ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Una piattaforma per la creazione e la gestione di servizi per IoT Collaborativo tramite Crowdsense

Relatore:
Chiar.mo Prof.
Luciano Bononi

Presentata da:
Davide Crestini

Co-relatori:
Dott. Luca Bedogni
Dott. Federico Montori

Sessione II
Anno Accademico 2015/2016

A Paolo, Giovanna, Ersilia

Chiunque abbia perso la nozione del tempo mentre usava un computer
conosce la propensione a sognare, il bisogno di realizzare i propri sogni
e la tendenza a saltare i pasti.

Tim Berners Lee

Indice

Elenco delle figure

Elenco dei codici

Capitolo 1

Introduzione

L'Internet of Things, termine coniato nel 1999 da Kevin Ashton, è l'insieme dei sensori ed attuatori integrati in oggetti fisici collegati tra loro attraverso reti cablate e wireless [1].

Nell'ultimo decennio l'utilizzo di dispositivi connessi ad Internet ha avuto una crescita dirompente. Nel 2008 il numero di tali dispositivi ha superato il numero delle persone connesse ad Internet e si stima che raggiungerà i 20.8 miliardi di unità nel 2020 [2]. Nel 2012 si stimava che il 5% degli oggetti costruiti dall'uomo fossero dotati di un microprocessore interno [3] in grado di registrare e trasmettere informazioni relative a suoni, temperatura, umidità ed altri valori. Tali informazioni vengono poi elaborate da sistemi centralizzati o da applicazioni installate direttamente sullo smartphone.

Allo stesso tempo, l'introduzione nel mercato di piattaforme a basso costo, Arduino[1] e Raspberry Pi[2] in particolare, integrabili con i più svariati tipi di sensore (DHT11, BMP180 *et al.*), ha permesso ad un numero sempre più ampio di persone di costruire delle soluzioni personali in grado di raccogliere e salvare dati, in forma privata o pubblica, su *datalogger* online. Questo aspetto rientra nell'ambito del Crowdsensing, una branca del Collaborative IoT, di cui fanno parte anche le applicazioni del *Mobile Crowdsensing (MCS)*, attraverso le quali dati provenienti

[1] https://www.arduino.cc/
[2] https://www.raspberrypi.org

da dispositivi mobili dotati di sensori (smartphone, tablets, wearables) degli utenti vengono aggregati e messi a disposizione di altri utenti [4].

Oltre ai dati forniti dai privati, in rete si trovano anche dati forniti da enti statali e regionali tra cui ARPAE[3], la quale mette a disposizione una grande quantità di dati tra cui valori ambientali relativi alla qualità dell'aria. Questi dati rispetto a quelli forniti dai privati hanno un'affidabilità maggiore.

È evidente che una mole di dati simile (con un trend di crescita altissimo) possano risultare interessanti sia per utenti privati che per enti pubblici quali comuni, province, regioni ed anche per alcuni tipi di aziende (si pensi a come potrebbe incidere sulle decisioni di imprese agricole). Per poter raggiungere questo risultato è necessario però che vi siano delle applicazioni che permettano a chiunque di utilizzare in maniera il più semplice possibile le informazioni disponibili.

Nel primo capitolo di questo studio introdurremo il concetto di Crowdsensing ed analizzeremo quali sono ad oggi le applicazioni che ne fanno parte cercando di capire dove si possa intervenire per creare un servizio che sia in grado di sfruttare la grande quantità di dati disponibili in rete. Nel secondo capitolo parleremo di SenSquare che rappresenta l'architettura di riferimento del nostro studio. Procederemo poi ad illustrare il lavoro svolto, il quale è stato suddiviso in due parti:

- Perfezionamento dell'algoritmo di classificazione [5] utilizzato per classificare automaticamente i dati provenienti dalle piattaforme ThingSpeak[4] e SparkFun[5], i quali non sono ben formati e necessitano quindi di una classificazione. Allo scopo è stata creata un'interfaccia che permettesse di semplificare sia la classificazione manuale che l'analisi delle metriche.

- Implementazione di Crowdservice, una piattaforma per SenSquare, il cui scopo è permettere ad ogni utente registrato di istanziare schemi, propri o altrui, in un particolare luogo, sfruttando la geolocalizzazione delle informazioni disponibili.

[3] http://www.arpae.it/
[4] https://thingspeak.com
[5] https://data.sparkfun.com

Capitolo 2

Stato dell'Arte

2.1 Internet of Things

L'Internet of Things (IoT) è una delle evoluzioni più interessanti della rete, riguarda l'interconnessione di dispositivi, veicoli, costruzioni, vestiti ed altri tipi di oggetto contenenti dei sistemi integrati dotati di sensori, attuatori, software e connessione di rete che gli permettono di collezionare, scambiare ed analizzare dati. L'IoT trova applicazione nei campi più disparati, quali la domotica, il settore medico, ingegneristico, ambientale e persino sportivo. La grande evoluzione elettronica degli ultimi anni ha permesso di costruire circuiti sempre più piccoli, sempre più potenti e sempre più economici, diffondendo l'idea che sia possibile inserirne uno in ogni oggetto creato dall'uomo.

Il mercato dell'IoT sta attirando l'interesse di tutte le grandi multinazionali che lavorano nel campo della tecnologia. Un caso celebre è Apple Watch, lo **smartwatch** lanciato da Apple nel 2014, dotato di sensori quali cardiofrequenzimetro, accelerometro e giroscopio ed in grado di comunicare con applicazioni per Ios fornendo all'utente tutte le informazioni ritenute necessarie. Ma gli *smartwatch* sono solo uno delle migliaia di prodotti creati negli ultimi anni nell'ambito dell'IoT. Per avere un idea più concreta dell'importanza di questo mercato basta considerare che la sua dimensione economica globale nel 2015 è stata valutata in 157 miliardi di dollari e secondo l'agenzia Research and Markets, salirà a 661

miliardi nel 2021 [6].

Questo trend è confermato non solo dal moltiplicarsi di oggetti creati dalle grandi multinazionali, ma anche dal moltiplicarsi dei sensori (DHT11, BMP180, DS18B20...) e dei moduli (ESP8266) a basso costo oltre che delle piattaforme come Arduino e Raspberry Pi. Questi sistemi se ben sfruttati permettono a qualsiasi appassionato di creare ad esempio un sistema di domotica personale, una piccola serra, un sistema di annaffiamento personalizzato per il proprio giardino ed altri servizi simili. Il tutto può essere fatto con dei costi molto contenuti.

L'importanza di questo settore è sottolineata anche dal grande numero di discipline che devono coordinarsi per la buona riuscita di un prodotto (sia esso software o hardware), tra esse figurano l'elettronica, le telecomunicazioni, l'Intelligenza Artificiale, il Semantic Web, il Cloud Computing, la sicurezza informatica e lo sviluppo software.

Iot Collaborativo

Il campo dell'IoT non attrae l'interesse solo delle grandi multinazionali. Di fatti la possibilità di poter disporre di una grande quantità di dati senza dover sostenere grosse spese per acquisirli è un aspetto molto interessante per tutti quei soggetti che siano intenzionati ad utilizzarli per scopi diversi da quelli meramente commerciali. Tra questi rientrano sia privati che enti pubblici quali comuni e regioni, che potrebbero utilizzarli per determinare quali interventi debbano essere fatti nel territorio di loro competenza.
Affinché tali dati siano a disposizione di tutti sono necessari dei meccanismi che permettano ad una grande massa di utenti di condividerli e reperirli. Proprio in questo settore si colloca l'IoT collaborativo, ovvero quella parte dell'IoT in cui i soggetti sono incentivati a collaborare per un interesse comune.

Una delle tecniche più interessanti di IoT Collaborativo è il *Crowdsensing* di cui parleremo in sezione 2.3 dopo aver introdotto il concetto di *Crowdsourcing*.

2.2 Crowdsourcing

Il Crowdsourcing è un modello economico basato sulla condivisione di conoscenze su larga scala per l'ideazione, lo sviluppo e la realizzazione di progetti lavorativi. La parola Crowdsourcing è una combinazione delle parole *crowd* (folla) e *outsourcing* (esternalizzazione di servizi). L'idea è quella di esternalizzare un lavoro ad una grande massa di utenti. Il termine Crowdsourcing venne utilizzato per la prima volta nel 2006 da Jeff Howe, giornalista statunitense esperto di *new economy* e lavoro digitale. In un articolo realizzato per la rivista **Wired**, Howe metteva in luce come stesse nascendo una nuova forma di collaborazione lavorativa, aperta a tutti e condivisa in rete. Nel 2012 Estellés e Gonzales hanno fornito una nuova definizione di Crowdsourcing, dove diviene centrale il ruolo delle due controparti in gioco. Per i due autori, sia il professionista che il crowdsourcer, ossia l'affidatario, vedranno soddisfatte le loro necessità grazie alla partecipazione al progetto stesso [7].

Uno dei progetti di Crowdsourcing meglio riusciti è Wikipedia, i cui ideatori invece di creare un'enciclopedia in proprio, assumendo scrittori ed editori, hanno preferito esternalizzare il lavoro concedendo a tutti gli utenti la possibilità di aggiungere le informazioni per proprio conto. L'esperimento ha avuto un successo così grande che ad oggi Wikipedia è l'enciclopedia più completa che sia mai stata scritta.

Esistono varie forme di Crowdsourcing tra cui: *Crowdsource Design*, *Crowdfunding*, *Microtask*, *Open Innovation* [8], *Code Hosting Platform* e *Crowdsensing*.

Crowdsource Design

Il Crowdsource Design è rivolto a tutti coloro che necessitano di un designer per la creazione di grafiche, loghi ed interfacce web. A tale scopo sono disponibili online alcune piattaforme, tra cui **Designcrowd**[1], in cui si mostra ad una folla di designer il lavoro richiesto, il compenso e la deadline. Tutti i designer interessati proporranno un proprio prodotto per il richiedente, il quale una volta

[1] http://www.designcrowd.com/

decorsa la deadline potrà scegliere tra tutte le proposte ricevute quella che più lo convince. Attraverso questo metodo si tende ad aumentare la qualità dei prodotti abbassandone i costi.

Crowdfunding

Il Crowdfunding è una delle forme di Crowdsourcing più conosciuta che consiste nell'utilizzare Internet per la raccolta di capitale da gruppi di persone con interessi comuni al fine di finanziare un progetto o un iniziativa. Da un punto di vista tecnico è definito come una pratica di microfinanziamento dal basso. Il Crowdfunding si può riferire a iniziative di qualsiasi genere, dall'aiuto in occasione di tragedie umanitarie al sostegno all'arte e ai beni culturali, al giornalismo partecipativo, fino all'imprenditoria innovativa e alla ricerca scientifica. Il Crowdfunding è spesso utilizzato per promuovere l'innovazione e il cambiamento sociale, abbattendo le barriere tradizionali dell'investimento finanziario.

Come ogni altra forma di Crowdsourcing, il Crowdfunding trova il proprio successo grazie alla disponibilità di piattaforme online le quali riescono a raggiungere un ampio numero di persone con costi piuttosto contenuti. Una delle più conosciute è **Kickstarter**[2]. In questa piattaforma ogni progetto ha una durata temporale definita, nella quale si prefigge di raggiungere un determinato obiettivo economico (goal) espresso solitamente in dollari. Questi due cruciali parametri sono stabiliti unicamente dal creatore del progetto. Alla fine del periodo temporale, se il goal è stato raggiunto (cioè sono stati raccolti abbastanza fondi), i soldi raccolti vengono versati da Kickstarter al creatore del progetto, che (se affidabile) li userà per attuare quanto promesso; in caso, contrario i soldi restano nelle tasche dei sostenitori (*backers*) del progetto.

Microtask

Il Microtasking è un metodo attraverso il quale un grosso lavoro viene suddiviso in tanti piccoli task, che vengono poi proposti ad una folla dietro pagamento

[2]https://www.kickstarter.com/

di una ricompensa. Per comprendere meglio mostriamo qui un esempio.

