Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Informatica

SenSquare: una piattaforma IoT di crowdsensing e sviluppo collaborativo di servizi personalizzati

Relatore:
Chiar.mo Prof.
Luciano Bononi

Presentata da:
Gianluca Iselli

Correlatori:
Dott. Luca Bedogni
Dott. Federico Montori

Sessione II
Anno Accademico 2016/2017

Abstract

Non è un segreto, l'Internet of Things si sta diffondendo a macchia d'olio e sta cambiando il nostro modo di vivere e di lavorare. Device eterogenei considerati **smart** collezionano e scambiano dati in ogni istante. L'IoT permette a persone, aziende, comunità e stati di trasformare questi dati in informazioni preziose. Inoltre, il continuo abbassarsi dei costi dei device, della bandwidth e del processing agevola l'interconnessione di sempre più "Things". Come risultato, esperti del settore hanno stimato che entro il 2020 ci saranno 212 miliardi [1] di sensori connessi a 50 miliardi [2] di device. Ciò denota che l'IoT offre e offrirà incredibili opportunità per chi è nel settore.

L'attenzione di questa tesi si focalizza sullo sviluppo collaborativo di servizi personalizzati, utilizzando fonti aggregate provenienti da risorse affidabili, non affidabili e da campagne di crowdsensing.

Il contributo di questo lavoro verterà sullo sviluppo, nella sua interezza, del Front End e del Back End della Web Application chiamata SenSquare. SenSquare si pone come obiettivo quello di stimolare l'utente ad usare, per scopi nobili, i dati eterogenei presenti in cloud attualmente non pienamente utilizzati. Tra queste ragioni si può citare la creazione di servizi per motivi comunitari, quali la qualità della vita, la salvaguardia dell'ambiente ed il risparmio di risorse. Esistono poi ulteriori elementi di interesse riguardanti gli ambiti della domotica e dell'automazione.

Indice

Introduzione

Nel corso degli ultimi anni è stato appurato che la diffusione dell'Internet of Things (IoT) ha permesso di aumentare le possibilità di realizzare nuovi servizi incentrati sulle persone e sull'ambiente. Sebbene si sia sentito parlare di Internet of Things, spesso non si conosce la ricca varietà di tecnologie incluse in questo concetto; alcune delle quali sono Internet of Services, Industry 4.0, Internet of Medical Things, Smart City.

A seguito della molteplicità dei campi applicativi di IoT, si ha a che fare con una sempre crescente mole di dati. Questo volume di dati deve essere interpretato, elaborato, aggregato e rappresenta una sfida per niente banale. Sono state ideate molte soluzioni con l'obiettivo di creare piattaforme atte al solo scopo di aggregare e mantenere questi dati. Esistono piattaforme con risorse affidabili come quelle di istituzioni governative o ufficiali; ve ne sono altre meno affidabili di Open Data, dove l'utente può caricare i propri rilevamenti e metterli a disposizione di tutti ed infine vi sono le campagne di crowdsensing, nelle quali viene richiesto di condividere i valori misurati dai propri dispositivi.

Questo progresso ha contribuito a presentare numerosi approcci destinati alle Smart City per la valorizzazione dell'IoT. Smart City è un termine relativamente nuovo che sta ad indicare una nuova generazione di città che impiega le nuove tecnologie per promuovere la competitività, la sostenibilità, la crescita economica, l'efficienza energetica ed il miglioramento della qualità della vita.

Viste tali premesse, è possibile combinare le motivazioni a supporto delle

Smart City con le soluzioni sopraccitate atte al mantenimento dei dati; la combinazione sopra esposta è l'oggetto di studio di questo lavoro di tesi.

SenSquare vuole offrire agli utenti l'opportunità di utilizzare i dati disponibili sulla piattaforma e fornire un mezzo con cui è possibile creare modelli di servizio -definiti template- ed istanziare servizi, chiamati "service".
Un template non è altro che il *codice sorgente del servizio* che si vuole eseguire. I Datastream, di cui SenSquare è a disposizione, invece, sono gli argomenti con cui viene lanciato il programma. Un template, perciò, deve essere considerato come un'entità astratta, infatti non possiede né i dati e né un'area con e su cui essere eseguito.
Istanziando un template in un servizio gli si da una concretezza; è quindi necessario scegliere i Datastream (stream di dati provenienti dai device di misurazione) e un'area geografica di operatività.
Ogni qualvolta un servizio viene eseguito[1], esso riceverà come argomenti solo i Datastream di cui ha bisogno, ovvero quelli disponibili nell'area precisata nello stesso servizio.
Grazie a questa logica è possibile, per esempio, creare servizi che permettano all'utente di monitorare tipi di dati di cui non possiede i sensori. E' altrettanto possibile fornire uno strumento su cui si possono effettuare analisi su un territorio molto vasto, richiedendo al sistema una media del monossido di carbonio presente in una certa area.
A differenza di altre piattaforme *service oriented*(sezione **1.4**), in SenSquare non si pretende dai sensori la compatibilità dei sensori con l'architettura o con le API; i dati, infatti, vengono presi dalle altre piattaforme di memorizzazione.
Quanto descritto permette un ingente eterogeneità dei dati, migliorandone sia la disponibilità che la varietà.
È importante sottolineare che il lavoro di questa tesi non verterà su come è stato possibile procurare questi dati ma su come utilizzarli e renderli

[1]Viene eseguito il programma del template da cui è stato istanziato

accessibile a chiunque.
SenSquare prevede che ogni utente possa definire, in maniera semplice e veloce, i propri template, ed allo stesso tempo, che i più esperti possano creare programmi complessi. Ogni utente ha quindi la possibilità di ideare i propri template che, se resi pubblici in fase di creazione, vengono messi a disposizione della collettività.
Chiunque deve essere in grado di instanziare, in maniera altrettanto semplice e veloce, i propri servizi. La piattaforma richiederà informazioni generali, come il titolo ed una descrizione ed anche i Datastream di interesse al servizio. Una volta portato a termine l'inserimento delle suddette preferenze, sarà possibile instanziare il servizio e seguirne gli aggiornamenti.
È possibile utilizzare una discreta gamma di tipologie di sensori (all'attivo più di 15), come per esempio sensori per il rilevamento di temperatura, di umidità, di pressione, di velocità e di direzione del vento, di intensità della pioggia ed anche di diversi gas (spesso presi in considerazione per il calcolo dell'inquinamento dell'aria).
Il progetto è quindi costituito da due parti: il *Back End* ed il *Front End*. Il primo, il *Back End*, si incaricherà della gestione dei dati, del loro salvataggio, dell'esecuzione dei servizi e dell'interfaccia REST. Il secondo invece, il *Front End*, dovrà offrire un'interfaccia tramite la quale permettere all'utente di registrarsi alla piattaforma, creare template, istanziare servizi e visualizzare i dettagli di template e servizi.

Parte I

Stato dell'arte

Capitolo 1

Piattaforme esistenti

In questo capitolo verrà effettuata una classificazione degli aspetti e dei concetti principali contenuti nel lavoro di tesi qui presentato.
Questa classificazione è stata introdotta perché, nel panorama odierno dell'IoT, non esiste un altro applicativo uguale a quello proposto. Questo è il motivo per cui si è deciso di paragonare quanto elaborato con lavori già esistenti, confrontandone uno ad uno gli aspetti che li contraddistinguono.
Per ogni categoria presente in questa classificazione verrà fornita, innanzitutto, una breve descrizione che aiuterà il lettore ad entrare nel merito di quanto trattato. Successivamente, verranno prese in considerazione una serie di piattaforme IoT già esistenti, facenti parte delle suddette categorie.
Per concludere, il capitolo terminerà con un breve ma dettagliato confronto tra le piattaforme citate ed il progetto di tesi esposto, indicando le novità, i vantaggi e i punti di forza di SenSquare [3].

1.1 Piattaforme per l'IoT

Una piattaforma "Internet of Things" (IoT) è un software di appoggio che collega le periferiche hardware, gli access point ed i dati ad altre parti della infrastruttura. Queste ultime, in genere, sono le applicazioni utilizzate dall'utente finale.

Le piattaforme IoT si occupano delle attività di gestione e della visualizzazione dei dati; tale procedimento consente agli utenti di automatizzare il proprio sistema di gestione.
Queste piattaforme possono essere definite come l'intermediario tra i dati rilevati dai sensori e l'interfaccia utente SaaS[1].

Xively

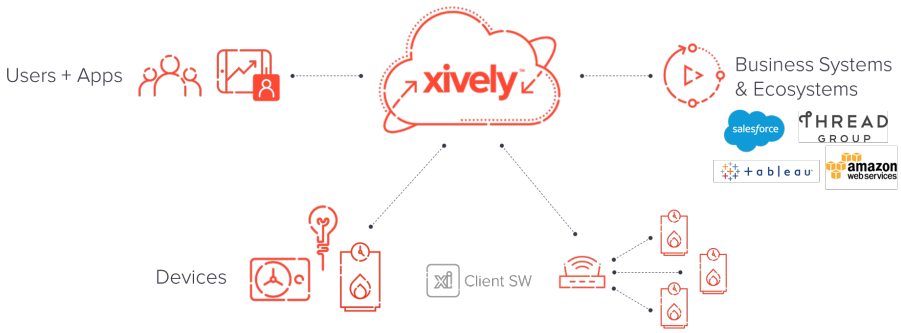

Figura 1.1: Architettura concettuale di Xively [4]

Xively [5] è un PaaS[2] che fornisce servizi middleware al fine di creare prodotti e soluzioni per l'IoT [6]. L'azienda offre molti strumenti e risorse che gli sviluppatori possono sfruttare per collegare i loro sensori e per raccogliere dati da questi ultimi [7]. Nel caso in cui, durante l'uso delle API, uno sviluppatore si trovi a dover fronteggiare problemi imprevisti, esiste anche una numerosa community di sviluppatori disponibile a offrire supporto. La mission generale di Xively è costituita dall'intenzione di aiutare gli sviluppatori e le aziende a progettare sensori fisici ed anche dall'idea di collegarli, in modo rapido e semplice, al loro servizio di cloud IoT.

[1] *Software as a service:* è un modello di concessione e distribuzione in cui il software viene fornito in licenza tramite abbonamento.

[2] *Platform as a service:* è un modello di cloud computing in cui un provider di terze parti fornisce agli utenti via Internet strumenti hardware e software, di solito quelli necessari per lo sviluppo delle applicazioni.

Xively viene fondamentalmente utilizzato per collegare i sensori desiderati al loro cloud. Se si richiede un Rule Engine in grado di interpretare i dati, sarà necessario svilupparne uno in autonomia o trovarne uno compatibile con Xively. L'obiettivo principale di Xively, in definitiva, collegare i sensori necessari al loro cloud, da dove è possibile estrarre facilmente i dati. Anche se l'aggiunta di sensori dovrebbe essere una questione semplice, l'API fornita può di difficile impiego, specialmente quando si utilizza un sensore che non è già supportato dal sito [8].

AllJoyn

Figura 1.2: Logo di AllJoyn [4]

AllJoyn è uno standard open source guidato da AllSeen Alliance, un consorzio intersettoriale dedicato a consentire l'interoperabilità di miliardi di dispositivi, servizi ed applicazioni per l'IoT. Più precisamente, AllJoyn è un framework [9] (indipendente dalla piattaforma) che facilita il collegamento in rete basato sulla vicinanza attraverso dispositivi eterogenei. AllJoyn definisce un protocollo comune per dispositivi ed applicazioni di modo da rilevarsi e per comunicare tra loro, indipendentemente dalla tecnologia di trasporto, dalla piattaforma o dal produttore. Il framework offre diversi vantaggi che, in precedenza, risultavano assenti nel campo dello sviluppo delle proximity network.

1.1.1 Service-Oriented Architecture

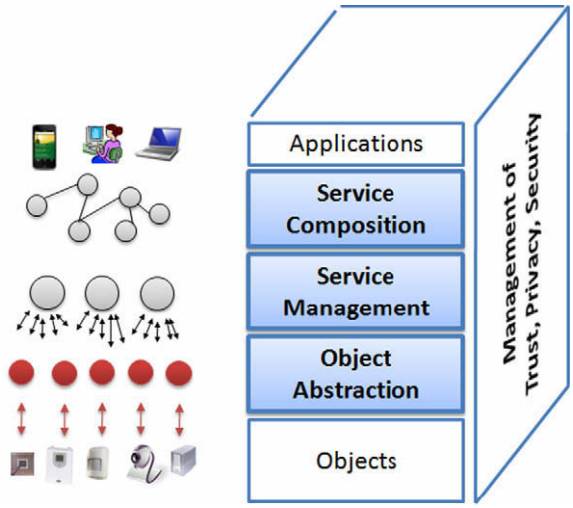

Figura 1.3: Architettura basata su SOA per il middleware IoT [10]

Il concetto di Service-Oriented Architecture (SOA) ha riscosso molto interesse da parte del mondo accademico e dell'industria. Infatti, SOA è uno stile di architettura che definisce come l'applicazione eterogenea debba essere sviluppata e integrata. Lo scopo principale di SOA è sia quello di allontanarsi dalle applicazioni monolitiche che quello di offrire un insieme di servizi riutilizzabili e componibili, di modo da permettere la costruzione di applicazioni Questi servizi non sono altro che agenti software distinti, caratterizzati da interfacce semplici, ben definite e organizzate per eseguire una funzione richiesta. Inoltre i suddetti servizi possono essere essenzialmente distinti in: produttori o consumatori. I produttori possono essere diversi e possono mettere a disposizione un servizio, mentre i consumatori ne usufruiscono, perché l'interfaccia (ovvero le API) è la stessa. La piattaforma SOA è un mediatore che mette in comunicazione le due tipologie di agenti software sopra citati.
Una definizione più formale di SOA potrebbe essere la seguente:

> "SOA è un paradigma per l'organizzazione e l'utilizzo delle risorse distribuite, le quali possono rientrare sotto la gestione di domini di proprietà differenti. Esso fornisce un mezzo uniforme per offrire, scoprire, interagire ed usare le capacità di produrre gli effetti voluti consistentemente con presupposti e aspettative misurabili. "
>
> (Organization for the Advancement of Structured Information Standards [11])

ArrowHead

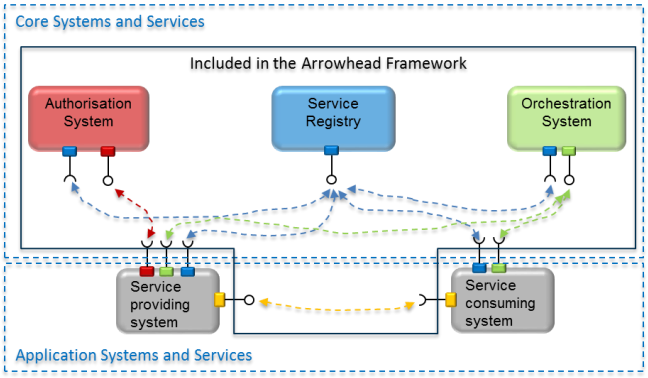

Figura 1.4: Il core del Framework Arrowhead e i servizi applicativi [12]

ArrowHead parte dal presupposto che un approccio basato sui servizi rappresenti la tecnologia necessaria al raggiungimento di un'automazione collaborativa in un ambiente di open-network (ovvero un ambiente in cui numerosi dispositivi embedded sono collegati). La filosofia di base dell'architettura è caratterizzata da un orientamento ai servizi e l'obiettivo del framework è sostenere efficacemente lo sviluppo, l'implementazione e il funzionamento di sistemi collaborativi ed interconnessi. Gli aspetti essenziali della struttura sono sistemi che forniscono e consumano servizi e che, allo stesso tempo, agiscono come sistemi di sistemi. Alcuni sistemi comunemente utilizzati, come l'*Orchestration System*, l'*Authorization System* o il *Service Registry* sono

considerati parte del core (come si può vedere in figura **1.4**). Questi possono essere utilizzati da qualsiasi sistema di sistemi che segue le linee guida del framework Arrowhead.

