Università degli Studi di Firenze

Scuola di Ingegneria

Dipartimento di Ingegneria Informatica e dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica

Software Architecture and Methodologies

Analisi e implementazione di un'architettura backend per IoT

Lorenzo Biotti

Alessio Danesi

Anno Accademico 2018/2019

Indice

Elenco delle figure

1 Introduzione

1.1 Contesto

Si vuole realizzare una web application che supporti l'utente nella gestione delle piante del giardino.
In particolar modo, si chiede che l'utente abbia a disposizione, principalmente, i valori di temperatura ed umidità dell'ambiente esterno, con possibilità di aggiungerne di altri in uno sviluppo futuro, come ad esempio la quantità di luce, parametro importante ai fini della "*buona salute*" della pianta (indispensabile per la fotosintesi clorofilliana) e dunque anche un valore di CO_2 presente nell'aria (più la pianta è *sana* e minore sarà la concentrazione di anidride carbonica nell'aria). Saranno altrettanto utili anche parametri che indichino la qualità dell'aria, come la presenza di polveri sottili o concentrazioni anomale di agenti inquinanti.
Oltre a parametri dell'aria e ambiente della pianta, centrale da un punto di vista sia dell'applicazione, che di importanza, sarà il valore di umidità del terreno o vaso in cui si trova la pianta in esame. L'utente deve poter avere a disposizione questi parametri, sia tramite desktop pc, ma anche da dispositivi mobili, nel caso in cui non si trovi a casa e voglia monitorare la situazione da remoto.
Si prevede la suddivisione in ambienti, ognuno dei quali verrà monitorato attraverso i principali parametri di qualità dell'aria circostante. Ogni ambiente (e.g. terrazzo, giardino, salotto, etc.) avrà alcune piante che dovranno essere *mappate* all'interno dell'ambiente in cui si trovano, in modo tale da averne una visione dettagliata della loro disposizione (e.g. giardino: rose, girasoli, ciclamini, etc.).
Aspetto importante sarà dunque la possibilità di poter gestire ambienti e piante, sia in modo automatico, che manuale. Per quanto riguarda la parte automatica, grazie all'utilizzo di particolari controllori, interconnessi con una rete wireless mediante un *gateway*, l'utente potrà decidere l'orario di irrigazione automatica, ovvero l'ora in cui il sistema dovrà bagnare la pianta o quelle per cui tale funzione sia stata attivata. Il sistema, costituito da particolari sensori e attuatori, connessi al controller, deve essere in grado di mantenere un livello di umidità del terreno adatto a quella tipologia di pianta in modo continuo, evitando che tale valore scenda troppo sotto una certa soglia, ovvero che sia troppo arido e secco, ma neppure sia troppo umido (valore eccessivo) e quindi si rischi di annegare la pianta: ogni vegetale infatti ha necessità di avere una determinata quantità di acqua giornaliera a cui corrisponde un preciso valore di umidità del terreno, che l'utente dovrà comunicare al sistema, tramite appositi pannelli di impostazione.
Per quanto riguarda la parte manuale, l'utente dovrà semplicemente disattivare l'opzione automatica e potrà decidere di irrigare, tramite appositi comandi, quando lo riterrà più opportuno: in questo caso il sistema avviserà l'utente nel caso in cui non sia necessario irrigare, ma in caso di conferma, il sistema provvederà comunque ad eseguire il comando impartito.

1.2 Scopo del progetto

Con questo progetto si intendono analizzare ed approfondire gli aspetti architetturali che riguardano la parte di backend in uno scenario IoT, in cui sono presenti sensori collegati tramite una rete. Si vuole analizzare come la struttura tradizionale viene modificata dall'inserimento di software che serve per comandare e far interagire macchine, sensori, etc. senza l'intervento umano. Essendo una struttura complessa ed articolata, con molti dispositivi interconnessi, si rende necessario suddividere il sistema in più sottosistemi in modo da poter garantire una migliore gestione degli stessi e bilanciare il carico di lavoro in modo più uniforme possibile.
I controller avranno il compito di gestire le informazioni provenienti dai sensori, che monitoreranno un ambiente o una pianta; il gateway, unico per ogni abitazione, dovrà gestire il flusso dati fra la casa monitorata e il server backend dell'applicazione; il cloud sarà necessario per lo storage dei dati dei sensori, che verranno richiesti dall'applicazione.
Si cerca di separare il più possibile informazioni e responsabilità di ogni sistema, in modo tale da semplificare la struttura e la gestione: ogni componente avrà le informazioni necessarie per svolgere le proprie funzioni, senza avere conoscenze ulteriori: ad esempio il gateway avrà i dati dal sensore che verranno passati a chi di dovere, senza sapere dove poi verranno effettivamente memorizzati.
Volendo analizzare nel dettaglio la parte backend e come essa viene modificata dal mondo IoT, la parte frontend viene soltanto menzionata e progettata a livello di mockup, cioè come si prevede possa risultare l'interfaccia utente, sia per la versione mobile, che desktop.

2 Analisi dei requisisti

2.1 Requisiti di sistema

Requisiti funzionali

Seguendo il contesto operativo di quest'applicazione, si ricavano i seguenti requisisti di sistema:

Req1 Il sistema deve permettere, ad utenti amministratori, di poter opportunamente configurare le schede con i relativi sensori con le impostazioni di base, come i parametri della rete wireless a cui tutti i controllori si dovranno necessariamente connettere.

Req2 Il sistema deve permettere all'utente, una volta impostati parametri iniziali, di poter decidere se aggiungere un nuovo ambiente oppure una nuova pianta da monitorare, collocata in un preciso luogo, che dovrà essere dichiarato.

Req3 Il sistema deve consentire, così come per l'aggiunta, all'utente di avere avere la possibilità di non seguire più ambienti o piante ad esso collegate; si dovrà prestare attenzione nella cancellazione di un ambiente, poiché potrebbero essere persi tutti i dati relativi alle piante che vi erano registrate.

Req4 Il sistema deve permettere all'utente di poter selezionare la pianta di cui controllare i valori principali, ma anche di gestire la frequenza di aggiornamento di questi valori, riguardanti la gestione del monitoraggio, sia delle piante che degli ambienti.

Req5 Il sistema deve, strettamente collegato al precedente, permettere all'utente di impostare dei valori di limite per i parametri disponibili, configurazioni riguardanti l'irrigazione, quali la durata e la possibilità di sfruttare l'irrigazione automatica con opportuni timer

Req6 Il sistema deve limitare o negare alcune operazioni, quali aggiunta o rimozione di sensori, all'utente base, poiché devono essere svolte soltanto da/in presenza dell'utente amministratore

Req7 Il sistema deve permettere una configurazione iniziale, servizi di gestione e monitoraggio del corretto funzionamento dell'applicazione, con interventi tecnici quando richiesti dallo stesso utente

Requisiti non funzionali

RNF1 Il sistema deve prevedere due interfacce principali, trattandosi di un applicativo per il monitoraggio: una mobile, che può adattarsi ad un dispositivo mobile quale smartphone o tablet per un utilizzo anche da remoto, ed una desktop, per l'utilizzo direttamente nell'ambiente monitorato.

- RNF1.1 Per quanto riguarda la prima interfaccia dovrà avere sufficiente spazio per la visualizzazione dei parametri di controllo, dedicando minor importanza alla parte di gestione e configurazione parametri, che sarà affidata principalmente alla versione desktop. Si prevedono funzionalità associate principalmente a pulsanti che identifichino univocamente l'azione a cui sono associati, cercando di ridurre al minimo il testo presente per la parte mobile.
- RNF1.2 La versione desktop, avendo a disposizione un monitor di dimensioni maggiori rispetto al mobile, prevederà la possibilità di visualizzare uno storico dei dati raccolti su di un grafico; inoltre, poiché installato nei pressi dell'ambiente controllato, dovrà consentire l'accesso alle configurazioni e impostazioni dell'amministratore per poter eseguire le richieste dell'utente di aggiunta/rimozione sensori.

RNF2 Il sistema, ma in particolare il software delle schede, dovrà essere in grado di scalare, poiché non si devono avere limiti riguardo il numero di sensori collegabili ad una pianta o ambiente. Deve essere possibile inserire e togliere sensori senza problemi, ma soprattutto deve essere possibile la coesistenza di sensori diversi con attuatori diversi, per avere maggiore flessibilità nella scelta da parte dell'utente di quale utilizzare in base alle proprie esigenze.

Vincoli

V1 Le schede che verranno utilizzate, richiedono che il firmware sia installato manualmente: per questo motivo, la struttura architetturale deve essere definita a priori. Poiché non è presente un sistema operativo (come per le schede Raspberry), la topologia della rete e l'organizzazione di schede con sensori deve essere predefinita e difficilmente modificabile: si deve prevedere quindi una struttura statica dei sensori e degli ambienti.

V2 L'architettura del sistema, prevedendo una rete di sensori, richiede la presenza di un dispositivo gateway che orchestri i vari controllori e instradi i vari messaggi da/per i vari sensori richiedendo opportunamente i valori registrati e inviare i comandi opportuni ai vari attuatori in base alle configurazioni e azioni dell'utente tramite l'applicazione. Il gateway sarà responsabile della persistenza dei dati registrati dai vari sensori sia su database interno, che su cloud (distribuito); dovrà mantenere copia della *mappa* di ambienti e relative piante, ma anche delle singole configurazioni di ogni elemento.

V3 La figura del tecnico, compreso nel ruolo di utente amministratore, si dovrà quindi occupare di installare il corretto software nelle schede e configurare opportunamente la posizione di tale scheda.

V4 Ogni sensore ha il proprio metodo di acquisizione dati che prevede la definizione a priori, sia del piedino fisico della scheda al quale è collegato, che varia a seconda della tipologia di circuito con cui è stato realizzato: sensori analogici devono necessariamente essere collegati a porte analogiche, mentre quelli digitali a uscite/ingressi digitali. Se questo vincolo non fosse rispettato si avrebbe inconsistenza dei dati o addirittura dei valori errati, a causa di un'errata conversione.
Si prevederà quindi che in ogni scheda sia *pre-caricato* il codice per il relativo sensore che verrà definito a priori. In caso di aggiunta di un nuovo sensore, si dovrà installare una nuova scheda con il nuovo sensore e caricare il software corretto: tale operazione non risulterà costosa, grazie al costo non proibitivo, sia delle schede che dei vari sensori; questo fatto inficerà sulla precisione delle misurazioni, le quali non saranno completamente errate, ma affette da una esigua percentuale di imprecisione, indicata dagli stessi costruttori.

2.2 Modello di dominio concettuale

Seguendo le specifiche del progetto si è inizialmente prodotto il seguente class diagram in forma concettuale:

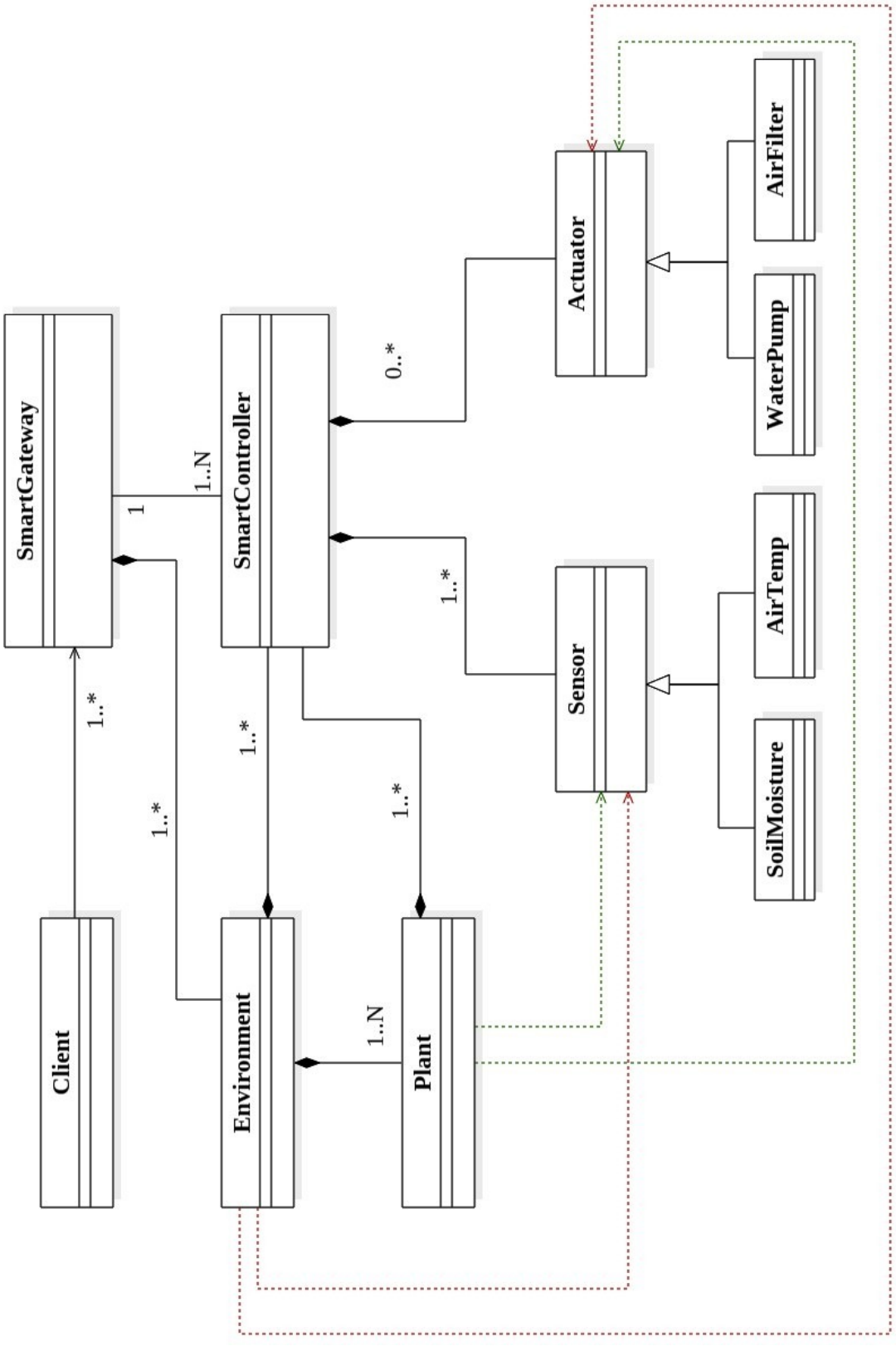


Figura 1: Class diagram concettuale

2.3 Casi d'uso

Per quanto riguarda la tipologia di utenti, si prevedono due categorie principali:

- **User** che rappresenta l'utente finale che utilizzerà l'applicazione. Tale utente, per come sarà strutturata l'applicazione e i casi d'uso previsti per lo stesso, non avrà necessità di contattare l'assistenza per l'aggiunta di una nuova pianta o di un nuovo sensore, ma potrà farlo in autonomia.
- **Admin** rappresenta l'utente amministratore dell'applicazione che interverrà in caso di necessità del cliente per risolvere alcune problematiche di natura tecnica e per le prime configurazioni iniziali, come il settaggio delle impostazioni della rete Wi-Fi a cui i microcontrollori dovranno collegarsi e l'installazione del programma sugli stessi.

Dalle specifiche del progetto e seguendo il modello del diagramma delle classi in forma concettuale riportato nella sotto sezione precedente (2.2), si identificano i seguenti casi d'uso, riportati nel seguente schema:

Figura 2: Use-Case diagram

Casi d'uso: User

1. **SelectPlant** funzionalità che permette di selezionare la piante, tra quelle configurate e gestite, in modo da poterne osservare i valori dei sensori che vi sono stati installati. Questa funzionalità permette anche di modificare i valori personalizzati delle varie impostazioni, tra cui, il più importante il valore limite dell'umidità del terreno.

