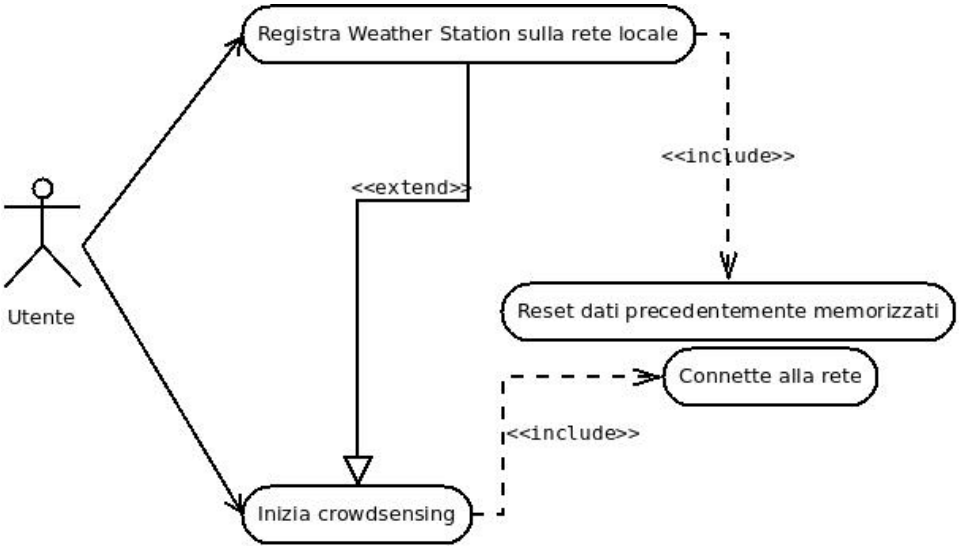


Figura 3.1: Diagramma dei casi d'uso

A livello *hardware* (come anche a livello *software*), la *Weather Station* è stata pensata in modo che consumi energia il meno possibile, infatti per la gran parte della "giornata tipo" reale del dispositivo prodotto, esso sarà in modalità "dormiente" (*deepSleep mode*).
Semplicemente, la *Weather Station* dà la possibilità di Registrarsi su una rete, cliccando il bottone B1 per almeno 3 secondi, o iniziare il *reporting* con le credenziali che ci sono in memoria (se esistono). Se il dispositivo è in *deepSleep mode* occorre svegliarlo prima con il bottone B2.
La fase di Registrazione consiste ne:

- ESP in modalità *STATION*: scansione delle reti *WiFi* vicine;

- ESP in modalità *SOFT AP*: il modulo crea una rete locale con SSID "WeatherStationESP" a cui collegarsi per vedere la pagina di registrazione;

- L'utente si connette alla WiFi "*WeatherStationESP*" attraverso qualunque dispositivo, e collegandosi alla pagina http://192.168.4.1 accederà al sito;
- L'utente sottomette il *form* e le credenziali vengono (sovra)scritte in memoria EEPROM, poi deve chiudere il server dal *browser*;
- ESP *RESET*: il modulo si riavvia.

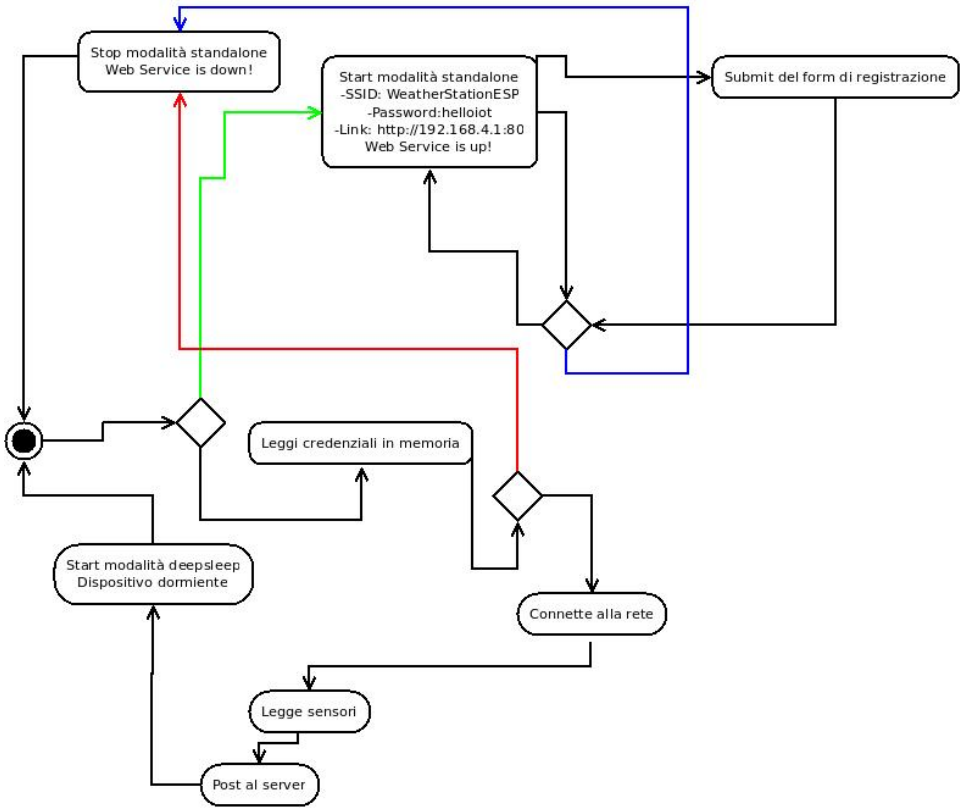


Figura 3.2: Diagramma degli stati

Ramo eseguito se il bottone sulla *Weather Station* viene premuto per almeno 3 secondi (ed il led giallo lampeggia).

Ramo eseguito se le credenziali di rete non vengono ritrovate in memoria sul dispositivo o la lettura non è andata a buon fine (Necessaria registrazione) .

■ Ramo eseguito se viene spento il *Web Service standalone* attraverso l'interfaccia utente(click su "*Close the server!*" o con una richiesta GET http://192.168.4.1/Exit+ "/*Exit*")

3.2 Realizzazione

Nella realizzazione, per tutta la fase di sviluppo *software* e di *testing*, è stato usato un Espduino[2] come *board* economica programmabile con un modulo ESP8266-12F integrato, sostanzialmente un modulo ESP con un ambiente di sviluppo facilmente interfacciabile da chiunque non abbia troppa dimestichezza con la circuiteria interna dei moduli ESP8266*, né con le componenti da acquistare e/o saldare su una PCB.

La costruzione, e realizzazione, è iniziata nel Settembre 2016 ed è terminata solo con il *deployment software* (per via di problematiche e *bug* testabili solo via *software*) nel Novembre 2016.
Il nostro percorso ha visto "nascere" 3 prototipi, ognuno più evoluto (e più piccolo) del precedente:

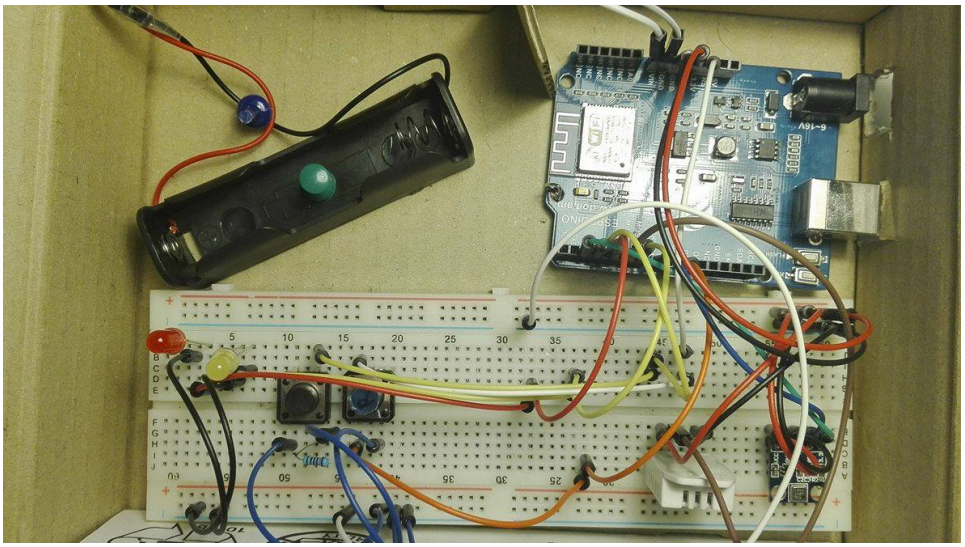


Figura 3.3: Prototipo 1
Espduino con *breadboard* e predisposizione per pila 18650 a Li.

[2]http://www.doit.am/

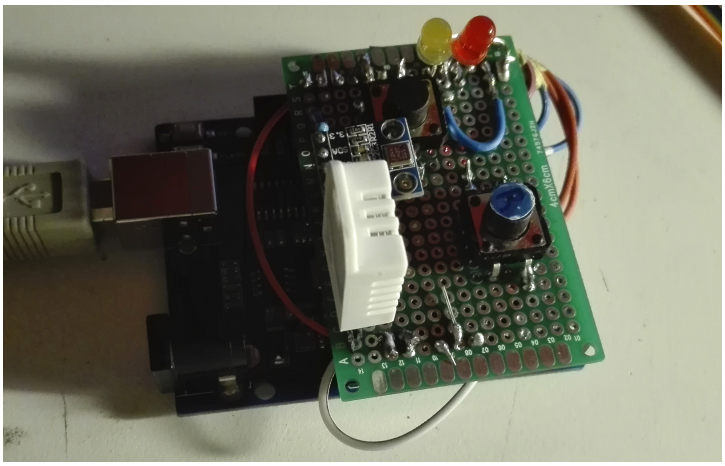


Figura 3.4: Prototipo 2
Espduino con *shield* creata ad hoc e predisposizione per pila 18650 a Li/3 pile AA a Ni-MH.

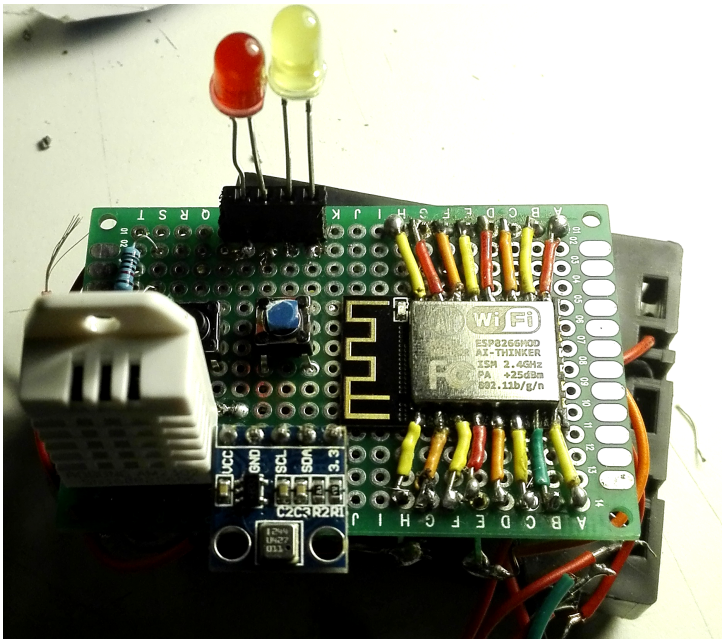


Figura 3.5: Prototipo 3
Board con ESP8266-12F integrato e predisposizione per pila 18650 a Li/3 pile AA a Ni-MH.