1.2 Piattaforme di crowdsensing

Il crowdsensing permette una rilevazione efficiente di dati eterogenei su grandi aree, sfruttando la collaborazione degli utenti iscritti che, a questo scopo, offrono la disponibilità dei propri sensori e device. Una piattaforma di crowdsensing si occupa di aggregare i dati derivanti dai propri membri e di utilizzarli per creare servizi. Tuttavia, la copertura che una piattaforma di crowdsensing può fornire per una determinata area dipende dalla disponibilità degli abbonati e dalla loro mobilità. Si parla di Mobile Crowdsensing (MSC) quando l'utente iscritto utilizza il cellulare per fornire dati al servizio. Il crowdsensing si suddivide principalmente nelle seguenti 2 categorie [13,14]:

- *Participatory Crowdsensing*, ovvero quando gli utenti hanno consapevolezza e sono coinvolti nel processo di rilevamento. Spesso viene richiesto loro di inviare opinioni e valutazioni su quello che stanno monitorando. Nessun dato, ad eccezione di tale feedback, deve essere inviato / raccolto.
- *Opportunistic Crowdsensing*, in questo caso il processo è autonomo e gli utenti hanno un coinvolgimento minimo, o nessuno, con il processo di propagazione. Quest'ultimo viene effettuato in maniera automatica e trasparente all'utente.

Vari esempi di piattaforme di crowdsensing verranno esposti di seguito.

SecondNose

SecondNose [15] è un servizio di mobile crowdsensing per il monitoraggio della qualità dell'aria: esso è finalizzato alla raccolta di dati ambientali ed al monitoraggio di alcuni indicatori di inquinamento atmosferico per favorire la

riflessione dei partecipanti sulla loro esposizione complessiva agli inquinanti. Il sistema è composto dai seguenti quattro componenti: un sensore di inquinamento atmosferico, un'applicazione per cellulare Android, un Back End con componenti di raccolta e analisi ed un'applicazione web per visualizzare i dati. L'applicazione mobile viene utilizzata per leggere in tempo reale il livello di l'inquinamento atmosferico, i parametri ambientali e per visualizzare i dati precedentemente raccolti. L'applicazione web visualizza l'aggregazione dei dati raccolti relativi alla qualità dell'aria corrente dell'ultima settimana.

iNaturalist

iNaturalist [16] è un sistema di identificazione delle specie in cui i volontari raccolgono dati sulla biodiversità; come ad esempio fotografie ed annotazioni sul campo sui loro dispositivi mobili, di modo da poterli condividere attraverso i social network con la community che condivide gli stessi interessi.

Siftr

Siftr [17] è un'applicazione social di fotografia -simile a Instagram [18]- progettata come strumento di rilevazione di dati su campo, ai fini di educazione della comunità. Siftr incoraggia i gruppi di utenti, come le scolaresche, ad uscire ed esaminare i loro quartieri e dintorni. Gli studenti possono scattare foto relative a un argomento o a un tema, creare categorie, aggiungere descrizioni e invitare altri a commentare e partecipare alla conversazione. Quella appena descritta è quindi un'applicazione semplice flessibile ed è progettata per essere utilizzata in diversi contesti.

Medusa

Medusa [19] è un framework di programmazione per il crowdsensing che fornisce supporto all'operatore per il processo di rilevamento o per rivedere i risultati. Il sistema riconosce la necessità di incentivare i partecipanti e si occupa delle loro preoccupazioni in materia di privacy e sicurezza. Medusa

fornisce un'astrazione di alto livello per specificare i passi necessari per completare un compito di crowdsensing, inoltre essa utilizza un sistema runtime distribuito che coordina l'esecuzione di questi compiti tra smartphone e un cluster sul cloud.

1.3 Visual programming

Con il termine *Visual programming* viene inteso un tipo di programmazione che permette agli utenti di descrivere i processi mediante una rappresentazione schematica e visiva. Se il tipico linguaggio di programmazione basato sul testo fa pensare allo sviluppatore come un computer, un linguaggio di programmazione visiva gli permette di definire il procedimento mediante un approccio comprensibile all'uomo.
Ogni sviluppatore è a conoscenza del fatto che i linguaggi di programmazione basati sul codice si concentrano interamente sull'implementazione: si tratta solo dei passi precisi che il computer deve compiere per poter ottenere l'esperienza che si vuole dare all'utente. È indubbio che i linguaggi di alto livello ed i framework moderni offrano diversi benefici ma il compito dello sviluppatore è quello di tradurre le esigenze umane in processi che si adattino alle limitate capacità del computer.
Alcuni strumenti di programmazione visiva seguono gli stessi processi e paradigmi della programmazione basata sul testo.

Scratch

Figura 1.5: Logo di Scratch [20]

Scratch [21] è un linguaggio di programmazione visiva gratuito sviluppato dal MIT Media Lab. Scratch è stato creato per aiutare i più giovani ad imparare a pensare in modo creativo, ragionare sistematicamente e lavorare in modo collaborativo.
Il suddetto linguaggio viene utilizzato per una vasta gamma di scopi educativi e di intrattenimento. Esso infatti può essere utilizzato per creare storie interattive, animazioni e giochi; inoltre fornisce un trampolino di lancio per attività scientifiche. Scratch permette infatti di realizzare progetti matematici e scientifici e simulazioni e visualizzazioni di esperimenti. Scratch presenta anche altre funzionalità più di carattere sociale o artistico.

Blockly

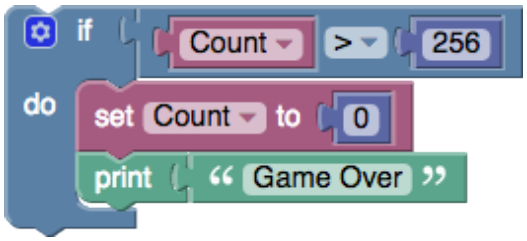

Figura 1.6: Programma di esempio scritto in Blockly [22]

Blockly [23] è un editor visivo che permette agli utenti di scrivere programmi collegando insieme blocchi, i quali sono elementi interattivi e grafici che l'utente può manovrare. Essi permettono di rappresentare, con enorme semplicità concetti complessi come variabili, espressioni logiche, loop, ed altro ancora. Questo esplicato concede agli utenti di concentrarsi solo sull'applicazione dei principi della programmazione e della semantica, senza doversi preoccupare della sintassi.

1.3.1 Flow-based programming

Inventata da J. Paul Morrison [24] negli anni Settanta, la programmazione flow-based è un modo per descrivere il comportamento di un'applicazione

come una rete di nodi (o scatole nere). Ogni nodo ha uno scopo ben definito, come esplicato di seguito: riceve alcuni dati, fare qualcosa con questi dati e poi trasmetterli. La rete risulta quindi responsabile del flusso di dati tra i nodi.
Quello descritto è di un modello che si presta molto bene ad una rappresentazione visiva e questo lo rende più accessibile ad una più ampia gamma di utenti. Se è possibile suddividere un problema in passi discreti, è altrettanto possibile guardare un flusso e farsi un'idea di quello che stia accadendo, senza dover capire le singole righe di codice all'interno di ogni nodo.

NodeRed

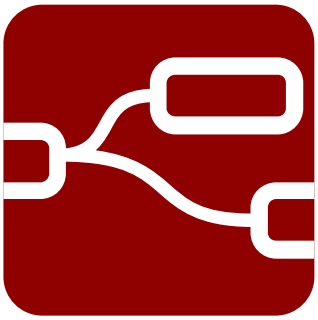

Figura 1.7: Logo di NodeRed [25]

Node-Red è un editor visivo di flusso basato su browser che semplifica il collegamento di dispositivi, API e servizi online utilizzando un'ampia gamma di nodi. I Nodi possono quindi essere trascinati e collegati tra loro. Ogni Nodo offre diverse funzionalità che possono spaziare da un semplice nodo di debug -in grado di vedere cosa stia succedendo nel flusso di dati- fino ad un nodo "Raspberry Pi" che permette di leggere e scrivere sui pin GPIO dei vostri Pi.

Figura 1.8: Esempio di flusso dei dati: *input* → *operazione* → *output* [26]

Come si può notare nella figura **1.8** tutti i nodi rientrano nella categoria di input, operation o output. Node-RED è stato prodotto da IBM ma ora è disponibile open source sul repository di github.

1.4 Vantaggi e motivazioni

Nelle sezioni precedenti sono state descritte diverse piattaforme che forniscono il proprio cloud e generano una rete di servizi distaccata dal resto del IoT nominata "Intranet of Things". Tali soluzioni sono particolarmente potenti se analizzate singolarmente ma, riferendosi all'IoT, peccano parzialmente o totalmente di interoperabilità con le altre piattaforme. Molto spesso vengono infatti utilizzate delle API proprietarie non utilizzabili da tutti. L'essere chiusi, è di vincolo all'utente poiché lo costringe a dover acquistare da produttori specifici. Questi ultimi potrebbero, prima di tutto, non possedere tutti i dispositivi di cui si ha bisogno (limitando così anche il potenziale di tale infrastruttura) e, in secondo luogo, costringe l'utente finale a subire il prezzo del prodotto poiché non può scegliere quello giusto per se stesso. Infine, con riferimento al monitoraggio ambientale, è possibile che questo comporti una ridondanza o una indisponibilità dei dati.
È facile intuire che queste lacune devono essere presto colmate. SenSquare [3] integra insieme fonti di dati eterogenee e le mette a disposizione dell'utente finale grazie all'impiego di un'interfaccia facilmente accessibile. Il beneficio sostanziale apportato da SenSquare è rappresentato dal fatto che non sia più richiesto nessun prodotto specifico o nessun sensore compatibile con l'architettura o le API. I dati vengono infatti raccolti dalla piattaforma stessa, di modo che l'utente debba unicamente utilizzarli creando servizi a lui utili.

La creazione e la gestione di servizi personalizzabili è una delle funzionalità più corpose del lavoro di tesi qui presentato. La "programmazione" di un servizio è un aspetto su cui si è riflettuto a fondo poiché avrebbe potuto ostacolare l'utilizzo della piattaforma. Si è perciò deciso di considerare il Visual programming come mezzo, preferendo l'editor visivo più intuitivo. Poiché Blockly viene utilizzato per l'insegnamento della programmazione ai bambini si è optato per questa libreria.

Capitolo 2

SenSquare

Nella sezione **1.4** sono state discusse le motivazioni alla base di SenSquare e i benefici che esso porta.
In questo capitolo verranno discusse come e quali fonti vengono considerate nello stato attuale dell'implementazione del sistema.
In particolare, si esaminerà in breve come e da dove la piattaforma riesca a raccogliere la grande quantità di dati eterogenei [27] che, successivamente, mette a disposizione all'utente finale.

2.1 Reliable

Vengono riconosciute come *risorse affidabili* quelle provenienti da istituzioni governative o ufficiali, per le quali il rilevamento ambientale è l'obiettivo principale del business.
Esse sono considerate affidabili per diversi motivi, come ad esempio i seguenti: l'alta precisione dei loro strumenti e la garanzia della loro continuità nel fornire informazioni sono tra le motivazioni più importanti.
Nel caso presentato è stata considerata l'Agenzia Regionale per la Tutela Ambientale in Italia (ARPA), che è l'Agenzia Amministrativa Pubblica per il monitoraggio ambientale.

2.2 Unreliable

Vengono contrassegnate come *risorse inaffidabili* quelle provenienti da piattaforme di Open Data, sulla quale normalmente gli utenti possono caricare i dati provenienti dai loro sensori per finalità personali.
Tali dati sono così definiti perché non vi è alcuna garanzia della loro veridicità bensì hanno però il vantaggio di essere in quantità significativa ed il loro numero è in costante crescita, rendendo possibile una copertura mondiale.
Inoltre, queste risorse inaffidabili tendono ad avere un tasso di aggiornamento significativamente più elevato delle risorse definite affidabili, riuscendo a fornire informazioni con una granularità soddisfacente.
Per far si che questi dati diventino più sicuri e veritieri, ogni misura deve essere elaborata attraverso algoritmi di *Machine Learning* [3].

2.3 Crowdsensing

Il principale contributo di dati viene fornito dalle *risorse di crowdsensing*.
L'ecosistema di Mobile Crowdsensing (MCS) (come descritto in [28]) rappresenta un sistema attualmente implementato (tramite una app mobile) come parte integrante della piattaforma. Tale sistema ha dimostrato di essere particolarmente utile e di rispettare i vincoli imposti dal server, con conseguente riduzione della ridondanza dei dati.

Parte II

Architettura del Sistema

Capitolo 3

Panoramica

In questo capitolo verranno delineati le componenti principali dell'architettura che caratterizza SenSquare.
Per rendere il tema trattato di più facile comprensione al lettore, si è scelto di fare un'introduzione dell'architettura completa, comprendendo i moduli e gli applicativi sviluppati precedentemente a questo lavoro di tesi.
Successivamente, verrà presentato ciò che è stato effettivamente prodotto durante questo progetto.