Supponiamo che un'azienda necessiti di dover tradurre 100.000 testi per un progetto interno, se affidasse il lavoro ai propri dipendenti, che supponiamo essere 100, ognuno di loro dovrebbe tradurre 1.000 testi impiegando una grande quantità di tempo e risorse. Per velocizzare il processo l'azienda potrebbe quindi decidere di affidarsi a piattaforme di microtasking, proponendo ad una vasta quantità di soggetti esterni di tradurre uno o più testi in cambio di una ricompensa monetaria. Così facendo i dipendenti potrebbero continuare a svolgere il proprio lavoro e nel frattempo l'azienda potrebbe parallelamente portare avanti il progetto di traduzione.

I campi di applicazione del Microtasking sono molteplici e non riguardano solo progetti industriali, ma anche progetti di ricerca nei più svariati campi.

Open Innovation

L'Open Innovation permette a persone operanti in ambiti differenti quali investitori, ingegneri, designer ed esperti di marketing di collaborare assieme per realizzare un'idea di profitto. Questo generalmente avviene attraverso piattaforme web dedicate a raccogliere punti di vista differenti. Una di queste è **Open Innovation by ENGIE**[3] che riunisce persone provenienti da diverse parti del mondo e diversi settori di business per lavorare insieme ad un progetto.

Il concetto di Open Innovation permette inoltre di coinvolgere maggiormente quelle figure, come gli investitori, che prima non venivano coinvolte nel processo di sviluppo, ma venivano considerate solo come una fonte di fondi da cui attingere.

Code Hosting Platform

Le Code Hosting Platform sono piattaforme online che permettono a sviluppatori di tutto il mondo di cooperare nello sviluppo di software. Attraverso queste piattaforme è infatti possibile condividere il codice sorgente di un progetto tra più sviluppatori. Il codice può essere condiviso in due forme diverse: *open* e *closed*.

[3] http://openinnovation-engie.com/en/

Attraverso la prima forma chiunque può vedere il codice sorgente e contribuire allo sviluppo. Si parla in questo caso di progetti *Open Source*. Per poter partecipare attivamente ad un progetto è necessario che chi ne possiede i diritti di amministrazione conceda agli utenti interessati il diritto a collaborare. La forma *closed* invece permette di condividere il codice solo tra i soggetti che vi lavorano o che fanno parte del progetto.

Le due piattaforme di code hosting più conosciute ed utilizzate sono **GitHub**[4] e **Bitucket**[5]. Entrambe permettono non solo di condividere il codice ma anche di mantenerne il versionamento attraverso il software **git**[6].

2.3 Crowdsensing

Il termine Crowdsensing si riferisce alla condivisione di dati collezionati da dispositivi di rilevamento con lo scopo di misurare un fenomeno di interesse comune. I dati possono provenire da smartphone e tablet, da piattaforme fisse come Arduino e Raspberry Pi, con i relativi sensori collegati, oppure da enti pubblici.

2.3.1 Tipologie

È possibile individuare 5 tipologie di Crowdsensing sulla base di due differenti criteri [9]. I criteri sono:

- coinvolgimento dell'utente nel processo;
- tipo di fenomeno misurato.

Sulla base del primo criterio possiamo distinguere tra *participatory crowdsensing* e *opportunistic crowdsensing*. Nel primo caso l'utente partecipa attivamente al processo inviando volontariamente i dati ad un server, nel secondo caso invece l'intervento dell'utente è pressoché nullo e i dati sono inviati in maniera del tutto automatica. Sulla base del secondo criterio invece possiamo distinguere tre

[4] https://github.com/
[5] https://bitbucket.org
[6] https://git-scm.com/

diverse categorie di crowdsensing: ambientale, infrastrutturale e sociale. Quello ambientale è utilizzato per misurare l'ambiente naturale (*e.g.* qualità dell'aria, temperatura, umidità), quello infrastrutturale viene usato per misurare le infrastrutture pubbliche (*e.g.* traffico, condizioni delle strade) mentre quello sociale colleziona dati che riguardano la vita sociale degli individui (*e.g.* musei e cinema visitati). Tale suddivisione è descritta in figura 2.1.

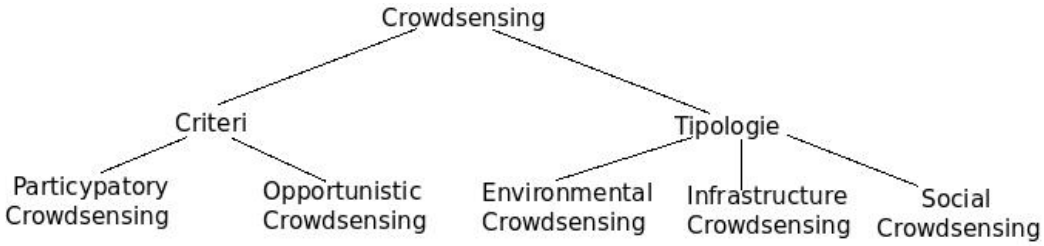


Figura 2.1: Suddivisione Crowdsensing.

2.3.2 Chiavi di sviluppo

Affinché tale paradigma possa funzionare è necessario assecondare il concetto di *user experience* sviluppando un modello che "ruoti attorno all'uomo" [10]. Nel particolare i problemi che devono essere analizzati sono i seguenti:

- **Incentivi alla partecipazione.** Questo argomento è stato centrale nel campo della filosofia e dell'economia. La promessa di guadagni finanziari e/o monetari è spesso un incentivo importante, ma ancora più importante è la partecipazione degli utenti per scopi meramente sociali ed etici, come ad esempio la socializzazione, la reputazione ed il riconoscimento da parte di altri utenti.

- **Sicurezza e privacy dell'utente.** La condivisione di dati personali (*e.g.* posizione) può comportare una minaccia importante alla sicurezza ed alla privacy dei partecipanti. Pertanto devono essere sviluppati nuovi approcci al

mantenimento della privacy per garantire che informazioni importanti non possano essere carpite o inferite da soggetti malevoli.

- **Sviluppo di interfacce user-friendly.** Se lo sviluppo di applicazioni di ausilio alla condivisione è già fortemente avanzato, mancano ancora delle piattaforme di facile utilizzo che permettano a chiunque di sfruttare queste informazioni. Nell'era del design è impensabile riuscire a realizzare un prodotto di larga diffusione che non sia attraente e soprattutto intuitivo per l'utente. Attualmente questo è forse il vero *tallone d'Achille del Crowdsensing*; ad oggi infatti non esistono piattaforme che permettano a chiunque sia interessato di usufruire dei dati messi a disposizione, a meno che non disponga di conoscenze specifiche. Lo scopo di questo studio è proprio quello di creare una base di piattaforma di facile utilizzo anche per i neofiti del settore.

2.3.3 Opportunistic Crowdsensing

L'*Opportunistic Crowdsensing* è un paradigma nel quale l'utente condivide i dati con il minimo sforzo possibile. Le applicazioni che si basano su questo paradigma infatti generalmente richiedono all'utente solo di impostare alcuni parametri iniziali o di concedere alcuni permessi, dopodiché non è richiesta alcun tipo di partecipazione. Applicazioni di questo genere vengono spesso utilizzate per condurre delle statistiche, potendo disporre di una grande mole di dati, in quanto coinvolgere l'utente a condividere i dati è generalmente facile visto che non viene richiesta una partecipazione attiva.

2.3.4 Participatory Crowdsensing

Il *Participatory Crowdsensing* in contrapposizione con l'*Opportunistic Crowdsensing* necessita di una partecipazione più attiva dell'utente. In questo paradigma è fondamentale riuscire a coinvolgere il più possibile gli utenti fornendo delle piattaforme che permettano di condividere ed utilizzare facilmente i dati. In

alcuni casi gli utenti sono spinti a collaborare solo per un "senso di appartenenza ad una comunità", ma spesso questo fattore da solo non risulta sufficiente a coinvolgere una vasta rete di persone.

Una delle applicazioni più famose che si basano su questo paradigma è **Waze**[7]. Waze permette agli automobilisti di risparmiare tempo alla guida, indicando eventuali incidenti, code, posti di blocco o pericoli presenti lungo il percorso. Queste indicazioni vengono rilevate proprio grazie alla partecipazione di altri automobilisti che decidono di segnalarle tramite l'app. Waze utilizza un sistema a punteggio e classifiche per coinvolgere gli utenti, consentendo di guidare su delle icone, situate in alcuni punti specifici, per guadagnare punti aggiuntivi. Sono inoltre previsti dei mini-giochi per favorire il coinvolgimento e la concorrenza, e ciò rende le informazioni stradali più aggiornate nelle zone in cui i dettagli sono comunque pochi o mancanti.

Data reliability

Un tema particolarmente delicato è invece il problema della *reliability* (affidabilità) dei dati. I dati condivisi in maniera autonoma posso infatti presentare diversi problemi da non sottovalutare. Innanzitutto i sensori utilizzati dagli utenti potrebbero essere di scarsa qualità, fornendo in tal caso misurazioni poco precise, oppure potrebbero essere messi in luoghi di scarso interesse per gli altri (*e.g* all'interno di una stanza). Infine potrebbe accadere che soggetti malintenzionati decidano di inserire nei *datastore* valori volutamente sbagliati, recando un danno ai soggetti che andranno ad utilizzarli.

Una soluzione a tale problema potrebbe essere quella di creare un sistema di reputazione tra gli utenti (metodo già ampiamente diffuso in altri campi), il quale significherebbe un valore aggiunto sia per i dati che per gli utenti, oltre che per i *datastore*.

[7] https://www.waze.com

2.3.5 Applicazioni

Nel mercato sono già presenti alcune applicazioni che sfruttano il meccanismo del Crowdsensing; tra queste figurano:

- **Creek Watch.** Sviluppata da IBM Almaden Research Center per monitorare il livello dell'acqua e la qualità dell'aria.
- **Nericell.** Sviluppata da Microsoft Reasearch per monitorare il traffico e le condizioni della strada.
- **DietSense.** Sviluppata ad UCLA per condividere informazioni relative alle abitudini alimentari degli utenti.

Le applicazioni sopra citate sono di semplice utilizzo ma non permettono all'utente di utilizzare i dati disponibili in maniera personalizzata, infatti ognuna di esse è relativa ad un singolo ambito e sfrutta dati omogenei forniti per un singolo scopo.

Apisense

APISENSE[8] è un progetto di Inria ADAM, che fornisce una soluzione per creare e distribuire servizi di Crowdsensing per la raccolta di dati sperimentali. La piattaforma distingue due ruoli diversi. Il primo, chiamato scienzato, è tipicamente un ricercatore che vuole definire ed effettuare un esperimento attraverso un gran numero di dati proveniente dai *mobile users*. Viene quindi fornito un ambiente online che permette ai ricercatori di descrivere i requisiti e le regole necessari all'esperimento attraverso un linguaggio di scripting, permettendo inoltre di collegarsi a servizi esterni in grado di processare ed analizzare i dati rilevati. Il secondo ruolo è rappresentato proprio dai *mobile users*, ovvero utenti dotati di dispositivi mobili. APISENSE fornisce a questi ultimi un'app[9], per Android e Ios, in cui

[8] https://apisense.com/

[9] Al momento della redazione della tesi l'app è ancora in fase di sviluppo e quindi non scaricabile sul proprio device.

vengono mostrati gli esperimenti inseriti a cui l'utente può decidere di collaborare se interessato, consentendo quindi all'app di inviare le rilevazioni necessarie [11]. APISENSE è un esperimento molto interessante, ma come è facile intuire, solo persone esperte (*i.e.* scienziati o ricercatori) possono definire nuovi servizi e agli utenti è lasciata solo la possibilità di seguire aggiornamenti su servizi ed esperimenti creati da altri. Neppure questo progetto dunque risponde pienamente al requisito di "interfaccia user-friendly ad ogni livello", proprio questo ci spinge ulteriormente a pensare che sia necessaria una piattaforma alla portata di tutti che permetta a chiunque di creare il proprio servizio e di seguirne gli aggiornamenti.