3.2.1 Schema

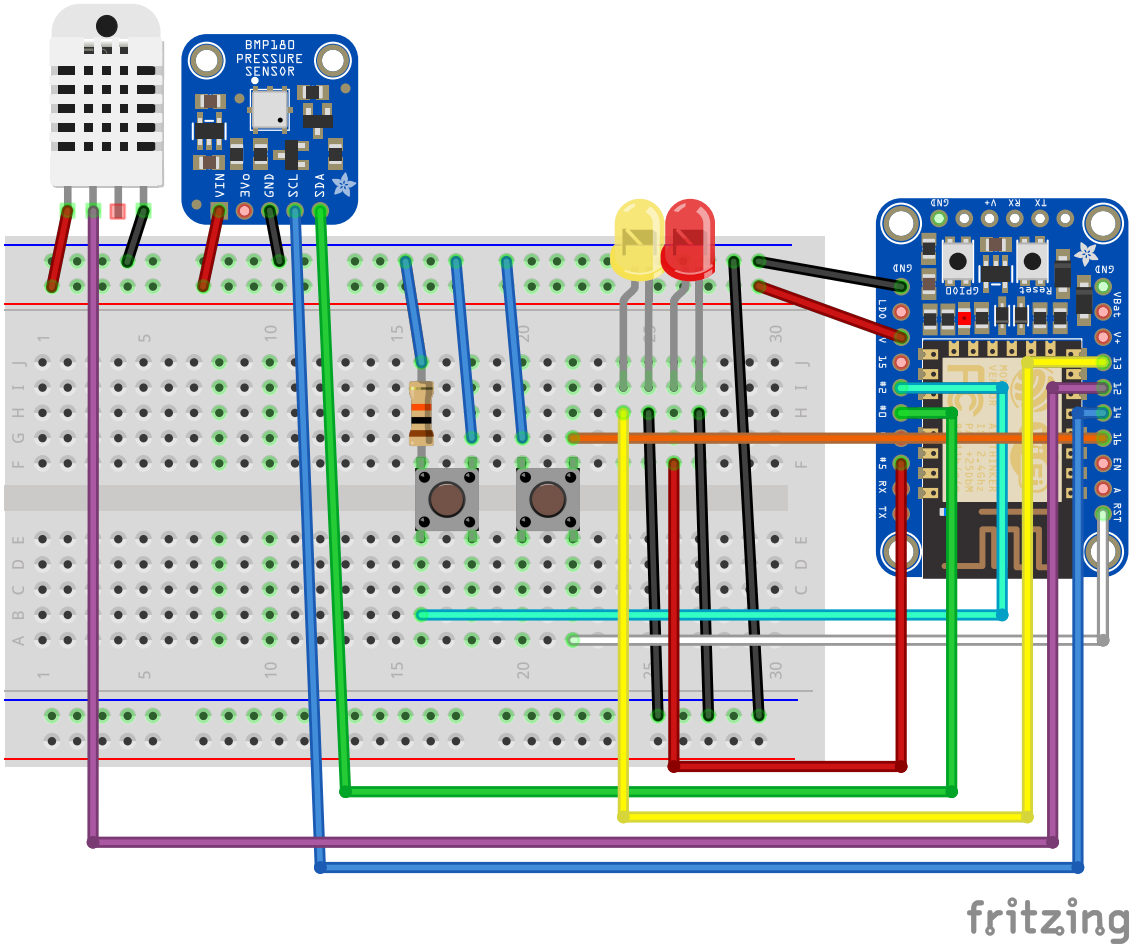


Figura 3.6: Schema completo di microcontrollore e componenti

3.2.2 Problematiche incontrate

La realizzazione *hardware* è stata ultimata solo dopo la progettazione e la realizzazione del software,in quanto sono emerse delle criticità risolvibili solo con un test software.

- Il primo, in ordine cronologico, è stato quello di inserire un bottone che, alla pressione per un dato tempo, facesse scattare una routine. La problematica sta nel fatto che se la *Weather Station* si trova in fase di "*posting*" dati sul server, è possibile rilevare cambiamenti di stato del bottone, mentre se è in fase di "*deepSleep*", il dispositivo non è in grado di capire se il bottone è stato premuto, e quindi non può far scattare la routine (che sarebbe la routine di registrazione sulla rete).
  Il problema è stato "aggirato" montando un secondo bottone che connette il GPIO16 dell'ESP al RESET, permettendone il risveglio (Il GPIO16 è quello dedicato al risveglio del dispositivo, il RESET è sempre *HIGH* anche in "*deepSleep mode*").

- Il secondo, nella fase finale, è stato quello di riportare tutto lo schema su PCB. Ciò ha comportato la perdita dei benefici che c'erano prima lavorando con una *board* invece che con un ESP su una scheda millefori.
  Una *board*, come Espduino, o Arduino, forniscono tutti i componenti utili "già pronti", come gli spinotti per l'alimentazione da USB con FT232 di FTDI[3], o la predisposizione di circuiteria di RESET e FLASH. Questo poiché l'ESP supporta tre modalità[4]: *Flash SPI*, che consiste nel caricamento del programma in memoria e della sua esecuzione, *UART* per caricare un nuovo *sketch*(*script Arduino-like* in C++) dalla seriale , e *SDIO*, che lancia il boot dalla SD. Per caricare uno sketch sull'ESP lo si deve impostare in modalità UART, resettarlo e caricarlo dall'IDE, cosa che con la nostra PCB non è stata possibile fino alla rea-

[3] http://www.ftdichip.com/

[4] https://github.com/esp8266/esp8266-wiki/wiki/Boot-Process

lizzazione di una circuiteria, seppur minimale, per resettare e settare in *UART mode* il nostro modulo *WiFi* ESP.

Capitolo 4

Software

4.1 Metodologia di sviluppo

Per quanto il lato *software* sia stato prodotto singolarmente da una persona, si è cercato di seguire un metodo di sviluppo che si confacesse il più possibile a quello che è il paradigma dell' *ASD (Agile Software Development)*, con maggior enfasi su alcuni punti dell' *"Agile Manifesto"*[1] e pratiche comuni legate alle *"Regole dell'Extreme Programming"*[2].
Dall' "Agile Manifesto":

- Rispondere al cambiamento più che seguire un piano;
- La collaborazione col cliente più che la negoziazione dei contratti;
- Il software funzionante più che la documentazione esaustiva.

Dalle "Regole dell'*Extreme Programming*":

- Scrivere le *User Stories*;
- Produrre piccoli e frequenti release software;
- Dividere il progetto in iterazioni;

[1] http://agilemanifesto.org/
[2] http://www.extremeprogramming.org/rules.html

- Rispettare gli standard nel codice;
- Creare un test quando un bug viene trovato;
- Creare test unitari per tutto il codice;
- Tutto il codice deve superare i test unitari prima della release;
- Rifattorizzare il codice (*Refactoring*) il più possibile.

4.2 Pianificazione

User Stories

Una buona "user story", secondo i canoni della programmazione Agile, deve essere INVEST, acronimo di:

- **Independent**, ogni storia deve essere *indipendente* dalle altre;
- **Negotiable**, non definisce una specifica, ma una "feature" ("Dovrebbe fare..."), quindi *negoziabile*;
- **Valuable**, deve essere *valutabile* da tutti gli attori coinvolti;
- **Estimable**, banalmente deve essere *stimabile*;
- **Small**, cioè abbastanza *corta* (e ragionevolmente esplicativa) da essere conseguita in poco tempo;
- **Testable**, naturalmente deve essere *testabile*.

Le U.S. sviluppate in fase di pianificazione sono state 3:

1. L'utente deve registrare la propria rete *WiFi* attraverso una routine;
2. L'utente può resettare le credenziali di rete attraverso una routine;
3. L'utente può visitare il *Web Service* sulla piattaforma.

In seguito a questa prima stesura, le "dipendenze" da risolvere sono aumentate: il modulo ESP, viste le esigenze, deve cambiare "modalità di *setup*", poiché oltre a leggere da memoria EEPROM e, se possibile, collegarsi alla *WiFi*, ora deve *switchare* (all'occorrenza) in modalità *STA_MODE* (*Station*) per fare una scansione delle reti *Wireless* rilevate e caricarle sulla *DropDown List* degli ESSID disponibili (e non *hidden*) a cui collegarsi, poi *switchare* in *SOFTAP_MODE* (*Software Access Point*) per servire un *Web Service* su una nuova rete locale creata e, alla chiusura del *gateway*/server, resettarsi e ripartire in *AP_MODE* (*Access Point*) per incominciare/riprendere l'attività solita (Post di dati attraverso una *WiFi*).

In un secondo momento, sono state aggiunte U.S. più specifiche, riferite alle informazioni di cui il dispositivo deve tenere traccia:

1. Il dispositivo deve misurare la potenza del segnale *WiFi* (*RSSI*, *Received Signal Strength Indication*) a cui è collegato;
2. Il dispositivo deve rilevare la potenza (in Volt) dell'alimentazione elettrica;
3. Il dispositivo deve prevedere una modalità *a basso consumo / deepSleep Mode* e risvegliarsi autonomamente;
4. Il dispositivo deve dare all'utente un minimo di *feedback* attraverso dei led.

4.3 Progettazione

La progettazione *software* è stata scandita in 6 aree di pertinenza:

1. **Setup**, funzione che fa parte delle *Structures* di Arduino insieme alla Loop, eseguita una volta sola al lancio di uno *sketch*; solitamente gli *sketches* per ESP sviluppano il *main* tutto all'interno di questa funzione; il nostro *Setup* inizializza la seriale per il *debugging* e i pin di

INPUT/OUTPUT per i led e il bottone B1, setta i pin per attivare il protocollo I^2C (necessario per la comunicazione con il sensore BMP), legge da EEPROM le credenziali (se esistono) e tenta di connettersi all' ESSID trovato con la relativa *password* di rete, scatta la routine di registrazione altrimenti.

```
void setup();
```

2. **Loop**, è il *main* della nostra *Weather Station*; dà la possibilità di registrare la il dispositivo, se premuto il bottone B1 in tempo, altrimenti inizializza i sensori, li legge (se inizializzati correttamente) e li posta al server. In seguito va in *deepSleep Mode* per 30 minuti e, al risveglio, riparte dalla *Setup*.

```
void loop();
```

3. **Connections**, funzioni che gestiscono le connessioni del modulo, cioè la connessione *Standalone* (Crea una nuova rete locale a cui collegarsi per registrare la *Weather Station*), la connessione alla rete *WiFi*, la scansione delle reti *WiFi* disponibili e la post al server.

```
String launchStandAlone(const char * essid,const char *
    pwd); // default: "WeatherStationESP","helloiot"
void connectToNetwork(const char essid[], const char
    password[], const char string[]);
char* scanNetworks();
int postToServer(String api_key, float f1, float f2,
    float f3, float f4); // key per scrivere sul server,
    fornita  dall' Unibo (o da ThingSpeak), temperatura
    DHT, umidita' DHT, temperatura BMP, pressione BMP
```

4. **Sensors**, funzioni per gestire l'inizializzazione e la lettura dei sensori, attraverso l'uso di librerie esterne.

```
void initializeDHT22();
float readDHT22(int sens);// per decidere quale dei due
    valori leggere (temp./umid.)
```

```
bool initializeBMP280();
float readBMP280(int sens);// per decidere quale dei due
    valori leggere (temp./press.)
```

5. **Web Service**, set di funzioni che gestiscono la pagina web; ogni funzione carica una pagina HTML diversa, con del CSS *embedded* e senza alcun linguaggio di *scripting*, e proprio per questo motivo si è cercato di creare/emulare un servizio HTTP *RESTful* (sebbene non sia presente nessun tipo di *scripting*) assegnando ad URI diverse servizi differenti; la *handleRoot()* è quella più importante, poiché è quella che contiene il *form* di registrazione del dispositivo.

```
void handleRoot(); // Home del servizio, contiene il form
     di registrazione
void handlePost(); // Pagina creata al momento di submit
    del form, redirect a handleExit()
void handleReset();// Permette di lanciare una routine
    per resettare le credenziali dalla memoria EEPROM
void handleExit(); // Permette di chiudere il server, la
    modalita' Standalone e di resettare il dispositivo
void handleNotFound(); //Pagina non trovata
void handleIoT(); //Carica una pagina informativa su IoT
void handleTools(); // Carica una pagina che contiene
    pulsanti a servizi (handleReset() e handleExit())
void handleAbout(); // Carica una pagina informativa sul
    progetto
void handleContact(); // Carica una pagina informativa
    sui nostri contatti e link utili
```

6. **Auxiliary** , routines utili e utilizzate più volte

```
int32_t getRSSI(const char* target_ssid); // Legge
    segnale WiFi
void blinkLed(int pin,int repeat);
void blink2Led(int pin1,int pin2,int repeat);
```

4.4 Sviluppo

4.4.1 Ambiente di sviluppo

Il progetto è stato sviluppato e testato interamente attraverso *tools* e *software open-source*:
abbiamo lavorato solo su piattaforme con *Linux* come sistema operativo (Linux Mint 17.3 Rosa-Cinnamon)[3], l'IDE utilizzato è stato*Arduino IDE*[4], e le librerie tutte scaricabili da GitHub.