3.1 Architettura Completa

Come viene mostrato in figura **3.1**, la piattaforma può essere divisa in quattro diverse parti ed ognuna di esse può, a sua volta, essere distinta per mezzo di un riquadro colorato. In questo diagramma viene raffigurata l'architettura completa di SenSquare.

La **Central Coordination Unit** (CCU) è l'unità "cervello" del sistema ed è colei che organizza il funzionamento di tutta la piattaforma.
L'architettura di SenSquare è strutturata come un sistema centralizzato client-server e la CCU può essere vista come il nodo centrale in una topologia a stella. Pertanto le entità client non hanno alcun collegamento di comunicazione

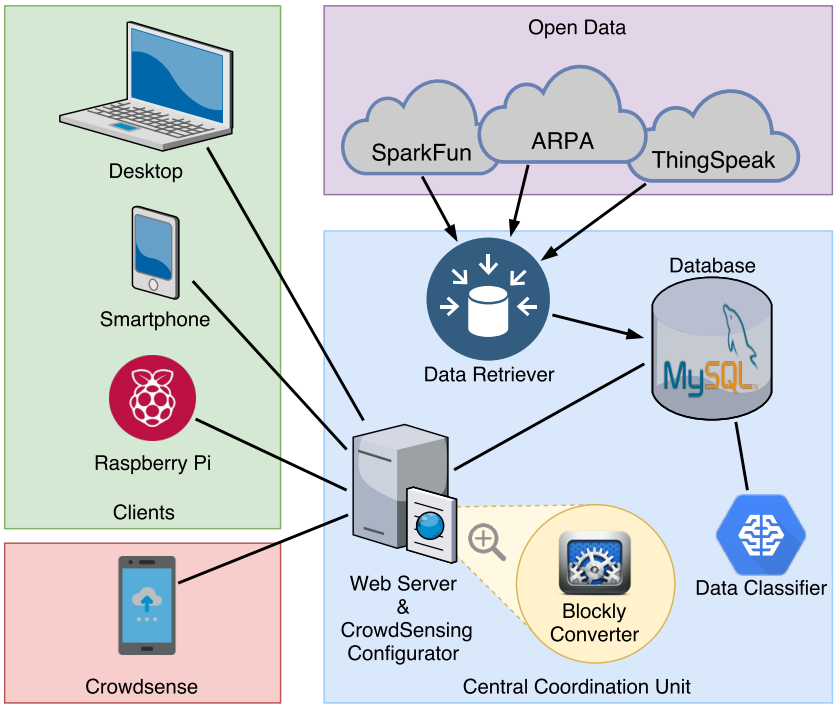

Figura 3.1: Architettura completa della piattaforma SenSquare

tra loro.
Come si può vedere nella figura **3.1**, l'entità centrale, raffigurata nel riquadro azzurro, è divisa in più moduli dedicati a compiti diversi.

Il modulo chiamato **Data Retriever** [3] viene impegnato per recuperare periodicamente i dati provenienti dalle "fonti statiche". Queste sorgenti di dati, come già esaminato nel capitolo **2**, sono costituite sia dalle fonti governative, che pubblicano nei propri repository aperti, e sia dalle piattaforme Open Data, da cui è possibile estrarre dati di grande valore. Le sorgenti vengono raffigurate nel riquadro violetto.

Il modulo chiamato **Data Classifier** viene impiegato per assegnare una classificazione a tutti i flussi di dati in cui non è indicata una Classe.
Per *Classe* si intende sia il tipo di dato, anche detto *Data Type*, (esempio:

temperatura, PM10, luminosità, etc.) sia la sua unità di misura (esempio per la temperatura: °C, °K, °F). Infatti, non tutte le fonti di dati specificano il tipo esatto di dato a cui si riferiscono le misurazioni, quindi è necessario dedurlo attraverso un approccio basato su NLP.
Per approfondimenti attenersi a [3].

Delle entità sviluppate precedentemente a questo lavoro di tesi, l'ultimo modulo che verrà discusso tratta un applicativo per Android[1].
Crowdroid [28, 30] è l'applicazione per fare Mobile CrowdSensing da tutti i dispositivi che collaborano alla condivisione dei dati con la piattaforma.
Lo scenario di riferimento è composto da due attori:

- I *participant*: sono utenti in grado di produrre dati e/o disposti a consumare dati sotto forma di servizi. Essi danno il consenso sia ad installare l'applicazione Crowdroid, sia a condividere la loro connessione internet per poter inviare i dati al server;

- Gli *stakeholders*: sono soggetti, aziende o enti territoriali che possono presentare la propria campagna ed anche richiedere dati. Gli utenti (i participant) possono quindi sottoscrivere o meno la richiesta degli stakeholder, a seconda della ricompensa e, se iscritti, possono condividere i dati richiesti.

Come è possibile notare, tutta l'architettura introdotta fin ora è rivolta alla raccolta dati.
Nei capitoli successivi verranno descritti quei componenti ancora non trattati e verrà introdotto e, successivamente, approfondito il sistema di come questi dati possono essere utilizzati dall'utente per creare servizi ad hoc per se stesso e la comunità.

[1] *Android* è un sistema operativo per dispositivi mobili sviluppato da Google Inc. e basato sul kernel Linux. Per dettagli riferirsi a [29]

3.2 Architettura del progetto

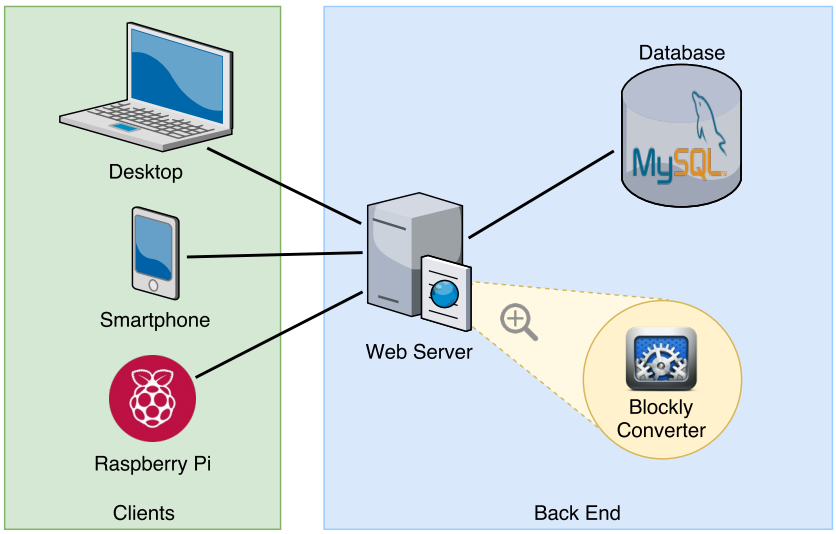

Figura 3.2: Architettura per lo sviluppo collaborativo di servizi personalizzati

L'obiettivo principale di questo progetto è, grazie ai dati raccolti dal resto della piattaforma, quello di riuscire a creare servizi personalizzati.
Per ottenere ciò c'è bisogno di un'architettura leggermente più estesa rispetto a quella vista fin ora.
Per precisione, il contributo di questo progetto viene mostrato in figura **3.2**. La principale discrepanza che si può notare tra i due diagrammi, presentati in precedenza, è che la *CCU* è stata rinominata *Back End*. Questo perché nel senso comune delle Web Application ci si riferisce al server, ed a tutti i programmi che servono per far funzionare l'applicativo, con il termine **Back End**.

Capitolo 4

Server

Come è possibile notare dall'architettura mostrata in figura **3.2**, la piattaforma ha un nodo centrale, ovvero il Server. Questo modulo si fa carico della comunicazione con tutte le altre risorse.
I client, che non risiedono sulla stessa macchina, sono connessi a lui mediante protocollo HTTP (per questo motivo viene indicato come Web Server) mentre i moduli *Database* e *Blockly Converter* sono servizi che vengono lanciati per il corretto funzionamento della piattaforma.
Nella fattispecie verrà eseguito il servizio del Database per il salvataggio dei dati, il Web Server per la comunicazione client-server ed infine uno script che converte il codice generato dalla libreria Blockly al linguaggio di programmazione *Python*.
La struttura modulare che ne deriva permette uno sviluppo più semplice. Tale beneficio è reso possibile anche dall'intenzione di voler utilizzare una interfaccia basata su REST API per la comunicazione con i client.

Capitolo 5

Database

Il Database è il componente che mantiene tutti i dati omogenei della piattaforme e farà riferimento a diverse tipologie di dato e differenti esigenze. Al fine di realizzare quanto prefissato si è optato per MySql [31], un RDBMS[1] open source sviluppato da Oracle.

Il database inerente a questo progetto presenta le seguenti tabelle:

- **Account** contiene i dati dell'utente iscritto alla piattaforma; per esempio: nome, cognome, email, password e il suo livello di accesso. Ogni *Account* possiede il riferimento al proprio ID nella tabella *Participant*. *Account* viene utilizzata unicamente per l'autenticazione ed il mapping da *Account* a *Participant*.

- **Participant** è quella entità che può interagire con la piattaforma. Il participant può essere un utente iscritto (con anche un *Account* associato) oppure un ente creato dalla piattaforma per aggiungere servizi e misurazioni (senza un *Account* associato).

[1] *Relational database management system*: è una tipologia di database che rappresenta i dati come relazioni, ovvero in forma tabellare. Cioè come una raccolta di tabelle dove ciascuna tabella è composta da un insieme di righe e colonne.

- **DataClass** contiene le informazioni base per la classificazione di un dato; per esempio il *Data Type* (introdotto sezione **3.1**) e la *Unit of Measure*.

- **Device** contiene le informazioni dei dispositivi. Un esempio di *Device* potrebbe essere una weather station, un cellulare, un Arduino con dei sensori connessi. Ogni *Device* può avere uno o più *DataStream*.

- **DataStream** è la tipologia di dato che viene emesso da un *Device*. Ogni *DataStream* può avere uno o più *Measurement*.

- **Measurement** contiene tutti i dati relativi alla misurazione; per esempio: il valore, la posizione in coordinate GPS ed il timestamp.

- **DataStreamOption** serve per distinguere se l'utente sceglie un *DataStream* preciso oppure no. Nella creazione di template l'utente ha la possibilità di selezionare se utilizzare uno solo o tutti i *DataStream* a disposizione nell'area che precisa.

- **CustomServiceTemplate** contiene tutti i dati relativi al template di un servizio, come ad esempio: il titolo, la descrizione, il codice del servizio e un valore *booleano* che indica se il template è pubblico o meno. Ogni template possiede anche il riferimento di chi lo ha creato, ovvero il *participant_ID*.

- **CustomServiceTemplateRating** viene utilizzata per memorizzare le valutazioni che gli utenti assegnano ad uno specifico *CustomServiceTemplate*.

- **CustomService** è la tabella che contiene le informazioni relative all'istanziazione di un *CustomServiceTemplate*. Vengono memorizzati: il titolo del servizio, una descrizione, le coordinate di dove l'utente vuole utilizzarlo ed il raggio entro il quale bisogna leggere i *Measurement*.

In figura **5.1** vengono mostrati i nomi e i tipi degli attributi all'interno di ciascuna delle tabelle appena presentate e le relazioni che ci sono tra esse. Come è possibile notare, le tabelle *DataClass* e *DataStreamOption* non hanno relazioni di tipo ForeignKey[2]. Quanto detto è motivato dal fatto che tali tabelle vengono utilizzate all'interno del campo *json_ args* di *CustomServiceTemplate* e *CustomService*.

[2]Un ForeignKey è un campo usato per collegare insieme due tabelle.

Figura 5.1: Diagramma che illustra le relazioni tra le entità del database.

Capitolo 6

REST API

I servizi web RESTful sono un modo per garantire l'interoperabilità tra i sistemi informatici disponibili su Internet. Tale metodo di comunicazione, introdotto nell'architettura presentata, punta a rendere accessibile qualsiasi tipo di client a tutte le risorse della piattaforma.
Le API REST sono importanti perché permettono al Back End di liberarsi di molti dati superflui e di elaborazioni che lo rallenterebbero. L'approccio REST rende possibili i seguenti benefici:

- **Stateless:** La comunicazione client-server è condizionata dal fatto che nessun contesto client viene memorizzato sul server tra le varie richieste. Le richieste dei client devono contenere tutte le informazioni utili affinché la richiesta possa andare a buon fine.

- **Cacheable:** I Client possono fare caching delle risposte in modo tale di migliorare la *User Experience* (visualizzando il contenuto richiesto più velocemente), come anche di consumare meno dati e di alleggerire il server da computazioni inutili.

- **Uniform interface:** Questa caratteristica permette di semplificare e disaccoppiare l'architettura *Front End* e *Back End*. Avendo un'inter-

faccia di comunicazione omogenea, è possibile evolvere i due applicativi separatamente e/o svilupparne di nuovi e personalizzati per differenti architetture (vedi l'esempio del Raspberry Pi in figura **3.2**).

Nello specifico, un'interfaccia di comunicazione RESTful non è altro che un insieme di URI che possono essere invocati dai client.
In base al *metodo HTTP*[1] utilizzato si richiede al server di eseguire una precisa azione.

