
Università di Padova


sweasabi@gmail.com

Corso di Ingegneria del Software A.A.:2022/2023

# Specifica architetturale

Versione documento: *V1.0.1*

| | |
|---|---|
| **Uso** | Esterno |
| **Destinatario** | Committente |
| | Cliente |

# Registro delle modifiche

| Versione | Data | Modifica | Persone |
|---|---|---|---|
| 1.0.1 | 23 ago 2023 | • Corretta immagine 6.1 sistema di autenticazione | **Approvazione** Romano Davide<br>**Redazione** Bonavigo Michele<br>**Verifica** Peron Samuel |
| 1.0.0 | 16 ago 2023 | • Approvazione del documento | **Approvazione** Romano Davide |
| 0.1.0 | 16 ago 2023 | • Revisione del documento | **Verifica** Peron Samuel |
| 0.0.12 | 30 lug 2023 | • Unione del database del sistema di coordinazione con quelli di anagrafe e logging, capitolo A. | **Approvazione** Peron Samuel<br>**Redazione** Romano Davide<br>**Verifica** Massarenti Alessandro |
| 0.0.11 | 24 lug 2023 | • Prima stesura dell'architettura del sistema di logging, capitolo 7; | **Approvazione** Peron Samuel<br>**Redazione** Romano Davide<br>**Verifica** Massarenti Alessandro |
| 0.0.10 | 23 lug 2023 | • Aggiunte sezioni relative all'analisi dei requisiti all'appendice della specifica delle basi di dati, capitolo A. | **Approvazione** Peron Samuel<br>**Redazione** Romano Davide<br>**Verifica** Massarenti Alessandro |
| 0.0.9 | 22 lug 2023 | • Aggiunta sezione 5 relativa al sistema di coordinazione e illuminazione. | **Approvazione** Romano Davide<br>**Redazione** Massarenti Alessandro, Peron Samuel, Bonavigo Michele<br>**Verifica** Massarenti Alessandro |
| 0.0.8 | 21 lug 2023 | • Aggiunte sezioni progettazione logica all'appendice della specifica delle basi di dati, capitolo A. | **Approvazione** Peron Samuel<br>**Redazione** Romano Davide<br>**Verifica** Massarenti Alessandro |
| 0.0.7 | 20 lug 2023 | • Aggiunte sezioni relative alla progettazione concettuale all'appendice della specifica delle basi di dati, capitolo A. | **Approvazione** Peron Samuel<br>**Redazione** Romano Davide<br>**Verifica** Massarenti Alessandro |
| 0.0.6 | 18 lug 2023 | • Aggiunta appendice della specifica delle basi di dati, capitolo A, con draft della struttura. | **Approvazione** Peron Samuel<br>**Redazione** Romano Davide<br>**Verifica** Massarenti Alessandro |

| | | | |
|---|---|---|---|
| 0.0.5 | 15 giu 2023 | • Prima stesura struttura webapp | **Approvazione** Romano Davide<br>**Redazione** Peron Samuel, Zarantonello Giorgio<br>**Verifica** Massarenti Alessandro |
| 0.0.4 | 14 giu 2023 | • Prima stesura dell'architettura del sistema di anagrafe, capitolo 4; | **Approvazione** Romano Davide<br>**Redazione** Alessandro Massarenti, Mattia Casarotto, Samuel Peron<br>**Verifica** Michele Bonavigo |
| 0.0.3 | 07 giu 2023 | • Prima stesura dell'architettura generale del sistema, capitolo 2; | **Approvazione** Romano Davide<br>**Redazione** Alessandro Massarenti, Mattia Casarotto<br>**Verifica** Samuel Peron |
| 0.0.2 | 06 giu 2023 | • Prima stesura del sistema di autenticazione | **Approvazione** Luca Pierobon<br>**Redazione** Alessandro Massarenti, Michele Bonavigo<br>**Verifica** Samuel Peron |
| 0.0.1 | 15 mag 2023 | • Prima stesura del documento e definzione struttura | **Approvazione** Luca Pierobon<br>**Redazione** Alessandro Massarenti<br>**Verifica** Samuel Peron |

# Indice

# Capitolo 1

# Introduzione

## 1.1 Scopo del Documento

Nel seguente documento viene illustrato in modo dettagliato la struttura e la composizione dei vari microservizi che compongo il prodotto. Il documento tratterà, in ordine, i seguenti punti:

- WebApp;
- sistema di logging;
- sistema di coordinazione;
- sistema di anagrafica;
- sistema di autorizzazione;
- specifica delle basi di dati;

## 1.2 Scopo dell'architettura

L'obiettivo di SWEasabi e dell'azienda ImolaInformatica S.p.A. è lo sviluppo di un sistema per l'ottimizzazione dell'illuminazione, il prodotto presenta differenti servizi che comunicano tra loro, ogni servizio ha un compito ben preciso e si occupa di una parte del sistema per questo necessita di un'architettura ben definita e strutturata. In questo documento verrà presentata l'architettura dei vari sistemi, i design pattern utilizzati e le tecnologie adottate.

## 1.3 Glossario

Per evitare ambiguità relative alle terminologie utilizzate è stato creato un documento denominato Glossario.

Questo documento contiene tutti i termini specifici di settore utilizzati nei documenti, con le relative definizioni.

## 1.4 Maturità del documento

Il presente documento ha raggiunto un buon grado di maturità, in quanto sono state definite le tecnologie e i design pattern utilizzati per lo sviluppo del prodotto. Inoltre sono state definite le interazioni tra i vari servizi che compongono il prodotto.

## 1.5 Riferimenti

### 1.5.1 Riferimenti Normativi

- Norme di progetto;
- Capitolato d'appalto C2.