2. **StartIrrigation** come richiesto nelle specifiche (seczione 1.1), il sistema deve prevedere l'avvio dell'irrigazione in modo manuale, quando richiesto dall'utente. In caso di non necessità o di situazione regolare, prima di procedere, l'utente sarà avvisato di tale situazione e in caso voglia comunque procedere, il sistema avvierà l'irrigazione della pianta selezionata, come richiesto, con tutte le conseguenze del caso (es. valore umidità terreno elevato).

3. **ConfigureParameters** questa funzionalità generale si compone di funzionalità più specifiche, che permettono la configurazione dei parametri per la gestione dei valori rilevati dai sensori delle varie piante.

   3.1 **SetDataUpdateFrequency** consente di impostare l'intervallo con cui i dati vengono aggiornati dai vari sensori. A seconda delle necessità, infatti, è possibile avere una frequenza di aggiornamento più o meno alta, impostando un determinato parametro da passare poi al sistema, che provvederà ad inviare/rendere disponibili tali informazioni con la cadenza richiesta.

   3.2 **SetWateringDuration** permette di regolare la durata dell'irrigazione, sia manuale che automatica. Questo consente di fornire alla pianta una maggiore o minore quantità di acqua a seconda delle necessità, anche in relazione alla stazione: ad esempio in estate, le verdure avranno bisogno di circa 30min di irrigazione tutti i giorni, mentre piante ornamentali/ da giardino soltanto 10 (in caso di irrigazione a goccia).

   3.3 **SetThresholds** collegato alla durata dell'irrigazione, ci sono le impostazioni dei valori limiti, in modo che il sistema possa avvisare l'utente nel caso in cui uno dei sensori rilevi un valore superiore o inferiore a determinati valori. Di particolare importanza per tenere sotto controllo tutti i valori principali delle varie **piante**, tra cui il valore di umidità del terreno, che fornisce un'importante indicazione sullo stato del terreno della pianta, ad esempio in estate come indice di aridità dello stesso e quindi impostare accuratamente la durata dell'irrigazione. Tale funzionalità viene specificata dai due casi d'uso principali:

      3.3.1 **SetEnvironmentThreshold**, che riguarda l'impostazione dei valori limite per le misurazioni relative a dati dell'ambiente esterno, come la temperatura, umidità, etc.

      3.3.2 **SetPlantThreshold** riguarda, come già accennato per il caso generale, l'impostazione dei valori di soglia per i valori rilevati dai sensori sulla pianta, come ad esempio l'umidità del terreno o la luminosità a cui tale pianta è esposta.

   3.4 **ManageAutomaticIrrigation** permette di attivare/disattivare l'irrigazione automatica. Se viene attivato il sistema provvederà ad irrigare automaticamente, per la durata impostata all'orario indicato. In caso di funzione disattivata, sarà compito dell'utente far partire l'irrigazione tramite apposita funzionalità.

      3.4.1 **SetIrrigationTime**[1] è considerata come estensione della precedente funzionalità, poiché permette di impostare l'orario di irrigazione automatica, diversa da quella impostata di def, in cui il sistema dovrà irrigare la pianta selezionata, in modo che possa essere regolata l'ora di irrigazione in base anche alla stagione: ad esempio in estate meglio la sera tardi per evitare di la pianta durante il giorno, in inverno da evitare la sera per via di gelate notturne.

4. **ShowSensorValues**[2] è stato inserito nei casi d'uso per diversificare il comportamento dell'applicazione in base alla tipologia di device su cui sarà in esecuzione: si prevede infatti un utilizzo sia da desktop con la possibilità di avere la visualizzazione di un grafico predefinito che mostri lo storico dei dati, mentre per mobile, si avrà una visualizzazione più compatta e immediata dei principali valori di riferimento richiesti dalle specifiche (sezione 1.1).

[1] Approfondita e dettagliata nella sottosezione successiva

[2] Per ulteriori approfondimenti si vedano le sottosezioni 3.5 e 3.5 della sezione 3.5 riguardanti le interfacce

Casi d'uso: Admin

5. **MakeInitialConfiguration** funzionalità che in realtà viene specificata da altri casi d'uso che riguardano le prime configurazioni iniziali necessarie per avviare l'applicazione che devono necessariamente essere svolte dall'utente con privilegi di amministratore.

   5.1 **UpdateSketchOnTheMicrocontroller** si tratta del caricamento dei vari programmi sui vari controllori e la prima configurazione dei sensori, anche per istruire l'utente sulle procedure da seguire, per le future configurazioni, sfruttando le funzionalità sopracitate.

   5.2 **ConfigureNewHardware** permette all'utente amministratore di poter aggiungere e configurare l'aggiunta di un nuovo microcontrollore, poiché, per quanto riguarda le schede WeMos (in generale quelle simili Arduino), il codice che deve implementare le funzionalità previste, deve necessariamente essere caricato ("*flashato*") sul firmaware della scheda tramite cavo seriale (usb); infatti, non è possibile eseguire tale upload tramite la rete (da remoto).

   5.3 **SetWiFiConfiguration** come la precedente, questa funzionalità specifica meglio quali saranno le funzioni che avrà a disposizione l'admin per procedere con le configurazioni iniziali dell'applicazione. Si ritiene questa funzionalità essenziale per la creazione della rete di sensori; infatti se nel programma caricato sul controllore non ci fossero le impostazioni della rete a cui collegarsi, la rete non si potrebbe creare e i vari sensori non potrebbero pubblicare i dati sul database.

6. **ProvideTechnicalAssistance** questa funzionalità prevede che l'utente amministratore, sia da remoto, che fisicamente presente, fornisca all'utente assistenza in caso di malfunzionamenti dei sensori o dell'applicazione, se avesse dubbi su come collegare i vari sensori oppure per installare eventuali aggiornamenti che devono essere installati sui microcontrollori: in quest'ultimo caso infatti gli aggiornamenti possono essere fatti soltanto manualmente, "*via cavo seriale*".

   6.1 **AddNewEnvironment** viene definito come estensione del caso d'uso precedente poiché consente di aggiungere un nuovo ambiente, in cui l'utente potrà registrare le piante che vorrà monitorare.

   6.2 **AddNewPlant** funzionalità disponibile soltanto nella versione desktop, che permette all'utente amministratore di configurare l'aggiunta di una nuova pianta: tramite procedura guidata, dovrà selezionere l'ambiente, tra quelli a lui disponibili, in cui ha posizionato la pianta (e quindi la scheda); successivamente indicherà il nome e procederà ad impostarne tutti i parametri di configurazione per la gestione automatica, se diversi da quelli di default proposti.

   6.3 **ConfigureNewSensor** una volta configurata una nuova pianta (oppure su una già esistente e precedentemente configurata), l'amministratore potrà aggiungere (o togliere) un sensore alla pianta, tramite procedura guidata: potrà definire un nuovo sensore che è stato applicato alla pianta, scegliendolo da una serie di tipologie di sensori predefinita (es. umidità e temperatura, umidità terreno, radiazioni luminose, CO_2 aria, etc.).

3 Specifiche di progetto

3.1 Specifica architetturale

UML Deployment diagram

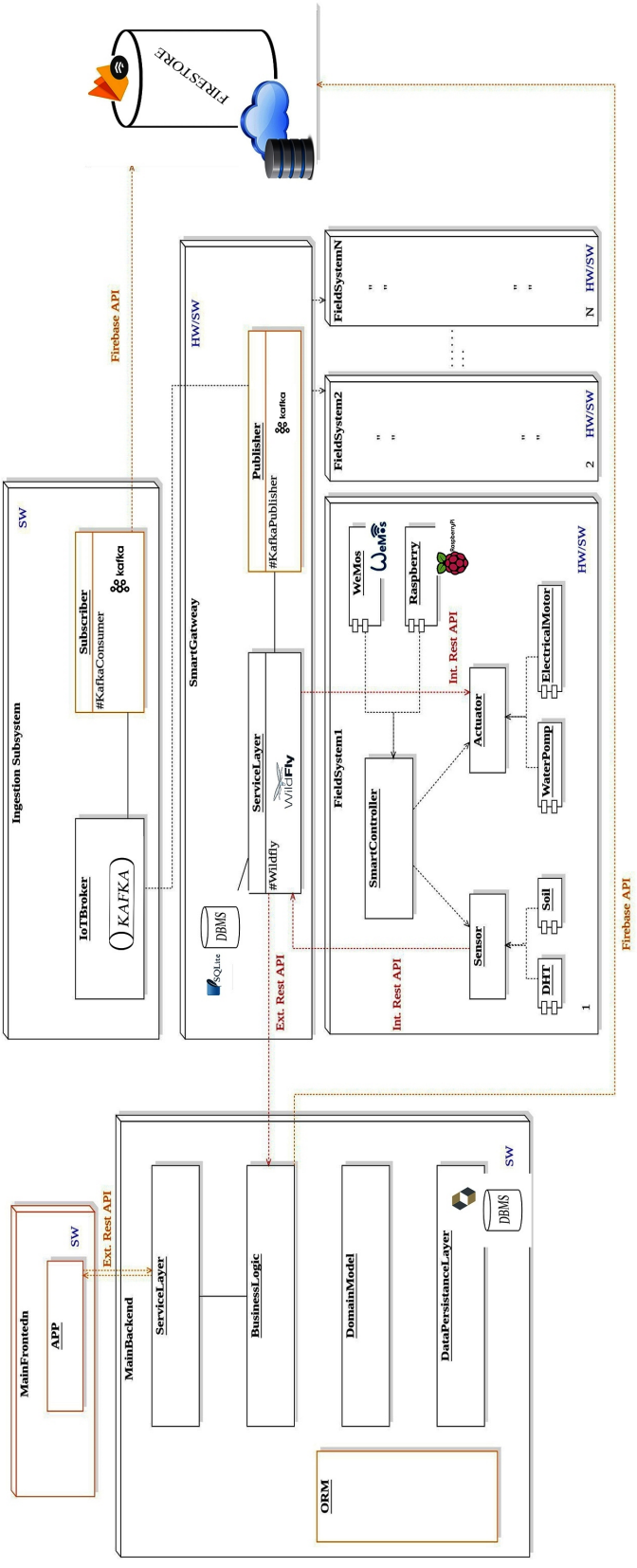


Figura 3: UML Deployment diagram

- ***FieldSystem*** rappresenta la parte hardware e software che opererà direttamente sul campo, in loco presso il cliente. Più elementi di Field System possono essere controllati da uno SmartGateway, in particolare si prevede che un gateway controlli tutto l'apparato installato presso un cliente. È costituito da:
  - **SmartController** ha il compito di gestire sensori e attuatori, inviando i dati raccolti dai sensori con una frequenza decisa dal cliente, impostata tramite interfaccia; ricevere i comandi e redirigerli verso l'attuatore controllato, in modo da compiere quanto impostato dall'utente.
  - **Sensor** hanno il compito di effettuare la lettura dei valori per cui sono stati istruiti (e.g. DHT-11 legge sia umidità che temperatura dell'aria, Soil soltanto umidità del terreno). Il controller chiederà con una certa frequenza il valore che dovrà essere il più preciso possibile.
  - **Actuator** sono dispositivi elettromeccanici molto semplici, come ad esempio piccoli motori elettrici o piccole pompe per l'acqua rispettivamente per il sistema di protezione dai raggi solari o per il sistema d'irrigazione, ad esempio. Sono anch'essi controllati dallo SmartController, che provvederà ad azionarli una volta ricevuto il comando dallo SmartGateway.
- ***SmartGateway*** è il core centrale di tutta l'applicazione installata presso l'utente finale. Ha il compito di gestire i comandi provenienti dall'interfaccia utente, sia di inviare messaggi, opportunamente formattati, al Consumer del DataIngestion System per la persistenza sul database distribuito. Deve riconoscere eventuali nuovi controller con relativi sensori e/o attuatori, mantenendo aggiornata una *mappa* degli stessi, per essere in grado di ricevere i dati dai sensori e inviare i comandi agli attuatori giusti, collegati al controller; deve interpretare i messaggi ricevuti e crearne di nuovi nel giusto formato da poter inviare all'unità preposta per la persistenza su database. Tramite opportune API deve interfacciarsi al Backend per ricevere comandi o impostazioni dall'app e tradurle in istruzioni operative per il field system sottostante.
  - **Internal DBMS** previsto in versioni future (si veda la sezione 7 per ulteriori approfondimenti); permetterà di mantenere copie aggiornate delle impostazioni e configurazioni, come una sorta di backup in locale.
  - **Service Layer** ha il compito di esporre le API necessarie per interfacciarsi sia con i Field System sottostanti sia con la Business Logic del Main Backend. Per quanto riguarda le prime, devono consentire operazioni di CRUD per sensori e attuatori, ma anche la gestione di comandi verso gli attuatori e la lettura (e conseguente invio) dei valori dei sensori. Le API verso il Backend servono principalmente per l'accesso da remoto alle impostazioni e configurazioni dei vari SmartGateway, per inviare opportunamente i comandi agli SmartController.
  - **Publisher** ha il compito di formattare i valori letti dai sensori in messaggi da inviare all'IoTBroker, in modo da poter essere persistiti su database distribuito (cloud).
- ***Ingestion System*** è il sistema che ha il compito di ricevere i dati provenienti dai vari sensori e persisterli sul cloud database.
  - **IoTBroker (Kafka)** rappresenta il *mezzo di trasporto* dei messaggi provienti dal gateway e diretti verso il cloud database.
  - **Consumer (Subscriber)** ha il compito di ricevere correttamente il messaggio dal gateway e di persistere tale dato in modo opportuno sul database, utilizzando le API fornite dallo stesso gestore del database. I messaggi dovranno essere tradotti in un formato compatibile alle API, in modo da poter memorizzare la lettura del sensore.
- ***Mid-Term storage*** è il database distribuito su cloud per la persistenza delle letture dei sensori di ogni gateway.
  - **FirebaseDB (Firestore)** viene utilizzato un cloud database offerto da terze parti, che mette a disposizione anche API per operazioni CRUD sui dati, opportunamente formattati (JSON). Si prevede una struttura unica per tutti i clienti, di cui verranno memorizzate soltanto le letture dei sensori, con rispettive etichette, di ogni pianta e ambiente: sul database si avranno collezioni di gateway, identificati tramite email di registrazione, all'interno dei quali ci saranno collezioni di ambienti e piante, con id e valore delle varie misurazioni.
    Questa struttura deve essere nota all'Ingestion System, in particolare al Consumer, che ha il compito di memorizzare i dati ricevuti dal gateway.
    Quando l'applicazione richiederà i valori misurati da un particolare sensore, sarà compito del Main Backend, in particolare del Service Layer, recuperare queste informazioni per far fronte alla richiesta del client.