6.1 Entry point

L'interfaccia incaricata di rispondere a tutte le suddette richieste, nel caso di SenSquare, è formata dai seguenti URI (dove **/api/v1/** è l'entry point delle API):

- **/api/v1/sign-up/**

  POST serve per effettuare la registrazione alla piattaforma

- **/api/v1/auth/login/**

  POST serve per eseguire il login nella piattaforma. Se il login avviene con successo, il sistema restituirà un *JSON Web Token* [32] al fine di permettere al client di autorizzarsi durante le query successive

- **/api/v1/password/change/**

  POST serve all'utente per cambiare password

- **/api/v1/password/set/**

  POST è un metodo disponibile solo agli admin per cambiare password a qualsiasi utente

[1] GET, HEAD, POST, PATCH, PUT, DELETE

- **/api/v1/users/**

  GET è un metodo accessibile solo agli admin per ricevere la lista di tutti gli utenti della piattaforma

- **/api/v1/users/me/**

  GET serve a recuperare le informazioni relative all'utente inserite durante la registrazione

- **/api/v1/participant/<pk>/**

  GET serve a recuperare le informazioni di un participant preciso

- **/api/v1/classes/**

  GET recupera tutte le informazioni delle classi di dato

  POST se l'utente è admin può aggiungere una classe di dato

- **/api/v1/classes/<pk>/**

  GET recupera le informazioni di una precisa classi di dato

  DELETE se l'utente è admin può eliminare una precisa classi di dato

- **/api/v1/datastream-options/**

  GET recupera le opzioni per i datastream

  POST se l'utente è admin può aggiungere un'opzione

- **/api/v1/datastream-options/<pk>/**

  GET recupera le informazioni di una opzione precisa

  DELETE se l'utente è admin può eliminare un'opzione

- **/api/v1/datastreams/**

  GET recupera le informazioni di tutti i datastream

- **/api/v1/datastreams/<pk>/**

  GET recupera le informazioni di un datastream preciso

- **/api/v1/streams/nearby/<lat>/<lng>/<radius>/<classes>/**

  GET recupera tutte le informazioni di tutti i datastream presenti nell'area selezionata appartenenti alle classi richieste

- **/api/v1/measurements/<pk>/**

  GET recupera tutte le informazioni di una specifica misurazione

- **/api/v1/templates/**

  GET recupera tutte le informazioni di tutti i template

- **/api/v1/templates/<pk>/**

  GET recupera tutte le informazioni di uno specifico template

- **/api/v1/templates/<template_pk>/rating/**

  POST valuta uno specifico template

- **/api/v1/user/templates/**

  GET recupera tutte le informazioni di tutti i template creati dall'utente

  POST crea un nuovo template

- **/api/v1/user/templates/<pk>/**

  GET recupera tutte le informazioni di un template creato dall'utente

- **/api/v1/instances/**

  GET recupera le informazioni di tutti i servizi istanziati

  POST crea un nuovo servizio, istanziandolo in una certa location

- **/api/v1/instances/<pk>/**

  GET recupera le informazioni di un servizio istanziato preciso

- **/api/v1/instances/nearby/<ne_lat>/<ne_lng>/<sw_lat>/<sw_lng>/**

  GET recupera le informazioni di tutti i servizi istanziati in una certa zona

- **/api/v1/instances/<instance_pk>/execute/**

  GET esegue un servizio preciso per ottenere il risultato

Tutte le REST API accetteranno e produrranno dati di tipo *application/json*.

Capitolo 7

Web App

L'utente avrà la possibilità di utilizzare l'applicazione web per interfacciarsi con la piattaforma.
Lo scenario di utilizzo di SenSquare sarà pressoché il seguente:

1. L'utilizzatore si registrerà alla piattaforma e, successivamente, effettuerà il login;

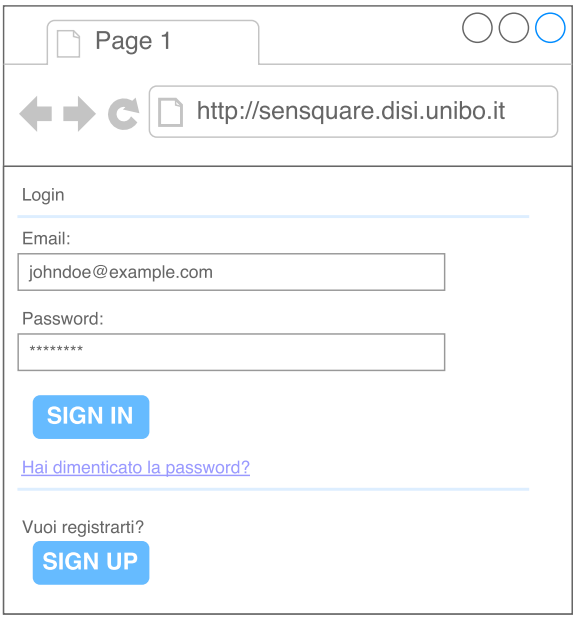

Figura 7.1: View mockup della pagina di login

2. L'applicazione richiederà l'accesso alla posizione e caricherà tutti i servizi istanziati in quella zona;

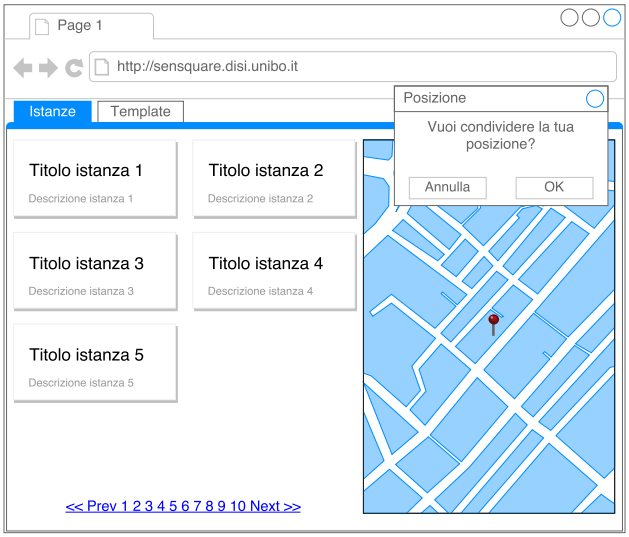

Figura 7.2: View mockup della pagina di home: lista delle istanze

3. L'utente potrà cliccare sulla card di un servizio istanziato per vederne i dettagli e per controllare i valori calcolati dall'esecuzione del servizio;

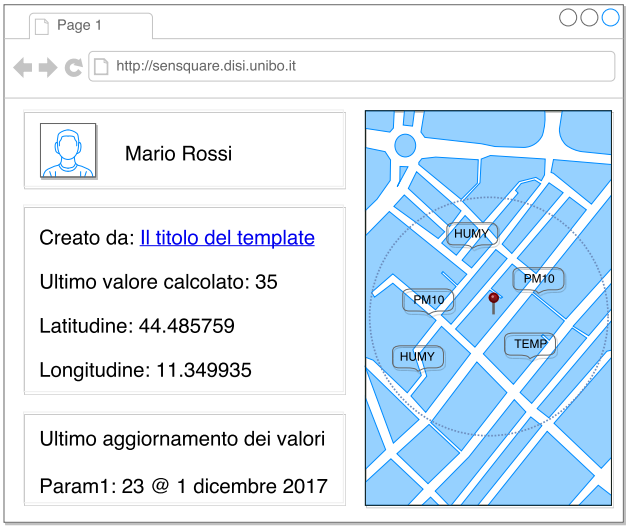

Figura 7.3: View mockup della pagina dettaglio di una istanza

4. Se l'utente lo dovesse desiderare, potrà andare nella sezione *template* mediante la scheda apposita mostrata in figura **7.2** oppure tramite il collegamento in figura **7.3**;

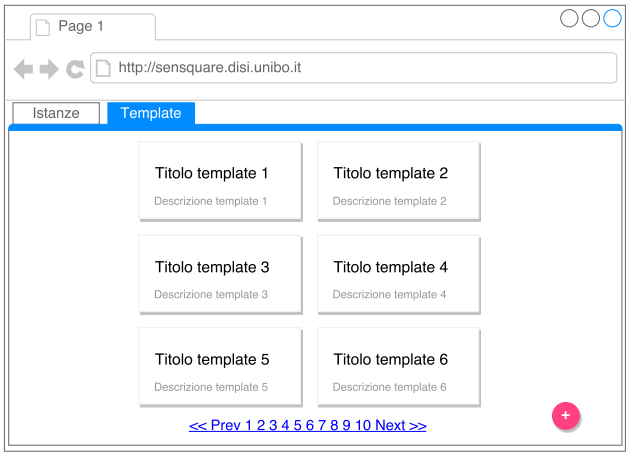

Figura 7.4: View mockup della lista dei template

5. L'utente potrà aggiungere un template cliccando sul pulsante *fabButton* in figura **7.4**. L'aggiunta prevederà la scelta di un titolo, una descrizione e la realizzazione del programma mediante l'editor di Blockly. Dopo l'aggiunta si verrà rediretti alla vista contenente la lista dei template;

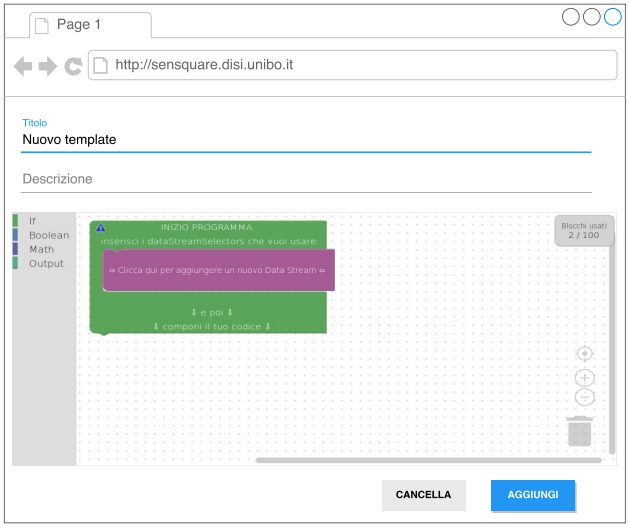

Figura 7.5: View mockup per l'aggiunta di un template

6. Se si clicca su un template apparirà la vista dettagli. Cliccando sulle stelline sarà quindi possibile dare una valutazione al template;

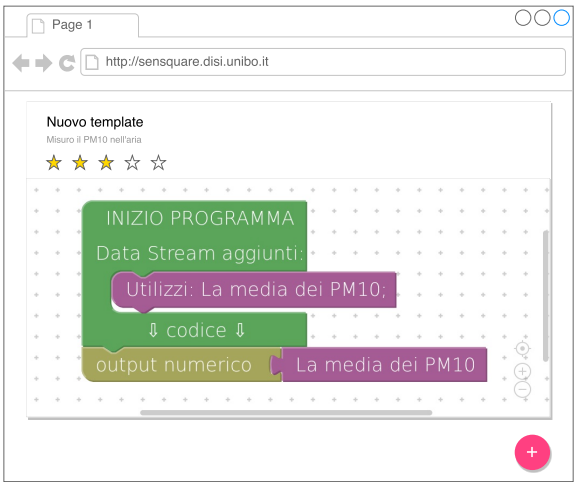

Figura 7.6: View mockup del dettaglio di un template

7. Dalla vista dettaglio di un template (figura **7.6**), cliccando sul pulsante *fabButton*, sarà possibile aggiungere un servizio. L'aggiunta prevederà la scelta scegliere un titolo, di una descrizione, la scelta della posizione di deploy e di un raggio (la larghezza del cerchio). Dopo l'aggiunta del template si verrà poi rediretti alla vista contenente la lista dei servizi;

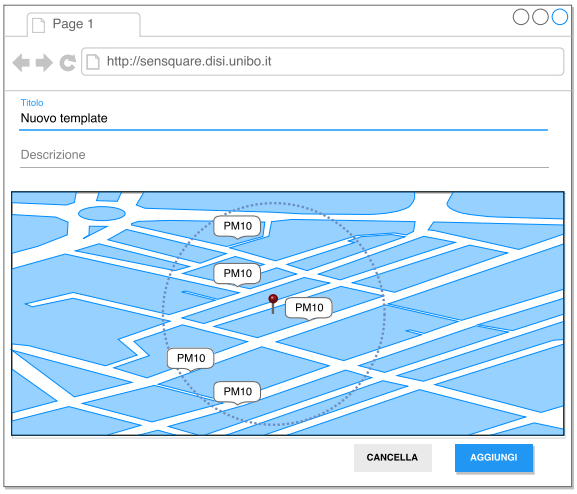

Figura 7.7: View mockup per l'aggiunta di un servizio

Capitolo 8

Convertitore Blockly → Python

La possibilità di dare all'utente un modo di poter scrivere del codice, che viene eseguito in seguito sul server, potrebbe far inorridire molti esperti della sicurezza informatica ma, la scelta di utilizzare la libreria Blockly è stata fatta scrupolosamente ed in maniera ponderata.
Fortunatamente la libreria è stata sviluppata in modo tale che il codice emesso da Blockly sia a tutti gli effetti un XML con tag specifici che si riferiscono ad id specifici. Questi identificativi, in fase di conversione, vengono mappati univocamente in spezzoni di codice. Il mapping ID-Codice è quello che rende sicura l'esecuzione del codice prodotto sul server dall'utente.

Dal momento che la conversione è uno degli aspetti più delicati, tale processo viene eseguito da un modulo specifico che permane sul server.

Il codice creato dagli utenti, se convertito con successo senza presentare errori, viene tradotto in uno script Python. È stato scelto questo linguaggio per via della sua compattezza e facilità di esecuzione. Quando la procedura di conversione di un codice termina con successo, esso diventa un nuovo template su cui è possibile istanziare un servizio.

Parte III

Implementazione

Capitolo 9

Scelte tecniche

Come segnalato nell'introduzione e mostrato in figura **3.2**, questo progetto di tesi è diviso in due parti. Le scelte tecniche perciò risultano svariate; di seguito verranno presentate le più importanti.

9.1 Python

Figura 9.1: Logo di Python [33]

Python è un linguaggio di programmazione ad alto livello e viene ampiamente utilizzato per la programmazione di uso generico. In quanto linguaggio interpretato, Python ha una propria filosofia progettuale che ne semplifica la leggibilità del codice; inoltre, il codice presenta una sintassi che permette ai programmatori di esprimere concetti complessi in poche righe.
Per questo progetto di tesi si è quindi scelto di impiegare l'ultima versione rilasciata, che ad oggi risulta essere Python 3.6

9.2 Django

Figura 9.2: Logo di Django [34]

Django [35] è un Web framework[1] di alto livello gratuito ed open source, altrettanto rilevante è il fatto che sia scritto in Python in modo da favorire uno sviluppo rapido e un design semplice e pragmatico.
Il framework si prende cura di gran parte dei problemi dello sviluppo Web, in modo da permettere allo sviluppatore di concentrare le forze sulla scrittura della applicazione senza bisogno di partire dalle basi.
L'obiettivo primario di Django è quello di facilitare la creazione di siti web complessi e basati su database.
Inoltre, Django mette in evidenza i seguenti tre principi chiave: la riutilizzabilità e "collegabilità" dei componenti, lo sviluppo rapido ed il non doversi ripetere. Python viene utilizzato in ogni sua parte, anche per le impostazioni di file e modelli di dati.
Django fornisce poi un'interfaccia grafica per l'amministrazione del sito opzionale che permette la creazione, lettura, aggiornamento ed eliminazione dei dati. Questa admin page viene generata dinamicamente, attraverso l'introspezione (possibile grazie a Python), e viene configurata tramite i modelli di amministrazione.
Un modello in Django è un tipo speciale di oggetto che viene salvato nel database; esso è il contenitore principale di informazioni sui tipi di dato

[1] Un web framework è un insieme di componenti che aiutano a sviluppare siti web più velocemente e facilmente.

salvati nel database. Il modello contiene i campi e i comportamenti essenziali dei dati che si stanno memorizzando.
Generalmente, ogni modello si mappa in una singola tabella di database ed ogni attributo del modello rappresenta un campo del database.