### 1.5.2 Riferimenti Informativi

- Analisi dei requisiti
- Software Architecture Patterns;
- JSON Web Token website;
- clean architecture with examples;
- Architettura esagonale

# Capitolo 2

# Architettura generale del sistema

## 2.1 Topologia generale

Per la definizione di questo sistema software è stato scelto di utilizzare un'architettura a microservizi, in quanto permette di avere un sistema scalabile e flessibile. Il pattern topologico scelto è quello REST.

Sono presenti all'interno del sistema due parti principali. Una parte è dedicata all'interazione con l'utente, l'altra parte è dedicata alla gestione dei dati e della logica di business, tra cui la comunicazione con i sistemi esterni quali lampioni, sensori e API meteo.

Ognuna delle responsabilità del sistema è stata suddivisa in un microservizio come segue:

- **WebApplication**: è il microservizio che si occupa di gestire l'interazione con l'utente;
- **Anagrafica**: è il microservizio che si occupa di gestire i dati anagrafici dei lampioni, dei gruppi e delle aree in cui questi sono collocati;
- **Coordinazione**: è il microservizio che si occupa di gestire l'accensione e lo spegnimento dei lampioni secondo regole specifiche e utilizzando le informazioni provenienti dai sensori e da API varie per ottimizzare l'illuminazione;
- **Autenticazione**: è il microservizio che si occupa di gestire l'autenticazione degli utenti e di fornire i token necessari per l'accesso agli altri servizi;
- **logging**: è il microservizio che si occupa di gestire i log dei lampioni e dei sensori. Da questi, inoltre, vengono estratte informazioni necessarie per la gestione dell'illuminazione e più in generale aggregazioni di dati utili per l'analisi e la presentazione all'utente.

Possiamo definire queste responsabilità come dei componenti. Ognuno di questi componenti ha un'interfaccia ben definita e può essere sostituito con un altro componente che implementa la stessa interfaccia. Questo ci permette di utilizzare al massimo le pratiche di CICD.

I componenti dell'applicazione, le interfacce utilizzate ed esposte e le funzionalità fornite sono mostrate in figura 2.1.

## 2.2 User interface

Come User Interface è stato scelto di utilizzare un'interfaccia web. Questa è stata scelta in quanto permette di avere un'interfaccia accessibile da qualsiasi dispositivo e non richiede l'installazione di software aggiuntivo. Inoltre, permette di avere un'interfaccia che può essere facilmente aggiornata e modificata.

Per l'implementazione è stato scelto di utilizzare il framework Angular.

La scelta di utilizzare una webapp come interfaccia utente risponde inoltre ai requisiti di vincolo: **RV_01**, **RV_02**, **RV_03**, **RV_04**, **RV_05**.

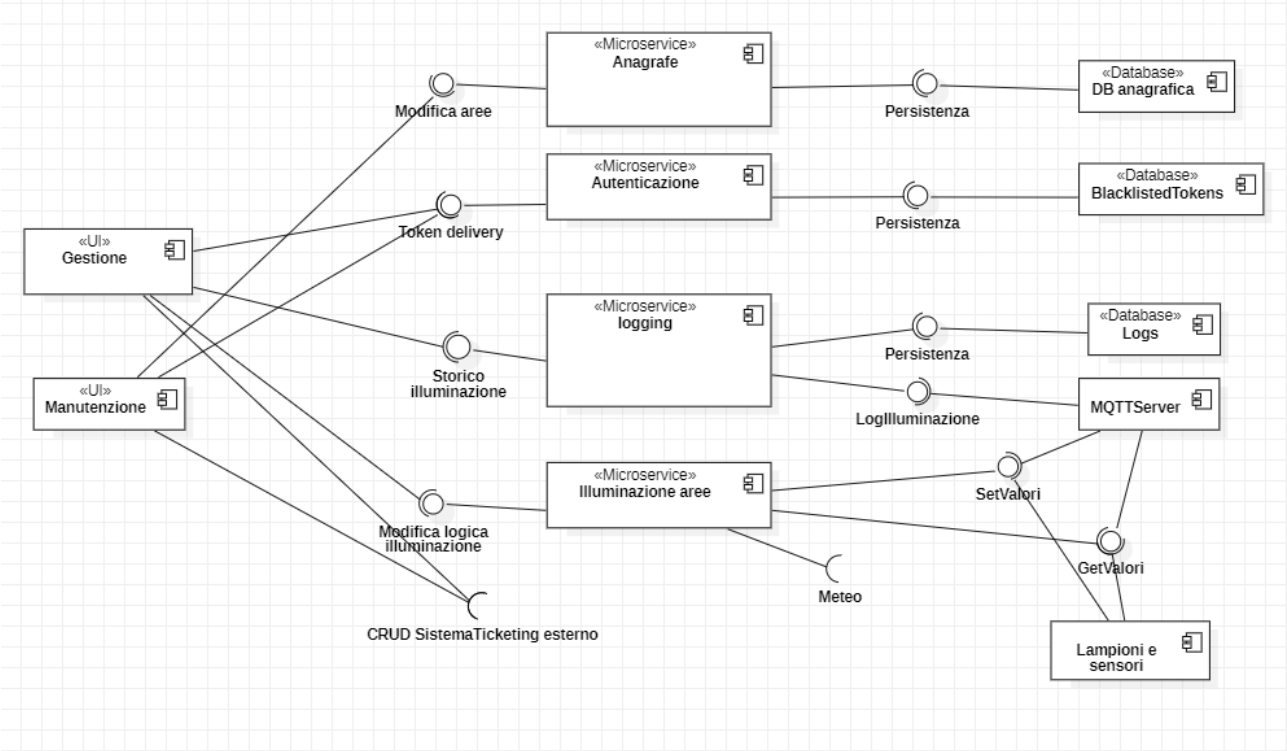



Figura 2.1: Diagramma dei componenti del sistema generale

In poche parole l'applicazione deve essere utilizzabile da mobile e dai maggiori browser nelle versioni più recenti. Questa è la definizione data dai requisiti sopra citati.