4.4.2 Librerie utilizzate

```
#include <Arduino.h>
#include <Wire.h>
```

La prima per includere il *core* di Arduino, cioè tutte le *macro*, le *costanti* e i controlli sul microcontrollore (versioni di *Atmel AVR*[5], famiglia di microcontrollori RISC);
la seconda permette di creare il protocollo I^2C su due pin dati in INPUT a piacere.
Link su GitHub[6].

```
#include <ESP8266WiFiMulti.h>
#include <ESP8266HTTPClient.h>
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
#include <WiFiClient.h>
```

Fanno parte del core *ESP8266*, citato precedentemente, forniscono primitive per sfruttare il modulo ESP nell'interfacciarsi alla rete. Inoltre ci hanno permesso di avere dei metodi fondamentali per il conseguimento in toto delle nostre *User Stories*:

[3]https://www.linuxmint.com/download.php
[4]https://www.arduino.cc/en/Main/Software
[5]http://www.atmel.com/products/microcontrollers/avr/
[6]https://github.com/arduino/Arduino/tree/master/hardware/arduino/avr/cores/arduino

```
ESP.getVcc(); //Per leggere la potenza dell'alimentazione
ESP.reset(); //Per resettare il dispositivo
ESP.deepSleep(uint32t timeusec); //Per far dormire il
    dispositivo per timeusec microsecondi
```

Link su GitHub[7].

```
#include <EEPROMforAuth.h>
```

Libreria scritta da noi *ad hoc* per leggere/scrivere le credenziali sulla memoria EEPROM della *board* , basandoci sulla libreria*EEPROM.h* di Arduino.
Link su GitHub[8].

```
#include <DHT.h>
#include <DHT_U.h>
```

Libreria per gestire i sensori DHT (*DHT_Unified* supporta DHT11, DHT21 e DHT22).
Link su GitHub[9].

```
#include <Adafruit_Sensor.h>//
#include <Adafruit_BMP085.h>//
```

Libreria per gestire i sensori BMP (*BMP085* supporta BMP180 e BMP280).
Link su GitHub[10] [11].

4.5 Testing

Il lavoro svolto nella fase di *testing* è stato incentrato nella produzione (manuale e via *software*) di test che emulavano un evento o un comportamento umano.
I primi ad essere stati prodotti sono stati i test di lettura dei sensori, per verificare la reale correttezza delle librerie importate e di queste ultime in concomitanza di altre routines e per assicurarci l'integrità dei nostri sensori.
In seguito, i test per creare un "*timed button*" per far scatenare la routine

[7] https://github.com/esp8266/Arduino/tree/master/libraries
[8] https://github.com/eldiablo100000/EEPROM4Auth
[9] https://github.com/adafruit/DHT-sensor-library
[10] https://github.com/adafruit/Adafruit_Sensor
[11] https://github.com/adafruit/Adafruit-BMP085-Library

di registrazione, che sono consistiti nel *testing* manuale con i *jumper* sulla *breadboard* per la circuiteria, e sempre manuali per assicurare il comportamento del bottone in caso di *deepSleep mode* o meno.
I test per le connessioni, invece, sono stati quelli più complicati da produrre: dopo aver esaminato le 3 modalità che avremmo utilizzato nello sketch singolarmente, cosa semplice visto che le librerie ESP8266* sono tutte corredate da esempi molto snelli da cui attingere, testare l'integrazione dei 3 moduli ha portato qualche conflitto risolvibile unicamente modificando l'ordine d'esecuzione di alcune chiamate a funzione.

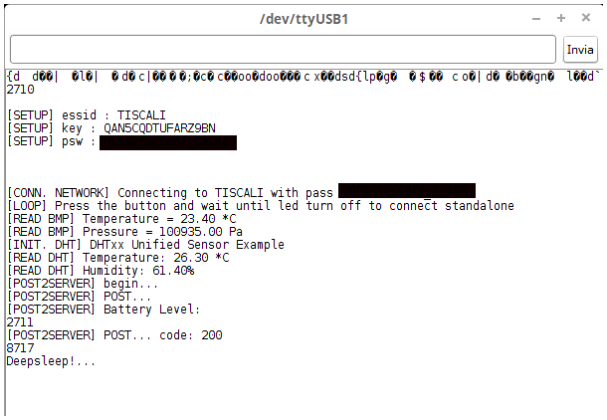


Figura 4.1: *Screenshot* della comunicazione via seriale tra computer (attraverso l'IDE) e la *Weather Station*.

Dispositivi difficili

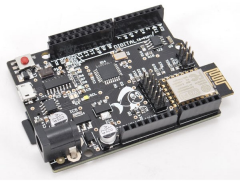
Lo sviluppo e, soprattutto, il *testing* si sono svolti su più dispositivi, diversi in alcune caratteristiche.
I *devices* su cui si è lavorato sono stati: modulo ESP8266-01, modulo ESP8266-12F, Arduino Uno, Fishino Uno, Espduino.

ESP88266-01

Il più semplice della famiglia ESP.
Problema: necessaria espansione per i pin visto che è fornito solo di GPIO0 e GPIO2.

ESP8266-12F

Il modulo più utilizzato, visto che può essere utilizzato senza *board*• supplementari (Vedi Prototipo 3).
Problema: FLASH e RESET necessitano di circuiteria da costruire; per utilizzarlo, inoltre, serve saldarlo.



Fishino UNO

Board italiana nata per unire l'*IoT* con l'idea di "*Plug and Play*" per l'utente con meno conoscenze di programmazione .
Problema: il modulo ESP12 integrato contiene un *firmware* dedicato, quindi zero compatibilità con qualunque altro *sketch* o libreria/*firmware* per ESP8266. Inoltre è caro.

Arduino UNO

La *board* più famosa, contiene già il *core* di Arduino nel *firmware*.
Problema: il meno pratico ed economico, specie se si pensa che è sfornito di ogni modulo di rete.

ESPduino

La board su cui si è svolta la maggiore attività di testing.
Problema: il FLASH dello *sketch* non può essere completato con la *board* (Vedi Prototipo 2) già inserita, poiché per lanciarlo in *UART* mode il RESET non deve essere collegato. Inoltre i moduli ESP non permettono dei periodi di *deepSleep* maggiori di 30 minuti.

4.6 Deployment

4.6.1 Sketch

Lo *sketch* è stato scritto interamente in C++, con parti di codice HTML e CSS, per strutturare il *Web Service*, inserite in variabili, allocate al momento della richiesta al server.
Idealmente, è la naturale implementazione della logica che era stata pensata in fase di Progettazione, non ci sono state modifiche sostanziali nel passaggio dalla fase di Progettazione a quella di *Deployment*.
A differenza degli *sketch* che si trovano spesso online per ESP, il nostro non esegue il "*main*" nella *setup()*, bensì nella *loop()*, perché nel nostro caso più di un' operazione deve essere eseguita solo una volta, quindi il paradigma della programmazione per Arduino ci viene in aiuto differenziando ciò che è "da fare sempre" da ciò che è "da fare una volta".

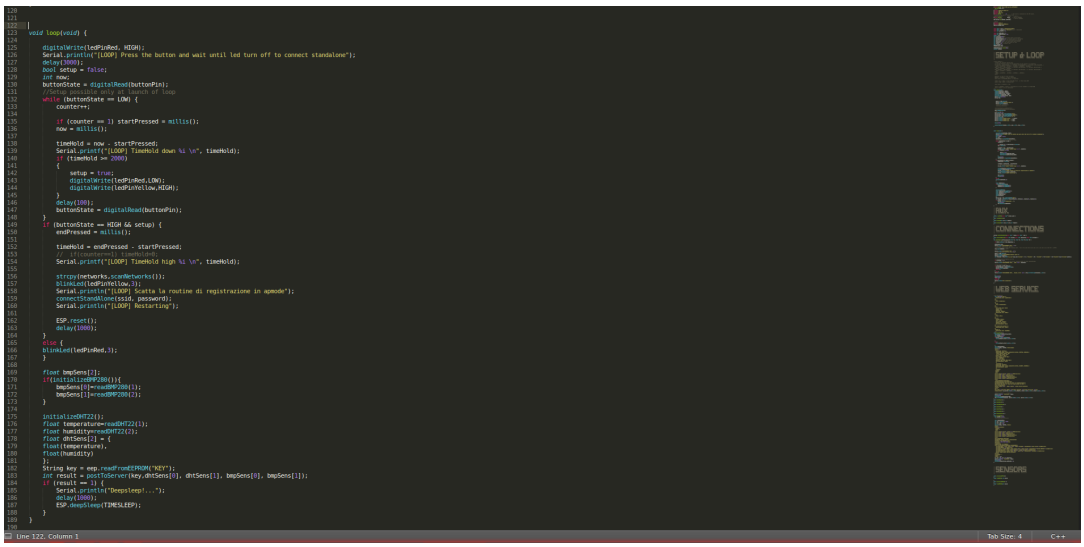


Figura 4.2: Screenshot dello sketch, focus sul loop.

4.6.2 Librerie prodotte

Le librerie prodotte sono 2:

1. **EEPROM4Auth**[12], libreria per scrivere/leggere credenziali sull'EEPROM di *board* Arduino-*like*, e per resettare i contenuti della stessa memoria.

2. **CrowdsensingLibrary**[13], libreria per interfacciarsi su Thingspeak attraverso le API fornite dal *Web Service*; permette di creare un canale e di fare *upload* di dati su un proprio canale (o un canale pubblico).

La prima, *EEPROM4Auth*, è costituita da soli 3 metodi pubblici, basati sugli ultimi aggiornamenti della libreria *EEPROM.h*, che tratta la scrittura come una transazione, quindi con metodi di *commit()*.

```
#define LENGTHEEPROM 512 //in EEPROM4Auth.h per modificare
    la grandezza della memoria
void clearEEPROM(); //Per resettare memoria EEPROM
void writeToEEPROM(String ess,String psw, String key); //
    Per scrivere credenziali nel formato "#myWiFi#myPass#
    myString#0000..."
String readFromEEPROM(String arg1); //Per leggere ess/psw/
    key dall'EEPROM, il parametro puo' variare fra ESSID/
    PSW/KEY
```

La seconda, *CrowdsensingLibrary*, è costituita da 4 metodi pubblici, di cui 2 sono dichiarati con *overloading* per poter supportare un *WiFi Client*, un *Ethernet Client* o un *Fishino Client*.

```
void UpdateChannel(WiFiClient/EthernetClient/FishinoClient
    client, String api_key,String name, float val[],int
    n_fields, String desc, int elevation,float lat, float
    lon, bool public_f, String tags, String url); //Per
    aggiornare canale, avendo l'api_key del canale
```

[12]https://github.com/eldiablo100000/EEPROM4Auth/
[13]https://github.com/eldiablo100000/CrowdsensingLibrary/

```
String CreateChannel(WiFiClient/EthernetClient/
    FishinoClient ,char* api_key,char* name,char** val, int
     n_fields, char* desc, int elevation, float lat, float
    lon, bool public_f, char* tags, char* url); //Per
    creare un nuovo canale, avendo l'api_key MASTER, cioe'
    quella legata all'account ThingSpeak, ritorna l'api_key
     in scrittura relativa al canale creato
String GetKey(); //Per ritrovare la chiave sull'EEPROM, nel
     formato "#myAPIkey"
bool KeyExists();
```