9.3 Django REST framework

Figura 9.3: Logo di Django Rest Framework [36]

Django REST framework [37] è un toolkit potente, sofisticato e sorprendentemente facile da usare. Esso offre una versione semplice e consultabile delle API che si stanno sviluppando, come anche la possibilità di restituire il risultato della query in un formato JSON grezzo. Django Rest Framework fornisce una potente serializzazione dei modelli e dei dati.

9.4 Typescript

Figura 9.4: Logo di TypeScript [38]

TypeScript [39] è un superset di JavaScript il cui scopo principale è fornire la tipizzazione statica, le classi e le interfacce, note nella grande maggioranza dei linguaggi ad oggetto.
Uno dei grandi vantaggi di utilizzare Typescript è quello di consentire agli IDE di fornire un contesto più ricco l'individuazione degli errori comuni durante la scrittura del codice.
Inoltre, TypeScript è open source e supportato da Microsoft. Per le sue caratteristiche sopra descritte, molti progetti di spessore sono oggi sviluppati utilizzando TypeScript, in particolare Angular e RxJs.

9.5 Angular

Figura 9.5: Logo di Angular [40]

Angular [41] è un framework di sviluppo web open source scritto e mantenuto dal team Angular di Google e viene utilizzato per la realizzazione di applicazioni client in HTML, CSS e TypeScript.
Angular ha diversi punti di forza rispetto ai concorrenti; i tre più importanti verranno presentati di seguito.

Agile platform La compattezza del codebase contribuisce alla versatilità della piattaforma. L'impegno si è spostato ad accogliere gli ultimi browser indipendentemente dal fatto che si tratti di browser desktop/laptop o mobile;

Ideale per lo sviluppo di applicazioni mobili Le ultime versioni si concentrano maggiormente sui dispositivi mobili, apportando miglioramenti alle prestazioni, al consumo di memoria ed alla reattività del controller per il touch screen.

Flessibilità grazie ai Web Components Angular mostra una elevata flessibilità verso tutti i Web Components. Diverse feature precedentemente non supportate, ora lo sono; una fra tutte lo Shadow DOM.

9.5.1 Component Based Development

Un componente è un'unità software indipendente che può essere composta da altri componenti per permettere la creazione di un sistema software più complesso.
Lo sviluppo basato su componenti è il futuro delle web application. Questa separazione in tante piccole unità consente la segmentazione dei componenti, all'interno dell'app, che possono essere scritti in modo indipendente.

9.5.2 zone.js

Zone.js [42] fornisce un meccanismo, chiamato *zones*, per incapsulare ed intercettare azioni asincrone nel browser (ad esempio setTimeout e promise). Queste zone rappresentano contesti di esecuzione, i quali consentono ad Angular di monitorare l'avvio ed il completamento delle attività asincrone, come anche di eseguire i compiti richiesti (ad es. rilevamento delle modifiche).
Angular è l'unico Web Framework che utilizza tale tecnica e ciò lo rende il più veloce a rilevare i cambiamenti di stato, permettendo un data binding reattivo.

9.6 Angular Material

Material Design [43] è una linea guida completa , sviluppata nel 2014 da Google per la progettazione di soluzioni visive, di movimento e di interazione tra piattaforme e dispositivi.
Angular Material [44] è un framework open source che può essere utilizzato per applicazioni personali e commerciali; è gratuito e presenta anche un forte sostegno da parte della community di google che lo sviluppa continuamente.

Il framework ormai conta più di 30 componenti e servizi IU, tutti essenziali per le applicazioni mobili e desktop.

Capitolo 10

Back End

In questo capitolo sarà mostrata una panoramica generale del lavoro atto allo sviluppo del Back End. Le righe di codice scritte per questo modulo sono oltre 3.500, prevalentemente in linguaggio Python (listato **7**). Per questo motivo si procederà alla discussione approfondita solamente delle parti ritenute di maggior rilievo per gli scopi di questa Tesi di Laurea.

10.1 Implementazione delle REST API

Le API per Django REST Framework non sono altro che delle ViewSet, ovvero delle classi "istruite" a compiere diverse operazioni di base.
Le viste implementate in questo progetto sono diverse ma tutte estendono i comportamenti della classe *GenericViewSet*.
Per implementare azioni precise si procede ad una specializzazione, con la quale si intende cosa debba fare una vista quando arriva un certo tipo di richiesta. Ad esempio, se la ViewSet creata estende anche la classe *RetrieveModelMixin*, quando arriva una request di tipo *GET*, questa si occuperà di serializzare i dati richiesti e di inviarli al client. Oppure se la ViewSet creata estende anche la classe *CreateModelMixin*, quando arriva una request di tipo *POST* questa si occuperà creare una nuova entry del modello interessato.
Un esempio di quanto appena descritto è quello riportato nel listato **10.1**,

dove è possibile notare come la classe *ParticipantViewSet* erediti da entrambe le classi citate in precedenza.

```
class ParticipantViewSet(mixins.RetrieveModelMixin, viewsets.GenericViewSet):
    authentication_classes = (JSONWebTokenAuthentication,)
    permission_classes = (IsAuthenticated,)
    queryset = Participant.objects.all()
    serializer_class = ParticipantSerializer
```

Listato 10.1: ParticipantViewSet: la classe che implementa la logica per restituire al client le informazioni del participant richiesto

Due aspetti da non tralasciare sono gli attributi *authentication_ classes* e *permission_ classes* di cui parleremo nella sezione **10.2**.
Mentre l'attributo *queryset* indica su quali modelli effettuare la ricerca. Poi, grazie a *Participant.objects.all()* viene richiesto al database di fare la query su tutta la tabella Participant.
L'attributo *serializer_ class* precisa quale classe deve occuparsi della serializzazione dei dati del participant. Nell'esempio esposto, la sua implementazione è riportata nel listato **10.2**

```
class ParticipantSerializer(serializers.ModelSerializer):
    name = serializers.SerializerMethodField()

    class Meta:
        model = Participants
        fields = ('id', 'name', 'description')

    def get_name(self, obj):
        return Account.objects.filter(
            participant__id=obj.id)[:1].get().get_full_name()
```

Listato 10.2: ParticipantSerializer: la classe che si occupa della serializzazione dei dati del participant richiesto

I campi di nostro interesse vengono elencati nell'attributo *fields* e sono i seguenti tre: l'*id*, il *nome* e la *descrizione*.

Quando un client effettua una richiesta *GET* alla API /api/v1/participant/<pk>/, indicando come *primary key* (pk) un id di un participant esistente, allora verranno restituiti tutti i dati specificati in precedenza.
Si è pensato di implementare una documentazione online al fine di permettere agli utenti di sfruttare le proprietà di questa piattaforma, per favorire sviluppi futuri della stessa o essendo di aiuto nello sviluppo di applicazioni proprie.
Tale documentazione è stata sviluppata tramite un framework chiamato *Swagger*. Swagger [45] è un framework software open source supportato da un ampio ecosistema di strumenti che aiuta gli sviluppatori a progettare, costruire, documentare e consumare i servizi Web RESTful.

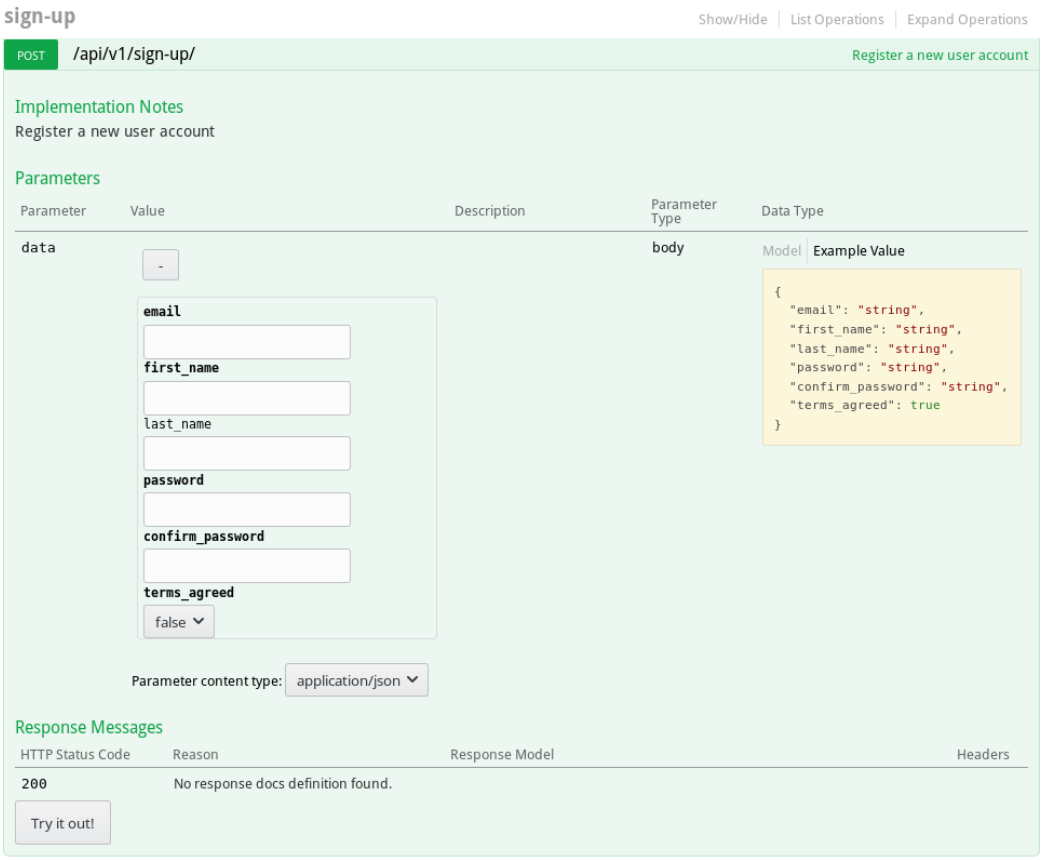

Figura 10.1: Vista di Swagger per la registrazione di un nuovo account.

In figura **10.1** è possibile notare un esempio della documentazione online

creata grazie all'aiuto di Swagger. Il caso presentato mostra quali sono i campi da inviare per una corretta registrazione. È possibile effettuare anche delle prove cliccando sul pulsante *Try it out!*.

10.2 Autenticazione e permessi

In *Django REST Framework* è possibile implementare, con estrema semplicità, meccanismi relativi all'autenticazione ed all'autorizzazione.
La libreria di appoggio *Django REST Framework JWT* [46] estende il framework per ammettere anche un'autenticazione basata su *JSON Web Token* (JWT), così rispettando i requisiti di scalabilità e RESTfulness, senza aver a che fare con le tradizionali sessioni o cookie.
L'attributo *authentication_ classes* introdotto nella sezione **10.1** serve per specificare che si richiede una autenticazione di tipo *JWT* e che deve essere verificata dalla classe *JSONWebTokenAuthentication*.

```
def post(self, request, *args, **kwargs):
    """
    API View that receives a POST with a user's email and password.

    Returns a JSON Web Token that can be used for authenticated requests.
    """
    serializer = self.get_serializer(data=request.data)

    if serializer.is_valid():
        user = serializer.object.get('user') or request.user
        token = serializer.object.get('token')
        response_data = jwt_response_payload_handler(token, user, request)
        return api_return_ok(response_data, status=status.HTTP_200_OK)

    return api_return_error(serializer.errors,
                            status=status.HTTP_400_BAD_REQUEST)
```

Listato 10.3: Codice che viene eseguito alla ricezione di una richiesta di login con metodo POST

Nell'esempio riportato nel listato **10.3** viene mostrato il codice eseguito quando viene ricevuta una richiesta di login con metodo POST. Prima di tutto

vengono inviati i dati della richiesta al serializer che verificherà la correttezza. Se sono validi l'API restituisce un *JSON* (come mostrato nel listato **10.4**) comprendente i dati dell'utente e il JSON Web Token.

```
{
  "ok": true,
  "data": {
    "token": "eyJ0eXAiOiJKV1QiLSC[...]g1IfYsIJ4mH-95F9Pw",
    "user": {
      "id": 1,
      "email": "email@example.com",
      "first_name": "Gianluca",
      "last_name": "Iselli",
      "date_joined": "2017-07-03T09:19:46Z",
      "is_staff": true,
      "participant": 5
    }
  }
}
```

Listato 10.4: Body della risposta di login in formato JSON

Come anticipato nel sezione **6.1** esistono alcune API che sono disponibili solo agli utenti admin. Nel caso descritto vengono utilizzate tecniche per limitare l'accesso basate sui permessi di ciascun utente.
Come per l'autenticazione, esiste un attributo per i permessi che si chiama *permission_ classes*.

```
class IsAdminOrReadOnly(BasePermission):
    """
    The request is authenticated as a user, or is a read-only request.
    """
    def has_permission(self, request, view):
        return (
            request.method in SAFE_METHODS or
            request.user and request.user.is_staff
        )
```

Listato 10.5: Classe che verifica se l'utente ha permessi amministrativi

Nel listato **10.5** è presente un esempio di quanto è semplice creare filtri per avere viste differenti in base al permesso.

10.3 Finezze implementative

Una delle problematiche che affligge solitamente i Back End sono i ritardi di risposta dovuti alla grossa quantità di richieste da parte dei client. Dal momento che, nel set di REST API elencato nel sezione **6.1** sono presenti anche richieste statiche si è pensato di ottimizzare almeno queste grazie a tecniche di caching.
Un'altro fattore non richiesto durante la definizione delle specifiche tecniche, ma comunque portato a termine, è stata la Admin Page. Questa sezione di sito, accedibile solo dallo staff, permette di leggere, creare, modificare ed eliminare (standard *CRUD*) ogni modello all'interno del database.