## 2.3 Business backend e microservizi

Il business backend è il componente che si occupa di gestire la logica di business dell'applicazione. Questo componente è composto dai microservizi citati precedentemente alla sezione 2.1.

La scelta di suddividere per responsabilità i microservizi serve a rispondere al requisito di vincolo **RV_06**, ovvero che il sistema deve essere scalabile orizzontalmente. Questi Microservizi sono infatti inoltre anche gestiti in modo tale da rimanere stateless rispetto all'utente ed utilizzando per ciascuno un database separato. In caso di aumenti di carico del sistema è quindi semplicemente necessario aggiungere nuove istanze dei microservizi.

Avendo poi basi di dati ridotte ed utilizzando però DBMS relazionali per via della loro facilità d'utilizzo non è necessario dover creare complesse strategie di sharding per la suddivisione del carico.

### 2.3.1 DNS

Per la struttura posta dietro alla REST based architecture, ci sarà bisogno di una definizione di nomi di dominio. Questa definizione è necessaria per poter utilizzare le risorse specifiche di ogni microservizio senza utilizzare un livello intermedio di API.

# Capitolo 3

# Webapp

## 3.1 Scopo del sistema

La webapp permette all'utente di avere un interfaccia grafica con la quale gestire e monitorare lampioni, sensori ed aree, inoltre permette di visualizzarne le informazioni relative.

## 3.2 Requisiti coperti dal sistema

### 3.2.1 RV_01

RV_01: l'applicazione deve essere visualizzabile su dispositivi mobile.

### 3.2.2 RV_02

RV_02: l'applicazione client deve poter essere utilizzata sulla versione più recente di Chrome (v. 110.0).

### 3.2.3 RV_03

RV_03: l'applicazione client deve poter essere utilizzata sulla versione più recente di Firefox (v. 110.0).

### 3.2.4 RV_04

RV_04: l'applicazione client deve poter essere utilizzata sulla versione più recente di Safari (v. 16.3).

### 3.2.5 RV_05

RV_05: l'applicazione client deve essere conforme almeno al livello AA delle WCAG.

## 3.3 Descrizione del sistema

La webapp è stata sviluppata utilizzando il framework Angular, il quale permette di creare applicazioni web single-page e di gestire le dipendenze tra le varie componenti dell'applicazione. Inoltre, è stato utilizzato il framework Bootstrap per la gestione della parte grafica. L'applicazione utilizza componenti di Angular, servizi e chiamate API per comunicare con il server. Le principali features dell'applicazione sono:

- Visualizzazione lista di lampioni, sensori ed aree;
- Cambiare lo status dei lampioni e dei sensori;

- Aggiungere lampioni, sensori ed aree;
- Gestire caricamenti ed errori durante le chiamate API;
- Implementare l'autenticazione;
- Proteggere i collegamenti all'applicazione;

## 3.4 Architettura del sistema

La classe Model implementa tre interfacce che definiscono lo status di lampioni,sensori ed aree, queste definiscono la struttura e forniscono le proprietà per immagazzinare informazioni quali status, identificativo e alias. Le tre interfacce sono:

- **LampStatus:** definisce lo stato di un lampione;
- **SensorStatus:** definisce lo stato di un sensore;
- **AreaStatus:** definisce lo stato di un'area;

### 3.4.1 Servizi

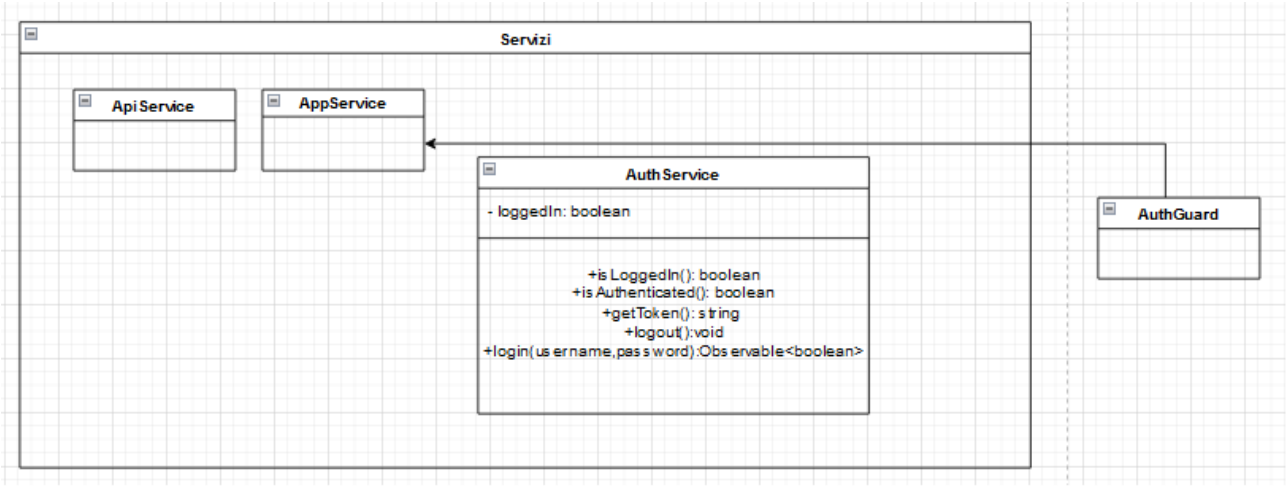



Figura 3.1: Servizi della webapp 3.1

La classe **Apiservice** definisce i metodi per effettuare le chiamate API e interagire con il server, questo servizio mette a disposizione i metodi per:

- Ottenere la lista di lampioni, sensori ed aree;
- Ottenere lo stato di un lampione, sensore o area;
- Modificare lo stato di un lampione, sensore o area;
- Aggiungere un lampione, sensore o area;
- Ottenere i dati di un sensore;
- Ottenere i dati di un'area;

L'applicazione presenta delle classi per l'utilizzo di altri microservizi:

- **AuthService:** controlla l'autenticazione dell'utente;

- **AppService:** gestisce lo stato dell'applicazione mantenendo lo status di lampioni, sensori ed aree, questo servizio fornisce i metodi per l'aggiunta di nuovi dispositivi e comunica con l'API per ottenere dati, aggiornare gli status e gestire caricamenti ed errori;

La classe **AuthGuard** permette di proteggere i collegamenti all'applicazione, in questo modo l'utente non autenticato non può accedere alle pagine dell'applicazione.