- ***Main Backend*** è la parte interna che gestisce le logiche dell'applicazione.
  - **Service Layer** Espone le API per interfacciare l'applicazione con il Field System, attraverso il Gateway (come il Service Layer dello SmartGateway!). Consente di effettuare, per prima cosa, il login e logout dell'utente: l'applicazione infatti, non è utilizzabile se non correttamente autenticati. Una volta autenticati deve essere possibile gestire le configurazioni e i vari dispositivi che sono stati (fisicamente) installati: opportune API, richiamate tramite applpicazione, consentiranno di avere una panoramica dei valori dei vari sensori e di impartire opportuni comandi ai vari controllori.
  - **Business Logic Layer** gestisce tutta la logica dell'applicazione, mediando opportunamente i vari messaggi ai vari gateway, con cui si interfaccia per poter permettere il controllo dei vari dispositivi connessi, invocando opportune API fornite dal Service Layer. Permette di controllare che sia stato eseguito correttamente il login (e logout) ed è in grado di recuperare le informazioni necessarie per l'applicazione, invocando opportuni metodi REST.
    Utilizzando opportunamente le API di Firestore, è in grado di reperire i dati dei sensori, richiesti dall'applicazione, poiché conosce la struttura del database distribuito.
  - **Domain Model** rappresenta la logica di gestione di utenti, gateway e del field system, descrivendo le varie entità che fanno parte o hanno rilevanza nel sistema e le loro relazioni e fornendo una descrizione della struttura (quindi una descrizione statica) del sistema.
  - **Object Relational Mapper** favorisce l'integrazione del sistema software con il RDBMS interno. Mediante un'interfaccia, fornisce tutti i servizi inerenti alla persistenza dei dati, astraendo nel contempo le caratteristiche implementative dello specifico RDBMS utilizzato.
  - **Data Persistance Layer** gestisce la lettura e la scrittura effettive dei dati nella memoria attraverso i DAOs, consentendo la creazione, la manipolazione e l'interrogazione efficiente del database.
  - **DMBS** come indicato nella sezione 7, tale database verrà utilizzato come backup delle principali informazioni di tutti gli utenti, poiché consentirà la persistenza di dati relativi a configurazioni sia utente che dei gateway (e.g. mappature di dispositivi connessi), in quanto i dati relativi alle misurazioni dei sensori sono salvate sul database distribuito.
- ***Main Frontend*** questa parte rappresenta l'interfaccia con l'utente, che consentirà di utilizzare tutte le funzionalità messe a disposizione dall'applicazione
  - **Angular app** consentirà all'utente di interagire con il Field System, impostando le principali configurazioni relative sia ai sensori che agli attuatori. L'applicazione utilizzerà i metodi REST messi a disposizione dal Service layer del Backend per contattare lo SmartGateway, il quale contatterà e comanderà l'ooportuno Field System. Utilizzando le stesse funzioni, ma in modo trasparente all'utente, si potranno avere tutti i valori forniti dai vari sensori sia per le piante, che per gli ambienti.

3.2 Modello di dominio dettagliato

Seguendo lo schema riportato dall'UML Deployment diagram precedentemente descritto, si procede a dettagliare le singole classi che costituiscono i vari componenti dello stesso deployment diagram.

SmartGateway-SmartController-IngestionSystem

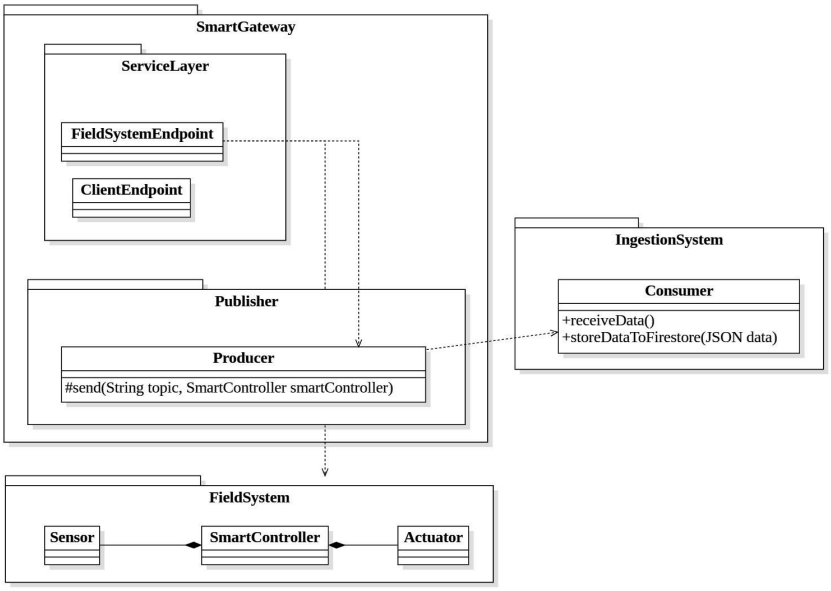


Figura 4: Class diagram in forma implementazione di SmartGateway-FieldSystem-IngestionSystem

Lo **SmartGateway** è costituito da:

- **Service Layer**, costituito a sua volta da
  - **ClientEndpoint**, responsabile di fornire le API per la registrazione iniziale dell'utente al sistema e successivamente per i login
  - **FieldSystemEndpoint**, responsabile di fornire le API verso il FieldSystem per consentire di aggiungere (o togliere) piante o ambienti da monitorare. Viene inoltre utilizzato per ricevere i dati provenienti dalle letture dei sensori che monitorano piante e ambienti, che verranno inviati all'IngestionSystem tramite il broker Kafka. Viene anche utilizzata per l'invio delle impostazioni da parte dell'utente allo SmartController.
- **Publisher**, costituito dalla sola classe
  - **Producer**, che viene utilizzata dal FieldSystemEndopint per inviare i dati provenienti dalle board all'IngestionSystem, dopo opportuna eleaborazione: al dato grezzo verrà associato un topic e lo SmartController che lo ha ricevuto.

Il **FieldSystem** è costituito da:

- **SmartController**, classe che è responsabile del controllo di validità dei dati, in quanto le operazioni principali vengono installate direttamente (e fisicamente) sulla stessa board.
- **Sensor** e **Actuator**, contengono i principali valori di impostazione, quali frequenza di aggiornamento e durata (rispettivamente). Le operazioni di elaborazione del dato vengono effettuate dalla scheda che le controlla.

L'**IngestionSystem** è costituito dalla sola classe

- **Consumer**, che ha il compito di ricevere il dato dal Producer, elaborarlo secondo il topic di provenienza. Una volta controllata e verificata la validità di quanto ricevuto, ha il compito di memorizzarlo opportunamente sul database distribuito Firestore.

Main Backend

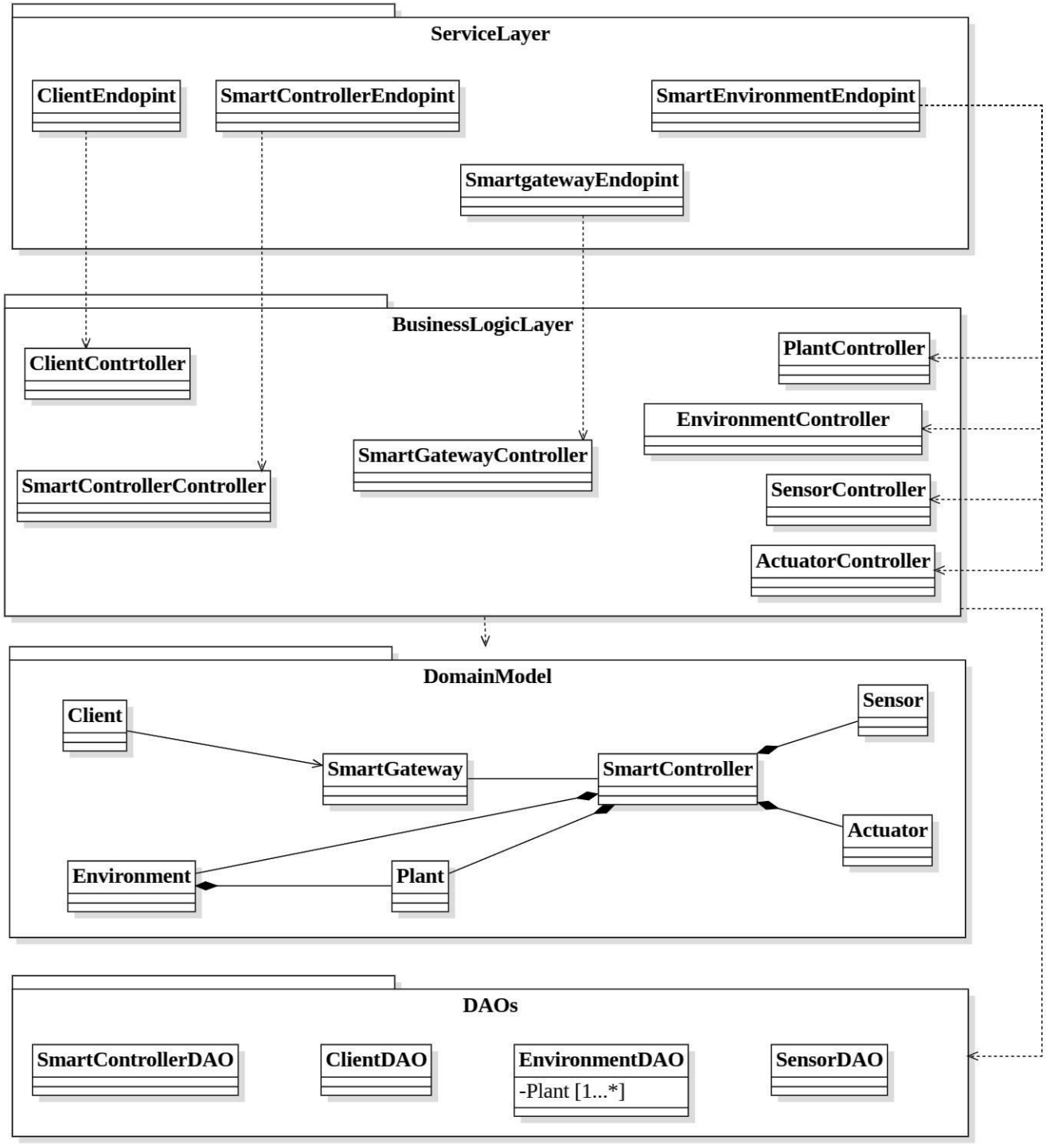


Figura 5: Class diagram in forma implementazione del Main Backend

Come mostrato dall'UML Deployment diagram, il Main backend si compone di più elementi:

- **Service Layer** è costituito da
  - **ClientEndopint** gestisce e implementa le API necessarie per la creazione e registrazione di un nuovo utente; inoltre consente anche di effettuare il login all'applicazione.
  - **SmartControllerEndpoint** gestisce e implementa le API necessarie per l'invio di nuove impostazioni e configurazioni per la gestione del monitoraggio di piante e ambienti, tramite applicazione. Le nuove impostazioni verranno poi passate fisicamente alle varie board interessate.
  - **SmartGatewayEndpoint** gestisce e implementa le API necessarie per controllare tutte le impostazioni di gestione del gateway all'interno del sistema installato presso ogni cliente.
  - **SmartEnvironment**, racchiude in realtà le funzionalità che sarebbero dovute esser implementate da PlantEndpoint, EnvironmentEndpoint, SensorEndpoint e ActuatorEndpoint. Contiene le API necessarie per la gestione delle impostazioni per la gestione di piante, ambienti, sensori ed attuatori.
- **Business Logic** costituita da
  - **ClientController** contiene tutte le operazioni che consentono la gestione degli utenti, come la registrazione e l'aggiunta di nuovi ambienti che si vogliono monitorare
  - **SmartControllerController** contiene tutte le operazioni che consentono la gestione delle board che controllano i sensori e attuatori in ambienti e piante monitorate, come ad esempio l'aggiunta (o rimozione) di un controller, di sensori e attuatori. Inoltre in questa classe viene gestita la frequenza di invio dei dati letti dai sensori verso lo SmartGateway
  - **SmartGatewayController** contiene tutte le operazioni che consentono la gestione della configurazione della rete interna, configurando opportunamente i vari hostname interni.
  - **PlantController** contiene tutte le operazioni che consentono la gestione delle piante.
  - **EnvironmentController** contiene tutte le operazioni che consentono la gestione degli ambienti.
  - **SensorController** contiene tutte le operazioni che consentono la gestione dei sensori
  - **ActuatorController** contiene tutte le operazioni che consentono la gestione degli attuatori.
- **Domain Model**
  - **Client** rappresenta la logica di dominio del client, ovvero l'applicazione frontend
  - **Environment** rappresenta la logica di dominio degli ambienti che vengono monitorati
  - **Plant** rappresenta la logica di dominio delle piante monitorate
  - **SmartController** rappresenta la logica di dominio delle board che controllano sensori e attuatori
  - **Sensor** rappresenta la logica di dominio dei sensori
  - **Actuator** rappresenta la logica di dominio degli attuatori
  - **SmartGateway** rappresenta la logica di dominio del gateway, che deve fare da tramite fra il sistema installato e il main backend.
- **DAOs**, che rappresentano il tramite tra l'applicazione e il database
  - **ClientDAO** consente di gestire le operazioni CRUD sull'entità del modello di dominio del client, che verrà persistita sul database interno.
  - **SmartControllerDAO** consente di gestire le operazioni CRUD sull'entità del modello di dominio dello SmartController, che verrà persistita sul database interno.
  - **EnvironmentDAO** consente di gestire le operazioni CRUD sull'entità del modello di dominio dell'ambiente, che verrà persistita sul database interno, con una lista di piante che sono presenti in quel determinato ambiente.
  - **SensorDAO** consente di gestire le operazioni CRUD sull'entità del modello di dominio dei sensori, che verrà persistita sul database interno, comprendente sia identificativi dei sensori stessi, ma anche loro configurazioni.

3.3 Interfacce di comunicazione

Kafka

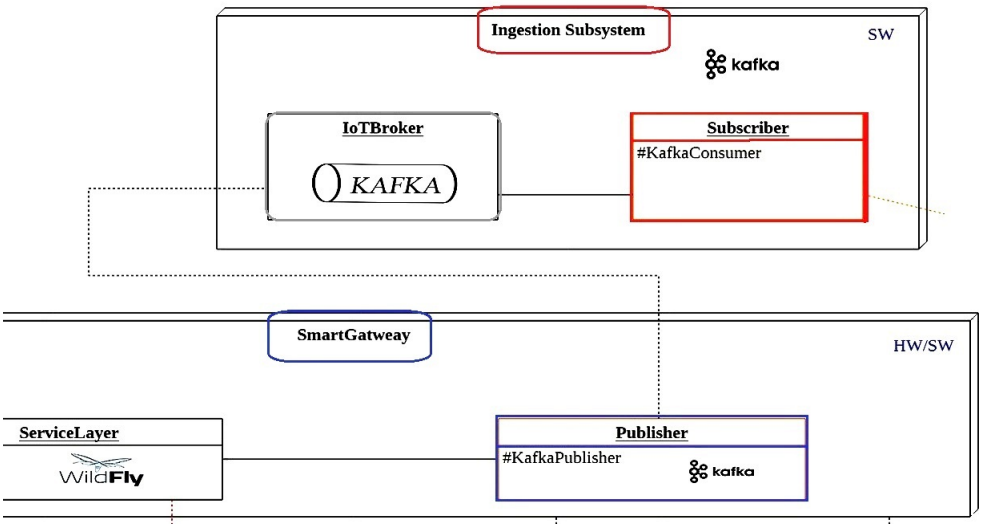


Figura 6: Utilizzo del *"pattern"* Kafka tra IoT Broker e SmartGateway

Il broker di messaggi Kafka è stato utilizzato nella fase di gestione della ricezione dei messaggi e permette di trattare dati di diversa tipologia e provenienza all'interno dello stesso cluster.