4.6.3 Misurazioni

Per tutta la durata del progetto, le rilevazioni sono state salvate su un canale/*stream* pubblico attraverso un account privato registrato su *ThingSpeak*.
Le misurazioni, inerenti questo ed altri progetti, possono essere trovati su questo canale.
Account: doctor100000
Password: ciaociao
Canale: **ESP-WeatherStation**[14]
Id Canale:180171

[14]https://thingspeak.com/channels/180171

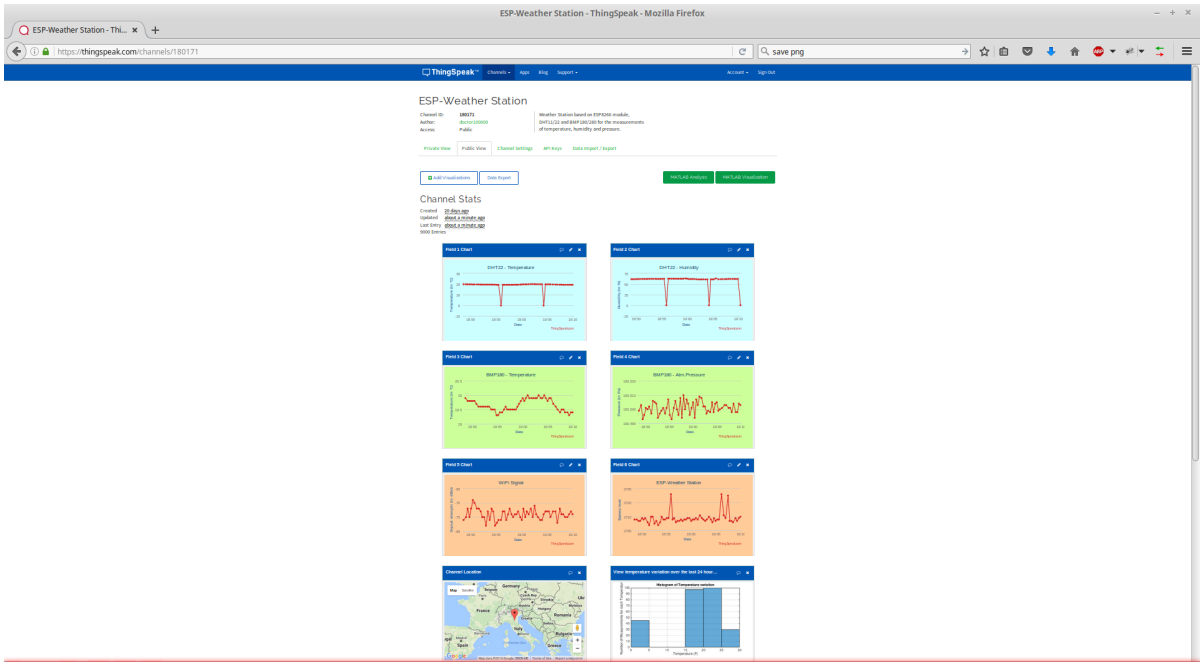


Figura 4.3: *Screenshot* canale ThingSpeak su cui, attualmente, vengono postate le rilevazioni.

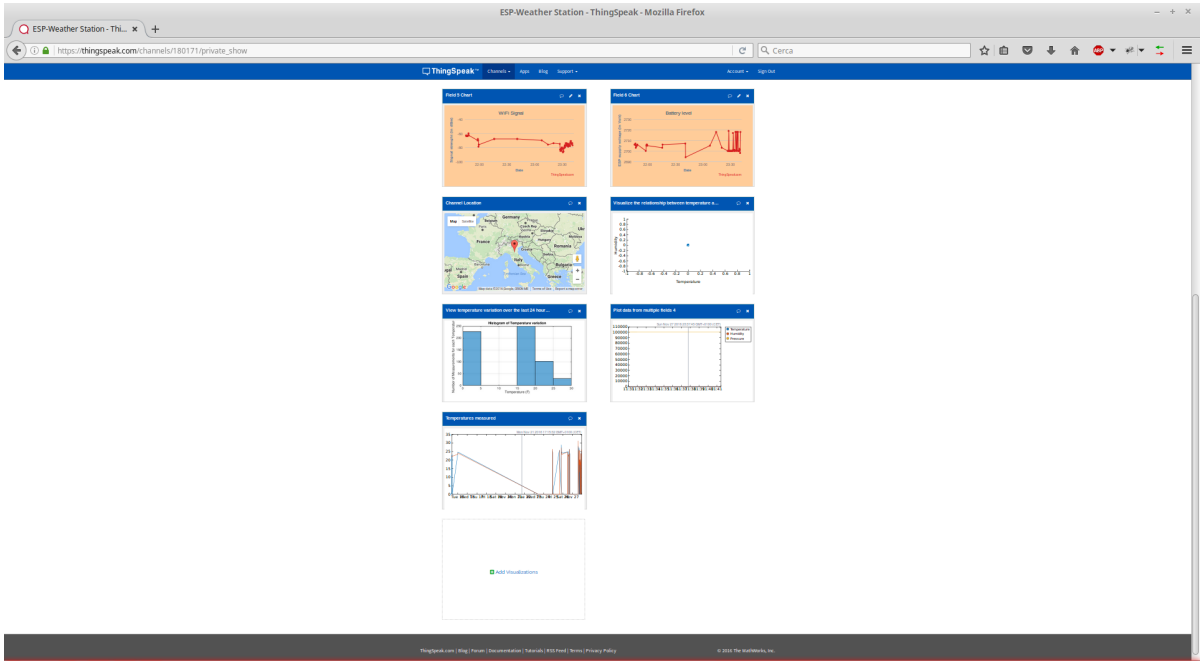


Figura 4.4: *Screenshot* canale ThingSpeak

4.6.4 Web Service

Figura 4.5: *Screenshot* pagina Home.
È la pagina che si presenta al lancio del *Web Service*, e contiene il form di Registrazione della *Weather Station*.
I dati da inserire sono: Nome, ESSID (Nome della rete a cui collegarsi), *Password* (di rete), *Key* (fornita dal proprietario del server su cui scrivere, per ora ThingSpeak).

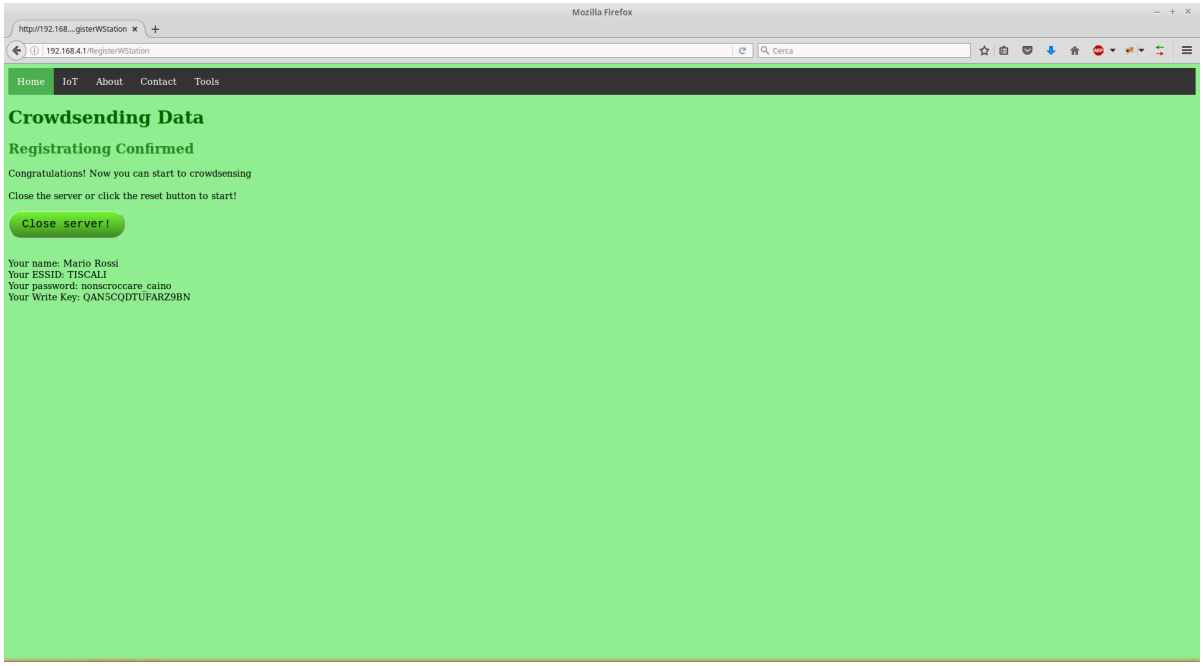


Figura 4.6: Screenshot pagina *RegisterWStation*

Pagina di atterraggio dopo aver sottomesso il form della Home.

Permette di chiudere il server ed iniziare il *Crowdsensing*, o visitare il *WebService* sulle altre pagine/schede.

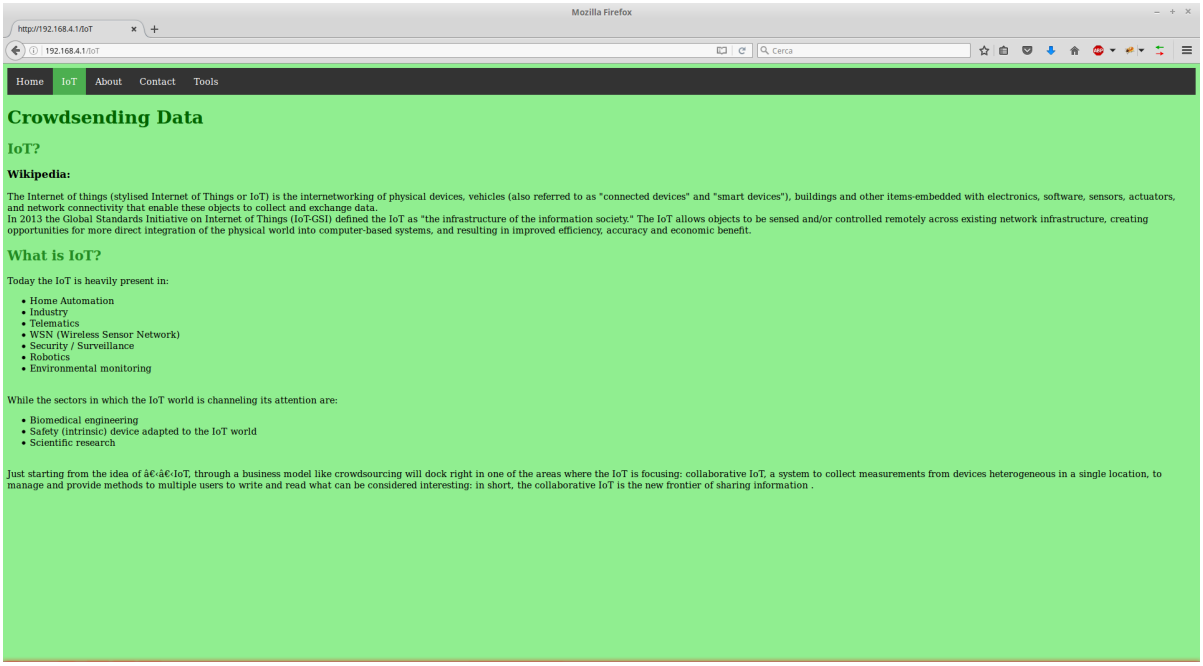


Figura 4.7: *Screenshot* pagina *IoT*
Informazioni sul mondo dell'*Internet of Things*.