Capitolo 3

L'architettura

SenSquare è stato sviluppato con lo scopo di raccogliere e rendere disponibile una grande quantità di dati eterogenei provenienti da soggetti diversi. Proprio per questo motivo la raccolta dei dati si dirama su sistemi differenti:

- Il Mobile Crowdsensing.

- Il *fetching di open data* dalle *open data platforms*.

Sensquare può essere utilizzato da due differenti tipi di utente. Da una parte troviamo gli *stakeholders*, ovvero soggetti che sono interessati ad utilizzare i dati disponibili per degli scopi propri. Dall'altra troviamo i *participants*, ovvero utenti che forniscono le misurazioni (effettuate dai propri devices) agli stakeholders[1]. Questi ultimi possono inoltre utilizzare i dati per creare schemi ed istanziare servizi, come verrà illustrato in sezione 5.3.

In questo capitolo illustreremo l'architettura del sistema SenSquare, descrivendone i moduli che la compongono ed un'applicazione precedentemente sviluppata.

[1] Il meccanismo di richiesta e fornitura dei dati sarà spiegato nella sezione 3.3.

3.1 Central Coordination Unit

Tale sistema è organizzato in una topologia a stella, il che significa che le entità client non hanno collegamenti tra loro, mentre tutte le operazioni sono svolte da un componente centralizzato chiamato **Central Coordination Unit (CCU)**. Nel dettaglio la CCU è composta da un unico database centrale e da più server "satellite" che si interfacciano con esso. Lo scopo di questo componente è quello di gestire sia la comunicazione con il database che la logica dei vari applicativi, infatti per ogni applicativo che la compone, la logica, così come i calcoli sono svolti in toto dai server della CCU.

3.2 Database

Il database è gestito mediante MySQL ed è composto dalle seguenti tabelle:

- **Stakeholders** contiene le informazioni degli stakeholders.
- **Participiants** contiene le informazioni degli utenti iscritti al servizio.
- **Subscriptions** contiene le iscrizioni degli utenti relative agli stakeholder.
- **Dataclasses** contiene tutte le classi di appartenenza[2] che sono state individuate.
- **Rules** contiene le regole definite dagli stakeholders. Ogni regola è descritta da una frequenza di aggiornamento sia spaziale che temporale ed è relativa ad un'unica classe.
- **Devices** contiene le informazioni, tra cui nome e tipo (*i.e.* mobile, fisso, governativo) dei dispositivi o dei canali da cui vengono prese le rilevazioni.
- **DataStreams** contiene, per ogni *Device*, le informazioni relative ad uno dei valori misurati (*e.g.* temperatura, umidità, pressione).

[2] Una spiegazione più approfondita sarà mostrata nel capitolo 4

- **Measurements** contiene tutte le misurazioni relative ad ogni *DataStream*; è in questa tabella che vengono salvate le coordinate relative ad ogni misurazione, le quali possono cambiare da una misurazione all'altra se effettuate da un dispositivo mobile.
- **CustomServicesTemplates** contiene le informazioni relative ad uno schema di servizio, tra cui l'espressione di cui si vuole tenere traccia e il tipo di output (numerico o booleano).[3] Ogni template viene creato da un unico utente ma può essere istanziato da qualsiasi utente.
- **CustomServices** contiene le informazioni relative all'istanziazione di un *CustomServiceTemplate*, in particolare contiene l'id dello schema a cui si riferisce, quello dell'utente che lo ha creato, il titolo del servizio, le coordinate del punto in cui vogliamo istanziarlo ed il raggio entro il quale vogliamo effettuare le misurazioni.
- **Multisensors** mette in relazione i *CustomServices* con i *DataStreams*.
- **PersonalServices** contiene le informazioni di servizi predefiniti dall'utente.

Durante lo sviluppo della piattaforma si è riscontrata la necessità di aggiungere alcuni attributi. Nel particolare è stato aggiunto l'attributo **title** alle tabelle *CustomServicesTemplate* e *CustomServices*, necessario per poter attribuire un titolo ai template e ai servizi creati dagli utenti, i quali altrimenti sarebbero stati riconoscibili solo tramite l'id e ciò avrebbe comportato un problema di usabilità. Per quanto riguarda la prima tabella il titolo è scelto a piacere dall'utente che lo crea, mentre nella seconda il titolo viene creato automaticamente dal sistema componendo il titolo della schema con l'indirizzo scelto dall'utente, separati dal carattere "**-**". Sono inoltre stati aggiunti i campi **password** ed **email** alla tabella degli utenti per poterne gestire l'autenticazione. Tali dispositivi sono chiamati **sensing clients**. Il database di cui sopra è mostrato in figura 3.1

[3] Questi aspetti saranno chiariti meglio nel capitolo 5.

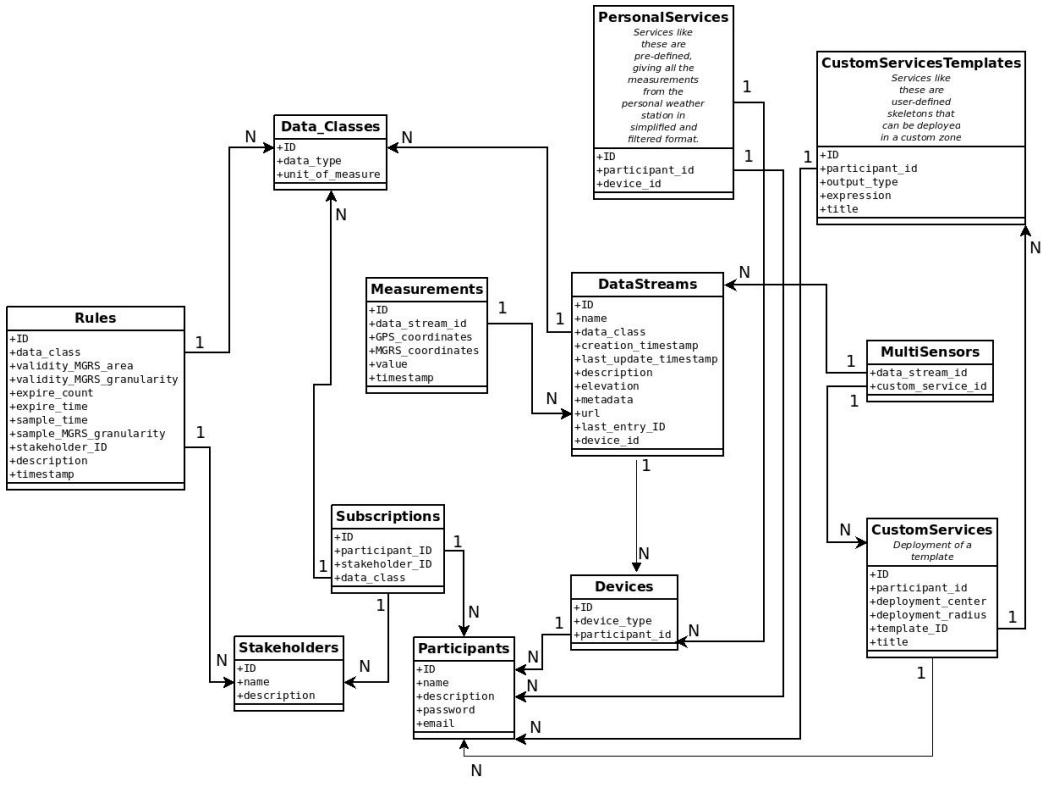


Figura 3.1: Database di SenSquare

3.3 Crowdroid

Crowdroid [12] [13] è una delle applicazioni, basata sull'architettura SenSquare, che rientra nel contesto del Mobile Crowdsensing. Quest'applicazione prevede uno scenario in cui un insieme di **sensing clients**, ovvero dispositivi appartenenti ad un utente, che dispongono di sensori e di una connessione ad Internet, vengono utilizzati per fornire al server le misurazioni richieste.

Lo scenario di riferimento è composto da due differenti entità:

- I *participants*, i quali dispongono di dispositivi mobili equipaggiati con dei sensori e con un'architettura di rete (*i.e.* i sensing clients).

- Gli *stakeholders*, ovvero soggetti che possono scrivere regole sul server descrivendo la frequenza di aggiornamento relativa al tempo e allo spazio di loro interesse.

Agli stakeholders i dati vengono forniti dai dispositivi degli utenti i quali possono decidere di iscriversi o meno a seconda della *revenue*, che generalmente dipende dalle necessità di entrambe le parti.

Dovendo l'applicazione gestire degli update spaziali oltre che temporali, si rende necessario un sistema di rilevamento della posizione ed a questo scopo è stato deciso di utilizzare le coordinate *Military Grid Reference System (MGRS)*, ritenute più opportune al conseguimento degli scopi preposti. Il server è un servizio CoAP, scritto in Python, ed è uno dei componenti della CCU il cui compito è quello di raccogliere e salvare le misurazioni effettuate dai *sensing clients*, ritornando ad ogni richiesta la successiva frequenza di aggiornamento sia temporale che spaziale.

Lo scenario di esecuzione è piuttosto semplice: l'utente deve scegliere con quali stakeholders condividere le proprie misurazioni, da qui in poi sono l'app Android ed il server a prendersi in carico il lavoro necessario a raccogliere e processare i dati necessari. Il tutto avviene tramite un meccanismo di regole ben definito ed ampiamente spiegato in [12] [13]. L'applicativo implementa una sorta di mix tra opportunistic e participatory crowdsensing, in quanto l'utente ha la fa-

coltà di scegliere gli stakeholders, lasciando poi che l’app esegua il proprio lavoro in background.

Capitolo 4

Algoritmo di Classificazione

Come accennato nel capitolo 3, il *fetching di open data* dalle *open data platforms* è uno dei punti focali di SenSquare. I dati raccolti su queste piattaforme possono essere suddivisi in **realiable** e **unrealiable**. Sono considerati **reliable** i dati provenienti da piattaforme di condivisione di dati istituzionali, mentre vengono considerati **unreliable** i dati provenienti da piattaforme come ThingSpeak e SparkFun. Gli *open data* hanno il grande pregio di essere già disponibili in grandi quantità ma presentano un problema da non sottovalutare. Questi infatti non sono ben formati e senza una classificazione ben precisa risultano inutili agli scopi degli applicativi che fanno parte di SenSquare. A questo scopo è stato quindi implementato un algoritmo di classificazione che analizzi i dati provenienti dalle due piattaforme e li classifichi secondo regole opportune. L'algoritmo verrà descritto in sezione 4.2, dopo aver introdotto il concetto di classificazione.

4.1 Classificazione

Il termine classificazione viene utilizzato per varie attività che si possono ricondurre alla gestione delle conoscenze. L'attività di classificazione è propria dell'essere umano. Molte classificazioni si incontrano nella vita quotidiana, nelle opere di riferimento come i trattati, i cataloghi, le collezioni, gli atlanti e le opere enciclopediche. In statistica, con il termine classificazione si intende l'insieme

delle attività che, facendo uso di un algoritmo di analisi dei dati, individuano una rappresentazione di alcune caratteristiche di una entità da classificare (oggetto o nozione) e le associano ad una etichetta classificatoria [14]. Nell'ambito dei metodi statistici di classificazione si è soliti distinguere tra **unsupervised** e **supervised classification**. I primi ricercano negli oggetti una struttura a gruppi che non esiste o almeno non è nota a priori, mentre i secondi mirano ad individuare quali attributi determinano l'appartenenza di un oggetto ad un gruppo o ad un altro, relativamente ad un insieme di gruppi predeterminato.

Supervised classification

La tecnica di classificazione utilizzata dal nostro algoritmo viene definita *Dictionary Learning*[1]. In questa tecnica si fa uso di una struttura, detta appunto **dizionario**, che viene riempito analizzando dati appartenenti ad un particolare insieme chiamato *Training Set* che viene utilizzato per classificare dati appartenenti ad un altro insieme chiamato *Test Set*. L'unione di questi due insieme costituisce il *Dataset*.