Analisi della struttura Kafka suddivide i messaggi in topic, identificati da un nome, e ogni topic in partizioni, identicate da un id progressivo. Un topic raggruppa messaggi dello stesso tipo e prevede la possibilità di fornire una configurazione adhoc potenzialmente differente da quella impostata per altri topic. Le configurazioni di default sono memorizzate in un file di configurazione del broker e sono utilizzate se non espressamente definite in fase di creazione di un nuovo topic. È comunque possibile modificare questi valori nel corso della vita del topic, pagandone il prezzo sotto forma di un tempo di riconfigurazione e ribilanciamento che variano in base alla quantità di log memorizzati e alla precedente configurazione. Le configurazioni riguardano il tempo per cui devono rimanere memorizzati i messaggi, il numero di partizioni, il numero di repliche, la politica di flush dei dati su disco, i timeout per vericare la coerenza dei log nel cluster, ecc.

Topic e partizioni Un producer può inviare messaggi a un certo topic e a una certa partizione. Tale messaggio viene memorizzato sotto forma di log appendendone il contenuto a un file sul file system. Tale file rappresenta una partizione. Un consumer può leggere messaggi da un topic e partizione arbitrari. Può esistere un solo consumer in lettura di una certa partizione e più consumer possono leggere da partizioni differenti dello stesso topic, per cui il livello di parallelizzazione ottenibile nella lettura dei messaggi è limitato dal numero di partizioni denite per quel topic. Per ogni consumer il broker tiene traccia dell'ultimo offset letto e dell'ultimo confermato in modo che sia possibile accorgersi, anche a seguito di disconnessioni forzate o guasti da parte di uno dei due attori, a quale messaggio si era arrivati.
In linea con la metodologia di memorizzazione basata su append, un messaggio all'interno della coda è identificato dal suo offset, cioè dalla sua posizione interna alla partizione.
Per quanto riguarda l'implementazione nel caso di studio in esame, il topic è stato implementato seguendo il seguente schema:

```
{
  "hostname": "myUname",
  "ID":51881166524,
  "Rose-Terrazzo": {
    "name": "DHT11",
    "temparature": 25,
    "threshold": 26
  },
  "frequency": 5
}
```

in modo da semplificare la creazione del pacchetto e successivo invio tramite la rete, con la conseguente facilitazione della lettura da parte del subscriber. Lo schema sopra riportato è un esempio semplificato, in quanto nel JSON potrebbero esserci più di un sensore (List¡Sensor¿), ognuno con gli stessi campi riportati sopra, ovvero nome, valore letto e soglia di tale misurazione.

Push vs. pull: pull-based! Una domanda iniziale che abbiamo preso in considerazione è se i consumer dovrebbero estrarre i dati dai broker o i broker dovrebbero *spingerli* verso il consumatore. A questo proposito Kafka segue un *design più tradizionale*, condiviso dalla maggior parte dei sistemi di messaggistica, in cui i dati vengono inviati al broker dal produttore ed estratti dal broker nel *buffer* del consumatore.
Alcuni sistemi incentrati sulla registrazione, come Scribe e Apache Flume, seguono un percorso basato sul push molto diverso, in cui i dati vengono trasferiti a valle. Ci sono pro e contro di entrambi gli approcci.
Tuttavia, un sistema basato su push ha difficoltà a gestire diversi consumatori poiché il broker controlla la velocità con cui i dati vengono trasferiti. L'obiettivo è generalmente che il consumatore sia in grado di consumare alla massima velocità possibile; sfortunatamente, in un sistema push questo significa che il consumatore tenda a essere sopraffatto, quando il suo tasso di consumo scende al di sotto del tasso di produzione (un attacco di negazione del servizio, in sostanza). Un sistema basato sul pull ha la proprietà più gradevole che il consumatore rimane semplicemente indietro e raggiunge il producer quando può. Ciò può essere mitigato con un qualche tipo di protocollo di backoff, mediante il quale il consumatore può indicare che è sopraffatto, ma ottenere la velocità di trasferimento per utilizzare pienamente (ma mai sovrautilizzare) il consumatore è più complicato di quanto sembri.
I precedenti tentativi di costruire sistemi in questo modo ci hanno portato ad adottare un modello pull più tradizionale. Un altro vantaggio di un sistema basato su pull è che si presta a un batch aggressivo di dati inviati al consumatore.
Un sistema basato su push deve scegliere di inviare immediatamente una richiesta o accumulare più dati e quindi inviarlo in un secondo momento senza sapere se il consumatore a valle sarà in grado di elaborarla immediatamente. Se ottimizzato per bassa latenza, ciò comporterà l'invio di un singolo messaggio alla volta solo per il buffer, che finirà comunque per essere bufferizzato, il che è dispendioso. Una progettazione basata su pull risolve questo problema, in quanto il consumatore estrae sempre tutti i messaggi disponibili dopo la sua posizione corrente nel registro (o fino a una dimensione massima configurabile): in questo modo si ottiene un batch ottimale senza introdurre latenza inutile.
La carenza di un ingenuo sistema basato su pull è che se il broker non ha dati, il consumatore può finire il polling in un circuito stretto, effettivamente occupato, in attesa dell'arrivo dei dati. Per evitare ciò, nella nostra richiesta pull abbiamo dei parametri che consentono alla richiesta del consumatore di bloccarsi in un *sondaggio lungo*, in attesa dell'arrivo dei dati (e opzionalmente in attesa che sia disponibile un determinato numero di byte per garantire grandi dimensioni di trasferimento).
Si potrebbe immaginare altri possibili design, che sarebbero solo pull, end-to-end. Il produttore avrebbe scritto localmente su un registro locale, e gli intermediari avrebbero attinto da quello con i consumatori attirati da loro. Viene spesso proposto un tipo simile di *produttore store-and-forward*: questo è interessante, ma non lo abbiamo ritenuto molto adatto per i nostri casi d'uso.

Web API

Seguendo lo schema dell'UML Deployment diagram, mostrato in figura 3, vengono utilizzate metodi REST nei seguenti sistemi (si vedano le figure 26, 27 e 28 per ulteriori dettagli implementativi):

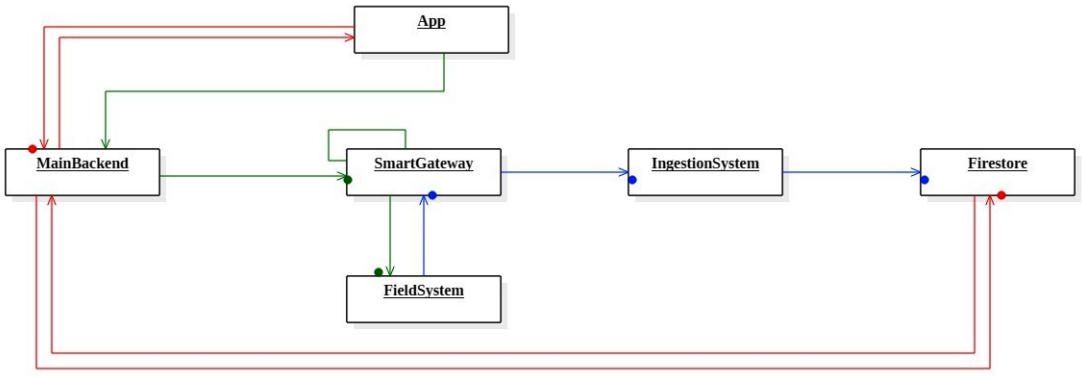


Figura 7: Schema generale Web API

- SmartGateway ⥋ FieldSystem: consentono ai sensori di inviare le letture, quindi allo SmartGateway di ricevere tali dati; è anche possibile comandare opportunamente gli attuatori collegati a quel particolare controller. Nel senso opposto, vengono esposte delle API dallo SmartController che permettono di inviare nuove impostazioni alla board stessa: queste API sono necessarie al gateway per segnalare la modifica delle impostazioni che regolano il monitoraggio delle piante e ambienti; tale modifica viene inviata dal cliente tramite applicazione.

- SmartGateway ⥋ MainBackend: servono come tramite fra l'interfaccia utente e il sistema composta da Gateway e FieldSystem, come una sorta di *bridge*; sono unidirezionali, con il flusso dati, che va dal Backend allo SmartGateway e sono necessarie per trasmettere le impostazioni e configurazioni, che regolano il monitoraggio da parte dei sensori e l'attivazione degli attuatori eventualmente collegati. È compito del MainBackend richiamare il giusto metodo dello SmartGateway con gli opportuni paramentri da modificare, che verranno poi passati al controller del FieldSystem in questione.

- MainBackend ⥋ App: consentono di eseguire il login agli utenti, per poter poi tradurre i comandi dell'interfaccia in azioni e operazioni da far eseguire ai controller, consentendo il controllo dei dispositivi da remoto: come ad esempio la richiesta di aggiornamento di un valore del sensore che verrà direzionata verso Firestore; mentre la modifica di un'impostazione di monitoraggio verrà instradata verso il *sistema di campo* e quindi verso il Gateway.

- IngestionSystem, MainBackend ⥋ Firestore: non sono state direttamente implementate, ma sono state utilizzate quelle messe a disposizione dal gestore del servizio cloud. Come per le precedenti, si individuano due direzioni del flusso informativo (e quindi delle richieste):

  - quelle *in entrata* verso Firestore, sono necessarie per consentire al Consumer dell'Ingestion System di memorizzare i dati dei sensori, che sono stati inviati tramite topic Kafka e sono principalmente metodi per eseguire l'upload di dati sul database cloud
  - quelle *in uscita* da Firestore, sono necessarie al Service Layer del Main Backend per recuperare l'informazione richiesta dall'applicazione e sono principalmente metodi per eseguire il download di dati dal database cloud

3.4 Mockup e descrizione

UC 3.4.1 Set Irrigation Time

Figura 8: Orologio per modificare orario di irrigazione

Nel caso in cui la funzione di irrigazione automatica sia stata attivata, l'utente potrà impostare l'orario in cui il sistema irrigherà la pianta selezionata.
Il valore di default dell'orario è impostato per le ore 18:00, ma l'utente potrà modificarlo, semplicemente premendo/click sull'icona a forma di orologio e comparirà un TimePicker (figura 8):

- In basso troviamo alcuni pulsanti che potrebbero facilitare il settaggio dell'orario:
  - "*Now*" che imposta automaticamente l'orario nell'ora attuale
  - "*Clear*" resetta l'orario inserito in caso di errori o ripensamenti
  - "*Cancel*" annulla ogni modifica fatta e fa scomparire l'orologio per terminare l'operazione di modifica (rimane impostata l'ora precedente)
- Si scelgono dapprima le ore scorrendo il cursore a forma di lancetta fino a raggiungere l'ora desiderato oppure semplicemente premendo sopra il numero corrispondente
- Successivamente comparirà una seconda freccia che permette di impostare i minuti (stesso procedimento delle ore)
- Dopo aver impostato anche i minuti l'orario comparirà in alto, sopra il quadrante (vedere figura 8) per confermare la modifica dell'orario basterà premere su "*Ok*"e una notifica informerà dell'avvenuta modifica.

UC 6.1 AddNewEnvironment

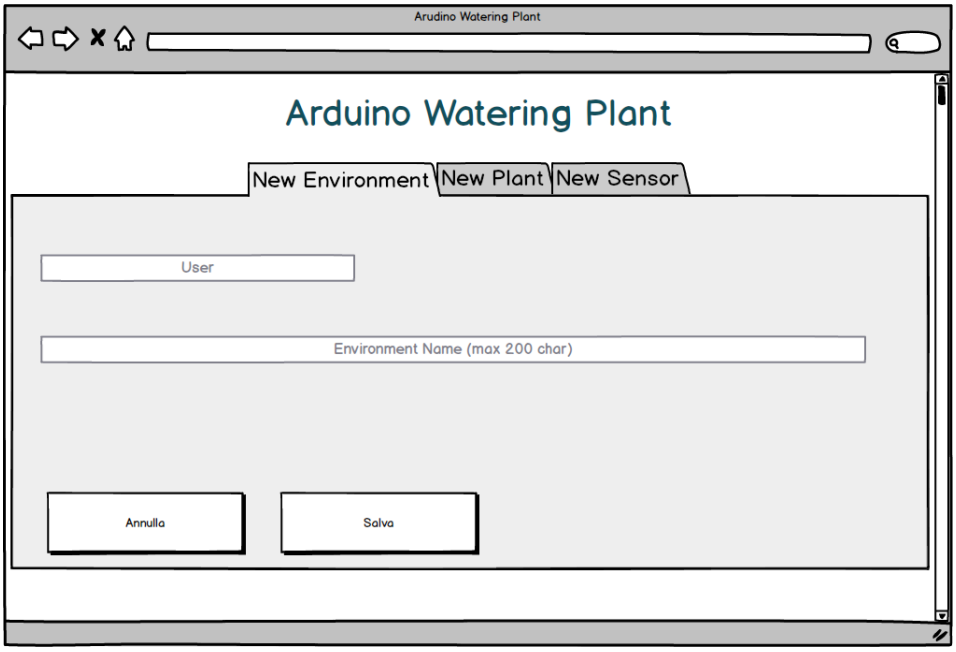


Figura 9: Mockup schermate per l'aggiunta di un nuovo ambiente

Questa funzionalità permette all'utente amministratore di poter aggiungere un nuovo ambiente da monitorare, seguendo la procedura guidata:

1. Per prima cosa dovrà accedere a tale menù tramite pulsante, per aggiungere un nuovo elemento, dal menu principale.
2. L'applicazione indirizzerà verso questa schermata in cui l'utente dovrà selezionare la scheda relativa all'aggiunta dell'ambiente.
3. Successivamente dovrà inserire un nome valido per il nuovo ambiente
   - 3.1 Se dovesse utilizzare un formato non supportato, verrebbe avvisato tramite opportuno messaggio al momento del salvataggio.
   - 3.2 Una volta salvato correttamente, l'utente verrà riportato sulla pagina principale.