### 3.4.2 Componenti

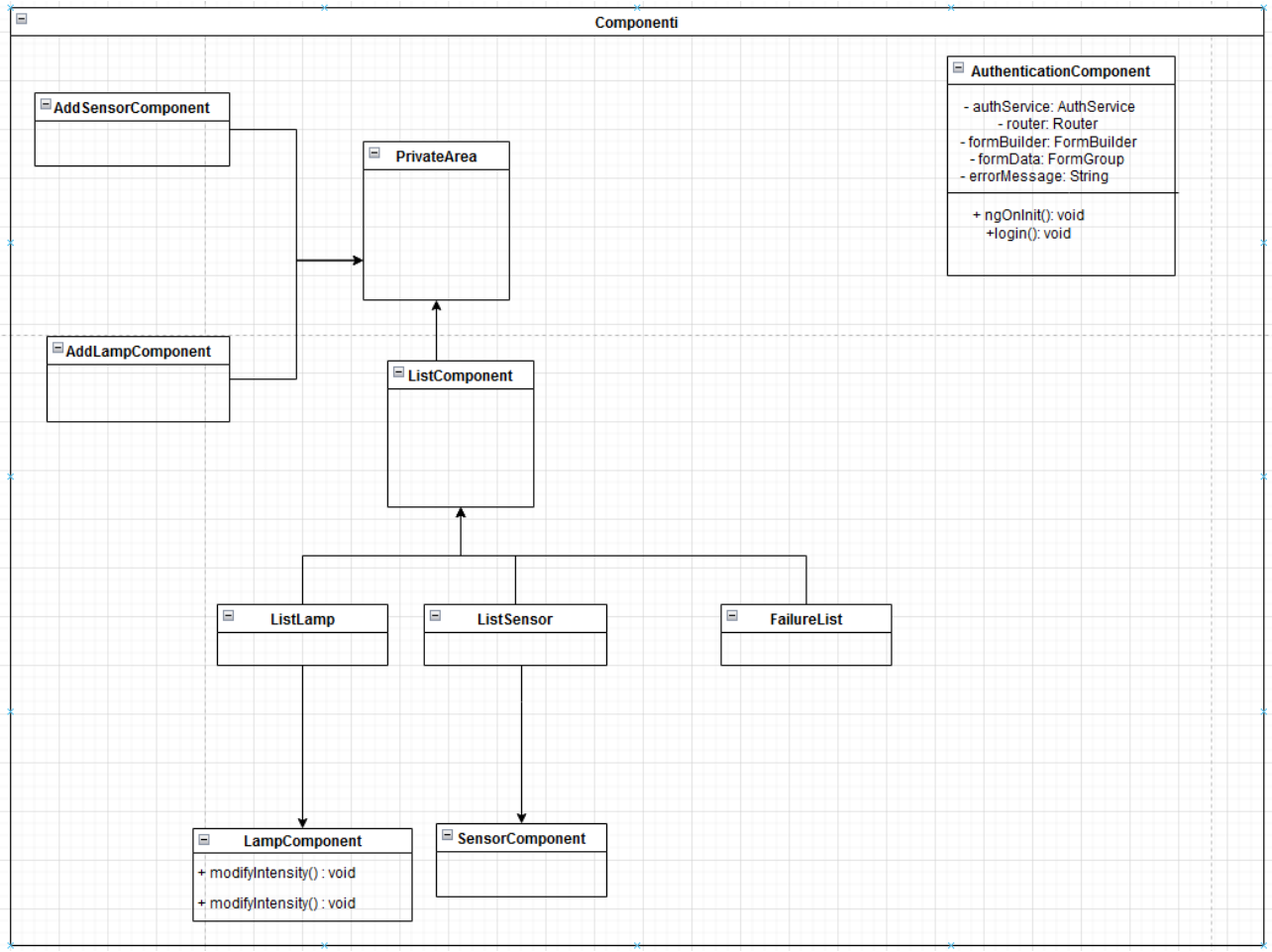



Figura 3.2: Componenti della webapp 3.2

L'applicazione utilizza dei componenti per definire e gestire le varie parti dell'applicazione, questi sono:

- **lamp.component:** rappresenta un lampione con il proprio stato e permette di modificarlo;
- **sensor.component:** rappresenta un sensore con il proprio stato e permette di modificarlo;
- **area.component:** rappresenta un'area di illuminazione;
- **lamps-list.component:** visualizza la lista di lampioni;
- **sensors-list.component:** visualizza la lista di sensori;
- **areas-list.component:** visualizza la lista di aree;
- **authentication.component:** gestisce l'autenticazione dell'utente;

- **private-area.component:** gestisce l'area privata;

# Capitolo 4

# Microservizio anagrafica

## 4.1 Scopo del sistema

Il sistema di anagrafica si occupa di gestire le informazioni anagrafiche dei sensori, delle aree e dei lampioni. Gestisce questi raggruppamenti e si occupa di renderli persistenti su database.

## 4.2 Requisiti coperti dal sistema

### 4.2.1 RF_03

RF_03: deve essere possibile aggiungere nuovi sensori a sistema.

Questo microservizio, copre il requisito persistendo i dati anagrafici del sensore.

### 4.2.2 RF_08

RF_08: l'utente deve poter inserire la locazione geografica del sensore nel sistema.