Nel nostro caso dopo aver riempito la struttura del dizionario con i dati del training set e aver selezionato il test set, viene classificata ogni parola presente nel test set confrontandola con la lista di termini presenti nel dizionario associati ad ogni classe. Il confronto avviene calcolando la distanza tra i termini, in particolare si calcola la **distanza di Damerau-Levenshtein** [15]. Tale algoritmo indica la distanza tra due parole riportando il numero di operazioni necessarie per trasformare una parola nell'altra.

4.2 Improvement algoritmo

In questa sezione illustreremo il lavoro svolto per affinare l'algoritmo di classificazione descritto in [5].

[1]La tecnica di Dictionary Learning trae ispirazione dalla tecnica del Classificatore Bayesiano Naif, una particolare forma di classificazione che si basa sull'applicazione del teorema di Bayes.

Uno dei punti cruciali degli open data è che essendo condivisi da privati, non viene seguito alcun tipo di standard nello scegliere i nomi da assegnare ai campi degli *stream/channel*. L'obiettivo è quindi quello di uniformare i dati per poterne estrarre le informazioni significative. A tale scopo è stato precedentemente condotto uno studio [5] attraverso il quale sono state individuate 32 classi e che ha portato ad una prima implementazione dell'algoritmo.
Il nostro studio quindi si è inizialmente incentrato sull'analisi di tale algoritmo al fine di capire dove intervenire per migliorarne la precisione. A fronte dell'analisi svolta si è deciso di lavorare sui seguenti aspetti:

- Perfezionamento del dataset.
- Sostituzione dei file CSV con un databse MySQL.
- Implementazione di un'interfaccia per gestire stream e risultati.
- Perfezionamento delle metriche di valutazione.
- Individuazione delle classi rilevanti.

4.2.1 Perfezionamento Dataset

Abbiamo appena visto che negli algoritmi di classificazione il dataset è un componente fondamentale, in quanto permette di testarne effettivamente la precisione e l'efficienza. Disporre di un dataset ben strutturato permette quindi di trarre conclusioni più realistiche.

Il precedente dataset era composto da 841 record di cui 601 appartenenti al training set ed i restanti 240 appartenenti al test set. Il numero di record era piuttosto basso per poter condurre un'analisi precisa e si è quindi preferito creare un nuovo dataset; allo scopo sono stati inizialmente analizzati 2.000 stream provenienti sia da ThingSpeak che da SparkFun. Tali stream sono stati poi sottoposti ad un processo di classificazione manuale, durante il quale ne sono stati scartati 800 perché ritenuti non idonei (*i.e.* non erano classificabili a mano nemmeno attraverso analisi più approfondite). Il passo successivo è stato suddividere gli stream

rimanenti in training e test set. Il training set è composto da 245 stream (circa 20% dataset) mentre il test set è composto da 955 stream (circa 80% dataset). Dei 1.200 stream, 590 provengono da SparkFun mentre i restanti 610 provengono da ThingSpeak.

4.2.2 Passaggio dai file CSV al database MySQL

Il CSV (Comma-Separated Values) è un formato di file utilizzato per l'importazione ed esportazione di una tabella di dati. In questo formato ogni record della tabella è rappresentato da una linea di testo, a sua volta divisa in campi separati da un apposito carattere (*i.e.* comma), ciascuno dei quali rappresenta un valore. I file CSV presentano ottime performance quando si tratta di inserire un record o di leggere un'intera tabella, ma a differenza dei database relazionali non permettono né di effettuare query né utilizzare indici nè tanto meno di effettuare *join* tra tabelle e risultano molto più lenti nelle operazioni di aggiornamento ed eliminazione. Proprio per questo motivo si è deciso di salvare i dati su un database MySQL composto dalle seguenti tabelle:

- **Channel.** Rappresenta i canali provenienti dalle Open Data Platform; sono rappresentati da un id, un nome, una descrizione e dalla provenienza (ThingSpeak o Sparkfun).

- **Field.** Rappresenta i field associati ad ogni canale, ognuno rappresentato da un proprio nome, l'unità di misura, l'id del canale a cui è associato, il tipo di set di appartenenza (test o training) e dalla classe di appartenenza, la quale è stata assegnata manualmente. Da qui in poi i field saranno chiamati stream.

- **TagChannel.** Rappresenta i tag relativi ad ogni canale.

- **Classification.** In questa tabella vengono salvati i risultati della classificazione. Ogni record è relativo ad uno stream e contiene l'id, il nome e la

classe effettiva (assegnata manualmente) dello stream, l'id, la descrizione ed il nome del canale, la classe attribuita dall'algoritmo ed infine la distanza[2].

- **ClassificationRelevant.** Equivalente alla tabella sopra ma relativa alle classi rilevanti, di cui parleremo in sezione 4.2.5.

- **Metric.** Rappresenta le metriche calcolate sulla base della classificazione svolta. In particolare vengono memorizzate Recall, Precision ed F-Measure, di cui parleremo in sezione 4.2.4.

- **MetricRelevant.** Equivalente alla tabella sopra ma relativa alle classi rilevanti.

- **ConfusionMatrix** Contiene le informazioni relative alla matrice di confusione di cui parleremo in sezione 4.2.4.

- **ConfusionMatrixRelevant** Equivalente alla tabella sopra ma relativa alle classi rilevanti.

La realizzazione di questo database ha inoltre permesso di realizzare in maniera piuttosto semplice anche l'interfaccia che andremo a spiegare nella prossima sezione.

4.2.3 Interfaccia di gestione

La classificazione degli stream, così come il processo di divisione in test e training set, richiedevano l'analisi di una grande quantità di informazioni e ciò ha fatto emergere la necessità di sviluppare un'interfaccia per semplificare il compito.

È stato quindi deciso di sviluppare una Web App minimale composta da un server, scritto in Python attraverso il framework Flask[3], il quale si occupa di interrogare il database e fornire i dati al client in formato JSON[4]. Per la realizzazione

[2]distanza di Damerau-Levenshtein come spiegato sopra.

[3]http://flask.pocoo.org/

[4]JSON (JavaScript Object Notation) è un semplice formato per lo scambio di dati. Per le persone è facile da leggere e scrivere, mentre per le macchine risulta facile generarlo ed analizzarne la sintassi[16].

del client si è deciso di utilizzare il framework AngularJS[5], il quale permette di sviluppare velocemente delle viste dinamiche.

L'applicazione, sviluppata seguendo un approccio *single page*, permette di navigare tra schermate differenti attraverso i link presenti nella *Navbar* posta in alto. Ogni schermata permette di effettuare alcuni tipi di operazione e/o di visualizzare determinate informazioni. Le schermate create permettono di:

1. Classificare e scartare manualmente gli stream.

2. Suddividere gli stream in test e training set.

3. Visualizzare informazioni riguardanti il numero degli stream classificati manualmente, il numero degli stream dei due set ed il numero degli stream appartenenti ad ogni classe.

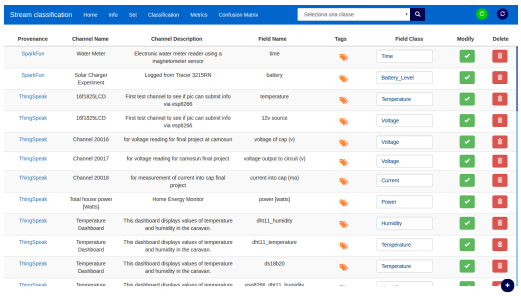
4. Visualizzare le classificazioni effettuate dall'algoritmo per ogni round della *k-fold validation*[6].

5. Visualizzare le metriche dell'algoritmo per ogni round della *k-fold validation*.

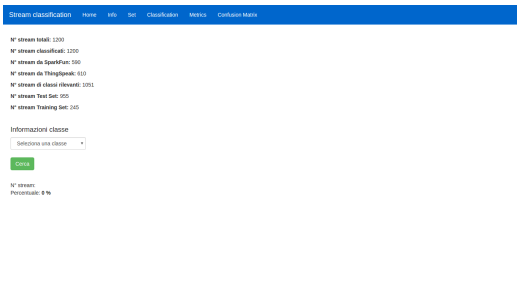
6. Visualizzare la matrice di confusione per ogni round della *k-fold validation*.

Relativamente alle ultime 3 sono state create anche le rispettive schermate che utilizzano solo dati delle classi rilevanti.
In figura. 5.5 sono mostrate alcune delle schermate.

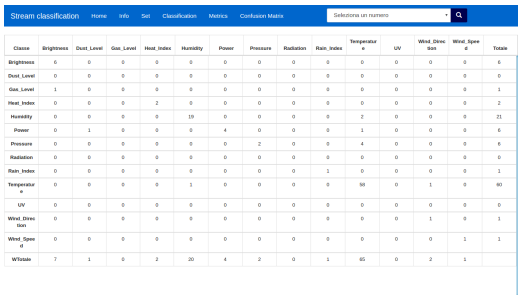
[5]https://angularjs.org/
[6]Ne parleremo in sezione 4.2.4



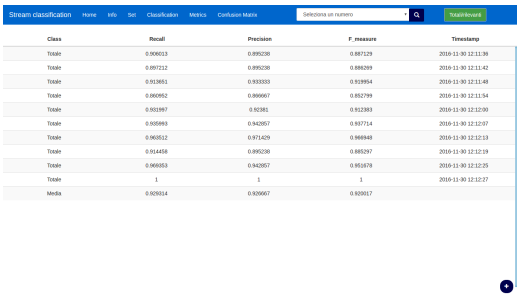
(a) Classificazione



(b) Info



(c) Confusion Matrix



(d) Metriche

Figura 4.1: Screenshot di alcune schermate dell'interfaccia di classificazione

4.2.4 Metriche

Per poter valutare la bontà e l'efficienza dell'algoritmo erano state precedentemente implementate le metriche: **Precision**, **Recall**, **F-measure**. Queste tre metriche sono strettamente correlate tra loro. I termini di misura utilizzati sono: numero di stream classificati in maniera corretta, numero di stream classificati in una certa classe, numero di stream realmente appartenenti ad una certa classe.

Precision

La precision nel contesto del *Data Mining* misura, relativamente ad una classe, il rapporto tra veri positivi (n° di volte che una classe è stata assegnata correttamente) e la somma tra veri positivi e falsi positivi (n° di volte che una classe è stata assegnata).

$$Precision = \frac{n^{\circ} veri\, positivi}{n^{\circ} veri\, positivi + n^{\circ} falsi\, positivi} \tag{4.1}$$

Il risultato della formula 4.1 è compreso tra 0 e 1 ed è considerato migliore quanto più si avvicina al valore 1.

Recall

La recall misura, relativamente ad una classe, il rapporto tra veri positivi e la somma tra veri positivi e falsi negativi (n° di volte che una classe è realmente presente nel set considerato).

$$Recall = \frac{n^{\circ} veri\, positivi}{n^{\circ} veri\, positivi + n^{\circ} falsi\, negativi} \tag{4.2}$$

Anche il risultato della formula 4.2 è compreso tra 0 e 1 ed è considerato migliore quanto più si avvicina al valore 1.

F-mesaure

Questa metrica rappresenta il rapporto tra le due appena descritte ed è data dalla formula 4.3.

$$Fmeasure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4.3}$$

Il risultato della formula 4.3 è compreso tra 0 e 1 ed è considerato migliore quanto più si avvicina al valore 1.

Nell'algoritmo precedente le metriche sopracitate venivano calcolate prima per ogni singola classe e successivamente ne veniva calcolata la media aritmetica al fine di ottenere le metriche totali. Tale media però non tiene conto del fatto che vi siano classi con un numero di stream nettamente superiore ad altre (*e.g.* il rapporto tra Temperature e RSSI è di 300:1). Dopo un'attenta analisi si è notato

che questa disparità influiva negativamente sulle metriche, in quanto le metriche relative alle classi con pochi stream risultavano molto inferiori rispetto a quelli delle classi più rappresentative. Si è perciò deciso di implementare la media ponderata, ponderando i risultati con il numero di stream effettivamente presenti nel dataset. Grazie a tale modifica si è notato come la bontà dell'algoritmo fosse in realtà migliore di quanto venisse mostrato con la precedente valutazione.