10.4 Blockly Converter

Anche se facente parte del Back End, il *Blockly Converter* è scollegato dal resto del codice di cui si è parlato fin ora.
Come anticipato nel capitolo **8** si ha la necessità di questo modulo per convertire l'XML della libreria di Blockly al codice Python.
Blockly è uno strumento potente perciò non avrebbe avuto senso riscriversi un nuovo compilatore. È stato più opportuno utilizzare questa libreria per questo scopo. La libreria è stata inclusa in un environment Nodejs, che è un runtime JavaScript basato sul motore V8 JavaScript di Chrome.
Il convertitore presenta un entry point che accetta due argomenti indispensabili: l'XML ed i *json_args*. L'XML è il codice del template mentre i *json_args* sono gli argomenti che bisogna dare al programma python una volta convertito.

Capitolo 11

Front End

Tenendo sempre presente il conteggio mostrato nel listato **7**, anche in questo capitolo verrà mostrata una panoramica generale del lavoro atto allo sviluppo del Front End. Le righe di codice scritte per questo modulo sono oltre le 10000 e prevalentemente in linguaggio Typescript, HTML e Sass. Per questo motivo si procederà alla discussione approfondita delle sole parti ritenute di maggior rilievo.

11.1 Interceptors

Da Angular 4.3 è disponibile una interfaccia per controllare al meglio le comunicazioni HTTP della Web Application. Gli *Interceptor* riescono ad interrompere la richiesta HTTP prima che sia stata inviata per poterla modificare.
Uno scopo degli *Interceptor* è quello di aggiungere l'header*Authorization* a tutte le richieste rivolte alle REST API. In questo modo si avrà un unico punto di controllo senza avere replicazione di codice. La porzione di codice utile a questo scopo è mostrata nel listato **11.1**.

```
@Injectable()
export class AuthInterceptor implements HttpInterceptor {

  intercept(request: HttpRequest<any>,
            next: HttpHandler): Observable<HttpEvent<any>> {

    if (jwtAuthenticationNeeded(request.url)) {
      request = request.clone({
        setHeaders: {
          Authorization: `JWT ${localStorage.getItem(JWT_TOKEN_NAME)}`
        }
      });
    }
    return next.handle(request);
  }
}
```

Listato 11.1: Interceptor che si occupa di aggiungere l'header *Authorization* all'interno di tutte le richieste effettuate dalla Web Application

Un altro meccanismo che si approccia bene all'utilizzo di un interceptor è l'aggiunta degli header *Content-Type* e *Accept-Language*. In questo caso, la porzione di codice utile a questo scopo è mostrata nel listato **11.2**.

Le classi mostrate nei due listati **11.1** e **11.2** mostrano elementi su cui bisogna fare attenzione. Primo, il JSON Web Token viene memorizzato nel *localStorage*. Questo ci permette una sessione molto lunga (dipendente dalla scadenza del JWT) senza evitare di eseguire il login ogni qual volta si accede al sito.
Considerato che le API accettano solo il *Content-Type* di tipo *application/json* quest'ultimo viene impostato per ogni richiesta. Un altro parametro che viene accettato tra gli header è l'*Accept-Language*.

```
@Injectable()
export class ApiHttpInterceptor implements HttpInterceptor {

  constructor(private translate: TranslateService) {
  }

  getUrl(url: string): string {
    return url.indexOf('http://') >= 0 || url.indexOf('https://') >= 0 ?
      url : environment.apiUrl + url;
  }

  intercept(request: HttpRequest<any>,
            next: HttpHandler): Observable<HttpEvent<any>> {
    request = request.clone({
      url: this.getUrl(request.url),
      setHeaders: {
        'Content-Type': 'application/json',
        'Accept-Language': this.translate.currentLang
      }
    });
    return next.handle(request);
  }
}
```

Listato 11.2: Interceptor che si occupa di aggiungere l'header *Content-Type* e *Accept-Language* all'interno di tutte le richieste effettuate dalla Web Application

11.2 Guards

È possibile scomporre il sito di SenSquare in diverse parti, ma di certo, la divisione più tangibile, è data da quello a cui si ha accesso quando si è autenticati e quando non lo si è.
Proteggere dei percorsi è un compito molto comune quando si creano delle applicazioni web, in quanto vogliamo impedire agli utenti di accedere ad aree alle quali non sono autorizzati ad entrare.
Il router di Angular fornisce una funzionalità chiamata *Navigation Guards* che cerca di risolvere esattamente questo problema ed assicurare questa

settorizzazione.

```
@Injectable()
export class AuthGuard implements CanActivate, CanActivateChild {

  constructor(private auth: AuthService, private router: Router) {
  }

  canActivate() {
    if (this.auth.isAuthenticated()) {
      return true;
    } else {
      this.router.navigate(['/about']);
      return false;
    }
  }

  canActivateChild(childRoute: ActivatedRouteSnapshot,
                   state: RouterStateSnapshot): boolean {
    return this.canActivate();
  }
}
```

Listato 11.3: Il servizio AuthGuard garantisce l'accesso solo agli utenti autorizzati reindirizzando, chi non lo è, alla pagina di about

11.3 i18n - Internazionalizzazione

La lingua con cui un programma si interfaccia con l'utente rischia di essere la fonte più importante di impedimento dell'utilizzo del software stesso. Infatti, la lingua utilizzata per l'interfaccia non sempre coincide con la lingua madre dell'utente, il quale perciò probabilmente si troverà a fronteggiare difficoltà di interpretazione.
Come sostegno per questa difficoltà è stata utilizzata una libreria supplementare ad Angular che si chiama *ngx-translate* [47].
Creando differenti file JSON, uno per ogni lingua che si vuole supportare, *ngx-translate* permette di caricare il file con la lingua richiesta dall'utente ed effettuare una traduzione runtime dell'applicazione web.

```
{
    "HELLO": "hello {{name}}"
}
```

Listato 11.4: Esempio di file json con traduzione in inglese. Il file si chiamerà *en.json*

```
{
    "HELLO": "ciao {{name}}"
}
```

Listato 11.5: Esempio di file json con traduzione in italiano. Il file si chiamerà *it.json*

Il funzionamento è semplice, nei listati **11.4** e **11.5** è possibile notare due JSON molto simili. Entrambi hanno un unica chiave *HELLO*, ma un differente valore rispetto a questa chiave. Il valore dipende dalla lingua specificata dal nome del file.

```
// per una traduzione in inglese
translate.use('en');
translate.get('HELLO', {name: 'Gianluca'}).subscribe((res: string) => {
    console.log(res);
    //=> 'hello Gianluca'
});

// per una traduzione in italiano
translate.use('it');
translate.get('HELLO', {name: 'Gianluca'}).subscribe((res: string) => {
    console.log(res);
    //=> 'ciao Gianluca'
});
```

Listato 11.6: Esempio di richiesta di traduzione con inerente interpolazione della chiave *name* con il valore 'Gianluca'

Sarà compito dello sviluppatore o di chi mantiene la web application aggiornare questi file tenendo il passo con l'evolversi dell'interfaccia grafica.

11.4 Schermate

Nel capitolo **7** è stato introdotto uno scenario di utilizzo del sito. In quanto le viste sono svariate ed illustrarle tutte avrebbe poco significato, è preferibile concentrarsi su quelle di maggior valore.

11.4.1 Header e navigazione

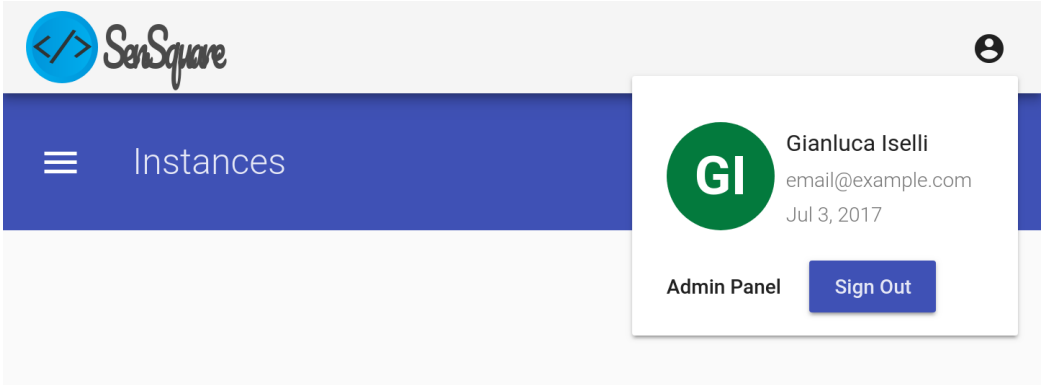

Figura 11.1: Fixed header e *card* con i dati dell'utente

Una costante di tutta la Web Application presentata è l'header con posizione *fixed* in in alto. Questa caratteristica permette di effettuare un reindirizzamento alla Home in qualsiasi istante premendo sul logo a sinistra di SenSquare. A destra invece è situato un pulsante che, se cliccato, fa visualizzare una *card* con i dati dell'utente.
Per permettere la navigazione nelle diverse sezioni dell'applicazione e per fornire funzionalità precise, specifiche di ogni singola schermata, sono state rese disponibili una *navbar* e una *sidebar*.
A sinistra della figura **11.2** è mostrata la sidebar che permette di muoversi all'interno delle sezioni dei template e dei servizi (*Instances*). La navbar di colore blu, invece, serve per mostrare all'utilizzatore in che sezione si trova e fornisce gli strumenti specifici ad ogni vista. Nel caso riportato in figura, il campo di ricerca viene utilizzato per filtrare i template in base al loro

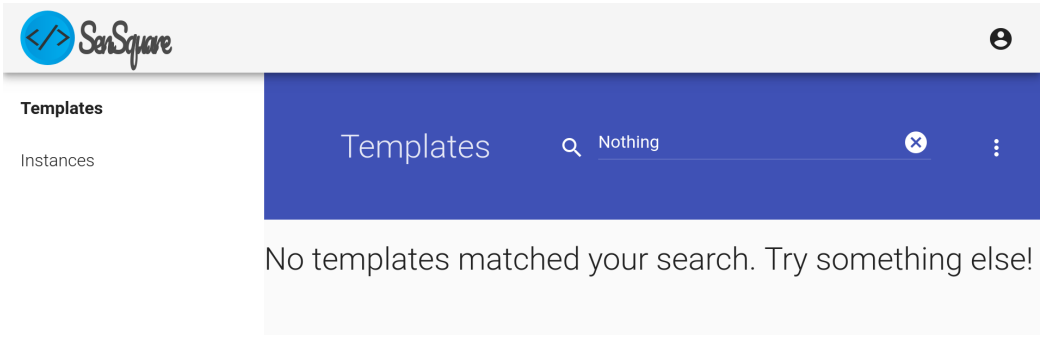

Figura 11.2: *Sidebar* e *navbar* con il campo ricerca

nome. La sidebar in condizioni particolari (schermo di piccole dimensioni o vista particolarmente piena di contenuti) rimarrà chiusa e perciò non visibile. Al suo posto comparirà un *hamburger button* a sinistra della navbar che se cliccato, mostrerà la sidebar.

11.4.2 Home - Lista servizi

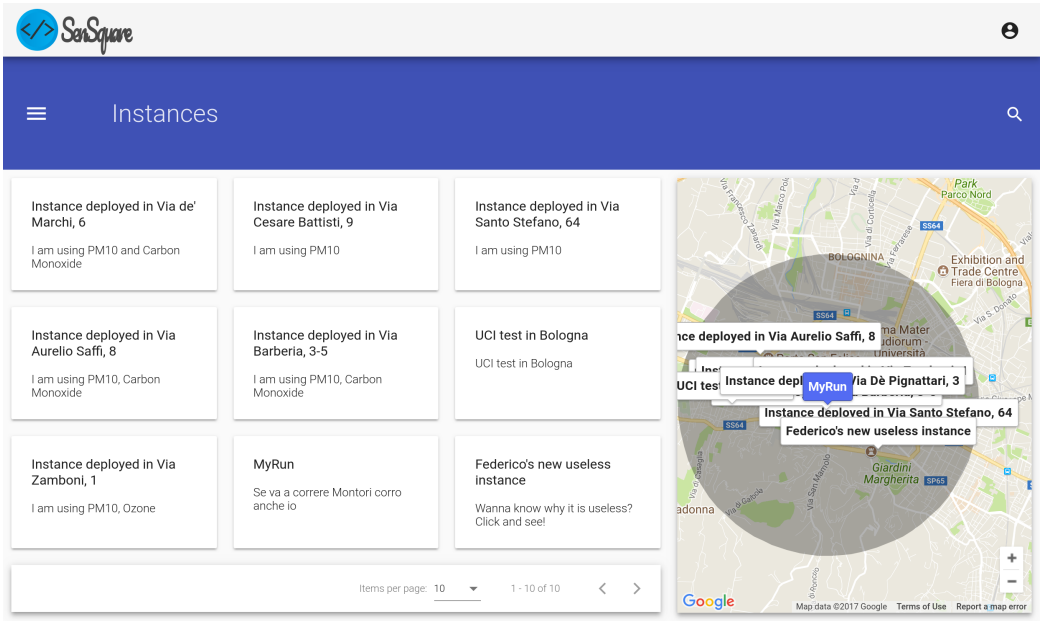

Figura 11.3: Schermata di home

Dopo aver effettuato la registrazione e, successivamente, il login l'utente potrà accedere alla schermata di home.
Questa schermata è composta da due sezioni: la prima, a sinistra, mostra una lista di servizi già istanziati, mentre la parte di destra, mostra una mappa con le specifiche posizioni dei servizi. I servizi della lista dipendono dall'area geografica selezionata nella mappa. Se l'utente muove la mappa i servizi della lista verranno ricaricati.