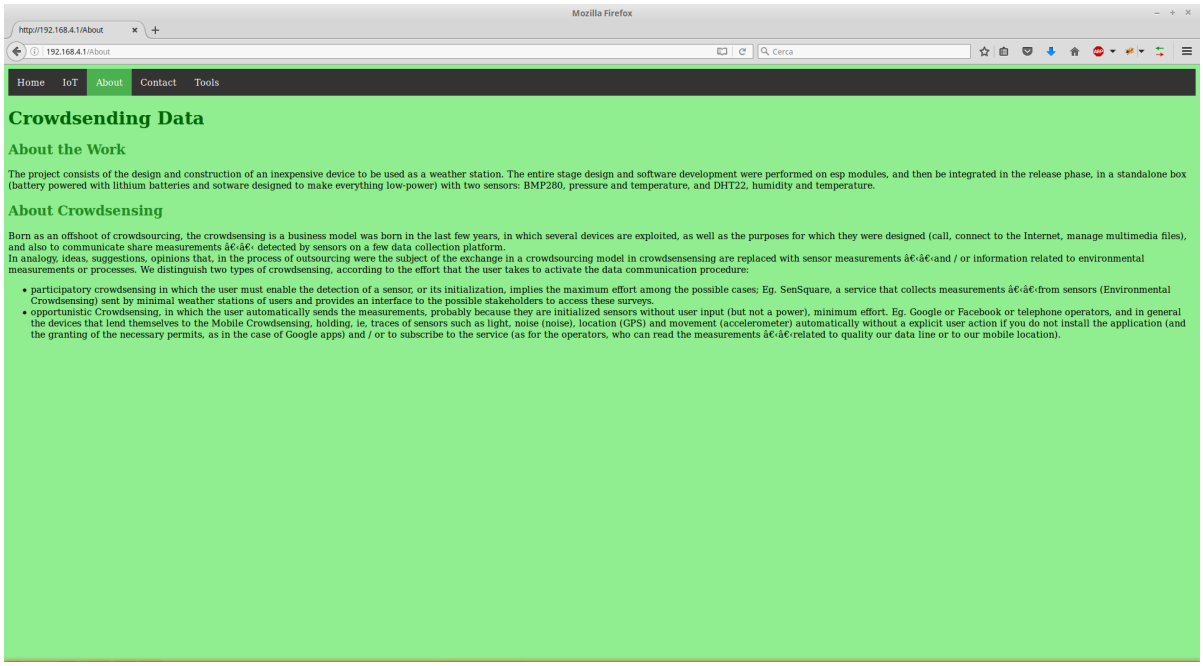


Figura 4.8: *Screenshot* pagina *About*
Informazioni sul progetto e sul*Crowdsensing*.

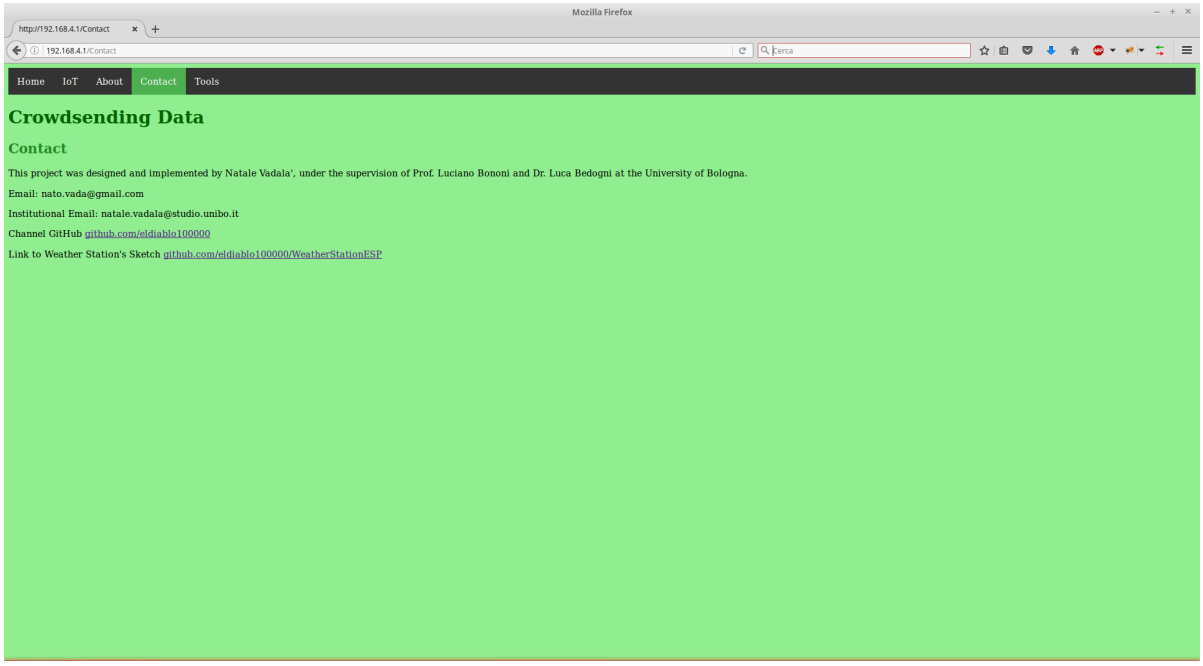


Figura 4.9: *Screenshot* pagina *Contact*

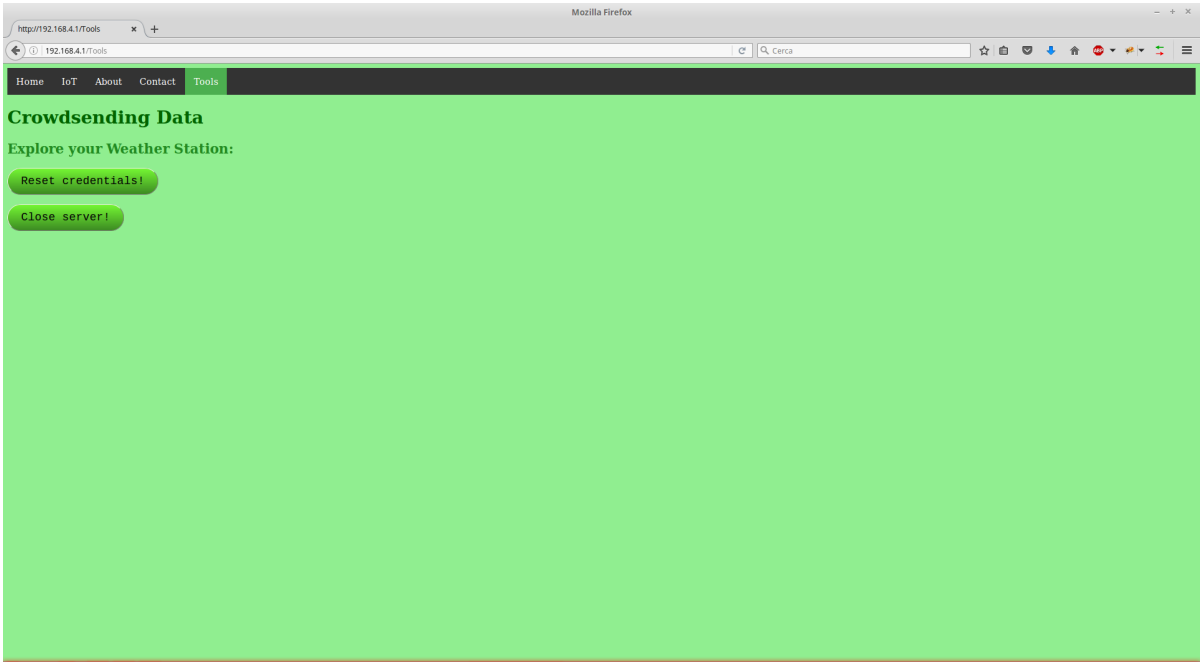


Figura 4.10: *Screenshot* pagina *Tools*

Pagina per resettare le credenziali o chiudere il server del dispositivo.

Conclusioni

Il presente lavoro, com' è evidente, affronta e prova ad analizzare uno scenario pesantemente attivo nell'ultimo decennio, l'*IoT*.
Sono stati analizzati i principi cardine dei modelli di business e sviluppo basati sull'impiego del *crowd* (folla), partendo dal *Crowdsourcing*, per arrivare al *Crowdsensing*, inglobato in un certo senso nel precedente.
In seguito, l'enfasi si è concentrata sul mondo dell'*Internet of Things* e del *Mobile CrowdSensing*, scenari tanto interessanti quanto vasti sia in termini di letteratura, che in termini di possibili sviluppi.
Dopo una prima analisi di ciò che sono questi fenomeni, ci si è concentrati sulla descrizione del progetto che abbiamo portato avanti, e cioè la progettazione e la realizzazione di una *Weather Station*. Questa descrizione è stata divisa in due macro aree di dominio: *Hardware* e *Software*.
Nella parte *Hardware*, oltre ad una esaustiva spiegazione del lavoro svolto per arrivare ai prototipi dianzi descritti, è riportata anche una piccola analisi delle problematiche incontrate durante la fase di creazione del nostro dispositivo, problematiche che si presuppone possano essere comuni a molti che, come chi scrive, non hanno un'esperienza decennale nel mondo dell'elettronica e dei microcontrollori, ma sono neofiti.
Nella parte *Software*, abbiamo descritto come è stato portato avanti lo sviluppo (Metodologia di sviluppo Agile), quali strumenti sono stati utilizzati e, naturalmente, abbiamo dato un feedback riguardo le piattaforme su cui abbiamo testato il nostro software, insieme alla spiegazione della logica e delle *User Stories* prese in considerazione.

Dal lavoro di tesi è emerso che, nonostante la letteratura e la ricerca siano in forte crescita nei settori qui trattati, esistono ostacoli di non poco conto che, di certo, non aiutano il neofita a misurarsi con questo mondo, né incentivano lo sviluppo da parte di chi già ci lavora. Questi ostacoli sono: l'assenza (visto il mercato giovane) di standard che regolino misurazioni e protocolli da utilizzare; l'assenza di banche dati pubbliche contenenti canali *reliable* (*ThingSpeak* e *SparkFun*, per uno *stakeholder*, lasciano a desiderare, poiché centinai di fattori possono rendere il canale di un utente privato non interessante, specie se l'utente non è a conoscenza di essere seguito da qualcuno che fa fede sulle sue misurazioni); l'assenza di *security* adatta, sia per la mancanza di protocolli, sia perché viene soppressa nel momento in cui altre funzioni hanno la priorità di essere memorizzate in un dispositivo piccolo come modulo ESP (con 1-4 MB di Memoria Flash); ciò rende il settore dell'IoT ancora poco affidabile; la difficoltà nel reperire alcune componenti, come le batterie a Litio.
Il presente progetto, più che un prodotto finito (nonostante lo sia), vuole essere inteso come uno studio di fattibilità, destinato ad Università, aziende o *stakeholder* che hanno interesse nel creare delle *WeatherStation* (piuttosto che *WiFiSignalStation*, *EarthQuakeStation*, *NoisePollutionStation*) a prezzi minimi da fornire alla gente per campagne di *Crowdsensing*, semplicemente con una minima (e ovvia) modifica all'*hardware* (basta cambiare sensori) e al *software* per adattarlo alle proprie esigenze.
Proprio per questo, il progetto presentato vuole essere una sorta di guida per creare piccoli ed economici dispositivi che comunichino ad server centrale delle informazioni con una certa frequenza, a prescindere dai sensori utilizzati.
Naturalmente, come ogni cosa (specie nel mondo *software*), anche il presente progetto è sicuramente migliorabile, da chi scrive o da chi se ne occuperà in futuro, infatti il *software* e lo schema sono disponibili già su GitHub[15] e in Appendice.