Matrice di confusione

Nell'ambito del *Data Mining*, la matrice di confusione (confusion matrix) restituisce una rappresentazione dell'accuratezza di una classificazione. Ogni colonna della matrice rappresenta i valori predetti, mentre ogni riga rappresenta i valori attuali. L'elemento sulla riga i e sulla colonna j è il numero di casi in cui l'algoritmo ha classificato la classe "vera" j come classe i. Attraverso questa matrice è osservabile se vi è "confusione" nella classificazione di diverse classi. In questa matrice i valori sulla diagonale rappresentano le classificazioni corrette, mentre tutti gli altri valori rappresentano errori. L'utilizzo della matrice di confusione mostra quali sono le classi che vengono più spesso confuse tra di loro, al fine di poterne analizzare i motivi per migliorare la classificazione.

Per implementare questa metrica è stata utilizzata la libreria **scikit-learn** [7], la quale consiste in un insieme di moduli Python per il machine learning e il data mining. Ai fini dell'applicazione è stato utilizzato il seguente metodo:

Codice 4.1: Algoritmo - Matrice di confusione

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_real,y_assigned,labels=all_labels)
```

dove **y_real** è il vettore che corrisponde ai valori attuali, nel quale per ogni stream classificato viene inserita la classe di appartenenza, mentre **y_assigned** è il vettore dei valori predetti, nel quale per ogni stream classificato viene inserita la classe attribuita dall'algoritmo. **all_labels** è un vettore contenente le classi su cui viene eseguito l'algoritmo.

[7] http://scikitlearn.org/stable/modules/classes.html

k-fold validation

Utilizzare un test ed un training set fissati è un buon modo per ottenere delle prime metriche di valutazione, ma spesso i risultati sono influenzati dalla composizione dei due insiemi e proprio per questo si rende necessario utilizzare un metodo di valutazione che prescinda dalla loro composizione.

La **k-fold validation** è un metodo di validazione degli algoritmi di classificazione che prevede di reiterare l'algoritmo per un numero di volte pari al parametro k. L'algoritmo viene eseguito sempre sullo stesso dataset ma il training ed il test set cambiano ad ogni iterazione e le metriche vengono prima valutate singolarmente per ogni iterazione e poi complessivamente attraverso la media aritmetica. Per ogni round le operazioni svolte sono le seguenti:

- Attraverso la funzione **azzera_metriche()** vengono inizializzate le strutture necessarie al calcolo delle metriche;
- Si calcolano gli indici su cui variare il dataset;
- Si utilizza la funzione **riempi_dizionario()** per istruire il dizionario tramite il training set;
- Viene lanciata la funzione **classificazione()** la quale prende in ingresso il test set e lo classifica sulla base delle informazioni contenute nel dizionario;
- Attraverso la funzione **verifica_metriche()** si calcolano *precision, recall* ed *f_measure* per ogni singola classe;
- Si utilizza la funzione **tot_metrics()** per calcolare la media ponderata di *precision, recall* ed *f_measure* valutate complessivamente.
- Si produce la matrice di confusione.

Infine per ogni round vengono salvate nel database la classificazione di ogni stream, le metriche e la matrice di confusione. A tale scopo sono stati aggiunti due attributi per ognuna delle relative tabelle:

- **Number.** Indica il numero del round. Risulta particolarmente utile per mostrare nell'interfaccia le informazioni relative ad ogni round.
- **Timestamp.** Riporta il timestamp in cui è iniziato il round.

Nell'implementazione di questo metodo abbiamo scelto *k=10* utilizzando ad ogni iterazione il 10% del dataset come test set ed il restante 90% come training set. Il codice che implementa la procedura è mostrato sotto.

Codice 4.2: Algoritmo - K-fold validation

```
for i in range(0,10):
  y_real = []
  y_assigned = []
  azzera_metriche()

  #Indici su cui variare il dataset
  min_index = (i*120)
  max_index = ((i+1)*120)

  riempi_dizionario(all_streams[:min_index] + all_streams[
      max_index:])
  test_set =  all_streams[min_index:max_index]

  #Esecuzione effettiva della classificazione
  classificazione(test_set)

  #Calcolo e salvataggio delle metriche
  metriche = verifica_metriche()
  tot_metriche(metriche)

  #Calcolo e salvataggio della matrice di confusione
  cm = confusion_matrix(y_real,y_assigned,labels=all_labels)
```

4.2.5 Classi rilevanti

Attraverso l'analisi delle matrici di confusione prodotte attraverso la **k-fold validation** è emerso che spesso l'algoritmo commetteva errori su classi che ap-

partenevano ad un numero esiguo di stream presenti nel database. Gli errori commessi causavano sia falsi positivi che falsi negativi e quindi influivano negativamente sia sul calcolo della *precision* che della *recall*. Per rimediare al problema si è quindi pensato di identificare un sottoinsieme delle classi che contenesse solo le classi più frequenti. Tale sottoinsieme contiene quindi le **classi rilevanti**, ovvero: Temperature, Dust_Level, Gas_Level, Brightness, Power, UV, Heat_Index, Pressure, Rain_Index, Radiation, Humidity, Wind_Direction, Wind_Speed. Questa decisione ha comportato anche un'importante modifica al database, sono infatti state aggiunte le tabelle:

- ClassificationRelevant.
- MetricsRelevant.
- ConfusionMatrixRelevant.

le cui funzionalità sono state descritte nella sezione 4.2.2.

4.2.6 Risultati

In tabella 4.1 vengono mostrate, per ogni metrica, la media e lo scarto quadratico medio. I risultati vengono mostrati sia per la classificazione svolta su tutte le classi che per quella svolta sulle sole classi rilevanti.

Tabella 4.1: Media e scarto quadratico medio dei risultati della 10-fold validation.

Tabella 4.2: Tutte le classi

Metrica	Media	S.Q.M.
Precision	0.88	0.04
Recall	0.89	0.04
F-Measure	0.87	0.05

Tabella 4.3: Classi rilevanti

Metrica	Media	S.Q.M.
Precision	0.93	0.04
Recall	0.93	0.04
F-Measure	0.92	0.04

Analizzando i risultati si vede come la precisione dell'algoritmo risulti piuttosto buona ed omogenea. Il valore dello scarto quadratico medio, prossimo allo

zero, evidenzia come vi siano poche disparità nei risultati pur prendendo test e training set differenti. Inoltre è possibile vedere come restringere la classificazione sulle sole classi rilevanti porti un incremento del 6% circa su tutte e tre le metriche.

4.2.7 Sviluppi futuri

Il lavoro svolto ha permesso di fare buoni passi avanti riguardo l'algoritmo, in particolare per quanto concerne le metriche. Attraverso il raffinamento di queste ultime si è notato come la precisione dell'algoritmo fosse in realtà migliore di quanto non risultasse con le metriche precedenti. L'algoritmo ha quindi dimostrato di lavorare con dei risultati piuttosto soddisfacenti anche se può ancora essere migliorato. Tale compito viene quindi lasciato come lavoro futuro. Una possibile evoluzione può essere quella di implementare una *bag of words* contenente i termini presenti nella descrizione dei canali associati agli stream e nei tag associati ai canali. Questa struttura dovrebbe poi essere utilizzata per riclassificare gli stream. Un'ulteriore evoluzione può poi essere rappresentata dall'analisi dei valori associati agli stream, cercando di inferire quale sia il valore che misura lo stream.

4.3 Integrazione con SenSquare

Essendo stata raggiunta una precisione piuttosto buona si è deciso di fare un primo passo di integrazione con l'architettura SenSquare. Lo scopo era quello di popolare il database con un insieme di dati reali per poter effettuare dei primi test sulle performance e sull'usabilità della piattaforma Crowdservice.

Nel particolare è stato scritto uno script Python che effettua una chiamata GET all'indirizzo *https://thingspeak.com/channels/public.json*, la quale ritorna tutti i canali pubblici di ThingSpeak. Dopodiché viene analizzata la geolocalizzazione dei canali e vengono scartati i canali non italiani[8]. Per ogni canale che si trova

[8] A causa del limite di chiamate imposto dalle API di Google la scrematura avviene attraverso un range riguardante latitudine e longitudine.

in Italia vengono estratti gli stream relativi i quali sono poi dati in pasto all'algoritmo; una volta identificata la classe lo stream viene salvato nel database di SenSquare e per ognuno di essi vengono salvate anche le misurazioni relative.

Capitolo 5

La Piattaforma

Come abbiamo descritto nel capitolo 3, SenSquare è formata da due componenti principali, da una parte la CCU, composta da un database MySQL e da più server “satellite”, dall’altra un insieme di applicazioni con funzionalità diverse. Scopo di questo capitolo è illustrare il processo di progettazione e sviluppo di Crowdservice un’applicazione web che permette a chiunque sia interessato di definire ed istanziare un proprio servizio, sfruttando gli stream presenti nel database.

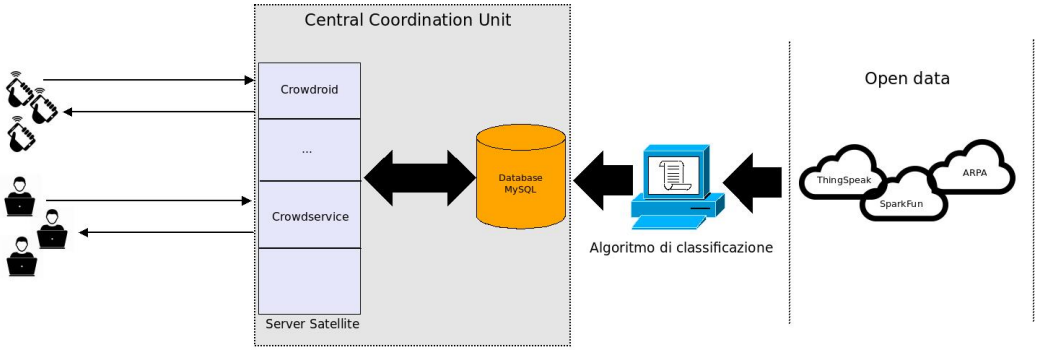


Figura 5.1: Architettura di SenSquare.

5.1 Specifiche

Crowdservice vuole permettere agli utenti di utilizzare i dati disponibili in SenSquare per definire degli schemi di servizio. Uno schema è definito tramite un'espressione aritmetica in cui le variabili sono rappresentate dalle 32 classi attribuite dall'algoritmo descritto precedentemente e da altre classi relative a dati provenienti da enti governativi.

Lo schema di un servizio viene creato da un singolo utente ma viene messo a disposizione di tutti gli altri.

Chiunque deve essere in grado di definire, in maniera semplice e veloce, i propri servizi, i quali possono successivamente essere istanziati in un determinato luogo. Una volta scelto il luogo, vengono mostrati gli stream disponibili nel minor raggio possibile.

L'utente sceglie quali stream utilizzare ed istanzia il servizio seguendone gli aggiornamenti.

Il progetto è diviso in due parti: server e client web. Il client web offre un'interfaccia tramite la quale è possibile registrarsi, creare schemi ed istanziarli come servizi. Il server gestisce l'autenticazione, le operazioni CRUD e le query di ricerca.

5.2 Server

Il server è uno dei componenti della CCU e quindi si interfaccia direttamente con il database MySQL, fungendo da tramite per il client. È necessario che questo componente sia solido, indipendente dalla parte client e in grado gestire tutte le richieste ricevute.

Si è voluto sviluppare un servizio REST completamente svincolato dal client, in modo da rendere l'applicazione scalabile ed eventualmente estendibile attraverso la creazione di un'app per dispositivi mobili.

L'intero server è stato sviluppato in Python 2.7, utilizzando due framework che si sono rivelati estremamente utili in termini di semplificazione dello sviluppo

e risparmio di tempo:

- **Django**[1]**.** Web framework Open Source che semplifica alcuni aspetti dello sviluppo web, come il routing e l'interfacciamento con il database.

- **Django-rest-framework (DRF)**[2]. Web framework Open Source che estende Django e permette di definire servizi RESTful.

5.2.1 Django-rest-framework.

Questo framework risolve molte delle problematiche relative all'autenticazione e al controllo dei permessi di un utente su un dato insieme di oggetti. DRF mette a disposizione tutta una serie di metodi e classi per mappare i metodi HTTP (GET, POST, PUT, DELETE) con le funzioni CRUD del database. Questo avviene attraverso i **Serializers** e le **View**.

I serializzatori permettono di mappare dati complessi (come queryset e istanze di modelli[3]) in tipi nativi del Python, i quali sono facilmente traducibili in JSON, XML o altri *content types*. È possibile inoltre effettuare l'operazione inversa. Viene qui mostrato il codice relativo al serializzatore dei servizi isantziati.

Codice 5.1: Server - Esempio di Serializzatore

```
from rest_framework import serializers
from services.models import Customservices

class CustomservicesSerializer(serializers.ModelSerializer):
  participant_id = serializers.ReadOnlyField(source='
     participant_id.id')
  deployment_radius= serializers.FloatField(required=False)

  class Meta:
    model = Customservices
```

[1]https://www.djangoproject.com/

[2]http://www.django-rest-framework.org/

[3]Un modello in Django rappresenta generalmente una tabella del database di riferimento. Mettono a disposizione metodi per interfacciarsi al database senza dover scrivere query SQL.