11.4.3 Dettaglio di un servizio

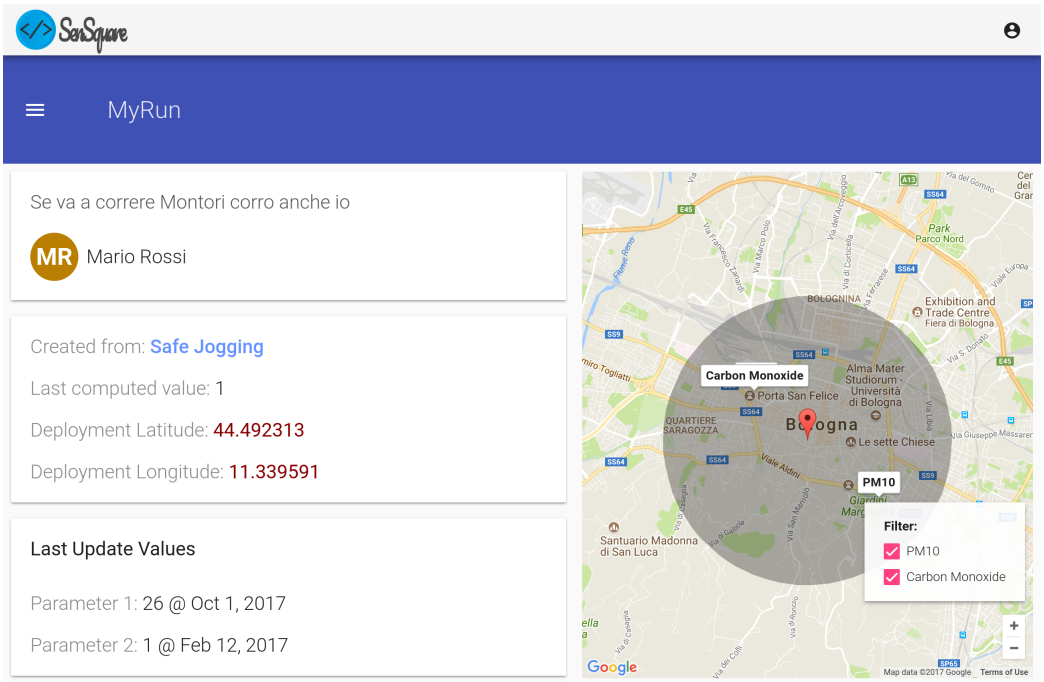

Figura 11.4: Schermata di dettaglio di un servizio - vista desktop

Questa schermata permette ad un qualsiasi utente di tenere traccia gli aggiornamenti dei valori calcolati dal servizio.

È possibile visualizzare tutti i dettagli, per esempio:

- il titolo e la descrizione che sono stati inseriti in fase di creazione;
- l'utente ideatore del servizio;
- il template utilizzato come base;
- la posizione e l'area dove è stato istanziato;
- i Datastream disponibili nella zona specificata;
- l'ultimo valore calcolato ed i parametri passati al codice python per eseguire il servizio.

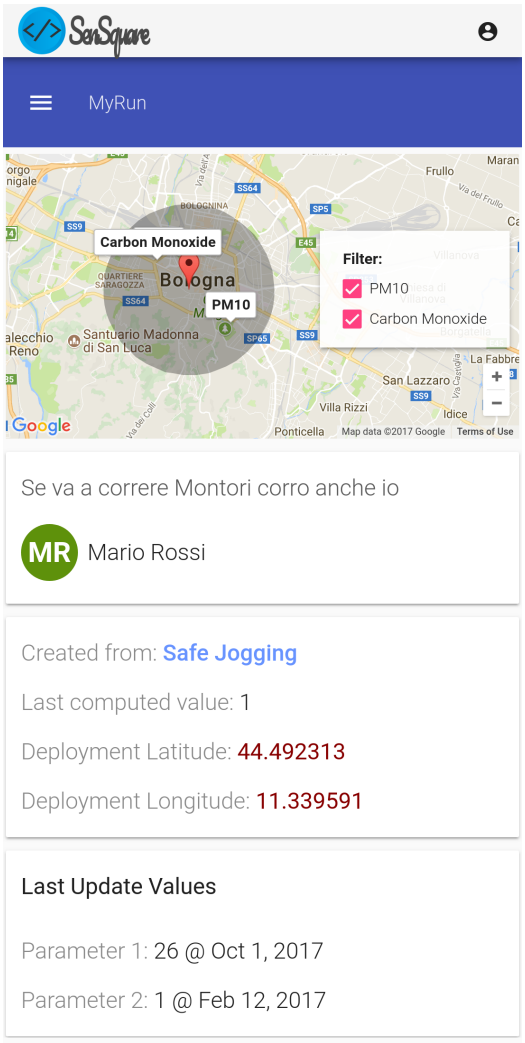

Figura 11.5: Schermata di dettaglio di un servizio - vista mobile

Questa schermata, come tutte le altre, sono disponibili anche in una versione più ristretta per dare la possibilità agli utenti di visualizzare la web application anche dal proprio smartphone o tablet. Nella figura **11.5** viene mostrato come sarebbe visualizzata la schermata su un device mobile.

11.4.4 Aggiunta di un template

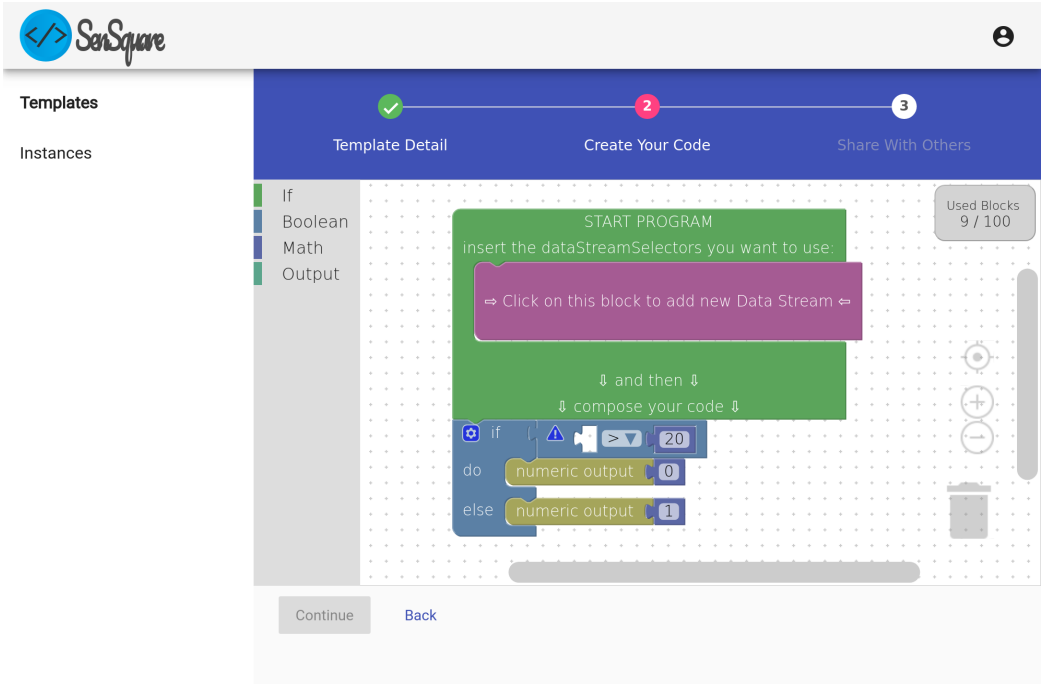

Figura 11.6: Wizard di aggiunta di un template

Nella schermata presentata in figura **11.6** è possibile notare un *wizard* di 3 fasi per la creazione di un template.
La prima pagina di wizard richiede di compilare solo due campi di testo, utili per decidere il titolo ed una descrizione.
Nella seconda pagina di wizard, invece, viene richiesto di creare il codice del template. È possibile definire formalmente un template come il codice sorgente del servizio e le tipologie di argomenti che riceve in ingresso. Come anticipato nella sezione **1.4** è stato scelto di utilizzare Blockly proprio per comporre il sorgente del template.
Nella centrale è possibile vedere i diversi blocchi che compongono il codice. In ogni template sarà presente un blocco di inizio programma (nello screenshot in figura **11.6** è colorato di verde) e un blocco di aggiunta di Datastream (nello screenshot è mostrato in rosa scuro).

Il programma verrà composto connettendo altri blocchi sotto il sotto il blocco di inizio. Come si può notare nello screenshot in figura **11.6**, il programma continua con un blocco *if-else*.
Aiutare l'utente alle prime armi a capire cosa manca o sta sbagliando è fondamentale. Nel caso mostrato, è possibile individuare gli errori tramite il triangolino di *warning che lo segnala*. Nel blocco di confronto, infatti, manca il primo valore.

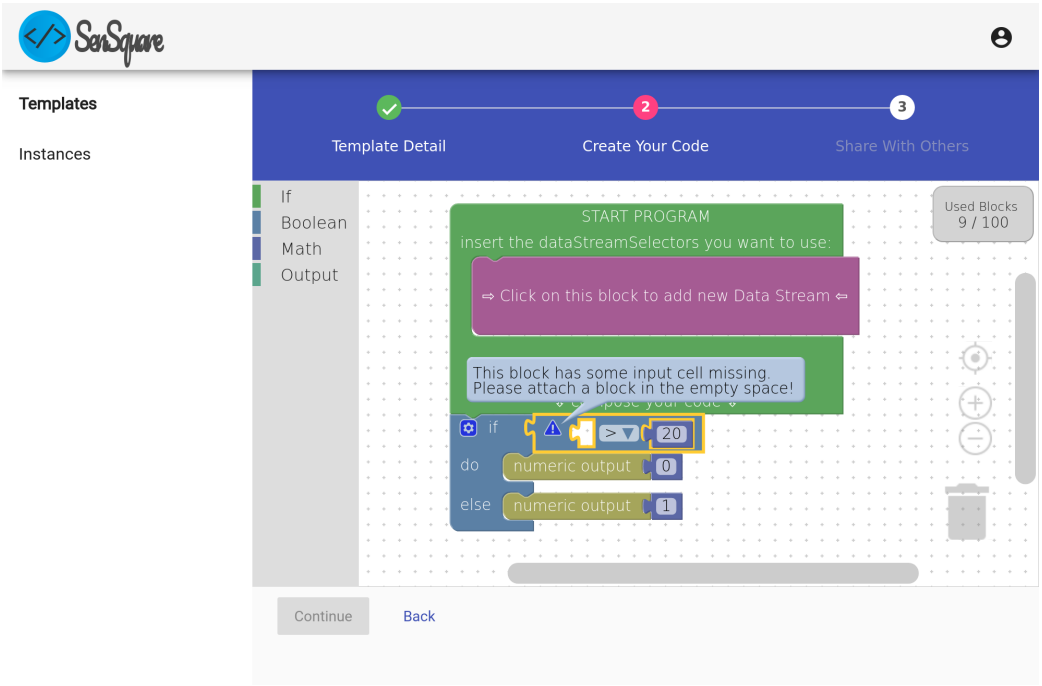

Figura 11.7: Tooltip di segnalazione di un errore

Se si clicca sul triangolino viene mostrato il tooltip con un messaggio di aiuto per l'utente, come mostrato in figura **11.7**.
L'obiettivo di questa tesi, però, è permettere ad un utente di utilizzare le informazioni sensoriali di cui SenSquare è in possesso.
Per far si di aggiungere uno stream di dati bisogna cliccare sul blocco di colore rosa. Così facendo apparirà un modal che chiederà il tipo di Datastream che si vuole inserire e le opzioni. In figura **11.8** viene mostrato il modal sopra citato.

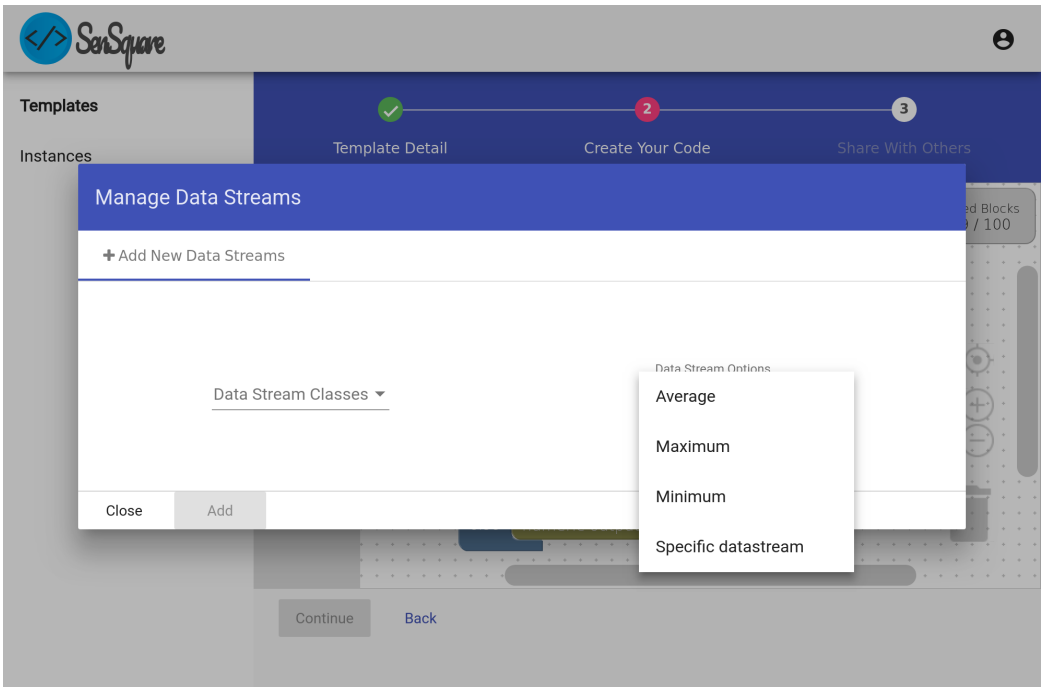

Figura 11.8: Modal per l'aggiunta di un Datastream

Quando si aggiunge un Datastream, nel menù laterale di scelta dei blocchi, risulta disponibile una nuova sezione chiamata, per l'appunto, *Data Stream*.

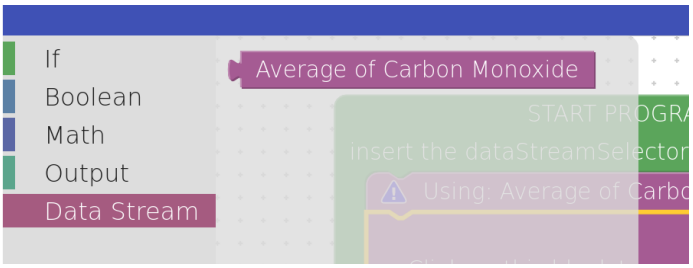

Figura 11.9: Sezione *Datastream* nel menù laterale

Cliccando sulla sezione, si apre una tendina che mostra tutti i datastream aggiunti. Nell'esempio di figura **11.9** è stato aggiunto un Datastream di tipo *Carbon Monoxide* con opzione *Average*. Questo significa che quando verrà istanziato questo template (creando così un servizio) in una certa area si

sta chiedendo la media di tutti i valori di monossido di carbonio nella zona selezionata.