UC 6.2 AddNewPlant

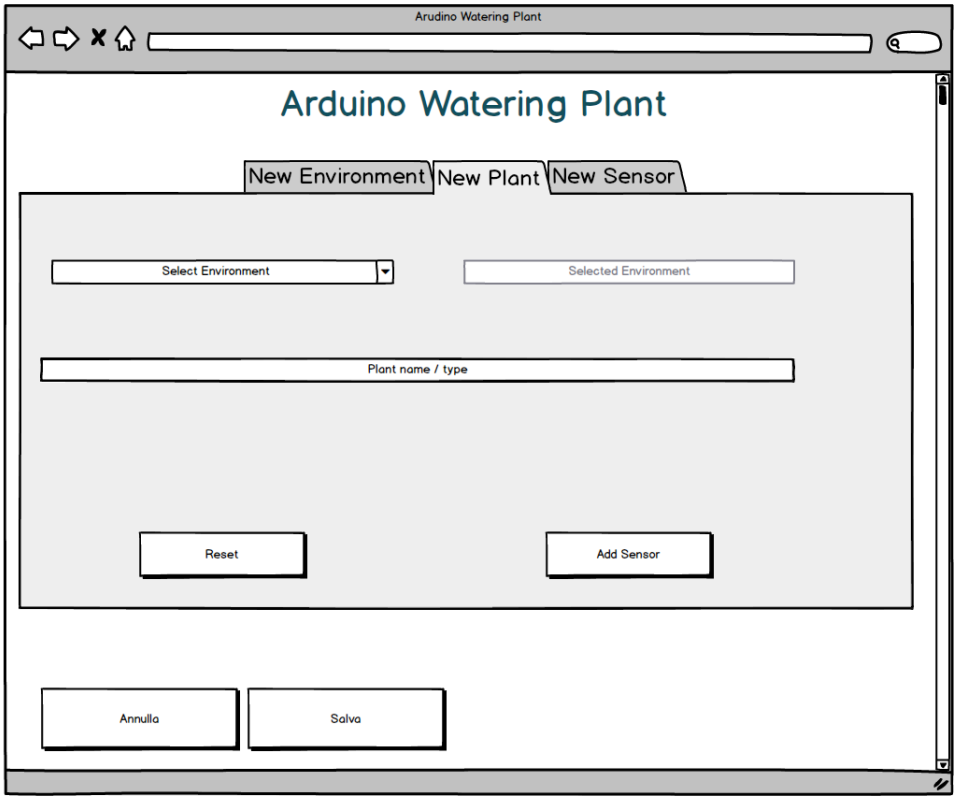
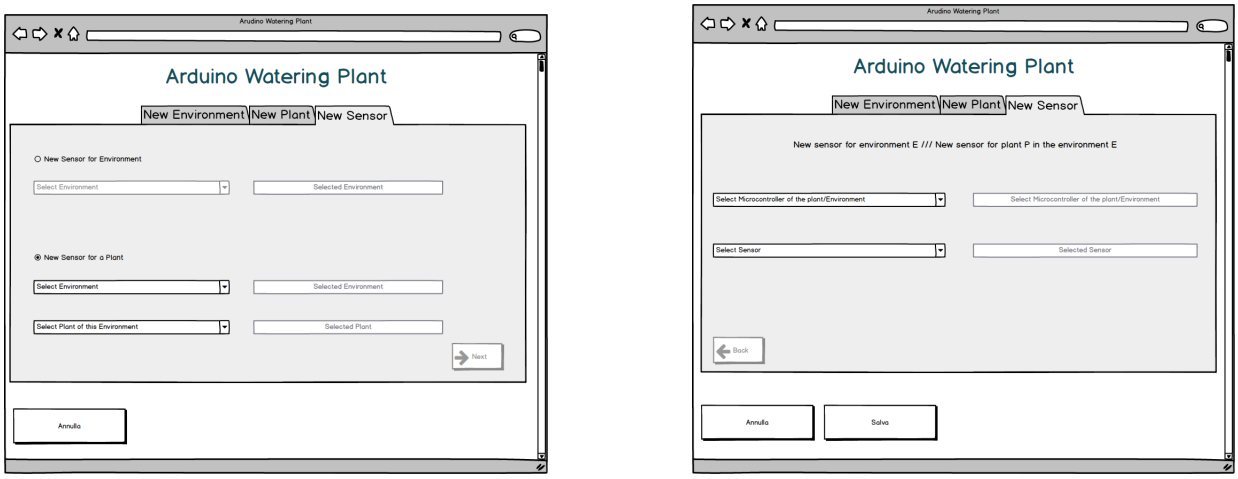


Figura 10: Mockup schermate per configurazione nuova pianta

Questa funzionalità permette all'utente di poter aggiungere una nuova pianta, seguendo la procedura guidata:

1. Per prima cosa dovrà accedere a tale menù tramite pulsante, per aggiungere un nuovo elemento, dal menu principale
2. L'applicazione indirizzerà verso questa schermata in cui l'utente dovrà dichiarare l'ambiente in cui verrà posizionata la pianta
3. Una volta selezionato l'ambiente, l'utente avrà a una casella di input per inserire il nome/tipo di pianta che vorrà inserire.
    - 3.1 Nel caso in cui non venga selezionato l'ambiente, l'utente non potrà effettuare ulteriori azioni se non quella di passare agli altri due voci del menu (uc 2.1 e 2.2) oppure tornare indietro.
    - 3.2 L'utente potrà ripetere questa operazione resettando i campi mediante apposito pulsante per il reset dei valori inseriti e selezionati.
4. Una volta dichiarato il nome della pianta l'utente potrà salvare le modifiche apportate, utilizzando il pulsante "salva", per poterle persistere sul database, ritornando alla pagina principale.
    - 4.1 Se dovessero esserci errori di scrittura sul database l'utente verrà avvisto tramite opportuno messaggio.
    - 4.2 Oppure invece di tornare alla pagina principale, l'applicazione suggerirà di impostare le configurazioni per tale pianta, soltanto nel caso in cui tale pianta sia monitorata da alcuni sensori. Sensor)

UC 6.3 ConfigureNewSensor



(a) Scelta ambiente o pianta

(b) Confgirazione microcontrollore

Figura 11: Procedura per configurazione di un nuovo sensore

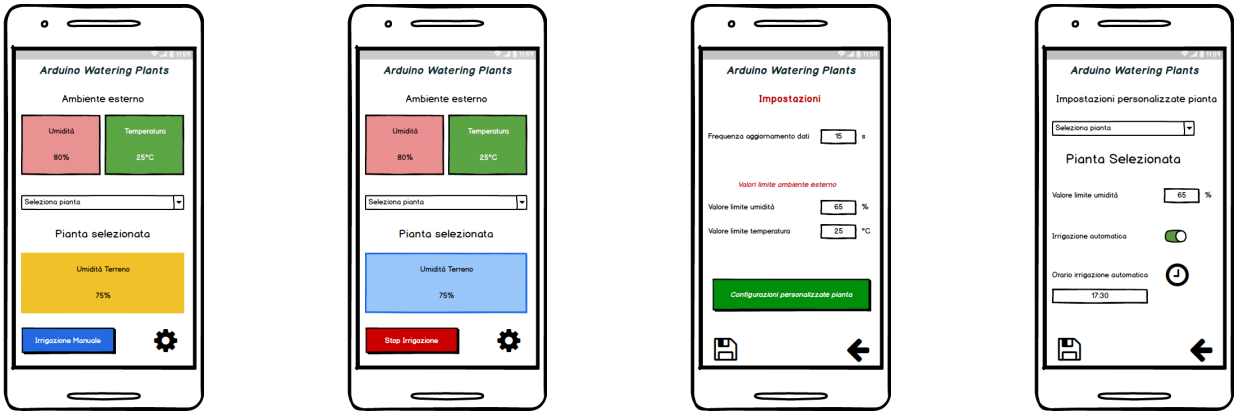
Questa funzionalità permette all'utente di poter aggiungere un nuovo sensore per monitorare una pianta, seguendo la procedura guidata:

1. Accedere a tale funzionalità in modo diretto
   - 1.1 Per prima cosa dovrà accedere a tale menù tramite pulsante, per aggiungere un nuovo elemento, dal menu principale
   - 1.2 L'applicazione indirizzerà verso questa schermata in cui l'utente dovrà selezionare la scheda relativa all'aggiunta del sensore
2. L'utente potrà arrivare in questo punto anche dopo aver configurato correttamente una nuova pianta da monitorare e utilizzando il pulsante per configurare un nuovo sensore, come suggerito dall'applicazione
3. La prima scelta che dovrà fare l'utente riguarda il luogo operativo del sensore:
   - Ambiente
   - Pianta
4. Effettuata la scelta, compariranno dei combobox appropriati per la selezione dell'ambiente oppure sia per l'ambiente che per la selezione della pianta dove verrà installato il sensore.
5. Una volta effettuate tutte le scelte, il pulsante per procedere alla fase successiva verrà attivato e sarà possibile passare alla configurazione successiva.
6. A questo punto l'utente avrà a disposizione un riepilogo della scelta precedente e la possibilità di definire
   - 6.1 Il tipo di microcontrollore a cui verrà collegato il sensore, fra quelli presenti nell'ambiente o pianta selezionati.
   - 6.2 Il tipo di sensore (ad esempio: dht11, 18b20, luminosità, etc.), poiché ogni sensore ha il proprio metodo di acquisizione dati.
   - 6.3 Il PIN al quale tale sensore è stato collegato, scegliendolo fra quelli disponibili nel microcontrollore scelto
7. Una volta eseguite queste scelte l'utente potrà salvare le configurazioni oppure annullare quanto fatto, tornando in ogni caso alla pagina principale
   - 7.1 Come per altri casi, in caso di errore l'utente verrà avvisato tramite opportuno messaggio
   - 7.2 L'utente può anche decidere di tornare al passo precedente, per modificare eventuali scelte fatte; in caso di modifica, le scelte prese nella fase precedente verranno cancellate e dovranno essere rifatte sulla base delle nuove scelte

3.5 Interfacce utente

In questa sezione sono state realizzate delle prime bozze schematiche delle interfacce risultati, che dovranno realizzare le funzioni richieste dalle specifiche (sezione 1.1) e concretizzate dal diagramma degli use-case (sezione 2.3 e mostrati in figura 2).

Mobile



(a) Schermata principale

(b) Irrigazione in corso

(c) Impostazioni generali

(d) Impostazioni personalizzate pianta

Figura 12: Progetto schermate applicazione versione mobile

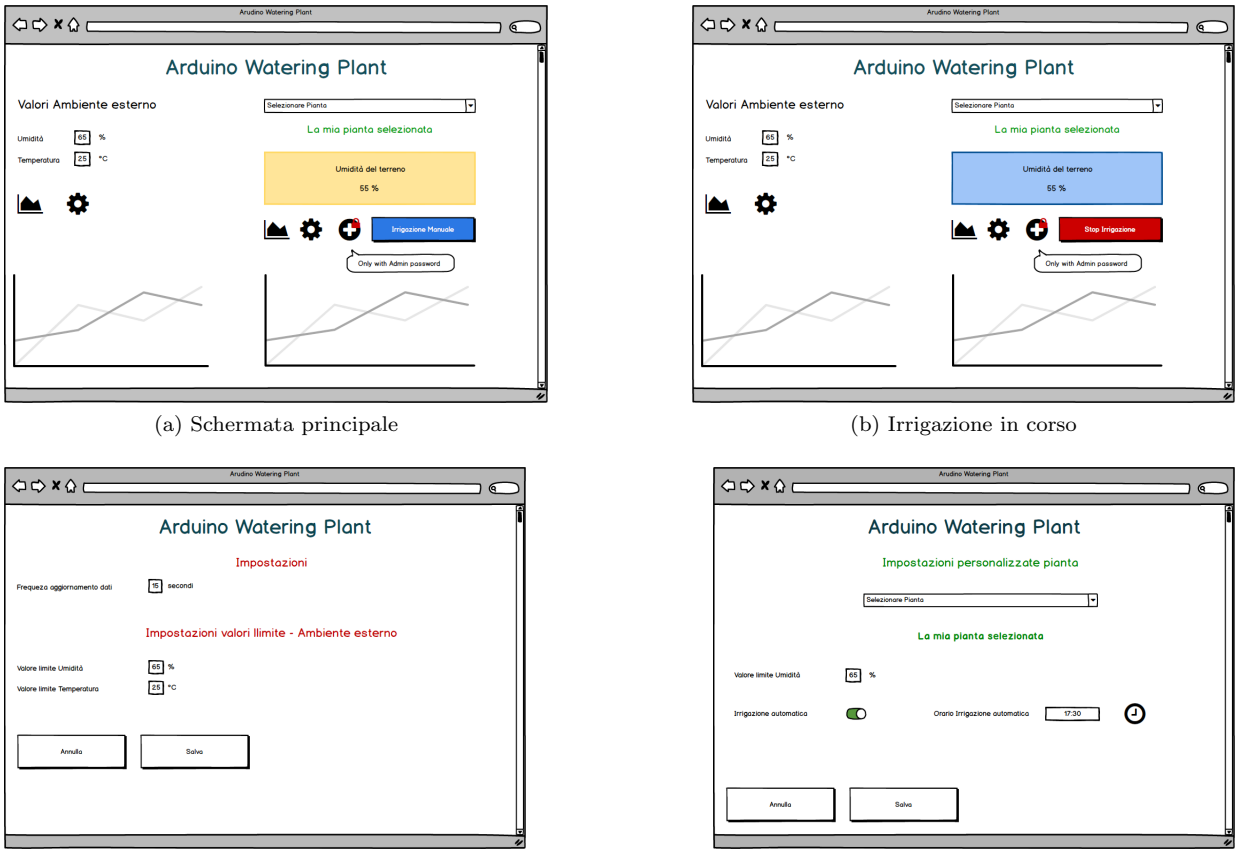
Nella schermata principale dell'interfaccia mobile l'utente avrà a disposizione i valori principali di umidità e temperatura dell'ambiente esterno alle piante, vicino cui saranno installati i sensori di umidità del terreno. Selezionando la pianta dall'elenco di quelle configurate, sarà possibile valutarne lo stato oppure irrigarla, qualora l'utente lo ritenga necessario.
Tramite apposito pulsante sarà possibile accedere alle configurazioni di principali valori per la gestione sia complessiva che specifica di ogni pianta. L'utente potrà settare la frequenza di aggiornamento dei dati prelevati dai sensori a seconda delle esigenze sia personali che in base alla pianta monitorata e impostare i valori di soglia dei parametri ambientali.
Tramite pulsante dedicato sarà inoltre possibile passare alla configurazione dei valori della pianta selezionata, tra cui il valore limite di umidità del terreno e la possibilità di attivare l'irrigazione automatica, modificando, se necessario l'orario di avvio della stessa[3].
Per quanto riguarda la funzione di irrigazione manuale da parte dell'utente, una volta attivata, si presenteranno due casi:

- se il valore di umidità del terreno è ottimale, l'utente verrà avvisato e sconsigliato nel procedere con tale operazione
- altrimenti, il sistema inizierà la procedura di irrigazione con la durata impostata.

In entrambi i casi, se fosse in corso tale operazione, l'utente potrà notarlo dal bordo del box contente il valore di umidità del terreno che verrà colorato di blu e il pulsante sarà disabilitato durante tutta la durata dell'irrigazione.

[3]Si veda la sottosezione 3.4 per ulteriori approfondimenti

Desktop



(a) Schermata principale

(b) Irrigazione in corso

(c) Impostazioni generali

(d) Impostazioni personalizzate pianta

Figura 13: Progetto schermate applicazione versione desktop

Le interfacce desktop, poiché anch'esse conformi alle specifiche (sezione 1.1) e ai casi d'uso (sezione 2.3), presentano funzioni e caratteristiche molto simili a quelle analizzate nella parte mobile (sottosezione 3.5), con la differenza di alcuni aspetti puramente grafici, in quanto si dispone di un display di maggiori dimensioni rispetto a quelle di un dispositivo mobile, come uno smartphone. L'aggiunta principale rispetto alla versione desktop riguarda la possibilità per l'utente amministratore di poter configurare un nuovo sensore per una pianta (aggiunta o rimozione) o di aggiungere una nuova pianta per un determinato ambiente.
Altre differenze riguardano principalmente l'impaginazione e la dimensione dei riquadri per mostrare i valori provenienti dai vari sensori, ma anche la possibilità di accedere separatamente alle impostazioni generali e specifiche per la pianta: nella versione mobile questa funzione non è presente in quanto è stata data la precedenza di occupazione dello spazio alla parte di visualizzazione dei dati e non è stato possibile aggiungere un bottone separato per le varie impostazioni. Una futura implementazione mobile potrebbe ridurre lo spazio dedicato alla visualizzazione dei valori, a vantaggio dell'aggiunta di un pulsante separato per le impostazioni, generali e specifiche della pianta.