```
    fields = ('id', 'participant_id','deployment_center_lat', '
       deployment_center_lon', 'deployment_radius', 'template_id
       ','title')
```

Le views implementano le funzioni che il server svolge ad ogni chiamata ricevuta da un client, si dividono in: **Generic views** e **Viewsets**. Entrambe permettono di implementare velocemente le funzioni CRUD, ma con le prime è necessario scrivere una view per ogni differente operazione. Le seconde invece permettono di gestire tutto in un'unica funzione.

Codice 5.2: Server - Esempio di modello

```
from services.models import Multisensors
from services.serializers import MultisensorsSerializer
from rest_framework import generics,permissions, viewsets
from rest_framework_jwt.authentication import
   JSONWebTokenAuthentication
from permissions import IsOwnerOrReadOnly

class MultisensorsViewSet(viewsets.ModelViewSet):
    authentication_classes = (JSONWebTokenAuthentication,)
    permission_classes = (IsOwnerOrReadOnly,)
    queryset = Multisensors.objects.all()
    serializer_class = MultisensorsSerializer
```

Il codice mostrato sopra corrisponde alla view relativa alla tabella **MultiSensors** del database, la quale mette in relazione l'istanza di un servizio con un insieme di stream. Attraverso questa view è possibile effettuare tutte le operazioni CRUD sulla tabella **MultiSensors**. A seconda delle operazioni che si intende gestire una view deve essere definita come sottoclasse di una particolare superclasse. Le superclassi disponibili sono:

- *ViewSet.* Non implementa alcuna funzione autonomamente, ma permette di definirle una per una.

- *ModelViewSet.* Implementa autonomamente tutte le funzioni CRUD. Possono essere ridefinite secondo le necessità specifiche.

- *ReadOnlyModelViewSet.* Implementa solo le funzioni in lettura (GET) sul singolo elemento o su una lista. Possono essere ridefinite secondo le necessità specifiche.

All'interno di una view è necessario soltanto definire i permessi, il modello su cui agire e il serializzatore[4] che deve essere utilizzato per impacchettare i dati da tornare al client.

5.2.2 Autenticazione e permessi

Le prime due righe del codice 5.2 servono per gestire l'autenticazione ed i permessi, due delle funzioni principali di cui si occupa il server.

Volendo rispettare i requisiti di scalabilità e di servizio RESTful si è scelto di gestire l'autenticazione tramite **JSON Web Token (JWT)**[5], invece che con le tradizionali sessioni o tramite cookie. Infatti con i JWT non si pone il problema di dover memorizzare e ritrovare i dati di sessione all'interno del database, snellendo in maniera significativa la comunicazione con esso. DRF mette a disposizione un modulo specifico per l'utilizzo dei JWT.

Lo scenario d'esecuzione è il seguente:

1. L'utente chiama il servizio di autenticazione fornendo username e password.

2. Il servizio di autenticazione verifica le credenziali ed in caso di successo restituisce un JWT firmato[6].

3. Ad ogni nuova richiesta l'utente invia il token ed un servizio middleware ne verifica autenticità e validità.

Attraverso l'id presente nel payload del token è possibile anche gestire i permessi sui singoli oggetti del database. DRF permette di definire delle funzioni

[4]Generalmente il serializzatore utilizzato è quello relativo al modello di riferimento.

[5]https://jwt.io/

[6]Il payolad dei token generalmente contiene l'id dell'utente ed una data di scadenza (al termine della quale deve essere richiesto un nuovo token), ma può contenere anche altre informazioni. Viene firmato mediante un algoritmo di cifratura come SHA,MD5 ed altri

middleware che verificano che l'oggetto su cui l'utente vuole effettuare un'operazione sia effettivamente di sua proprietà. Di seguito è mostrato il codice che implementa il permesso *IsOwnerOrReadOnly*, il quale concede, relativamente ad un oggetto, il diritto in lettura a tutti gli utenti ma in scrittura solo al proprietario.

Codice 5.3: Server - Esempio di permesso

```
from rest_framework import permissions

class IsOwnerOrReadOnly(permissions.BasePermission):
    #Object-level permission to only allow owners of an object
       to edit it.
    def has_object_permission(self, request, view, obj):
        # Read permissions are allowed to any request,
        if request.method in permissions.SAFE_METHODS:
            return True

        return obj.participant_id == request.user
```

5.2.3 Moduli

I moduli che compongono il server di Crowdservice sono i seguenti:

- **settings.py** file di configurazione del server in cui vengono specificati alcuni parametri tra cui quelli relativi al database e ai JWT;
- **models.py** definisce i modelli dell'applicazione, mappandoli con quelli del database;
- **serializers.py** definisce i serializzatori;
- **permissions.py** contiene il codice che implementa i permessi sugli oggetti;
- **views.py** definisce le view. La maggior parte di queste forniscono solo le operazioni CRUD e sono quindi sviluppate tramite viewsets. La view più interessante è quella che ritorna gli stream disponibili per l'istanziazione di un certo servizio;

- **urls.py** definisce il routing delle chiamate al server. Ogni indirizzo viene mappato con una specifica view. Se viene inserito un indirizzo non presente nella lista ritorna la pagina *index.html*.

Ricerca degli stream. La funzione che effettua la ricerca degli stream è implementata mediante una viewset che prende in ingresso latitudine, longitudine ed un array con le classi presenti nell'espressione relativa allo schema di riferimento. Cerca gli stream necessari nel minimo raggio possibile, partendo da un raggio minimo di 100 metri, raddoppiato ad ogni iterazione fino ad un massimo di 50 chilometri . Viene eseguito un ciclo while che termina nel caso in cui siano state trovate tutte le classi necessarie o sia stato raggiunto il massimo valore possibile per il raggio.

Gli stream trovati vengono tornati all'interno di un array in formato JSON.

5.3 Client Web

Lo scenario d'utilizzo tipico di Crowdservice è il seguente:

1. L'utente effettua la registrazione o il login;
2. L'utente cerca lo schema d'interesse tra quelli già disponibili. Se non lo trova allora crea il proprio schema;
3. Una volta scelto uno schema, l'utente decide dove istanziarlo;
4. L'utente sceglie quali stream utilizzare tra quelli disponibili ed istanzia il servizio;
5. L'utente controlla i valori calcolati dal servizio.

L'interfaccia vuole quindi permettere di gestire il flusso di esecuzione in maniera semplice ed intuitiva. È stata sviluppata attraverso i framework Javascript:

- **Angular.** Permette di creare applicazioni moderne che sfruttano le più recenti tecnologie, come ad esempio le **Single Page Application**, cioè applicazioni le cui risorse vengono caricate dinamicamente su richiesta, senza necessità di ricaricare l'intera pagina. Implementa inoltre il binding bidirezionale e supporta il pattern *Model View Controller (MVC)*.

- **Angular Material**[7]**.** Estende Angular mettendo a disposizione una serie di componenti grafici basati sul *Matrial Design.*

5.3.1 Model View Controller

MVC è un pattern che permette di organizzare il codice dividendolo in tre parti:

- **Model:** fornisce i metodi per accedere ai dati dell'applicazione;

- **View:** si occupa di mostrare i dati all'utente e gestisce l'interazione fra quest'ultimo e l'infrastruttura sottostante;

- **Controller:** riceve i comandi dell'utente attraverso la View e reagisce eseguendo delle operazioni che possono interessare il Model e che portano generalmente ad un cambiamento di stato della View.

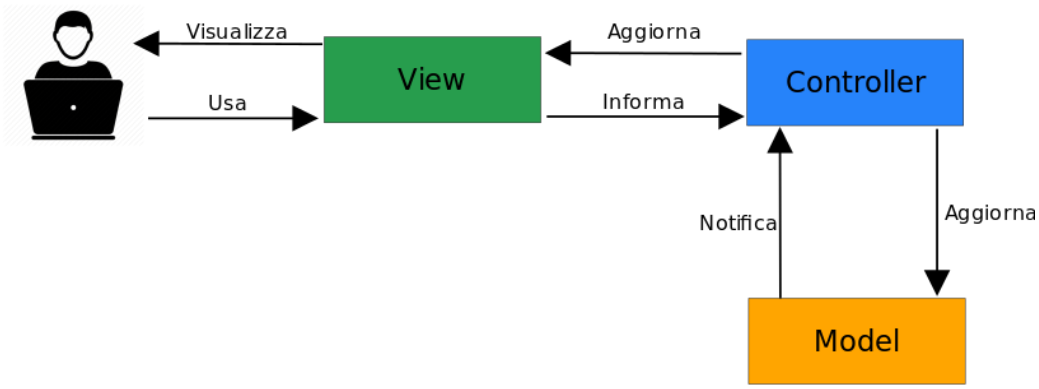


Figura 5.2: Schema Model View Controller.

[7]https://material.angularjs.org/latest/

5.3.2 Material Design

Il Material Design è una nuova forma di design per app ed interfacce web sviluppata da Google. Matìas Duarte, il designer che ha gestito il processo creativo che ha portato alla teorizzazione ed alla realizzazione del Material Design ha spiegato che "Proprio come la carta, il nostro materiale digitale si può espandere o restringere riformandosi in modo intelligente. I materiali hanno superfici fisiche e bordi. Cose come ombre e cuciture forniscono il significato di quello che tocchi."

Il Material Design evolve il **flat design** unendo ad esso la metafora del materiale e quindi, di fatto, inserendo un po' di **scheumorfismo**[8] in tutto ciò. Vuole quindi diventare un punto di riferimento per la creazione e l'aggiornamento di app web e mobile. In questo modo, nel passaggio tra diverse piattaforme e dispositivi, l'esperienza rimane unificata.

Angular Material mette a disposizione dello sviluppatore tutta una serie di tag che permettono di creare oggetti HTML disegnati secondo i concetti del Material Design. Attraverso questo framework si è quindi cercato di creare un'interfaccia piacevole ed intuitiva per l'utente.

5.3.3 Moduli

Il client è stato suddiviso in quattro moduli, ognuno dei quali è composto da una View e da un Controller che agiscono su un particolare modello, secondo le regole del MVC.

Home

Questo modulo gestisce la schermata home della piattaforma, la quale spiega in breve il funzionamento della piattaforma. Contiene un breve tutorial che mostra il funzionamento della piattaforma.

[8]Uno scheumorfismo è un ornamento fisico o grafico apposto su un oggetto allo scopo di richiamare le caratteristiche estetiche di un altro.

Premendo l'icona dell'utente posta nella **navbar** compare un menù a tendina dal quale è possibile effettuare il login o la registrazione. Se il login va a buon fine il server ritorna un token che viene salvato nel browser e viene inserito nell'header di tutte le chiamate al server che richiedono autenticazione.

Schema List

Questo modulo gestisce la visualizzazione degli schemi istanziati dagli utenti e permette all'utente di definire un proprio schema.

La view del modulo è composta da una **navbar** posta in alto, una **sidebar** posta sulla sinistra e da un corpo centrale nel quale vengono mostrati tramite **tile** tutti gli schemi presenti nel database di SenSquare. Uno screenshot è mostrato in figura 5.3.

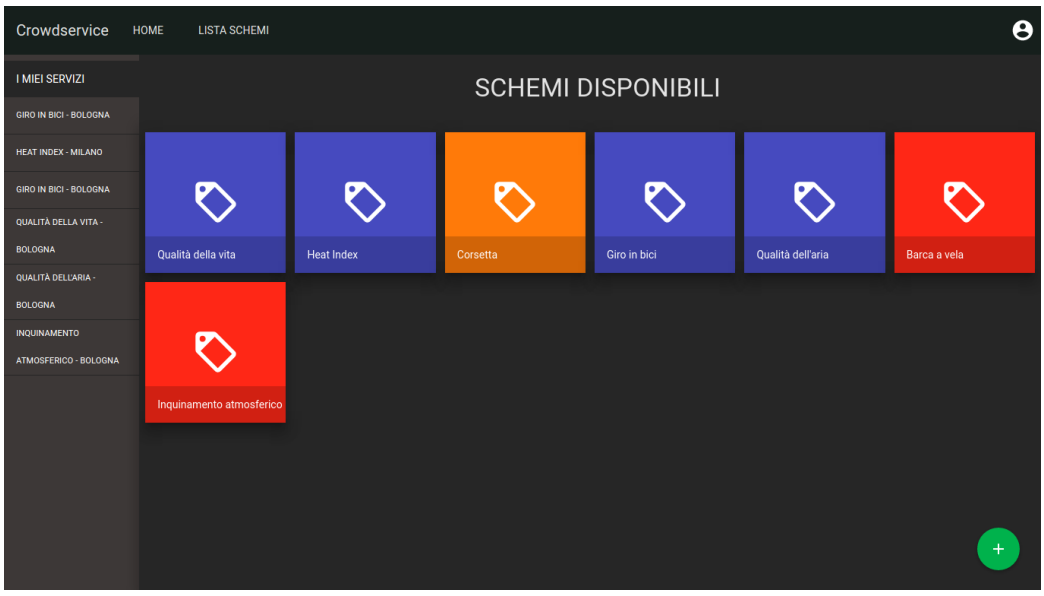


Figura 5.3: Schermata lista degli schemi.

È la schermata che viene mostrata all'utente una volta effettuato il login. Al caricamento della schermata il controller si occupa di effettuare una chiamata Ajax

al server richiedendo tutti gli schemi disponibili, i quali una volta ricevuti vengono inseriti in **vm.templates[]**, l'array che contiene i modelli di schema, il quale tramite binding viene associato alla lista di tile mostrata all'utente.

Codice 5.4: Web Client - Visualizzazione dei servizi in tile