Questo microservizio, copre il requisito persistendo i dati di locazione del sensore.

### 4.2.3 RF_09

RF_09: l'utente deve poter inserire il raggio di azione del sensore.

Questo microservizio, copre il requisito persistendo i dati di raggio di azione del sensore.

### 4.2.4 RF_10

RF_10: l'utente deve poter essere in grado di visualizzare quali aree sono illuminate in un dato momento

Questo microservizio, copre parte del requisito fornendo l'informazione di quali lampioni si trovano in una determinata area.

## 4.3 Descrizione del sistema

Il sistema principalmente fornisce le informazioni conservate su base di dati tramite un'interfaccia REST. Questo è poi in grado di fornire i dati in forma aggregata, come ad esempio il numero di lampioni per area, o il numero di sensori per lampioni.

Per queste informazioni ha inoltre il compito e la responsabilità di tenere aggiornati i dati nel database, organizzando i dati in modo da poterli fornire in modo efficiente.

## 4.4 Architettura del sistema

L'architettura base del sistema è sviluppata in ottica KISS.

Il sistema è composto da un'interfaccia REST che espone i dati e da un insieme di classi che si occupano di gestire i dati nel database.

Il servizio nasce molto semplice e per questo motivo non è stato ritenuto necessario l'utilizzo di un'architettura più complessa e strutturata.

Ognuna delle classi rispetta però SOLID e soprattutto il principio della singola responsabilità.

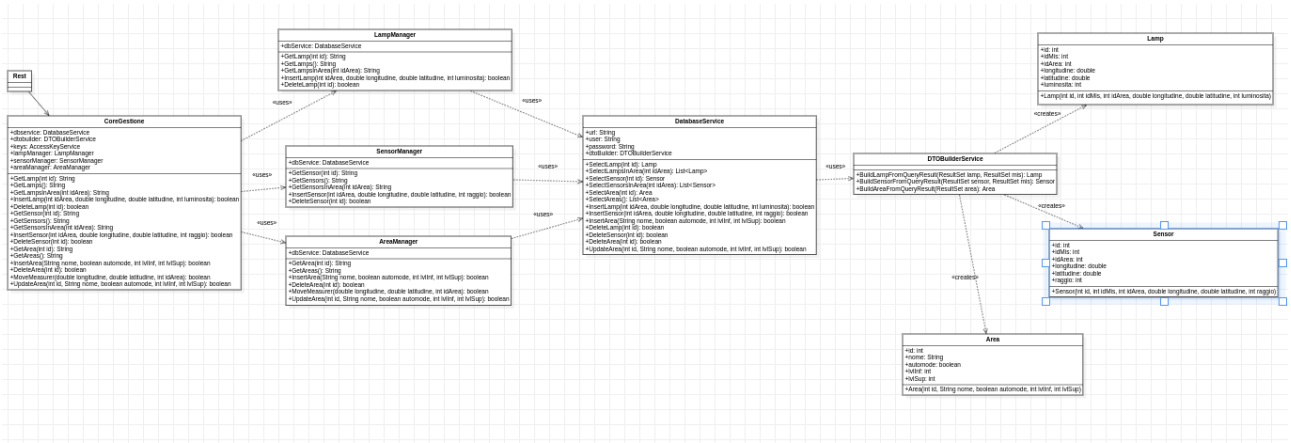



Figura 4.1: Vista generale del sistema di anagrafe, si consiglia visione nel dettaglio delle immagini 4.2, 4.3 e 4.4

## 4.5 Servizi

I servizi sono in generale le interfacce ad alto livello delle funzionalità del sistema. Sono composti da classi che si occupano di gestire le richieste e di fornire i dati richiesti. Questi possono essere visionati nell'immagine 4.2.

### 4.5.1 DatabaseService

DatabaseService si occupa di effettuare tutte le operazioni di connessione e interazione con il database.

Sono presenti operazioni di Select, Delete, Insert e Update. In futuro sarà possibile espanderle in base al tipo di operazione che si desidera eseguire.

### 4.5.2 DTOBuilderService

DTOBuilderService si occupa di creare i diversi DTO dell'applicazione, in questo caso Area, Lampione e Sensore, che verranno utilizzate all'interno dell'applicazione per muovere i dati tra le varie parti del sistema.

### 4.5.3 AccessKeysService

L'AccessKeysService si occupa di verificare che la chiave di accesso dell'utente non sia scaduta. Se la chiave è valida l'utente può utilizzare le varie funzionalità offerte dal microservizio.

## 4.6 Interfaccia REST

L'interfaccia REST viene esposta all'esterno e permette di effettuare le operazioni discusse nelle sotto-sezioni successive.