4 Hardware utilizzato

Board WemosD1

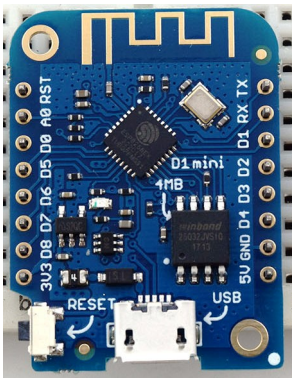


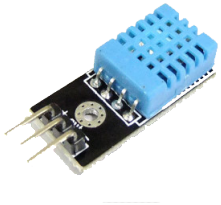
Figura 14: WeMos D1 mini

Questa mini scheda è una versione *clone* di Arduino uno, veramente molto piccola, ma è anche molto più potente e possiede delle caratteristiche in più, prima fra tutte ha il WiFi incorporato ed è così accessibile tramite indirizzo IP in una rete LAN. Questo significa che potete far girare su di essa un programma e vederne gli output in remoto per esempio tramite un browser (client). Wemos possiede anche una serie di interfacce e sensori adattabili che ne fanno un piccolo kit per IoT.
Wemos D1 Mini Possiede un microcontrollore esp8266 che è molto potente e che si può programmare tramite la IDE di Arduino proprio come fate con Arduino e addirittura è possibile utilizzare la Ide a blocchi proprio come Scratch. Sul modulo D1 c'è l'interfaccia USB con cui la collegate al computer come succede con Arduino. Una cosa molto importante è che i PIN sono tutti a $3.3V$ e quindi bisogna stare attenti perché molti dispositivi usati per Arduino qui non vanno bene.

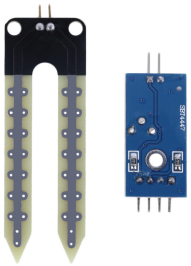
Specifiche tecniche

- CPU Esp8266
- 11 PIN digitali
- Un pin analogico
- Un connettore micro USB
- Compatibilità con Arduino e NodeMCU
- Dimensioni 2 cm x 3 cm
- Interfaccia i2c D2 e D1 che realizzano un bar su cui è possibile collegare 127 nodi.
- Interfaccia SPI (PIN D5 D6 e D7) su cui è realizzato un collegamento Masters/Slave

Sensori di umidità



(a) DHT11



(b) SOIL

DHT11 Il sensore DHT11 è un sensore di temperatura e umidità con uscita dei dati in formato digitale. Il sensore utilizza una tecnica digitale esclusiva che unita alla tecnologia di rilevamento dell'umidità, ne garantisce l'affidabilità e la stabilità. I suoi elementi sensibili sono connessi con un processore 8-bit single-chip. Ogni sensore di questo modello è compensato in temperatura e calibrato in un'apposita camera di calibrazione che determina in modo preciso il valore di calibrazione il cui coefficiente viene salvato all'interno della memoria OTP. Le sue piccole dimensioni e suo basso consumo unite alla lunga distanza di trasmissione (20 m) permettono al sensore DHT11 di essere adatto per molti tipi di applicazioni.
Di seguito, vengono riportate alcune specifiche di tale sensore:

Alimentazione	$3 - 5.5VDC$
Segnale di uscita del segnale	digitale tramite single-bus
Elemento sensibile	Resistenza in Polimero
Campo di misura umidità	$20 - 90\%$ di umidità relativa
Campo di misura temperatura	$0 - 50^oC$
Precisione umidità	$\pm 4\% RH$ (Max + -5% di umidità relativa)
Precisione temperatura	$\pm 2.0^oC$
Risoluzione o la sensibilità	umidità 1% di umidità relativa, temperatura 0.1^oC

Tabella 1: Tabella con alcune delle specifiche più significative del sensore DTH11

SOIL Questo sensore non misura l'umidità dell'aria, come per esempio con un sensore $DHT11$, ma l'umidità del terreno, e si può usare, per esempio, per misurare l'umidità nel terreno delle piante. Il valore viene elaborato elettronicamente nel sensore e trasmesso sotto forma di un segnale analogico ad un ingresso analogico della scheda: in questo caso la scheda, non misura la tensione elettrica come tale, ma converte il segnale analogico presente in ingresso in un valore numerico. Per cui una valore da $0 \div 5V$ corrisponde a un valore numerico da 0 a 1023 (1024 valori, poiché lo zero è considerato come il primo numero). Quando il sensore di umidità è completamente immerso in acqua, il valore numerico sarà circa 700. Una calibrazione accurata dipende tuttavia dal sensore e dal tipo di liquido che viene misurato: per esempio l'acqua leggermente salata ha una migliore conducibilità e il valore sarebbe corrispondentemente più elevato. Un esempio di range di valori misurati potrebbe essere il seguente:

- Sensore in aria: $valore = 0$
- Sensore in terreno asciutto: $0 < valore < 300$
- Sensore nel terreno umido: $300 < valore < 700$
- Sensore in acqua: $valore \simeq 700$

5 Tecnologie utilizzate

Firebase

Firebase è un potente servizio on line che permette di salvare e sincronizzare i dati elaborati da applicazioni web e mobile. Si tratta di un database NoSQL dalle grandissime risorse, ad alta disponibilità ed integrabile in tempi rapidissimi in altri progetti software, semplicemente sottoscrivendo un account al servizio. Questo articolo ne presenta le caratteristiche e mostra i passi necessari da compiere per iniziare ad utilizzarlo subito. Nel mese di ottobre 2014 un comunicato apparso sul sito ufficiale del progetto a firma di James Tamplin, CEO e cofondatore, annunciava che Firebase entrava a fare parte dei servizi Google. L'acquisizione, di cui non sono stati rivelati i dettagli economici, permetterà a Firebase di crescere ben oltre le proprie possibilità e alla società di Mountain View di inserirlo nella Google Cloud Platform.
Tamplin stesso dichiarava nel post che in soli tre anni la sua "pazza idea che avrebbe potuto funzionare" si era trasformata in una realtà su cui 110.000 sviluppatori basavano il proprio lavoro. Il successo di Firebase è legato indiscutibilmente alle sue caratteristiche peculiari:

- capacità di **sincronizzazione dei dati** oltre che di **storage**: Firebase è in grado di aggiornare i dati istantaneamente, sia se integrato in app web che mobile. Non appena l'app recupera la connettività sicronizza i dati mostrati con le ultime modifiche apportate;
- disponibilità di **librerie** client per integrare Firebase in ogni app. Android, Javascript e framework con esso realizzati (Node.js, Angular.js, Ember.js, Backbone.js e molti altri), Java e sistemi Apple: per tutte le più comuni tecnologie web e mobile esistono librerie già pronte per essere importate nei propri progetti;
- **API REST**: rendono disponibili le funzionalità di Firebase per ogni tecnologia per cui non esistano librerie apposite o in caso di operazioni non contemplate in esse;
- **Sicurezza**: i dati immagazzinati in Firebase sono replicati e sottoposti a backup continuamente. La comunicazione con i client avviene sempre in modalità crittografata tramite SSL con certificati a 2048-bit;
- **costi differenziati** in base all'uso e alle capacità richiesti. Si parte dal "Hacker Plan", gratuito, ed è sufficiente sottoscriverlo per avere a disposizione un massimo di 50 connessioni, $5GB$ di trasferimento dati e $100MB$ di storage[4]. L'uso di questo piano gratuito è esclusivamente didattico e di sperimentazione, ma utilissimo per imparare ad integrare Firebase nei propri progetti.
  Per contesti di produzione, esistono altri piani (Candle, Bonfire, Blaze, Inferno Plan) con prezzi mensili variabili e caratterizzati da capacità di memorizzazione crescenti.

[4] Saranno sufficienti in quanto si prevede l'invio di dati in modo formattato e ogni volta saranno sovrascritti a quelli precedenti.

JWT - JSON Web Token

Il JSON Web Token (JWT) è uno standard open (RFC 7519) che definisce uno schema in formato JSON per lo scambio di informazioni tra vari servizi. Il token generato può essere firmato (con una chiave segreta che solo chi genera il token conosce) tramite l'algoritmo di HMAC, oppure utilizzando una coppia di chiavi (pubblica / privata) utilizzando gli standard RSA o ECDSA.
I JWT sono molto utilizzati per autenticare le richieste nei Web Services e nei meccanismi di autenticazione OAuth 2.0 dove il client invia una richiesta di autenticazione al server, il server genera un token firmato e lo restituisce al client che, da quel momento in poi, lo utilizzerà per autenticare le successive richieste.

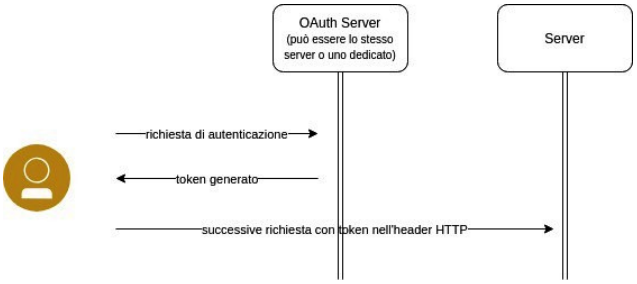


Figura 15: Meccanismo di autenticazione OAuth 2.0

La struttura del token Il token è composto da 3 parti fondamentali: Header, Payload e Signature.

Header L'header contiene due informazioni principali: la tipologia del token (in questo caso valorizzata a JWT perchè si tratta di un JSON Web Token) e il tipo di algoritmo di criptazione utilizzato.

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Payload Il payload contiene le informazioni di interscambio, possiamo categorizzarle in tre blocchi:

1. parametri registrati: sono delle proprietà predefinite che indicano delle informazioni sul token:
   - iss (issuer): è una stringa che contiene il nome identificativo dell'entità che ha generato il token
   - sub (subject): è una stringa che contiene l'oggetto del messaggio, può essere usato ad esempio singoli contesti e azioni di un applicativo
   - aud (audience): è un array di valori che indicano le abilità del token. In fase di validazione del token è possibile indicare un specifico audience, in questo caso, se il token non possiede il valore richiesto, verrà bocciato. Possiamo pensare ad esempio all'abilitazione o meno di un'utente su alcuni moduli di un applicativo
   - exp (expiration): è un numero intero (timestamp in secondi) che indica fino a quando il token sarà valido
   - nbf (not before): è un numero intero (timestamp in secondi), il token sarà valido solo dopo la data indicata in questo campo
   - iat (issued at): è un numero intero (timestamp in secondi) che indica il momento in cui il token è stato generato, serve per conoscere l'età di un token
   - jti (jwt id): identificativo univoco del token, utile per prevenire la generazione accidentale di token uguali
2. parametri pubblici: fanno riferimento a parametri definiti nel IANA JSON Web Token Registry, possono essere compilati a piacimento stando attenti al contenuto che si inserisce per evitare conflitti parametri privati: qui si ci può sbizzarrire inserendo quello che si vuole avendo piena flessibilità grazie alla struttura JSON

```
{
  "iss": "Gladiator_APP",
  "name": "Massimo Decimo Meridio",
  "iat": 1540890704,
  "exp": 1540918800,
  "user": {
     "profile": "editor"
  }
}
```

Signature La generazione del token avviene codificando in base 64 l'header e il payload e unendo i due risultati separandoli da un ".", successivamente si applica l'algoritmo indicato nell'heder alla stringa ottenuta utilizzando una chiave segreta.
Ad esempio, utilizzando l'algoritmo HMAC SHA256, il token si potrebbe ottenere come segue:

$$HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)$$

Utilizzando l'header e il payload degli esempi precedenti e applicando una secret key (per esempio "chiavesegreta") il token generato sarà il seguente:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJOT01FX0FQUCIsIm5hbWUi
OiJNYXJpbyBSb3NzaSIsImlhdCI6MTU0MDg5MDcwNCwiZXhwIjoxNTQwOTE4ODAwLCJ1c
2VyIjp7InByb2ZpbGUiOiJlZGl0b3IifX0.ZUlQ_W8U5oW4bRTEuUEwcSNcvsazitYoXr
rDjtP6wkY
```

È possibile verificare e scompattare online un jwt utilizzando il sito ufficiale. Fortunatamente non dobbiamo re-implementare la logica di cifratura, ci sono molte libreria per generare jwt a seconda del linguaggio di programmazione che si utilizza. È importante sapere che il contenuto del token non è cifrato si può scompattare e leggerne il contenuto perciò non bisogna inserire dati sensibili come password. La sicurezza viene garantita dal fatto che il token viene firmato con una chiave segreta lato server, perciò se viene corrotto o modificato da un agente esterno, non passerà la validazione.

DynamicDNS

Il DNS Dinamico è una tecnologia che permette ad un nome DNS in Internet di essere sempre associato all'indirizzo IP di uno stesso host, anche se l'indirizzo cambia nel tempo.
I nomi DNS sono normalmente associati stabilmente ad indirizzi IP, i quali a loro volta sono stabilmente assegnati ad host che hanno funzioni di server. Molti host, in particolare quelli che si collegano ad internet utilizzando i servizi di uno (o più) ISP, ricevono invece un indirizzo diverso ad ogni connessione. Pertanto è impossibile raggiungerli da internet, perché non si conosce il loro indirizzo IP. Ciò preclude la possibilità di amministrarli remotamente e di offrire servizi su questi host.
Il DNS dinamico permette a questi host di essere sempre raggiungibili attraverso il loro nome DNS, e quindi rende possibile amministrarli remotamente ed erogare servizi raggiungibili da chiunque su internet.

Implementazione Un servizio di DNS dinamico è costituito da una popolazione di client dinamici (host con indirizzo IP dinamico che vogliono che il loro IP attuale sia registrato nel DNS), da uno o più server DNS dinamici e da un protocollo di comunicazione tra le due parti. Quando un client dinamico ottiene un indirizzo IP, contatta uno dei server e lo informa del suo IP attuale. Il server inserisce allora un record DNS che punta al nuovo indirizzo del cliente. In questo modo, altri host sono in grado di ottenere l'indirizzo IP attuale del client dinamico utilizzando il normale servizio DNS, e quindi senza essere consci che l'host che contattano ha un indirizzo IP registrato dinamicamente. La comunicazione tra client dinamico e server, che permette ad un client di aggiornare il proprio record DNS, è realizzata mediante diversi protocolli proprietari, che normalmente richiedono l'autenticazione del client dinamico per poter effettuare l'aggiornamento. Esiste anche una estensione standardizzata del protocollo DNS, che rende possibile effettuare query di UPDATE per aggiornare una zona DNS.
Numerosi fornitori di servizi su internet offrono servizi di DNS dinamico gratuiti e a pagamento.
Il gestore del sito con IP dinamico ha la responsabilità di mantenere aggiornato il proprio record DNS. Questo può essere effettuato tramite una pagina web apposita del sito del fornitore di servizio, ma questa soluzione richiede un intervento manuale ogni volta che l'IP cambia. Nella pratica si utilizzano dei programmi che aggiornano autonomamente il database del DNS dinamico, quando l'IP cambia oppure periodicamente. Questi programmi sono adeguati soprattutto quando girano direttamente sull'host a cui è assegnato l'indirizzo pubblico, in quanto possono facilmente conoscere l'indirizzo assegnato dal provider, e spesso possono anche essere informati dal sistema operativo dei cambi di indirizzo IP.
Quando invece si utilizza un router per la connessione ad internet, l'indirizzo pubblico è normalmente configurato su questo router, e i PC collegati a questo router accedono ad internet mediante qualche forma di network address translation, pertanto non possono facilmente conoscere l'indirizzo IP pubblico con cui si presentano su internet (e sul quale possono essere raggiungibili se sono configurate delle redirezioni sul router). Alcuni router utilizzati per la connessione ad internet includono un client di DNS dinamico, e possono quindi provvedere autonomamente all'aggiornamento dell'indirizzo IP.
Se invece il client di DNS dinamico deve essere installato su un calcolatore che utilizza un indirizzo IP privato, è necessario scoprire l'indirizzo IP assegnato al router NAT per poterlo registrare nel DNS dinamico. Alcune tecniche prevedono di effettuare connessioni verso server esterni, che riportano l'IP da cui vedono provenire la connessione (una tecnologia di questo tipo è STUN). In alternativa, alcuni programmi di DNS dinamico sono in grado di collegarsi autonomamente ad un router per scoprire l'indirizzo di rete pubblico assegnato dal provider.
Nella realizzazione del DNS dinamico è necessario inoltre impostare il massimo tempo di caching del dominio per un periodo breve (tipicamente pochi minuti). Questo serve a prevenire che gli altri server DNS mantengano a lungo nella loro cache il vecchio indirizzo, rendendo meno probabile che una query DNS per un indirizzo gestito dal DNS dinamico ottenga come risposta un indirizzo obsoleto.

Wildfly

WildFly, precedentemente noto come JBoss AS o semplicemente JBoss, è un application server open source che implementa le specifiche Java EE. WildFly è un sistema multipiattaforma, interamente realizzato in Java. Originariamente creato dalla società "JBoss Inc.", nel 2006, il sistema è stato acquistato da Red Hat per 420 milioni di dollari e viene gestito come progetto open source, sostenuto da un'enorme rete di sviluppatori.
A WildFly sono associati[non chiaro] una quantità di altri prodotti, incluso Hibernate, Undertow, JBoss ESB, jBPM, JBoss Rules (ex Drools), Infinispan, JGroups, Gatein, SEAM, JBoss Transaction, e ActiveMQ.

Estensioni Un'estensione è un modulo che estende le funzionalità di base del server. Il core WildFly è molto semplice e leggero; la maggior parte delle funzionalità che le persone associano a un server delle applicazioni sono fornite tramite estensioni. Un'estensione è impacchettata come modulo nella cartella dei moduli. L'utente indica che desidera che sia disponibile una particolare estensione includendo un elemento $< extension/ >$ che denomina il suo modulo nel file domain.xml o standalone.xml.