[15]https://github.com/eldiablo100000/WeatherStationESP

Appendice A

Sketch

WeatherStationESP.ino

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
#include <EEPROM.h>
#include <Arduino.h>
#include <ESP8266WiFiMulti.h>
#include <ESP8266HTTPClient.h>
#include <EEPROMforAuth.h>
#define NMINSLEEP 30
//#define TIMESLEEP 1000000 * 5 // in useconds
#define TIMESLEEP 1000000 * 60 * NMINSLEEP
#define SERVER "http://184.106.153.149/update"
ADC_MODE(ADC_VCC); //To read Voltage
//DHT22
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>
#define DHTPIN          12          // Pin which is connected
    to the DHT sensor.
// Uncomment the type of sensor in use:
//#define DHTTYPE           DHT11     // DHT 11
#define DHTTYPE           DHT22     // DHT 22 (AM2302)
```

```
//#define DHTTYPE           DHT21     // DHT 21 (AM2301)
DHT_Unified dht(DHTPIN, DHTTYPE);

//BMP280
#include <Wire.h>
#include <Adafruit_BMP085.h>
Adafruit_BMP085 bmp;


const char * ssid = "WeatherStationESP";
const char * password = "helloiot";//min. 8 characters
const int buttonPin = 2;

const int ledPinRed = 5;
const int ledPinYellow = 13;
int32_t rssi;
char networks[500]="";
int buttonState = 0; // current state of the button
int lastButtonState = 0; // previous state of the button
int startPressed = 0; // the time button was pressed
int endPressed = 0; // the time button was released
int timeHold = 0; // the time button is hold
int counter = 0; // counter loop
bool closeAP = 0;
EEPROMforAuth eep;
//ESP8266WiFiMulti WiFiMulti;
ESP8266WebServer server(80);
uint32_t getVcc;
//*********************************************
//*********************************************
//SETUP AND LOOP*****************************
//*********************************************
//*********************************************
void setup(void) {
  pinMode(buttonPin, INPUT);
  pinMode(ledPinRed, OUTPUT);
  pinMode(ledPinYellow, OUTPUT);
  digitalWrite(ledPinRed, LOW);
```

```
  digitalWrite(ledPinYellow, LOW);
  Serial.begin(115200);
  String res;


    getVcc = ESP.getVcc();
  Serial.println("Battery Level:");
  Serial.println(getVcc);
  eep= EEPROMforAuth();
  //SDA 0, SCL 14
  Wire.begin(0,14);
  String psw = eep.readFromEEPROM("PSW");
  String essid = eep.readFromEEPROM("ESSID");
  String key = eep.readFromEEPROM("KEY");
  Serial.println();
  Serial.println("[SETUP] essid : " + essid);
  Serial.println("[SETUP] key : " + key);
  Serial.println("[SETUP] psw : " + psw);
  delay(1000);

  connectToNetwork(essid.c_str(), psw.c_str(), key.c_str())
     ;
}
void loop(void) {

  digitalWrite(ledPinRed, HIGH);
  Serial.println("[LOOP] Press the button and wait until
     led turn off to connect standalone");
  delay(3000);
  bool setup = false;
  int now;
  buttonState = digitalRead(buttonPin);
  //Setup possible only at launch of loop
  while (buttonState == LOW) {
  counter++;

  if (counter == 1) startPressed = millis();
  now = millis();
```

```
timeHold = now - startPressed;
Serial.printf("[LOOP] TimeHold down %i \n", timeHold);
  if (timeHold >= 2000) {
  setup = true;
  digitalWrite(ledPinRed,LOW);
  digitalWrite(ledPinYellow,HIGH);

  }
delay(100);
buttonState = digitalRead(buttonPin);

}
if (buttonState == HIGH && setup) {
endPressed = millis();

timeHold = endPressed - startPressed;
//  if(counter==1) timeHold=0;
Serial.printf("[LOOP] TimeHold high %i \n", timeHold);

strcpy(networks,scanNetworks());
blinkLed(ledPinYellow,3);
Serial.println("[LOOP] Scatta la routine di registrazione
    in apmode");
launchStandAlone(ssid, password);
Serial.println("[LOOP] Restarting");

ESP.reset();
delay(1000);



}else {
//startPressed = millis();

blinkLed(ledPinRed,3);

}
```

```
   float bmpSens[2];
  if(initializeBMP280()){
    bmpSens[0]=readBMP280(1);
    bmpSens[1]=readBMP280(2);

  }


  initializeDHT22();
  float temperature=readDHT22(1);
  float humidity=readDHT22(2);

  float dhtSens[2] = {
  float(temperature),
  float(humidity)
  };


  String key = eep.readFromEEPROM("KEY");
  int result = postToServer(key,dhtSens[0], dhtSens[1],
     bmpSens[0], bmpSens[1]);
  if (result == 1) {
  Serial.println("Deepsleep!...");
  delay(1000);
  ESP.deepSleep(TIMESLEEP);
  }
}



//***********************************************
//***********************************************
//AUX*****************************************
//***********************************************
//***********************************************
int32_t getRSSI(const char* target_ssid) {
  WiFi.mode(WIFI_STA);
```

```
 WiFi.disconnect();
  byte available_networks = WiFi.scanNetworks();

  for (int network = 0; network < available_networks;
     network++) {
    if (strcmp(WiFi.SSID(network).c_str(), target_ssid) ==
       0) {

      return WiFi.RSSI(network);
    }
  }
  return 0;
}
char* scanNetworks(){
  char networks[500];
  digitalWrite(ledPinRed, HIGH);
  digitalWrite(ledPinYellow, HIGH);
  Serial.println("[SCAN] Scan start");

  // WiFi.scanNetworks will return the number of networks
     found
  WiFi.mode(WIFI_STA);
 WiFi.disconnect();
  int n = WiFi.scanNetworks();
  Serial.println("[SCAN] Scan done");
  if (n == 0)
    Serial.println("[SCAN] No networks found");
  else
  {
    Serial.print(n);
    Serial.println(" networks found");
    strcpy(networks,"<select name=essid>");
    for (int i = 0; i < n; ++i)
    {
      // Print SSID and RSSI for each network found
      Serial.print(i + 1);
      Serial.print(": ");
      Serial.print(WiFi.SSID(i));
```

```
      Serial.print(" (");
      Serial.print(WiFi.RSSI(i));
      Serial.print(")");
      Serial.println((WiFi.encryptionType(i) ==
         ENC_TYPE_NONE)?" ":"*");
  strcat(networks,"<option value=\"");

  strcat(networks,WiFi.SSID(i).c_str());
  strcat(networks,"\">");
  strcat(networks,WiFi.SSID(i).c_str());
  strcat(networks,"</option>");

      delay(10);
    }
    strcat(networks,"</select>");
  }
  Serial.println("");
  return networks;
}
void blinkLed(int pin,int repeat){
  if(repeat>0){
  for(int i=0;i<repeat;i++){
  digitalWrite(pin,LOW);
  delay(500);
  digitalWrite(pin, HIGH);
  delay(500);
  digitalWrite(pin, LOW);
  delay(500);

  }
  }
}
void blink2Led(int pin1,int pin2,int repeat){
  if(repeat>0){
  for(int i=0;i<repeat;i++){
  digitalWrite(pin1,LOW);
  digitalWrite(pin2,LOW);
  delay(500);
```

```
  digitalWrite(pin1, HIGH);
  digitalWrite(pin2,HIGH);
  delay(500);
  digitalWrite(pin1, LOW);
  digitalWrite(pin2,LOW);
  delay(500);

  }
  }
}



//*********************************************
//*********************************************
//CONNECTIONS**********************************
//*********************************************
//*********************************************

String launchStandAlone(const char * essid,const char * pwd
    ) {
     closeAP=0;

  WiFi.softAP(essid, pwd);
  Serial.println();

  // Wait for connection

  Serial.println();
  Serial.print("[CONN. STANDALONE] Connected to ");
  Serial.println(essid);
  Serial.print("[CONN. STANDALONE] IP address: ");
  Serial.println(WiFi.softAPIP());

  if (MDNS.begin("esp8266")) {
    Serial.println("MDNS responder started");
  }
```

```
  server.on("/", handleRoot);
  server.on("/RegisterWStation", handlePost);
  server.on("/IoT", handleIoT);
  server.on("/About", handleAbout);
  server.on("/Contact", handleContact);
  server.on("/Tools", handleTools);
  server.on("/Exit", handleExit);
  server.on("/Reset", handleReset);

  server.onNotFound(handleNotFound);
  server.begin();

  Serial.println("[CONN. STANDALONE] HTTP server started");
  while (true&&!closeAP) {
    server.handleClient();

  }
  Serial.println("[CONN. STANDALONE] Server closed!Exit
     !...");

  char buf[250];
   char network[68];
  if(server.arg(2).length()>0){
    strcpy(network,server.arg(2).c_str());

  }
  else{
    strcpy(network,server.arg(1).c_str());

  }
  sprintf(buf, "Fullname:%s, ESSID:%s, Password:%s, Write
     Key:%s", server.arg(0).c_str(), network, server.arg
     (3).c_str(), server.arg(4).c_str());
  WiFi.disconnect();

  return buf;
}
```

```
void connectToNetwork(const char essid[],const char
    password[],const char string[]) {

  Serial.println();
  Serial.println();
  Serial.println();
  Serial.printf("[CONN. NETWORK] Connecting to %s with pass
      %s\n", essid, password);
  rssi = getRSSI(essid);

    WiFi.begin(essid,password);
 // WiFiMulti.addAP(ssid, password);
}
int postToServer(String api_key,float f1, float f2, float
   f3,float f4) {
  // wait for WiFi connection
  if ((WiFi.status() == WL_CONNECTED)) {

  HTTPClient http;
  Serial.print("[POST2SERVER] begin...\n");
  // configure traged server and url
  //http.begin("https://192.168.1.12/test.html", "7a 9c f4
     db 40 d3 62 5a 6e 21 bc 5c cc 66 c8 3e a1 45 59 38");
      //HTTPS
  http.begin(SERVER); //HTTP

  Serial.print("[POST2SERVER] POST...\n");
  // start connection and send HTTP header
  getVcc = ESP.getVcc();
  Serial.println("[POST2SERVER] Battery Level:");
  Serial.println(getVcc);
  int httpCode = http.POST("api_key="+api_key+"&field1=" +
     f1 + "&field2=" + f2 + "&field3=" + f3+"&field4=" +
     f4+"&field5="+rssi+"&field6="+getVcc);

  // httpCode will be negative on error
  if (httpCode > 0) {
```

```
  // HTTP header has been send and Server response header
      has been handledPinRed
  Serial.printf("[POST2SERVER] POST... code: %d\n",
      httpCode);

  // file found at server
  if (httpCode == HTTP_CODE_OK) {
    String payload = http.getString();
    Serial.println(payload);
    return 1;
  }
  } else {
  Serial.printf("[POST2SERVER] POST... failed, error: %s\n"
      , http.errorToString(httpCode).c_str());

  }
  delay(5000);
  http.end();
  return 0;
  } else {
  Serial.println("Not connected");
  }
}



//************************************************
//************************************************
//WEB SERVICE************************************
//************************************************
//************************************************
//CSS DECLARATION
  char css[540]="body {\
    background-color: LightGreen;\
  }\
  h1 {\
    color: DarkGreen;\
  }\
```

```
   h2 {\
     color: ForestGreen;\
   }\
   ul.bar {\
     list-style-type: none;\
     margin: 0;\
     padding: 0;\
     overflow: hidden;\
     background-color: #333;\
   }\
   li.bar {\
     float: left;\
   }\
   li.bar a {\
     display: block;\
     color: white;\
     text-align: center;\
     padding: 14px 16px;\
     text-decoration: none;\
   }\
   li.bar a:hover:not(.active) {\
     background-color: #111;\
   }\
   .active {\
     background-color: #4CAF50;\
   }";
void handlePost() {
   digitalWrite(ledPinYellow,HIGH);
   int lenHTML = 2500;
   char network[68];
   if(server.arg(2).length()>0){
     strcpy(network,server.arg(2).c_str());

   }
   else{
     strcpy(network,server.arg(1).c_str());

   }
```

```
char temp[lenHTML];
snprintf(temp, lenHTML,"<html><head>\
<style>%s\
.btn {\
  background: #75f734;\
  background-image: linear-gradient(to bottom, #75f734,
     #3d8c23);\
  -webkit-border-radius: 30;\
  -moz-border-radius: 30;\
  border-radius: 30px;\
  font-family: Courier New;\
  color: #040504;\
  font-size: 20px;\
  padding: 10px 20px 10px 20px;\
  text-decoration: none;\
  }\
  .btn:hover {\
  background: #3cb0fd;\
  background-image: linear-gradient(to bottom, #3cb0fd,
     #3498db);\
  text-decoration: none;\
  }\
  </style>\
 </head>\
 <body>\
 <ul class=\"bar\">\
<li class=\"bar\"><a class=\"active\" href=\"/\">Home</a
   ></li>\
<li class=\"bar\"><a href=\"/IoT\" >IoT</a></li>\
<li class=\"bar\"><a href=\"/About\">About</a></li>\
<li class=\"bar\"><a href=\"/Contact\">Contact</a></li>\
<li class=\"bar\"><a href=\"/Tools\">Tools</a></li>\
</ul>\
 <h1>Crowdsending Data</h1>\
<h2>Registrationg Confirmed</h2>\
<p>Congratulations! Now you can start to crowdsensing<br
   />\
```