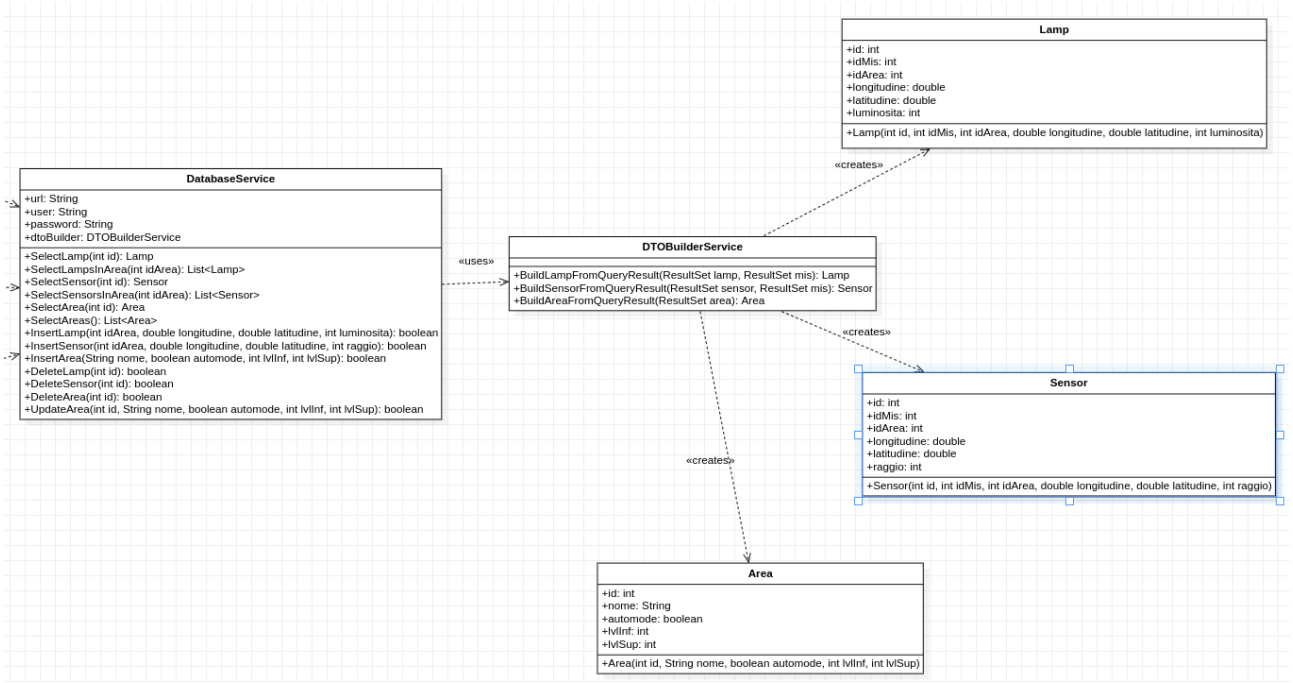



Figura 4.2: Diagramma delle classi del modello all'interno del sistema di anagrafe

Questa interfaccia comunica al resto del sistema tramite il componente *CORE* definito nella sezione 4.8.

- GET /getLamp/id
- GET /getLampsInArea/idArea
- PUT /addLamp
- PUT /deleteLamp/id
- GET /getSensor/id
- GET /getSensorsInArea/idArea
- PUT /addSensor
- PUT /deleteSensor/id
- PUT /moveMeasurer/idMeasurer
- GET /getArea/id
- GET /getAreaList
- PUT /addArea
- PUT /updateArea/id
- PUT /deleteArea/id

### 4.6.1 GET /getLamp/id

GetLamp permette di ottenere tutti i dati relativi a un lampione. L'utente deve fornire l'id del lampione in questione.

Il sistema risponderà con un JSON contenente tutti i dati relativi al lampione richiesto.

### 4.6.2 GET /getLampsInArea/idArea

GetLampsInArea permette di ottenere i dati relativi a tutti i lampioni in un'area. L'utente deve fornire l'id dell'area della quale si desidera visualizzare i lampioni.

Il sistema risponderà con un JSON contenente tutti i dati relativi ai lampioni presenti in quell'area.

### 4.6.3 PUT /addLamp

AddLamp permette di inserire nel database un nuovo lampione. All'utente è richiesto di fornire i dati relativi al lampione da inserire in un JSON che viene inviato al sistema.

Il sistema risponderà con un boolean ad indicare se l'operazione è andata a buon fine o meno.

### 4.6.4 PUT /deleteLamp/id

DeleteLamp permette di eliminare dal database un lampione. L'utente deve fornire al sistema l'id del lampione da eliminare.

Il sistema procede poi a tentare di eliminare il lampione, rimuovendo prima il riferimento al misuratore, e ritorna un boolean positivo in caso di successo, negativo in caso di fallimento.

### 4.6.5 GET /getSensor/id

GetSensor permette di ottenere tutti i dati relativi a un sensore. L'utente deve fornire l'id del sensore in questione.

Il sistema risponderà con un JSON contenente tutti i dati relativi al sensore richiesto.

### 4.6.6 GET /getSensorsInArea/idArea

GetSensorsInArea permette di ottenere i dati relativi a tutti i sensori in un'area. L'utente deve fornire l'id dell'area della quale si desidera visualizzare i sensori .

Il sistema risponderà con un JSON contenente tutti i dati relativi ai sensori presenti in quell'area.

### 4.6.7 PUT /addSensor

AddSensor permette di inserire nel database un nuovo sensore. All'utente è richiesto di fornire i dati relativi al sensore da inserire in un JSON che viene inviato al sistema.

Il sistema risponderà con un boolean ad indicare se l'operazione è andata a buon fine o meno.

### 4.6.8 PUT /deleteSensor/id

DeleteSensor permette di eliminare dal database un sensore. L'utente deve fornire al sistema l'id del sensore da eliminare.

Il sistema procede poi a tentare di eliminare il sensore, rimuovendo prima il riferimento al misuratore, e ritorna un boolean positivo in caso di successo, negativo in caso di fallimento.

### 4.6.9 PUT /moveMeasurer/id

MoveMeasurer permette di spostare un misuratore (che può essere un lampione o un sensore) da un'area a un'altra. L'utente deve fornire l'id del misuratore da spostare e un JSON contenente le nuove coordinate e l'area dove va inserito.

Il sistema ritorna infine un boolean positivo in caso di successo, negativo in caso di fallimento.