```
djangoData.getServiceTemplates().success(function(data) {
  for(template in data){
    span = randomSpan()
    tileTemplate = {
      id: data[template].id,
      title: data[template].title,
      color: randomColor(),
      rowspan: span,
      colspan: span,
    }
    vm.templates.push(tileTemplate)
   }
}).error(function (e) {
     alert('Problema nel reperire gli schemi');
});
```

Espressioni. All'interno di uno schema, l'espressione rappresenta ciò di cui si vuole tener traccia. È costruita e memorizzata come un'espressione aritmetica le cui variabili sono le classi a cui appartengono gli stream. All'interno di un'espressione possono essere presenti, oltre alle variabili, anche delle costanti.

Composizione delle espressioni

- L'espressione è costruita componendo una o più espressioni binarie;
- Ogni espressione binaria è composta da due valori (variabili e/o costanti) uniti mediante un operatore aritmetico. Può essere composta anche da un unico valore;
- Le espressioni binarie sono a loro volta unite tra loro tramite operatori aritmetici;

- Un'espressione deve essere composta da almeno un'espressione binaria e da massimo cinque.

Nella parte in basso a destra della schermata è presente un **fabButton** al click del quale appare all'utente il dialog mostrato in figura 5.4. Attraverso il dialog si può creare un nuovo schema di servizio definendone il titolo, il tipo[9] e l'espressione. Sono presenti inoltre i bottoni "Aggiungi Regola" ed "Elimina ultima regola" che permettono, rispettivamente, di aggiungere un'espressione binaria o di eliminare l'ultima che è stata inserita.

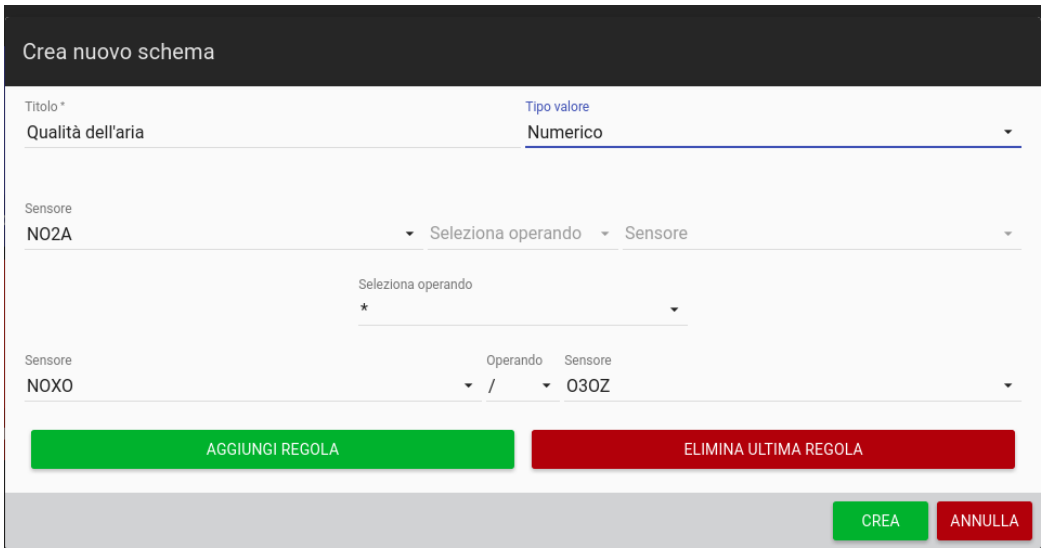


Figura 5.4: Dialog per la creazione di uno schema.

Dopo aver cliccato il bottone di conferma viene effettuato un controllo sintattico sull'espressione, al superamento del quale viene effettuata una POST asincrona al server per salvare lo schema nel database.

[9] Attualmente il tipo delle espressioni è solo numerico, ma in futuro potrebbe essere esteso anche ad un tipo booleano.

Schema Detail

Questo modulo permette all'utente di cercare e scegliere gli stream necessari all'istanziazione dello schema in un determinato luogo, nonché di istanziare effettivamente lo schema.

Il modello di riferimento è il singolo schema. Al caricamento della view il controller si occupa di richiedere al server i dati relativi allo schema in esame (titolo ed espressione). Al ricevimento dei dati il controller si occupa di fare il parsing dell'espressione per individuare quali sono le classi che la compongono.

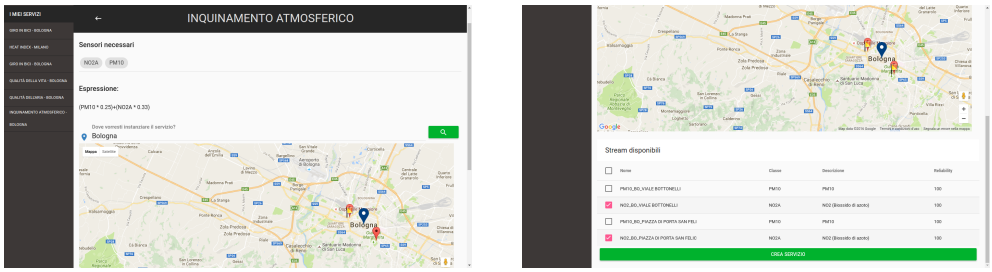


Figura 5.5: Screenshot della schermata di dettaglio di uno schema

Nella view è presente una barra di ricerca all'interno della quale l'utente digita un indirizzo. L'indirizzo viene poi convertito in coordinate GPS, attraverso le *Google Maps APIs*. L'utente può inoltre individuare un luogo cliccando direttamente in un punto specifico della mappa. Le coordinate e l'insieme delle classi necessarie vengono poi utilizzate per compiere una GET al server che effettua la query che ricerca gli stream descritta nel paragrafo 5.2.3. La chiamata ritorna un array di stream. Per ogni stream tornato viene creato un **marker** all'interno della mappa, al quale è associato una **infoWindow** contenente le informazioni relative ed un bottone mediante il quale è possibile scegliere di aggiungere/rimuovere lo stream dalla lista degli stream che l'utente vuole utilizzare per il proprio servizio.

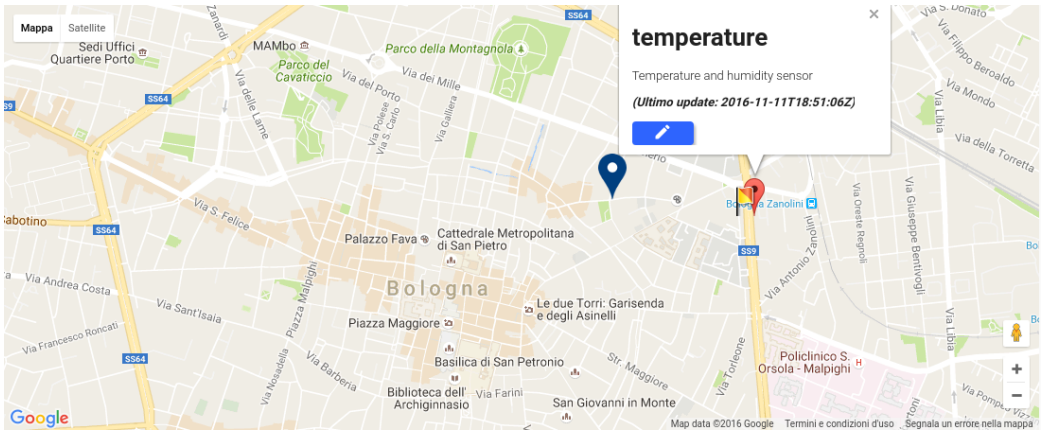


Figura 5.6: Marker con infowindow.

Gli stream sono inoltre mostrati in una tabella collocata sotto la mappa. È possibile selezionare gli stream anche attraverso la tabella. Ogni volta che uno stream viene aggiunto alla mappa viene modificata l'icona relativa al marker che rappresenta lo stream. L'icona viene modificata nuovamente se si decide di deselezionarlo.

Per istanziare il servizio si preme il bottone "Istanzia Servizio", al click del quale viene controllato che per ogni classe necessaria sia stato selezionato uno ed un solo stream. Se il controllo è superato si procede all'istanziazione attraverso una chiamata POST al server, in caso negativo viene mostrato un messaggio d'errore.

Service Detail

Questo modulo permette all'utente di tenere traccia dei propri servizi. Il modello di riferimento è rappresentato dall'istanza di un servizio creato dall'utente. Al caricamento della view il controller effettua una GET al server per ricevere i dati relativi al servizio. Il server ritorna i dati solo se l'utente che fa la richiesta è il proprietario del servizio.

All'interno della view vengono mostrati:

- l'espressione dello schema di cui il servizio è istanza;

- il valore dell'espressione calcolato sulla base dell'ultimo aggiornamento degli stream scelti;
- una tabella con le informazioni degli stream utilizzati, i quali vengono anche geolocalizzati su mappa tramite i marker.

Calcolo dell'espressione. Il calcolo dell'espressione avviene lato client attraverso la libreria *math.js*[10]. Nel particolare viene fatto uso della funzione **eval** la quale prende in ingresso un'espressione ed un contesto (valore delle variabili) e ne calcola il risultato. Viene qui mostrata una porzione di codice utile a chiarire meglio il funzionamento.

Codice 5.5: Web Client - Valutazione di un espressione