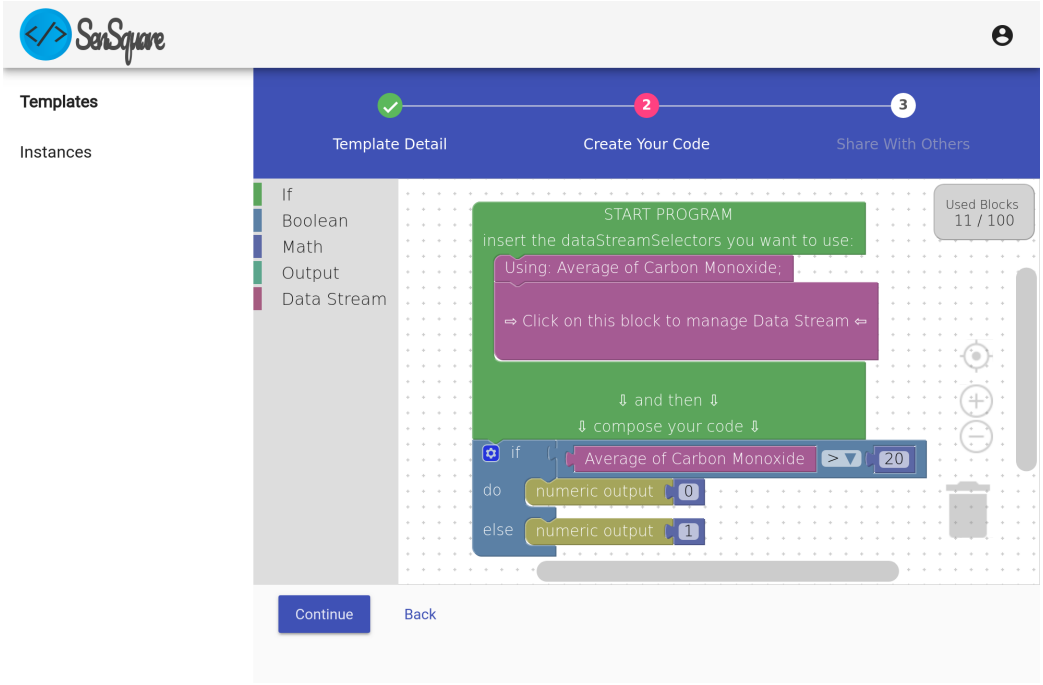

Figura 11.10: Codice del template completo

Quando quest'ultimo blocco di Datastream viene trascinato ed inserito come primo parametro del confronto > 20, il sistema capisce che il programma è sintatticamente corretto, perciò rimuove il triangolino di warning ed abilita il pulsante *Continue* che permetterà di passare alla terza pagina di wizard. Nella terza ed ultima pagina di wizard viene esclusivamente richiesto all'utente se vuole rendere pubblico il template appena creato.

Conclusioni

In questo documento, inizialmente, sono stati introdotti i concetti dell'Internet of Things, insieme a diversi ambiti applicativi inerenti a questo settore. Successivamente è stata descritta l'importanza della collaborazione nel mondo IoT, concentrandosi sui pregi e sui difetti delle piattaforme già esistenti. Nello specifico esistono enti pubblici per la condivisione di *dati ufficiali*, progetti di Open Data e campagne di crowdsensing, i quali ospitano enormi quantità di dati eterogenei.
È possibile constatare, infatti, che la tendenza è quella di adoperare piattaforme per l'immagazzinamento di dati, dove l'utente deve sottostare alle API proprietarie dell'infrastruttura, che solitamente non sono conformi a tutti i dispositivi. Ciò comporta che l'utente debba acquistare da produttori specifici, i quali potrebbero non avere prezzi concorrenziali o non disporre di tutti i dispositivi (o sensori) di cui si ha bisogno; limitando così anche il potenziale di tale piattaforma.
Infine, quanto sopra esplicato è possibile che comporti, relativamente al monitoraggio ambientale, una ridondanza o una indisponibilità dei dati. Potrebbero esserci zone con una sovrabbondante quantità di misurazioni ed altre con nessuna.
In questo lavoro di tesi viene presentata SenSquare: una piattaforma IoT di crowdsensing e sviluppo collaborativo di servizi personalizzati.
Questa piattaforma, a differenza delle controparti, non richiede all'utente

l'impiego di sensori compatibili con l'architettura o con le API. I Datastream e le relative misurazioni vengono ottenuti dalle altre piattaforme di memorizzazione sopraccitate.
Il lavoro di questa tesi non riguarda il processo di ottenimento di questi dati ma su come poterli utilizzare. Il progetto elaborato comprende due parti: il Back End ed il Front End di una Web Application.
Con la progettazione ed infine lo sviluppo di questa applicazione si è voluto dare all'utente uno strumento semplice per creare servizi per se e per la collettività grazie ai dati in possesso di SenSquare.
All'utente viene dapprima richiesto di scrivere il "programma" per il servizio, creando così un modello, chiamato *template*. In secondo luogo, gli viene richiesto di istanziare il suddetto modello.
L'installazione di un servizio significa essenzialmente lanciare il programma scritto dall'utente sul server della piattaforma. Questo potrebbe essere un elemento di sgomento, ma come è stato illustrato nel capitolo **8** è stato studiato e conseguentemente sviluppato un metodo per cui non dovrebbero sussistere problemi di sicurezza.

In conclusione è possibile affermare che, senza ombra di dubbio, il progetto realizzato risponde a pieno gli obiettivi che erano stati prefissati; infatti il lavoro svolto contribuisce ad accrescere le potenzialità, già ingenti, di SenSquare.

Bibliografia

[1] D. Nagel. (2013) 212 billion devices to make up 'the internet of things' by 2020. [Online]. Available: https://thejournal.com/articles/2013/10/07/212-billion-devices-to-make-up-the-internet-of-things-by-2020.aspx (Accessed 2017-12-6).

[2] Cisco. (2013) Connections counter: The internet of everything in motion. [Online]. Available: https://newsroom.cisco.com/feature-content?type=webcontent&articleId=1208342 (Accessed 2017-12-6).

[3] F. Montori, L. Bedogni, and L. Bononi, "A collaborative internet of things architecture for smart cities and environmental monitoring," *IEEE Internet of Things Journal*, 2017.

[4] Xively. Conceptual architecture of xively for a connected product solution. [Online]. Available: https://developer.xively.com/v1.0/docs (Accessed 2017-11-23).

[5] I. LogMeIn, "Xively," 2015.

[6] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for internet of things: a survey," *IEEE Internet of Things Journal*, vol. 3, no. 1, pp. 70–95, 2016.

[7] M. Köhler, D. Wörner, and F. Wortmann, "Platforms for the internet of things–an analysis of existing solutions," in *5th Bosch Conference on Systems and Software Engineering (BoCSE)*, 2014.

[8] J. Boman, J. Taylor, and A. H. Ngu, "Flexible iot middleware for integration of things and applications," in *Collaborative Computing:*

Networking, Applications and Worksharing (CollaborateCom), 2014 International Conference on. IEEE, 2014, pp. 481–488.

[9] A. Alliance. Alljoyn open source project wiki. [Online]. Available: https://wiki.alljoyn.org/ (Accessed 2017-11-24).

[10] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[11] Open Data Institute. What is open data? [Online]. Available: https://theodi.org/what-is-open-data (Accessed 2017-11-23).

[12] F. Blomstedt, L. L. Ferreira, M. Klisics, C. Chrysoulas, I. M. de Soria, B. Morin, A. Zabasta, J. Eliasson, M. Johansson, and P. Varga, "The arrowhead approach for soa application development and documentation," in *Industrial Electronics Society, IECON 2014-40th Annual Conference of the IEEE.* IEEE, 2014, pp. 2631–2637.

[13] B. Guo, Z. Yu, X. Zhou, and D. Zhang, "From participatory sensing to mobile crowd sensing," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on.* IEEE, 2014, pp. 593–598.

[14] Y. Chon, N. D. Lane, F. Li, H. Cha, and F. Zhao, "Automatically characterizing places with opportunistic crowdsensing using smartphones," in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing.* ACM, 2012, pp. 481–490.

[15] C. Leonardi, A. Cappellotto, M. Caraviello, B. Lepri, and F. Antonelli, "Secondnose: an air quality mobile crowdsensing system," in *Proceedings of the 8th Nordic Conference on Human-Computer Interaction: Fun, Fast, Foundational.* ACM, 2014, pp. 1051–1054.

[16] A. Bowser, A. Wiggins, L. Shanley, J. Preece, and S. Henderson, "Sharing data while protecting privacy in citizen science," *interactions*, vol. 21, no. 1, pp. 70–73, 2014.

[17] Siftr. Explore your world, share your discoveries. [Online]. Available: https://siftr.org/ (Accessed 2017-11-30).

[18] K. Systrom and M. Krieger. Instagram. [Online]. Available: https://www.instagram.com/ (Accessed 2017-11-30).

[19] M.-R. Ra, B. Liu, T. F. La Porta, and R. Govindan, "Medusa: A programming framework for crowd-sensing applications," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM, 2012, pp. 337–350.

[20] https://scratch.mit.edu/. (2010) Create stories, games, and animations share with others around the world. [Online]. Available: https://commons.wikimedia.org/wiki/File:Scratch_Logo.svg (Accessed 2017-11-30).

[21] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman *et al.*, "Scratch: programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[22] N. Fraser. Google blockly - the web-based visual programming editor. [Online]. Available: https://github.com/google/blockly (Accessed 2017-11-30).

[23] Google. Blockly - a library for building visual programming editors. [Online]. Available: https://developers.google.com/blockly/guides/overview (Accessed 2017-11-30).

[24] J. P. Morrison, *Flow-Based Programming: A new approach to application development*. CreateSpace, 2010.

[25] Node-RED. (2015) Node-red github repository. [Online]. Available: https://github.com/node-red/node-red/tree/master/editor/images (Accessed 2017-11-30).

[26] P. Through. (2017) What is node-red? [Online]. Available: https://punchthrough.com/bean/docs/guides/node-red/what-is-node-red/ (Accessed 2017-12-1).

[27] F. Montori, L. Bedogni, and L. Bononi, "On the integration of heterogeneous data sources for the collaborative internet of things," in *Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI), 2016 IEEE 2nd International Forum on.* IEEE, 2016, pp. 1–6.

[28] F. Montori, L. Bedogni, A. Di Chiappari, and L. Bononi, "Sensquare: A mobile crowdsensing architecture for smart cities," in *Internet of Things (WF-IoT), 2016 IEEE 3rd World Forum on.* IEEE, 2016, pp. 536–541.

[29] Google. (2017) Android. [Online]. Available: https://www.android.com/ (Accessed 2017-12-3).

[30] A. D. Chiappari, "A collaborative mobile crowdsensing system for smart cities," Ph.D. dissertation, University of Bologna, 2016. [Online]. Available: http://amslaurea.unibo.it/11874/ (Accessed 2017-12-4).

[31] Oracle. (2017) Mysql. [Online]. Available: https://www.mysql.com/ (Accessed 2017-12-4).

[32] M. Jones, J. Bradley, and N. Sakimura, "Json web token (jwt)," Tech. Rep., 2015.

[33] Python Software Foundation. (2017) The python logo. [Online]. Available: https://www.python.org/community/logos/ (Accessed 2017-12-6).

[34] Django Software Foundation. (2017) Official django logos. [Online]. Available: https://www.djangoproject.com/community/logos/ (Accessed 2017-12-6).

[35] Django Software Foundation. (2017) Django makes it easier to build better web apps more quickly and with less code. [Online]. Available: https://www.djangoproject.com/ (Accessed 2017-12-6).

[36] Django REST framework. (2017) Github repository of django rest framework. [Online]. Available: https://github.com/encode/django-rest-framework/tree/master/docs/img (Accessed 2017-12-6).

[37] Django REST framework. (2017) Django rest framework is a powerful and flexible toolkit for building web apis. [Online]. Available: http://www.django-rest-framework.org/ (Accessed 2017-12-6).

[38] Remo H. Jansen. (2017) A community logo for typescript. [Online]. Available: https://github.com/remojansen/logo.ts (Accessed 2017-12-6).

[39] Microsoft. (2017) Javascript that scales. [Online]. Available: https://www.typescriptlang.org/ (Accessed 2017-12-6).

[40] Google. (2017) Angular logo. [Online]. Available: https://angular.io/presskit (Accessed 2017-12-6).

[41] Google. (2017) One framework. mobile & desktop. [Online]. Available: https://angular.io/ (Accessed 2017-12-6).

[42] Google. (2017) Implements zones for javascript. [Online]. Available: https://github.com/angular/zone.js/ (Accessed 2017-12-6).

[43] Google. (2017) Material design is a unified system that combines theory, resources, and tools for crafting digital experiences. [Online]. Available: https://material.io/ (Accessed 2017-12-6).

[44] Google. (2017) Material design components for angular. [Online]. Available: https://material.angular.io/ (Accessed 2017-12-6).

[45] Swagger. (2017) The world's most popular api tooling. [Online]. Available: https://swagger.io/ (Accessed 2017-12-7).

[46] Blimp. (2017) Json web token authentication support for django rest framework. [Online]. Available: https://github.com/GetBlimp/django-rest-framework-jwt (Accessed 2017-12-7).

[47] NGX-Translate. (2017) The internationalization (i18n) library for angular. [Online]. Available: https://github.com/ngx-translate (Accessed 2017-12-7).

Elenco delle figure

Elenco dei listati

Appendice

```
/workspace/SenSquare> cloc .  --exclude-list-file=.clock-exclude-files
     388 text files.
     382 unique files.
      42 files ignored.

github.com/AlDanial/cloc v 1.74  T=1.03 s (351.9 files/s, 19827.8 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
TypeScript                     187           1681            870           7615
Python                          70           1087            483           3678
Sass                            36            323             89           1651
HTML                            38            296             37           1357
JSON                            21              6              0            946
XML                              6              0              0            161
CSS                              2             11             12            102
JavaScript                       3              6              4             68
Markdown                         1             13              0             15
-------------------------------------------------------------------------------
SUM:                           364           3423           1495          15593
-------------------------------------------------------------------------------
```

Listato 7: Esecuzione del programma cloc (Count Lines of Code) sulla directory root del progetto. Si vuole far notare che, nel calcolo, viene considerato solo il codice prodotto per questo lavoro di tesi. Librerie o framework di supporto non sono inclusi nel conteggio.