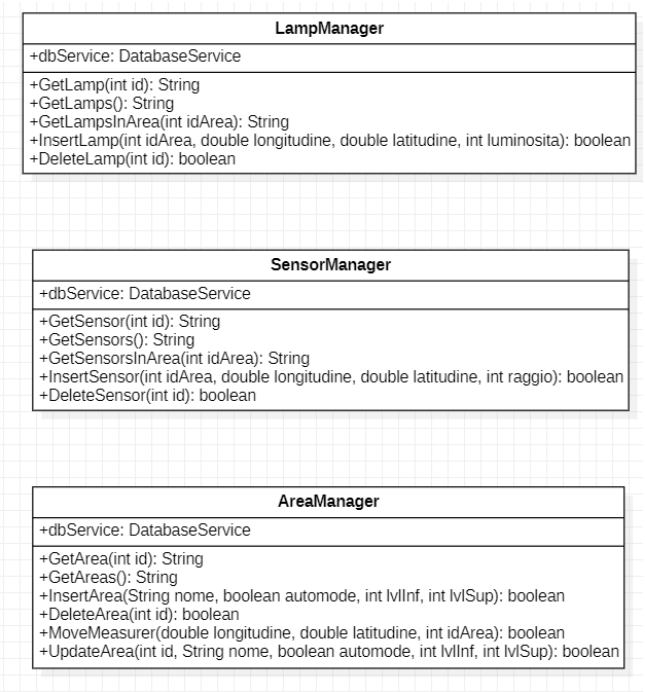



Figura 4.3: Diagramma delle classi relative ai servizi del sistema di anagrafe

## 4.7 Managers

Si occupano di fare da tramite tra il servizio che colloquia fisicamente con il DB e il controller che si occupa di gestire le richieste REST. Si possono vedere i metodi dei servizi nel diagramma delle classi 4.3.

- **LampManager**: si occupa della gestione (ottenimento, inserimento, modifica o cancellazione) dei dati relativi ai lampioni nel database.
- **SensorManager:** si occupa della gestione (ottenimento, inserimento, modifica o cancellazione) dei dati relativi ai sensori nel database.
- **AreaManager:** si occupa della gestione (ottenimento, inserimento, modifica o cancellazione) dei dati relativi alle aree nel database.

**LampManager** Ha la responsabilità di gestire l'inserimento e la rimozione di un lampione dal database, oltre che di ottenere i dati relativi a un lampione o a tutti i lampioni in un'area.

**SensorManager** Ha la responsabilità di gestire l'inserimento e la rimozione di un sensore dal database, oltre che di ottenere i dati relativi a un sensore o a tutti i sensori in un'area.

**AreaManager** Ha la responsabilità di gestire l'inserimento e la rimozione di un'area dal database, oltre che di ottenere i dati relativi a un'area o a tutte le aree.

## 4.8 Core

L'interfaccia accentra tutte le funzionalità base del sistema rendendo semplice l'implementazione di nuove interfacce esterne all'utente.

Tutti i suoi metodi sono statici e permettono di ottenere i dati dal database in modo semplice e veloce.

I metodi che questa componente offre sono visibili nel diagramma delle classi 4.4.

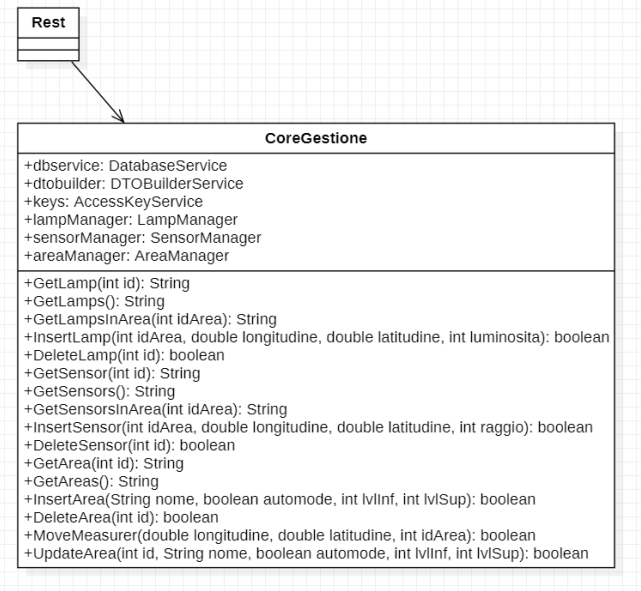



Figura 4.4: Diagramma delle classi relative al core del sistema di anagrafe

# Capitolo 5

# Sistema di coordinazione

## 5.1 Scopo del sistema

Il sistema di coordinazione è il componente che si occupa di gestire i lampioni, regolandone l'accensione e lo spegnimento in base alle informazioni che ha a disposizione.

Tramite algoritmi sofisticati, il sistema di coordinazione è in grado di gestire l'illuminazione in modo efficiente.

### 5.1.1 Requisiti coperti dal sistema

**RF_05**

RF_05: Il sistema deve accendere un'area per un lasso di tempo preconfigurato quando rileva persone in prossimità dello stesso.

**RF_06**

RF_06: Il sistema deve riportare l'intensità luminosa dell'area al valore di default una volta passato il tempo impostato.

**RF_19**

RF_19: Il sistema deve essere in grado di ricevere informazioni dal sensore in modalità push.

## 5.2 Descrizione del sistema

Il sistema utilizza molteplici paradigmi. L'architettura generale è di tipo esagonale.

Il programma eseguirà azioni quando attivato da eventi esterni, quali l'arrivo di un nuovo stato da MQTT oppure l'arrivo di una richiesta da parte dell'utente collegato alla webapp[1].

Le componenti principali saranno:

- **Porta ricevente MQTT**: si occupa di ricevere i messaggi da MQTT e di inoltrarli al sistema ad eventi;
- **Interfaccia REST**: si occupa di ricevere le richieste degli utenti e di inoltrarle al sistema ad eventi;
- **Pila di payload**: È una pila contenente i payload di dati da analizzare;
- **Algoritmo di analisi**: Dato un payload si occupa di analizzarlo e di generare un nuovo stato;

[1] Si veda sezione 3

La pila di Payload usa il paradigma del **Producer-Consumer**, in cui il produttore è la porta ricevente, mentre il consumatore è l'algoritmo di analisi.

È poi presente un database[2] al quale il sistema si connette, per mantenere gli stati anche in caso di reboot del sistema.