```
<extensions>
    [...]
    <extension module="org.jboss.as.transactions"/>
    <extension module="org.jboss.as.webservices" />
    <extension module="org.jboss.as.weld" />
    [...]
    <extension module="org.wildfly.extension.undertow"/>
</extensions>
```

Hibernate

Hibernate (talvolta abbreviato in H8) è una piattaforma middleware open source per lo sviluppo di applicazioni Java, attraverso l'appoggio al relativo framework, che fornisce un servizio di Object Relational Mapping (ORM) ovvero gestisce la persistenza dei dati sul database attraverso la rappresentazione e il mantenimento su database relazionale di un sistema di oggetti Java.
Come tale dunque, nell'ambito dello sviluppo di applicazioni web, tale strato software si frappone tra il livello logico di business o di elaborazione e quello di persistenza dei dati sul database (Data Access Layer).
È stato originariamente sviluppato da un team internazionale di programmatori volontari coordinati da Gavin King; in seguito il progetto è stato proseguito sotto l'egida di JBoss, che ne ha curato la standardizzazione rispetto alle specifiche Java EE.

Caratteristiche Hibernate è distribuito in licenza LGPL sotto forma di librerie software da linkare nel progetto di sviluppo software. Lo scopo principale di Hibernate è quello di fornire un mapping delle classi Java in tabelle di un database relazionale; sulla base di questo mapping Hibernate gestisce il salvataggio degli oggetti di tali classi su database (tipicamente attributi di oggetti per ciascun campo dati della tabella). Si occupa inoltre al rovescio del reperimento degli oggetti dal database, producendo ed eseguendo automaticamente le query SQL necessarie al recupero delle informazioni e la successiva reistanziazione dell'oggetto precedentemente "ibernato" (mappato su database).
L'obiettivo di Hibernate è quello di esonerare lo sviluppatore dall'intero lavoro relativo alla persistenza dei dati. Hibernate si adatta al processo di sviluppo del programmatore, sia se si parte da zero sia se da un database già esistente. Hibernate genera le chiamate SQL e solleva lo sviluppatore dal lavoro di recupero manuale dei dati e dalla loro conversione, mantenendo l'applicazione portabile in tutti i database SQL. Hibernate fornisce una persistenza trasparente per Plain Old Java Object (POJO); l'unica grossa richiesta per la persistenza di una classe è la presenza di un costruttore senza argomenti. In alcuni casi si richiede un'attenzione speciale per i metodi equals() e hashcode(). Hibernate è tipicamente usato sia in applicazioni Swing che Java EE facenti uso di servlet o EJB di tipo session beans.
La versione 3 di Hibernate arricchisce la piattaforma con nuove caratteristiche come una nuova architettura Interceptor/Callback, filtri definiti dall'utente, e annotazione stile JDK 5.0 (Java's metadata feature). Hibernate 3 è vicino anche alle specifiche di EJB 3.0 (nonostante sia stato terminato prima di EJB 3.0 le specifiche erano già state pubblicate dalla Java Community Process) ed è usato come spina dorsale per l'implementazione EJB 3.0 di JBoss. Nel dicembre 2011 è uscita la versione 4.0, e a gennaio 2012 la versione 4.01. Nel mese di agosto 2013 è stata resa disponibile la versione 4.2.4.

6 Implementazione

In questa sezione andreamo ad analizzare, come sono state effettivamente implementate le classi che costituiscono i vari moduli dell'UML Deployment diagram (si veda Figura 3), a cui faremo costante riferimento, per mostrare come sia effettivamente realizzato ogni suo modulo. Come prima divisione, il progetto è stato suddiviso in 3 "*sotto-progetti*", uno per ogni *macro modulo*, ad eccezione dell'applicazione front end, che non è stata fisicamente implementata[5], ma soltanto simulata. Tale suddivisione si è resa necessaria, in un primo luogo per consentire una gestione e manutenzione più semplice, in quanto un unico progetto avrebbe avuto una complessità maggiore, rispetto a 3 sotto progetti più semplici. In secondo luogo, perché tutti e tre i moduli sarebbero andati in esecuzione in tre luoghi (server, portatili, microcontrollori, Raspberry, etc.): questo verrà analizzato nei sotto paragrafi successivi.

Field System

```
void makeJson() {
  StaticJsonDocument<SIZE_BUFF> doc;
  JsonArray sensors = doc.createNestedArray("sensors");
  JsonObject sensors_0 = sensors.createNestedObject();
  doc["type"] = "WemosEnv";
  doc["frequency"] = frequency;
  doc["name"] = myName;
  doc["lastread"] = stamp;

  sensors_0["type"] = "DHT11";
  sensors_0["name"] = "temperatura";
  sensors_0["threshold"] = threshold;
  sensors_0["valueReaded"] = temp;
  serializeJson(doc,buffJson,SIZE_BUFF);
}
```

(a) make JSON

```
void loop() {

  HTTPClient http;

  http.begin("http://aledns:8080/SmartGateway/api/data");
  http.addHeader("Content-Type","application/json");

  timeClient.update();

  client = server.available();
  rest.handle(client);

  delay(frequency);
  temp = dht.readTemperature();
  stamp = timeClient.getFormattedTime();