```
var scope = {
  Temperature: 20,
  Humidity: 50
};

var res = math.eval('(Temperature * Humidity) / 10',scope);
console.log(res) //100
```

Direttive Angular

Le direttive in Angular rappresentano dei componenti grazie ai quali possiamo creare nuovi attributi per tag HTML esistenti o creare dei tag HTML completamente nuovi. Nel caso in esame sono state create due differenti direttive: **navbar**, **sidebar**.

Navbar è il componente collocato in alto all'interno di ogni view. Contiene alcuni link ed un'icona al click della quale è possibile effettuare il login o la registrazione. La direttiva prende in ingresso un parametro, **location**, utilizzato per mostrare link differenti a seconda che l'utente si trovi nella pagina home o in un'altra pagina.

[10] http://mathjs.org/index.html

Sidebar è il componente collocato sulla sinistra in ogni view esclusa la home. Al caricamento della view il controller della sidebar effettua una chiamata GET per ricevere la lista dei servizi istanziati dall'utente e mostrarli. Al click di uno di essi si viene reindirizzati alla pagina di dettaglio descritta sopra.

Servizi Angular

I servizi in Angular permettono di condividere dati e funzioni tra più controller. Per la realizzazione di questo progetto sono stati creati due servizi:

- **authentication** si occupa di gestire le problematiche relative all'autenticazione. Nel particolare attraverso questo servizio è possibile salvare, rimuovere e ritrovare il JWT. Fornisce inoltre le funzioni per registrarsi e ed autenticarsi con il server.
- **djangoData** fornisce ai controller tutte le funzioni per interagire con il server, occupandosi di inserire il JWT in tutte le chiamate che richiedono autenticazione.

Il codice 5.6 mostra come viene effettuata una chiamata che richiede autenticazione. Il JWT è recuperato attraverso il servizio **authentication**.

Codice 5.6: Web Client - Esempio di chiamata con autenticazione

```
var getServices = function () {
   return $http.get('/api/services/', {
     headers: {
       Authorization: 'JWT '+ authentication.getToken()
     }
   });
};
```

5.3.4 Sviluppi futuri

Ci sono vari aspetti che non sono stati presi in considerazione in questo progetto, ma che potrebbero ampliare questo sistema per generare una piattaforma più completa.

Testing intensivo

Sulla piattaforma in esame sono stati condotti alcuni test al fine di individuare eventuali bug nelle funzioni create. Mancano però dei test mirati ad individuare eventuali anomalie in un contesto reale e con un ampio numero di utenti che usufruiscono contemporaneamente della piattaforma. Uno dei problemi che si potrebbero incontrare in questa direzione è la scarsa densità di stream attualmente presenti nel territorio italiano, ma considerando il trend di crescita del crowdsensing in Italia, ciò non dovrebbe più comportare un problema nel futuro.

Implementazione app mobile

Come spiegato nella sezione 5.2 si è deciso di sviluppare un servizio RESTful che fosse svincolato dal client. Il motivo di tale scelta è stato proprio quello di poter permettere lo sviluppo di client differenti senza la necessità di dover riscrivere da capo il server.

Uno dei primi sviluppi della piattaforma potrebbe dunque essere l'implementazione di un'app per smartphone e tablet, sia Ios che Android. L'app potrebbe inoltre svolgere compiti più complessi del client per browser. A titolo di esempio potremmo immaginare un'app del tipo *If This Then That (IFTT)*. Un possibile caso d'uso potrebbe essere quello di uno sportivo a cui piace andare a correre la mattina, ma solo in presenza di determinate condizioni meteorologiche (*e.g* non piove, poco vento, pochi pollini nell'aria...). Lo scenario d'esecuzione potrebbe essere il seguente:

1. L'utente istanzia un servizio relativo ad alcune condizioni meteorologiche nel tragitto che intende percorrere.

2. Prima di coricarsi l'utente imposta l'app richiedendo di analizzare il valore del servizio la mattina successiva.

3. Se il valore indica che ci sono condizioni favorevoli per andare a correre allora l'app farà suonare la sveglia all'orario impostato.

4. Altrimenti lascerà che l'utente si riposi ancora facendo suonare la sveglia più tardi, evitandogli di alzarsi in anticipo senza motivo.

La stessa app potrebbe inoltre essere utilizzata dall'utente durante la propria corsa per rilevare alcuni valori attraverso i sensori dello smartphone, inviandoli alla CCU per essere processati ed utilizzati in futuro, stimolando in questo senso anche la partecipazione al Mobile Crowdsensing.

Potenziamento delle espressioni

Al momento non è possibile costruire espressioni booleane, le quali ritornando un valore di verità anziché numerico potrebbero ampliare ulteriormente il campo d'azione di Crowdservice. Il problema però non è banale, in particolare si deve tener di conto di due criticità principali: da una parte lo sviluppo di un componente dell'interfaccia che permetta a chiunque (anche e soprattutto persone senza competenze nel campo della programmazione) di costruire delle condizioni formate da valori ed operatori sia aritmetici che booleani, dall'altra l'implementazione di una funzione in grado di effettuare il parsing e calcolare il risultato delle espressioni così create. Il metodo utilizzato al momento infatti funziona nella risoluzione di espressioni composte solo con operatori aritmetici, ma non è in grado si risolvere espressioni contenenti gli operatori booleani (*i.e.* and, or, not) e/o gli operatori di confronto (*i.e.* $<$, $>$, =).

Conclusioni

In questo studio abbiamo innanzitutto introdotto il concetto di IoT ed IoT collaborativo. È stato poi introdotto il paradigma del crowdsensing come forma specifica del concetto di crowdsourcing. Nel particolare abbiamo cercato di individuare quali sono le tipologie di crowdsensing e quali sono i punti su cui lavorare affinché tale paradigma possa perdurare nel tempo producendo risultati utili. Abbiamo inoltre analizzato quali sono al momento le applicazioni già sviluppate che sfruttano questo paradigma.
Dopodiché è stata illustrata l'architettura di SenSquare, mostrandone i componenti principali, ovvero CCU e database, ed un'applicazione già esistente, Crowdroid, che lavora nell'ambito del Mobile Crowdsensing.
Abbiamo poi analizzato il lavoro svolto per perfezionare l'algoritmo di classificazione utilizzato da SenSquare per raccogliere dati dalle *Open Data Platforms*, illustrando punto per punto le modifiche apportate. In questo senso ci siamo concentrati soprattutto nel cercare di creare un ambiente che permettesse di lavorare sull'algoritmo in maniera snella ed infine è stato migliorato il calcolo delle metriche per poter avere valutazione più precise.
Infine abbiamo introdotto Crowdservice, una piattaforma che sfrutta i dati disponibili in SenSquare e li mette a disposizione degli utenti, i quali possono utilizzarli per creare servizi personalizzati. Della piattaforma abbiamo analizzato sia il server che il client, facendo inoltre una breve introduzione alle tecnologie utilizzate.
Come già visto sopra, non esiste una piattaforma che permetta a chiunque di utilizzare dati sensoristici per creare dei servizi personalizzati. Esiste un'applicazione simile, ma i servizi possono essere definiti solo da persone con competenze nel

campo della programmazione, un numero di utenti troppo limitato. Un altro obiettivo è stato quello di rendere l'interfaccia di Crowdservice piacevole ed intuitiva, proprio per poter invogliare il maggior numero di persone possibile ad utilizzarla. Sono infine stati proposti alcuni possibili miglioramenti che potrebbero essere effettuati sia sull'algoritmo che sulla piattaforma, in modo da renderla sempre più affidabile e completa per l'utente finale.

Gli obiettivi che erano stati prefissati sono quindi stati raggiunti, ed il lavoro svolto ha quindi contribuito ad incrementare le potenzialità ed il valore di SenSquare, la quale, pur offrendo già servizi distinti, può ancora crescere molto e diventare un'architettura centrale nello scenario del Crowdsensing.

Bibliografia

[1] Michael Chui, Markus Löffler e Roger Roberts. *The Internet of Things*. URL: http://www.mckinsey.com/industries/high-tech/our-insights/the-internet-of-things.

[2] Rob van der Meulen. *Gartner Says 6.4 Billion Connected Things Will Be in Use in 2016, Up 30 Percent From 2015*. URL: http://www.gartner.com/newsroom/id/3165317.

[3] Vernor Vinge, cur. *Who's Afraid of First Movers?* San Francisco, CA, USA. URL: https://docs.google.com/document/d/1W7Vs8e3MNwjU_1ZZZqmq0O-dFKCwD4VRyww-qOvDdRU/pub.

[4] J. Burke, M. Estrin D.and Hansen, A. Parker, N. Ramanathan, S. Reddy e Srivastava M. B. "Participatory Sensing". In: *Center for Embedded Networked Sensing (CENS)* (2006).

[5] Silvia Perrino. "Internet of Things collaborativo: progettazione ed analisi di una piattaforma di aggregazione di dati sensoristici". Università di Bologna, 2016.

[6] Carlo Di Benedetti. *Perché l'≪Internet delle cose≫ è la frontiera della crescita*. URL: http://www.ilsole24ore.com/art/commenti-e-idee/2016-04-27/perche-l-internet-cose-e-frontiera-crescita--073803.shtml?uuid=ACFKJPGD&refresh_ce=1.

[7] Fastweb. *Cos'è e come funziona il crowdsourcing*. URL: http://www.fastweb.it/social/cos-e-il-crowdsourcing-nuova-frontiera-dei-rapporti-lavorativi/.

[8] David Bratvold. *What is Crowdsourcing?* URL: https://dailycrowdsource.com/training/crowdsourcing/what-is-crowdsourcing.

[9] Daniel Dimov. *Crowdsensing: State of the Art and Privacy Aspects*. URL: http://resources.infosecinstitute.com/crowdsensing-state-art-privacy-aspects/.

[10] Bin Guo, Zu Whang, Zhiwen Yu, Yu Wang, Neil Y. Yen, Runhe Huang e Xingshe Zhou. "Mobile Crowd Sensing and Computing: The Review of an Emerging Human-Powered Sensing Paradigm". In: *ACM Computing Surveys* (2015).

[11] Nicolas Haderer, Romain Rouvoy, Christophe Ribeiro e Lionel Seinturier. *APISENSE: Crowd-Sensing Made Easy*. URL: http://ercim-news.ercim.eu/en93/special/apisense-crowd-sensing-made-easy.

[12] Alain Di Chiappari. "A Collaborative Mobile Crowdsensing system for Smart Cities". University of Bologna, 2016.

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

[14] Massimo Aria. *Introduzione alla cluster analysis*. URL: http://www.federica.unina.it/economia/analisi-statistica-sociologica/introduzione-cluster-analysis/.

[15] Fred J Damerau. "A technique for computer detection and correction of spelling errors." In: *Communications of the ACM* (1964).

[16] *Introduzione a JSON*. URL: http://www.json.org/json-it.html.

[17] Federico Montori, Luca Bedogni e Luciano Bononi. “On the Integration of Heterogeneous Data Sources for the Collaborative Internet of Things”. In: *IEEE RTSI 2016*.

Ringraziamenti

Ringrazio innanzitutto i miei genitori, Paolo e Giovanna, che mi hanno permesso di intraprendere questo percorso, sperando di poter ripagare un giorno tutto il sacrificio e il supporto che mi hanno sempre dimostrato.

Un ringraziamento speciale va a Beatrice che mi ha supportato e sopportato durante l'intero percorso.

Voglio inoltre ringraziare i miei coinquilini Luca, Antonio e Luca,con i quali ho condiviso gioie e dolori di questi ultimi tre anni.

Un sentito ringraziamento va inoltre ai miei compagni di corso Natale, Giovanni e Andrea, che mi hanno dato ed insegnato molto in questi tre anni e non solo in campo informatico.

Infine voglio ringraziare il professor Bononi per essere stato relatore di questa tesi, ma soprattutto ringrazio il dottor Luca Bedogni ed il dottor Federico Montori, per l'idea ed il prezioso aiuto ricevuto durante la realizzazione e la stesura di questa mia tesi.