```
  <br/>Close the server or click the reset button to start!
      \
  <form action=/Exit>\
  <button class=\"btn\"  type=\"submit\" >Close server!</
     button>\
  </form>\
  <br/>\
  Your name: %s<br/>Your ESSID: %s<br/>Your password: %s<br
     />Your Write Key: %s</p>\
  </body></html>",css,server.arg(0).c_str(),network, server
     .arg(3).c_str(), server.arg(4).c_str());

  //Serial.println("----->"+out);
  server.send(200, "text/html", temp);
  delay(1000);
  digitalWrite(ledPinYellow,LOW);
  eep.writeToEEPROM(network, server.arg(3).c_str(), server.
     arg(4).c_str());
strcpy(temp,"");
}
void handleReset() {
  blinkLed(ledPinRed,3);
  String temp;
  temp="<html><head>\
  <style>";
  temp+=css;
  temp+="</style>\
   </head>\
   <body>\
  <ul class=\"bar\">\
  <li class=\"bar\"><a href=\"/\">Home</a></li>\
  <li class=\"bar\"><a href=\"/IoT\" >IoT</a></li>\
  <li class=\"bar\"><a href=\"/About\">About</a></li>\
  <li class=\"bar\"><a href=\"/Contact\">Contact</a></li>\
  <li class=\"bar\"><a class=\"active\" href=\"/Tools\">
     Tools</a></li>\
  </ul>\
   <h1>Crowdsending Data</h1>\
```

```
  <h2>Resetting</h2>\
  <p>Your Weather Station was cleared from every old data.\
  <br/>Click Home to register from zero.</p>\
  </body></html>";

  Serial.println("----->Reset");
  server.send(200, "text/html", temp);
  eep.clearEEPROM();
}
void handleExit() {
  blinkLed(ledPinRed,3);
  String temp;
  temp="<html><head>\
  <style>";
  temp+=css;
  temp+="</style>\
   </head>\
   <body>\
  <ul class=\"bar\">\
  <li class=\"bar\"><a href=\"/\">Home</a></li>\
  <li class=\"bar\"><a href=\"/IoT\" >IoT</a></li>\
  <li class=\"bar\"><a href=\"/About\">About</a></li>\
  <li class=\"bar\"><a href=\"/Contact\">Contact</a></li>\
  <li class=\"bar\"><a class=\"active\" href=\"/Tools\">
     Tools</a></li>\
  </ul>\
   <h1>Crowdsending Data</h1>\
  <h2>Stopping server</h2>\
  <p>Your AccessPoint was closed: <br/> you can start
     crowdsensing\
  (if you registred your Weather Station) <br/> you must
     reset the Weather Station otherwise.</p>\
  </body></html>";
  Serial.println("----->Exit");
  server.send(200, "text/html", temp);
  WiFi.disconnect();
  server.close();
  closeAP=1;
```

```
}
void handleNotFound() {
  digitalWrite(ledPinRed, HIGH);
  String message = "File Not Found\n\n";
  message += "URI: ";
  message += server.uri();
  message += "\nMethod: ";
  message += (server.method() == HTTP_GET) ? "GET" : "POST"
      ;
  message += "\nArguments: ";
  message += server.args();
  message += "\n";
  for (uint8_t i = 0; i < server.args(); i++) {
  message += " " + server.argName(i) + ": " + server.arg(i)
      + "\n";
  }
  server.send(404, "text/plain", message);
  digitalWrite(ledPinRed, LOW);
}
void handleIoT() {
  digitalWrite(ledPinYellow, HIGH);
  int lenHTML = 2900;
  String temp;
  temp="<html>\
  <head>\
  <style>";
  temp+=css;
  temp+="</style>\
   </head>\
   <body>  \
   <ul class=\"bar\">\
  <li class=\"bar\"><a href=\"/\">Home</a></li>\
  <li class=\"bar\"><a class=\"active\" href=\"/IoT\" >IoT
     </a></li>\
  <li class=\"bar\"><a href=\"/About\">About</a></li>\
  <li class=\"bar\"><a href=\"/Contact\">Contact</a></li>\
  <li class=\"bar\"><a href=\"/Tools\">Tools</a></li>\
  </ul>\
```

```
<h1>Crowdsending Data</h1>\
<h2>IoT?</h2>\
<h3>Wikipedia:</h3>\
<p>The Internet of things (stylised Internet of Things or
    IoT) is the \
internetworking of physical devices, vehicles (also
   referred to as \"connected devices\" and \"smart
   devices\"),\
 buildings and other items-embedded with electronics,
    software, sensors, actuators, and network \
 connectivity that enable these objects to collect and
    exchange data. <br/>In 2013 the Global Standards \
 Initiative on Internet of Things (IoT-GSI) defined the
    IoT as \"the infrastructure of the information
    society.\"\
 The IoT allows objects to be sensed  and/or controlled
    remotely across existing network infrastructure,\
 creating opportunities for more direct integration of
    the physical world into computer\-based systems, \
 and resulting in improved efficiency, accuracy and
    economic benefit.</p>\
<h2>What is IoT?</h2>\
   <p>\
Today the IoT is heavily present in:\
<ul>\
 <li>Home Automation</li>\
 <li>Industry</li>\
 <li>Telematics</li>\
 <li>WSN (Wireless Sensor Network)</li>\
 <li>Security / Surveillance</li>\
 <li>Robotics</li>\
 <li>Environmental monitoring</li>\
</ul>\
<br/>While the sectors in which the IoT world is
   channeling its attention are:\
<ul>\
 <li>Biomedical engineering</li>\
```

```
    <li>Safety (intrinsic) device adapted to the IoT world</
       li>\
    <li>Scientific research</li>\
   </ul>\
   <br/ >Just starting from the idea of a business model
      like crowdsourcing will dock right/
    in one of the areas where the IoT is focusing:
       collaborative IoT, a system to collect measurements\
     from devices heterogeneous  in a single location, to
        manage and provide methods to multiple\
      users to write and read what can be considered
         interesting: in short, the collaborative IoT is\
       the new frontier of sharing information .</p>\
     </body> \
   </html>";

   server.send(200, "text/html",temp);
   delay(5000);
   digitalWrite(ledPinYellow, LOW);
   //free(temp);
}
void handleTools() {
   digitalWrite(ledPinYellow, HIGH);
   String temp;
   temp="<html>\
   <head>\
   <style>";
   temp+=css;
   temp+="\
   .btn {\
     background: #75f734;\
     background-image: -webkit-linear-gradient(top, #75f734,
         #3d8c23);\
     background-image: -moz-linear-gradient(top, #75f734, #3
        d8c23);\
     background-image: -ms-linear-gradient(top, #75f734, #3
        d8c23);\
```

```
    background-image: -o-linear-gradient(top, #75f734, #3
        d8c23);\
    background-image: linear-gradient(to bottom, #75f734,
        #3d8c23);\
    -webkit-border-radius: 30;\
    -moz-border-radius: 30;\
    border-radius: 30px;\
    font-family: Courier New;\
    color: #040504;\
    font-size: 20px;\
    padding: 10px 20px 10px 20px;\
    text-decoration: none;\
    }\
    .btn:hover {\
    background: #3cb0fd;\
    background-image: -webkit-linear-gradient(top, #3cb0fd,
         #3498db);\
    background-image: -moz-linear-gradient(top, #3cb0fd,
        #3498db);\
    background-image: -ms-linear-gradient(top, #3cb0fd,
        #3498db);\
    background-image: -o-linear-gradient(top, #3cb0fd,
        #3498db);\
    background-image: linear-gradient(to bottom, #3cb0fd,
        #3498db);\
    text-decoration: none;\
    }\
  </style>\
   </head>\
   <body>  \
   <ul class=\"bar\">\
  <li class=\"bar\"><a href=\"/\">Home</a></li>\
  <li class=\"bar\"><a href=\"/IoT\" >IoT</a></li>\
  <li class=\"bar\"><a href=\"/About\">About</a></li>\
  <li class=\"bar\"><a href=\"/Contact\">Contact</a></li>\
  <li class=\"bar\"><a class=\"active\" href=\"/Tools\">
      Tools</a></li>\
  </ul>\
```

```
  <h1>Crowdsending Data</h1>\
  <h2>Explore your Weather Station: </h2>\
  <form action=/Reset>\
  <button class=\"btn\"  type=\"submit\" >Reset credentials
     !</button>\
  </form>\
  <form action=/Exit>\
  <button class=\"btn\"  type=\"submit\" >Close server!</
     button>\
  </form>\
  </body> \
  </html> ";
  server.send(200, "text/html", temp);
  delay(5000);
  digitalWrite(ledPinYellow, LOW);
}
void handleAbout() {
  digitalWrite(ledPinYellow, HIGH);
  String temp;
  temp="<html>\
  <head>\
  <style>";
  temp+=css;
  temp+="</style>\
   </head>\
   <body>  \
   <ul class=\"bar\">\
  <li class=\"bar\"><a href=\"/\">Home</a></li>\
  <li class=\"bar\"><a  href=\"/IoT\" >IoT</a></li>\
  <li class=\"bar\"><a class=\"active\" href=\"/About\">
     About</a></li>\
  <li class=\"bar\"><a href=\"/Contact\">Contact</a></li>\
  <li class=\"bar\"><a href=\"/Tools\">Tools</a></li>\
  </ul>\
  <h1>Crowdsending Data</h1>\
  <h2>About the Work</h2>\
  <p>The project consists of the design and construction of
      an inexpensive device to be used as a weather
```

```
    station.\
  The entire stage design and software development were
    performed on esp modules, \
  and then be integrated in the release phase, in a
    standalone box (battery powered \
  with lithium batteries and sotware designed to make
    everything low-power) with \
  two sensors: BMP280, pressure and temperature, and DHT22,
     humidity and temperature.\
  <br/></p>\
  <h2>About Crowdsensing</h2>\
  <p>Born as an offshoot of crowdsourcing, the crowdsensing
     is a business model was born in the last few \
  years, in which several devices are exploited, as well as
     the purposes for which they were designed.\
 <br/>\
  We distinguish two types of crowdsensing, according to
    the effort that the user takes to activate \
  the data communication procedure:\
  <ul>\
  <li>participatory crowdsensing in which the user must
    enable the detection of a sensor, or its
    initialization, implies the maximum effort among the
    possible cases;\
  Eg. SenSquare, a service that collects data (
    Environmental Crowdsensing) sent by minimal \
  weather stations of users and provides an interface to
    the possible stakeholders to access these surveys.</
    li>\
   <li>opportunistic Crowdsensing, in which the user
     automatically sends the measurements, probably
     because they\
    are initialized sensors without user input (but not a
      power), minimum effort.\
  Eg. Google or Facebook or telephone operators, and in
    general the devices that lend themselves to the \
  Mobile Crowdsensing, holding, ie, traces of sensors such
    as light, noise (noise), location (GPS) and \
```