  makeJson();
  int httpResponseCode = http.PUT(buffJson);
```

(b) Loop function

Figura 16: Implementazione del Field System

È il componente software che viene direttamente caricato sui vari SmartController (board WeMos), per consentire il monitoraggio dei parametri della pianta quali umidità del terreno, temperatura dell'ambiente, etc. in base ai sensori che sono stati installati sulla board. Le due funzionalità principali di questo modulo sono:

- invio dei dati provenienti dai sensori allo SmartGateway
- ricezione di nuove impostazioni (e.g. frequenza di aggiornamnto dati da 50 a 10 min) o azioni da svolgere (e.g. irrigare pianta) provenvienti dallo SmartGateway

[5]Le ragioni della non implementazione sono state già fornite nell'Introduzione, nella sezione dedicata allo Scopo del Progetto

Ingestion System

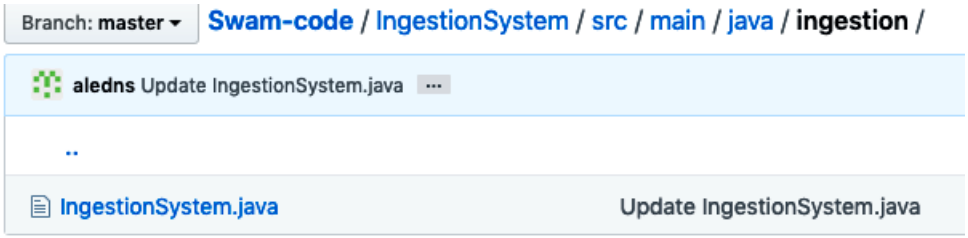


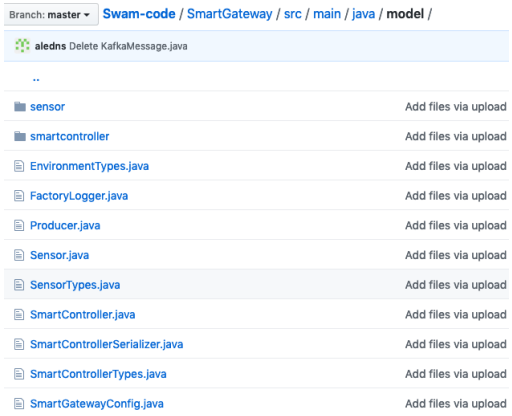
Figura 17: Implementazione dell'Ingestion System

Contiene una sola classe, la quale rappresenta il Consumer di Kafka. I topic sono etichettati in base allo Smartgateway che lo invia. Una volta ricevuto il messaggio, questo viene salvato su Firestore tramite le API messe a disposizione da Google.

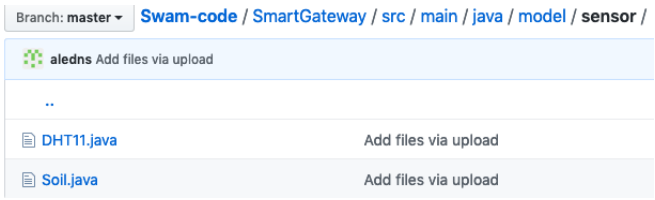
SmartGateway

È il componente software che sarà in esecuzione su un Raspberry, installato in ogni abitazione, che svolgerà le funzioni di gateway tra l'applicazione front-end e il Main Back-end (server centrale). Questo modulo è costituito dai seguenti *sotto-moduli*

Model



(a) Struttura generale model dello SmartGateway



(b) Struttura sensor del model dello SmartGateway

(c) Struttura smartcontroller del model dello SmartGateway

Figura 18: Implementazione dello SmartGateway - Model

Rappresenta il dominio dell'applicazione. In questo caso, a differenza del Main Backend, le classi non fanno uno di annotazioni JPA in quanto non è necessaria la persistenza. Il dominio in questa applicazione rappresenta degli SmartController generici, a differenza di quanto fatto nel MainBackend in cui sono presenti due classi astratte separate. Oltre a questo, è stato necessario rappresentare gli SmartController in modo che, a livello generale, non ci sia differenza se lo SmartGateway riceve un messaggio da quello che è effettivamente uno SmartController o uno SmartEnvironment, infatti sono entrambi dotati di sensori e si vuol tener traccia dei dati letti dai sensori sul field system e del timestamp di tale operazione. È inoltre presente la classe che rappresenta il Producer di Kafka.

Controller

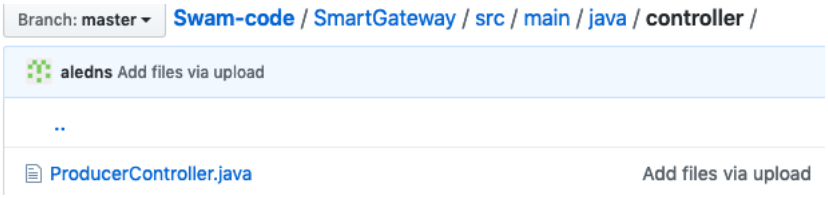


Figura 19: Implementazione dello SmartGateway - Controller

Contiene la sola classe ProducerController, usata dal livello superiore per permettere l'invio corretto dei dati tramite messaggi su topic di Kafka.

Rest

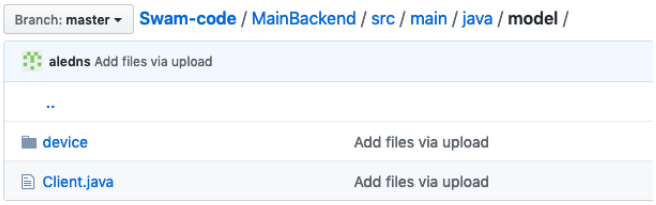
Figura 20: Implementazione dello SmartGateway - Rest

Contiene gli endpoint, che sono utilizzati, oltre che per la registrazione dello SmartGateway nei confronti del MainBackend, anche per ricevere i dati dal field-system. Sono presenti anche api rest per aggiornare i parametri degli smartcontroller in base all'API invocata; queste API saranno invocate dal Main Backend, invocate a loro volta dal MainFrontend.

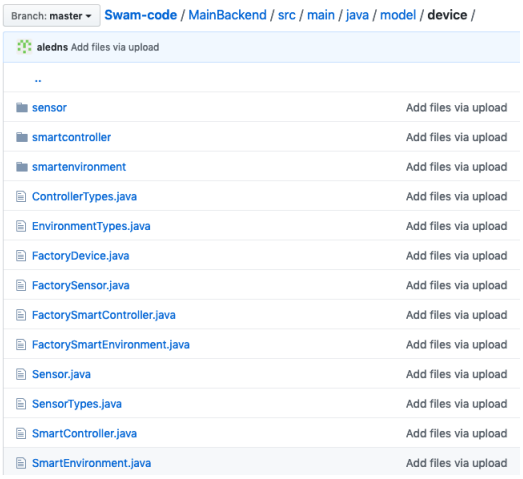
Main Backend

È il componente software che sarà in esecuzione sul server centrale, avrà il compito di gestire tutto il traffico dati dall'applicazione allo SmartGateway di competenza e recuperare i valori richiesti dal database cloud distribuito Firestore: l'utente avrà l'impressione di dialogare direttamente col gateway, in realtà tutte le richieste e i dati che vengono inviati, passano tutti da qui. Questo modulo è costituito dai seguenti *sotto-moduli*

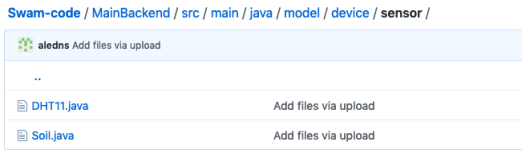
Model



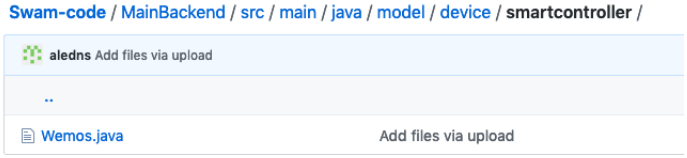
(a) Struttura generale model del Main Backend



(b) Struttura di model/device del Main Backend



(c) Struttura di model/device/sensor del Main Backend



(d) Struttura di model/device/smartcontroller del Main Backend

(e) Struttura di model/device/smartenvironment del Main Backend

Figura 21: Implementazione del Main Backend - Model

Tale package descrive il modello di dominio. Il modello viene rappresentato tramite una classe Client ed un insieme di classi astratte e relative implementazioni per poter rappresentare correttamente sia l'utente che gli smartcontroller e gli smartenvironment (sottopackage device). Le classi del dominio di applicazione sono annotate con annotazioni JPA, per poterle persistere su un database relazione tramite l'ORM Hibernate. Le classi sono composte da codice Java, annotazioni JPA e CDI.

Controller

Figura 22: Implementazione del controller del Main Backend

Rappresentano la business logic per il modello di dominio rappresentato. Tali classi sono composte da codice Java e annotazioni CDI.

Rest

Figura 23: Implementazione del modulo rest del Main Backend

Rappresenta la parte esterna dell'applicazione e fa uso delle classi appartenenti al package controller. Per questo livello si è deciso di utilizzare una architettura rest. Sono presenti 4 endpoint, uno per il client, uno per gli smartcontroller, uno per gli smartenvironment e data-endpoint (per il recupero dei dati letti, che sono salvati su Firestore). Le Web API sono utilizzate per comunicare sia con i frontend, sia per comunicare con i vari SmartGateway: ad esempio, al momento di attivazione di uno SmartGateway, lo stesso SmartGateway invocherá le API esposte dall'endpoint client del MainBackend).

DAO

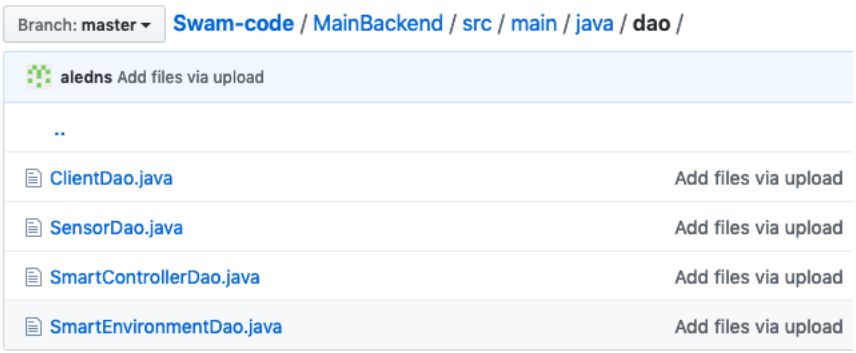


Figura 24: Implementazione del modulo DAO del Main Backend

Formato da i Data Access Object: per ogni classe del dominio si ha un oggetto che permette di mappare un POJO in forma relazione (quindi su db).

Utils

Figura 25: Implementazione del modulo utils del Main Backend

Contiene varie classi di utilitá. Di rilievo è la classe FirestoreDb, la quale si occupa di inizializzare l'SDK e permette il recupero dei dati.

Interazione e flusso dati

In questa sezione del capitolo dedicato all'implementazione dell'architettura precedentemente descritta, si intende analizzare l'utilizzo delle web API e del conseguente flusso dati che ne deriva. Per evitare di ripetere lo stesso concetto per ogni parte, si precisa che tale assunzione si denota essere vera per le 3 parti che verranno descritte, in particolare: si identifica uno schema di tipo request-response, tramite l'invio e la ricezione di messaggi in formato JSON. Una volta inviata la richiesta, tramite chiamata del metodo opportuno, non è necessario richiamare un ulteriore metodo di risposta, ma verrà fornito un messaggio di reply alla request seguita alla chiamata dell'API.

App → FieldSystem: changeSettings

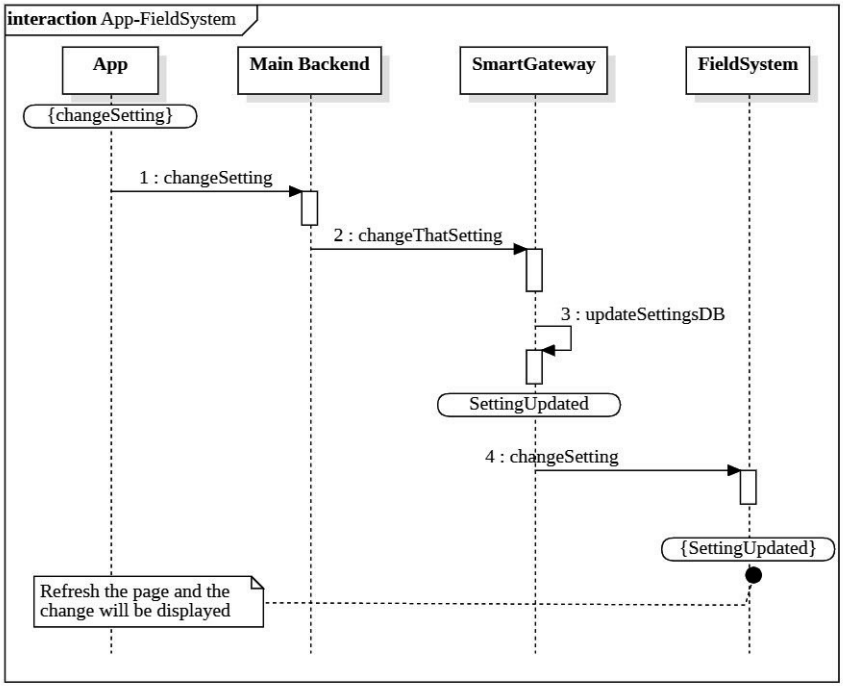


Figura 26: Flusso dati per invio di nuove impostazioni

Questa interazione è stata sviluppata per consentire l'invio delle impostazioni che l'utente ha modificato tramite l'app. Queste impostazioni, riguardanti il monitoraggio delle piante, dovranno essere trasmesse al FieldSystem, in modo che possa comunicarle ai sensori o attuatori interessanti: un esempio potrebbe essere la modifica della durata dell'irrigazione che deve essere inviata all'attuatore, oppure la variazione di una soglia di umidità, da inviare al sensore interessato.
Si identifica dunque il seguente percorso:

1. L'utente, tramite app, invia la richiesta di modifica dell'impostazione
2. Questo parametro viene elaborato dal Main Backend, che provvederà ad inviarlo allo SmartGateway
3. Una volta ricevuto il messaggio dal Main Backend, lo Smartgateway provvederà ad inviarlo allo Smart-Controller che gestisce il dispositivo interessato.
   Lo Smartgateway provvederà anche ad ggiornare il proprio database interno delle impostazioni e configurazioni

FieldSystem → Firestore: storeData

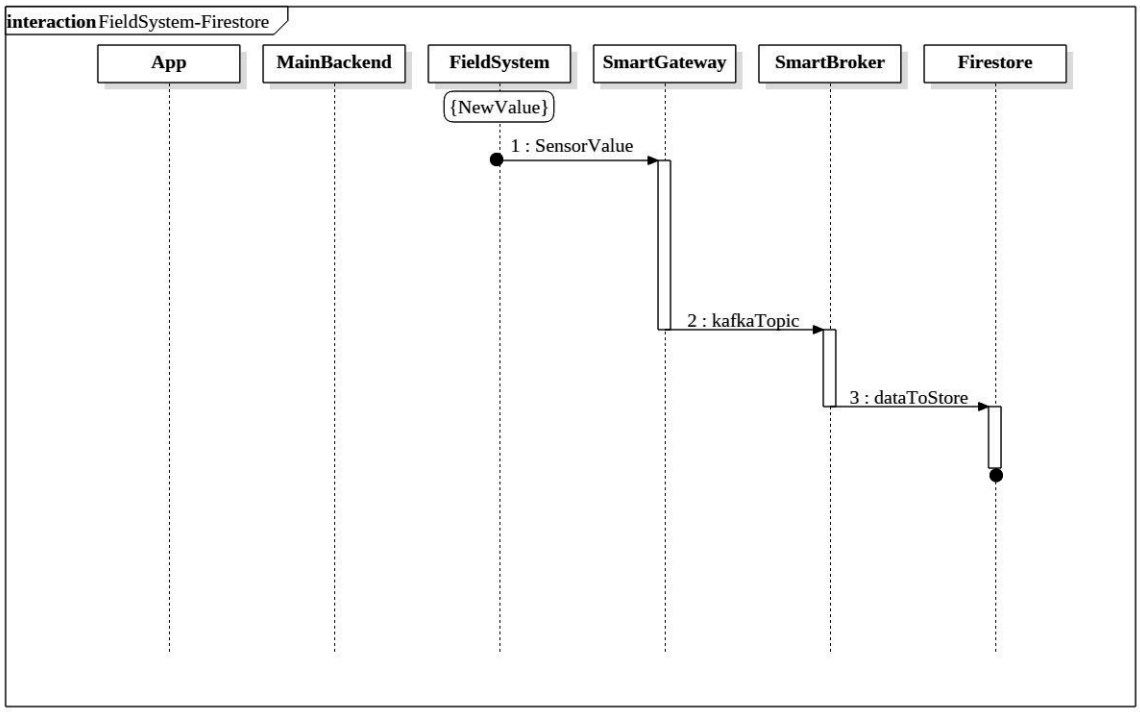


Figura 27: Flusso dati per lo storage del dato inviato dal sensore sul cloud Firestore

Questa interazione è stata sviluppata per consentire l'invio del valore letto dal sensore per poter essere salvato sul cloud Firestore. Si identifica dunque il seguente percorso:

1. Il sensore, attraverso lo SmartController che lo gestisce, invierà un messaggio, opportunamente strutturato, contenente il valore letto con ulteriori informazioni accessorie
2. Lo SmartGateway provvederà all'elaborazione di tale messaggio per la creazione del topic Kafka da inviare all'Ingestion System, seguendo lo schema procedurale di Kafka
3. Una volta ricevuto dall'IoTBroker dell'Ingestion System, questo messaggio dovrà essere formattato in modo tale da poter essere memorizzato sul cloud Firestore utilizzando le API necessarie e messe a disposizione dallo stesso cloud

App → Firestore: getData

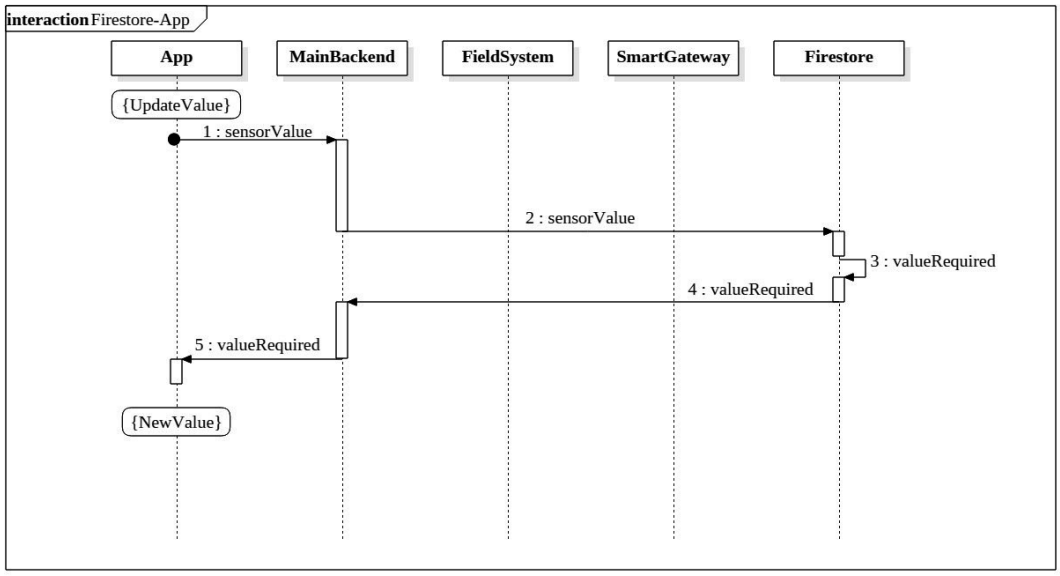


Figura 28: Flusso dati per la richiesta di un valore su Firestore

Questa interazione è necessaria per poter consentire all'app di avere a disposizione l'ultimo valore del sensore collegato alla pianta o ambiente che si intende monitorare. Si identifica dunque il seguente percorso:

1. L'app invia la richiesta del valore di un determinato sensore, con informazioni accessorie, utili per risalire al richiedente
2. Il Main Backend riceve la richiesta e utilizza le informazioni che ha a disposizione per richiamare opportunamente le API di Firestore per ottenere il dato richiesto
3. Firestore provvederà a rispondere a tale richiesta, fornendo il dato secondo il formato standard, che sarà interpretato e opportunamente formattato dal Main Backend
4. Una volta ricevo, il Main Backend provvederà ad inviare il dato richiesto all'applicazione, che lo mostrerà all'utente tramite interfaccia

7 Sviluppi futuri

Si prevedono sviluppi futuri di questa architettura, con le seguenti funzionalità:

- Si prevedono miglioramenti per quanto riguarda la scalabilità di hardware utilizzato, in quanto si prevede di poter migliorare il software caricato su ogni scheda, in modo tale che gli *SmartController* siano compatibili con una vasta gamma di sensori, per permettere il monitoraggio di ambienti diversi, come ad esempio un piccolo orticello, in cui la *situazione operativa* è molto diversa da quella di un ambiente domestico.
- Si prevede l'implementazione di una copia di backup delle configurazioni anche in database locali, su ogni *SmartGateway*: in questo modo, si avranno copie ridondate di impostazioni e configurazioni sia su ogni gateway dell'utente, che su *Main Backend*, così da poter eseguire più facilmente e velocemente il ripristino, in caso di sostituzione o malfunzionamento dello stesso gateway.
- Poiché ci potrebbero essere più utenti che *interagiranno con lo stesso gateway*, monitorando ambienti e piante diverse l'uno dall'altro, si prevede la creazione e l'utilizzo di più "*sotto-account*": ovvero, ogni utente avrà a disposizione il proprio account, registrato sempre con lo stesso nominativo[6], per poter configurare e monitorare l'ambiente o pianta che desidera.

[6]Ogni utente accederà col proprio account, ma tutti questi sotto-account sono registrati con lo stesso identificativo con cui è registrato lo *SmartGateway*: email e password sono le stesse, cambia soltanto l'etichetta del nome

References

[1] Lorenzo Biotti Alessio Danesi. *GitHub repository*. URL: https://github.com/aledns/Swam-code.

[2] Apache Software Foundation. *Apache kafka - A distributed streaming platform*. URL: https://kafka.apache.org/.

[3] M. Fowler. *UML Distilled - A Brief Guide to the Standard Object Modeling Language*. A cura di Addison-Wesley Professional. 2003.

[4] Fabio Golfarelli. «JWT — Incrementiamo la sicurezza con i JSON Web Tokens». In: *Medium.com* (ottobre 2018). URL: https://medium.com/@fabiogolfarelli/jwt-incrementiamo-la-sicurezza-con-i-json-web-tokens-cd0f2f9880da.

[5] Google. *Firebase Realtime Database*. URL: https://firebase.google.com/docs/database.

[6] Google. *Firebase Realtime Database - User Based Security*. URL: https://firebase.google.com/docs/database/security/user-security.

[7] Red Hat. *Wildfly Admin Guide*. URL: https://docs.wildfly.org/18/Admin_Guide.html#.

[8] M. Jones, J. Bradley, N. Sakimura. «JSON Web Token (JWT) - RFC7519». In: *Internet Engineering Task Force (IETF)* (). URL: https://tools.ietf.org/html/rfc7519.

[9] SEACOM. «Perché scegliere Apache Kafka? Scopriamo le potenzialità del Data Streaming». In: (6 dicembre 2008). URL: https://www.seacom.it/apache-kafka-data-streaming/.

[10] Wikipedia. *Apache Kafka*. URL: https://en.wikipedia.org/wiki/Apache_Kafka.

[11] Wikipedia. *Dynamic DNS*. URL: https://en.wikipedia.org/wiki/Dynamic_DNS.

[12] Wikipedia. *Hibernate (framework)*. URL: https://en.wikipedia.org/wiki/Hibernate_(framework).

[13] Wikipedia. *JSON Web Token*. URL: https://en.wikipedia.org/wiki/JSON_Web_Token.

[14] Wikipedia. *WildFly*. URL: https://en.wikipedia.org/wiki/WildFly.