```
  movement (accelerometer) automatically without a explicit
      user action if you do not install the \
  application (and the granting of the necessary permits,
     as in the case of Google apps) and / or to \
  subscribe to the service .</li></ul></p>\
  </body> \
  </html>";
  server.send(200, "text/html", temp);
  delay(5000);
  digitalWrite(ledPinYellow, LOW);
}
void handleContact() {
  digitalWrite(ledPinYellow, HIGH);
 String temp;
 temp="<html>\
  <head>\
  <style>";
  temp+=css;
  temp+="</style>\
   </head>\
   <body>  \
   <ul class=\"bar\">\
  <li class=\"bar\"><a href=\"/\">Home</a></li>\
  <li class=\"bar\"><a href=\"/IoT\" >IoT</a></li>\
  <li class=\"bar\"><a href=\"/About\">About</a></li>\
  <li class=\"bar\"><a class=\"active\" href=\"/Contact\">
     Contact</a></li>\
  <li class=\"bar\"><a href=\"/Tools\">Tools</a></li>\
  </ul>\
  <h1>Crowdsending Data</h1>\
  <h2>Contact</h2>\
  <p>This project was designed and implemented by Natale
     Vadala', under the supervision of Prof. Luciano
     Bononi and Dr. Luca Bedogni at the University of
     Bologna.</p>\
    <p><a href=\"mailto:nato.vadagmail.com">Email</a></p> <p><a
       href=mailto:natale.vadalastudio.unibo.it\">Institutional
        Email</a></p>\
```

```
  <p>Channel GitHub <a href=\"https://github.com/
     eldiablo100000\"> github.com/eldiablo100000</a></p>\
  <p>Link to Weather Station's Sketch <a href=\"https://
     github.com/eldiablo100000/WeatherStationESP\"> github
     .com/eldiablo100000/WeatherStationESP</a> </p>\
  </body> \
  </html> ";
  server.send(200, "text/html", temp);
  delay(5000);

  digitalWrite(ledPinYellow, LOW);
}
void handleRoot() {
  int lenHTML = 2000;
  //blinkLed(ledPinYellow,2);
  String temp;

  int sec = millis() / 1000;
  int min = sec / 60;
  int hr = min / 60;
  temp="<html>\
  <head>\
  <style>";
  temp+=css;
  temp+="</style>\
   </head>\
   <body>  \
   <ul class=\"bar\">\
  <li class=\"bar\"><a class=\"active\" href=\"/\">Home</a
     ></li>\
  <li class=\"bar\"><a href=\"/IoT\" >IoT</a></li>\
  <li class=\"bar\"><a href=\"/About\">About</a></li>\
  <li class=\"bar\"><a href=\"/Contact\">Contact</a></li>\
  <li class=\"bar\"><a href=\"/Tools\">Tools</a></li>\
  </ul>\
  <h1>Crowdsending Data</h1>\
  <h2>Weather Station based on Arduino</h2>\
  <h2>Hello from ESP8266!</h2>\
```

```
  <p>Uptime: ";
  temp+=hr;
  temp+=":";
  temp+=min%60;
  temp+=":";
  temp+=sec%60;
    temp+="</p>\
  <fieldset>\
  <legend>Sign in:</legend>\
  <form action=/RegisterWStation method=POST>\
    <p><label>Name: <input type=\"text\" name=\"fullname\"
       placeholder=\"Alan Turing\"></label></p>\
    <p><label>ESSID: ";
    temp+=networks;
    temp+="</label></p>\
    <p><label> or hidden ESSID: <input type=\"text\" name
       =\"essid\" placeholder=\"Fastweb-9876527\"></label
       ></p>\
    <p><label>Password: <input type=\"password\" name=\"
       password\"></label></p>\
    <p><label>Write Key: <input type=\"text\" name=\"key\"
       placeholder=\"lachiave\"></label></p>\
    <button type="submit">Write!</button>\
  </form>\
  </fieldset>  \
  </body> \
  </html> ";
  server.send(200, "text/html", temp);
  delay(5000);
  blink2Led(ledPinYellow,ledPinRed, 3);
}



//************************************************
//************************************************
//SENSORS*************************************
//************************************************
```

```
//*****************************************
void initializeDHT22(){
  // Initialize device.

  dht.begin();
  Serial.println("[INIT. DHT] DHTxx Unified Sensor Example"
      );
  // Print temperature sensor details.
  sensor_t sensor;
  dht.temperature().getSensor(&sensor);
  //Serial.println("------------------------------------");
  //Serial.println("Temperature");
  //Serial.print  ("[INIT. DHT] Sensor:       ");
  //Serial.println(sensor.name);
  //Serial.print  ("[INIT. DHT] Driver Ver:   ");
  //Serial.println(sensor.version);
  //Serial.print  ("[INIT. DHT] Unique ID:    ");
  //Serial.println(sensor.sensor_id);
  //Serial.print  ("[INIT. DHT] Max Value:    ");
  //Serial.print(sensor.max_value);
  //Serial.println(" *C");
  //Serial.print  ("[INIT. DHT] Min Value:    ");
  //Serial.print(sensor.min_value);
  //Serial.println(" *C");
  //Serial.print  ("[INIT. DHT] Resolution:   ");
  //Serial.print(sensor.resolution);
  //Serial.println(" *C");
  //Serial.println("------------------------------------");
  // Print humidity sensor details.
  dht.humidity().getSensor(&sensor);
  //Serial.println("------------------------------------");
  //Serial.println("Humidity");
  //Serial.print  ("[INIT. DHT] Sensor:       ");
  //Serial.println(sensor.name);
  //Serial.print  ("[INIT. DHT] Driver Ver:   ");
  //Serial.println(sensor.version);
  //Serial.print  ("[INIT. DHT] Unique ID:    ");
  //Serial.println(sensor.sensor_id);
```

```
  //Serial.print  ("[INIT. DHT] Max Value:    ");
  //Serial.print(sensor.max_value);
  //Serial.println("%");
  //Serial.print  ("[INIT. DHT] Min Value:    ");
  //Serial.print(sensor.min_value);
  //Serial.println("%");
  //Serial.print  ("[INIT. DHT] Resolution:   ");
  //Serial.print(sensor.resolution);
  //Serial.println("%");
  //Serial.println("------------------------------------");
}
float readDHT22( int sens){
  // Get temperature event and print its value.
  sensors_event_t event;
  if(sens==1){
      dht.temperature().getEvent(&event);
  if (isnan(event.temperature)) {
  Serial.println("[READ DHT] Error reading temperature!");
  }
  else {
  float temp=event.temperature;
  Serial.print("[READ DHT] Temperature: ");
  Serial.print(temp);
  Serial.println(" *C");
  return temp;
  }
  }
  else if(sens==2){
    // Get humidity event and print its value.
  dht.humidity().getEvent(&event);
  if (isnan(event.relative_humidity)) {
  Serial.println("[READ DHT] Error reading humidity!");
  }
  else {
  Serial.print("[READ DHT] Humidity: ");
  float hum=event.relative_humidity;
  Serial.print(hum);
  Serial.println("%");
```

```
  return hum;
  }
  }
  else{
    return 0;
  }

}
bool initializeBMP280( ){


  if (!bmp.begin()) {
  Serial.println("[INIT. BMP] Could not find a valid BMP085
      sensor, check wiring!");
  return false;
  }
  return true;
}
float readBMP280(int sens){
  float buf[2];
  if(sens==1){
    Serial.print("[READ BMP] Temperature = ");
    buf[0]=bmp.readTemperature();
    Serial.print(buf[0]);
    Serial.println(" *C");
    return buf[0];
  }

  else if(sens==2) {
    buf[1]=bmp.readPressure();
    Serial.print("[READ BMP] Pressure = ");
    Serial.print(buf[1]);
    Serial.println(" Pa");
    return buf[1];
  }

    else return 0;
```

```
  // Calculate altitude assuming 'standard' barometric
  // pressure of 1013.25 millibar = 101325 Pascal
  //Serial.print("[READ BMP] Altitude = ");
  //Serial.print(bmp.readAltitude());
  //Serial.println(" meters");

  //Serial.print("[READ BMP] Pressure at sealevel (
      calculated) = ");
  //Serial.print(bmp.readSealevelPressure());
  //Serial.println(" Pa");

  // you can get a more precise measurement of altitude
  // if you know the current sea level pressure which will
  // vary with weather and such. If it is 1015 millibars
  // that is equal to 101500 Pascals.
  //Serial.print("[READ BMP] Real altitude = ");
  //Serial.print(bmp.readAltitude(101500));
  //Serial.println(" meters");

  //Serial.println();
}
```

Bibliografia

[1] Estellés Arolas, E.; González Ladrón-de-Guevara, F. "Towards an integrated crowdsourcing definition", Journal of Information Science. Vol 38. no 2. pp. 189-200, 2012.

[2] Federico Montori, Luca Bedogni, Alan Di Chiappari, Luciano Bononi, "SenSquare: a Mobile Crowdsensing Architecture for Smart Cities", in: EEE 3rd World Forum on Internet of Things (WF-IoT) (WF-IoT 2016).

[3] F. Montori, L. Bedogni, L. Bononi (2016) "On the Integration of Heterogeneous Data Sources for the Collaborative Internet of Things", in: IEEE RTSI 2016.

[4] C. Bormann, A. Castellani, Z. Shelby, "CoAP: An Application Protocol for Billions of Tiny Internet Nodes", Internet Computing IEEE, vol. 16, no. 2, pp. 62-67, March 2012.

[5] Z. Shelby, K. Hartke, and C. Bormann, "Constrained Application Protocol", IEEE Internet-Draft, draft-ietf-core-coap-18, June 28, 2013.

[6] P. Penial, M. Franekova,"Model of integration of embedded systems via CoAP protocol of Internet of Things",2016 International Conference on Applied Electronics (AE), pp. 201-204, 2016.

[7] P. Bellavista, S. Zanni, "Towards Better Scalability for IoT-Cloud Interactions via Combined Exploitation of MQTT and CoAP", In: IEEE 2016.

[8] Shu-yuan Ge; Seung-Man Chun; Hyun-Su Kim; Jong-Tae Park, "Design and implementation of interoperable IoT healthcare system based on international standards", 2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC), pp. 119-124, 2016.

[9] Zheng Liu; Gustavo Monte; Victor Huang "ISO/IEC/IEEE P21451-001 standard for signal treatment of sensory data", IEEE 25th International Symposium on Industrial Electronics (ISIE), pp. 766-771, 2016.

[10] D. Bonino; M. Delgado Alizo; A. Alapetite; T. Gilbert; M. Axling; H. Udsen; J. Carvajal Soto; M. Spirito, "ALMANAC: Internet of Things for Smart Cities", 3rd International Conference on Future Internet of Things and Cloud, pp. 309-316, 2015.

[11] Gaurav Sarin, "Developing smart cities using Internet of Things: An empirical study", 3rd International Conference on Computing for Sustainable Global Development (INDIACom), pp. 315-320, 2016.

[12] C. Leonardi, A. Cappellotto, M. Caraviello, B. Lepri, F. Antonelli, "SecondNose: an air quality mobile crowdsensing system", in Proceedings of th 8th Nordic Conference on Human Computer Interaction Fun, Fast, Fundational - NorCHI'14, pp. 1051-1054, 2015.

[13] R. Sanpietro, L. Bedogni, M. Di Felice, L. Bononi, "Park Here! A smart parking system based on smartphones' embedded sensors and short range Communication Technologies", in IEEE World Forum on Internet of Things, WF-IoT 2015 - Proceedings, pp. 18-23, 2015.

Ringraziamenti

"*La nuova interdipendenza creata dall'elettronica ricrea il mondo ad immagine di un villaggio globale.*" - Marshall McLuhan

Desidero innanzitutto ringraziare il Prof. Bononi ed il Dr. Bedogni per i preziosi insegnamenti durante il corso seguito alla laurea triennale, nel percorso di tirocinio e le per le numerose ore dedicate alla mia tesi, che senza il loro aiuto non esisterebbe.
Inoltre, ringrazio sentitamente chi è stato sempre disponibile a dirimere i miei dubbi durante la stesura di questo lavoro e vorrei esprimere la mia sincera gratitudine ai miei compagni di corso, in particolare Davide e Giovanni, per i numerosi consigli durante la ricerca e lo sviluppo di questo progetto.
Il ringraziamento va anche all'Università per avermi dato la possibilità, i mezzi e le conoscenze per affrontare questo percorso al meglio.
Infine, ho desiderio di ringraziare con affetto i miei genitori per il sostegno ed il grande aiuto che mi hanno dato, per essermi stato vicino ogni momento durante questo percorso.