Inoltre, se il programma farà comunque del caching dei dati, sarà comunque nel database che verranno depositati i dati quando la cache diventa troppo grande.

## 5.3 Architettura del sistema

Il sistema è composto da un'interfaccia REST, un'interfaccia MQTT, un collegamento ad un db e un algoritmo di analisi.

### 5.3.1 Entità del sistema

- **AreaAnagrafica**: Contiene le informazioni relative ad un'area, quali la posizione, il nome, il tipo di illuminazione, ecc.
- **LampAnagrafica**: Contiene le informazioni relative ad un lampione, quali la posizione, il nome, il tipo di illuminazione, ecc.
- **Misuratore**:
- **SensoreAnagrafica**: Contiene le informazioni relative ad un sensore, quali la posizione, il nome, il tipo di sensore, ecc.

### 5.3.2 Classi legate al database

Il database viene gestito come repository, e ognuna delle entità viene gestita come tale.

Le classi relative ai repository sono:

- **AreaRepository**: Fornisce l'accesso al database per le aree;
- **LampRepository**: Fornisce l'accesso al database per i lampioni;
- **MisuratoreRepository**:
- **SensoreRepository**: Fornisce l'accesso al database per i sensori;

Le classi sopra descritte sono visibili in figura 5.1

Queste classi effettuano le operazioni sul db tramite JPA.

### 5.3.3 Classi di configurazione

La classe **Beanconfig** contiene tutti i bean di Spring relativi alla configurazione per mqtt e per il database.

### 5.3.4 Classi legate ad MQTT

Per la gestione di MQTT sono presenti le seguenti classi visibili in figura 5.2

- **MqttController**: Si occupa di gestire la ricezione dei messaggi da MQTT e di inoltrarli al sistema ad eventi;
- **Producer**: Questa classe rimane in ascolto delle rilevazioni MQTT, quando dal topic sensore arriva un cambiamento, questo viene segnalato al logging[3]. In ultimo genera un payload e lo aggiunge alla coda dei payload da processare;
- **Consumer**: Quando vede che la coda dei payload non è vuota raccoglie le operazioni da fare, analizza lo stato e modifica la luminosità dei lampioni via mqtt, inoltre aggiorna il database con il nuovo stato;

[2] Vedi sezione relativa al database e alla sua progettazione A.2
[3] Il logging poi loggerà l'informazione

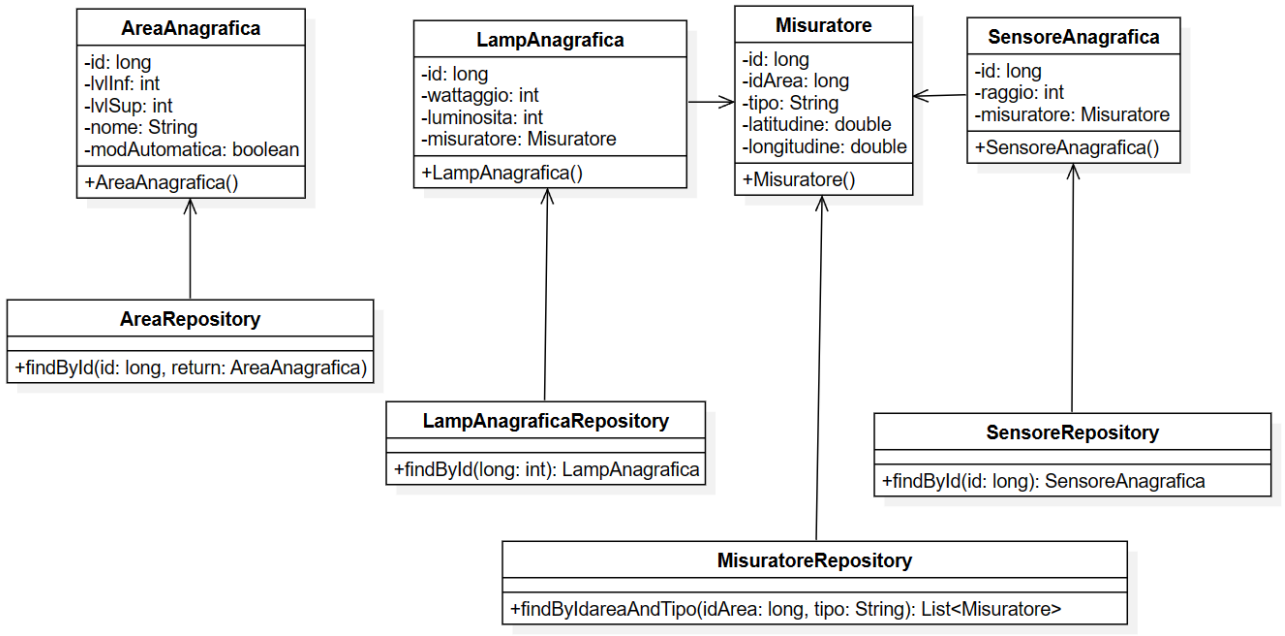



Figura 5.1: Diagramma delle classi relative ai repository e al sistema connesso al database

**Payloads**

I payloads sono un oggetto che contiene le informazioni necessarie per l'analisi e la modifica dello stato di illuminazione.

Queste classi vengono descritte nell'immagine 5.3.

**Payload** è un'interfaccia che contiene i metodi comuni a tutti i payloads.

**PayloadQueue** è una coda di payload, che contiene i payload da processare.

**PayloadThread** è un thread che si occupa di processare i payload.

**PayloadStatus** è un enum che contiene i possibili stati dei payload.

**PayloadManual** è un payload che contiene le informazioni per un'analisi manuale.

**PayloadAuto** è un payload che contiene le informazioni per un'analisi automatica.

I payload manuali e automatici sono thread che fanno le loro operazioni in parallelo, controllano se c'è da modificare uno stato e in caso positivo preparano le informazioni per il consumer.

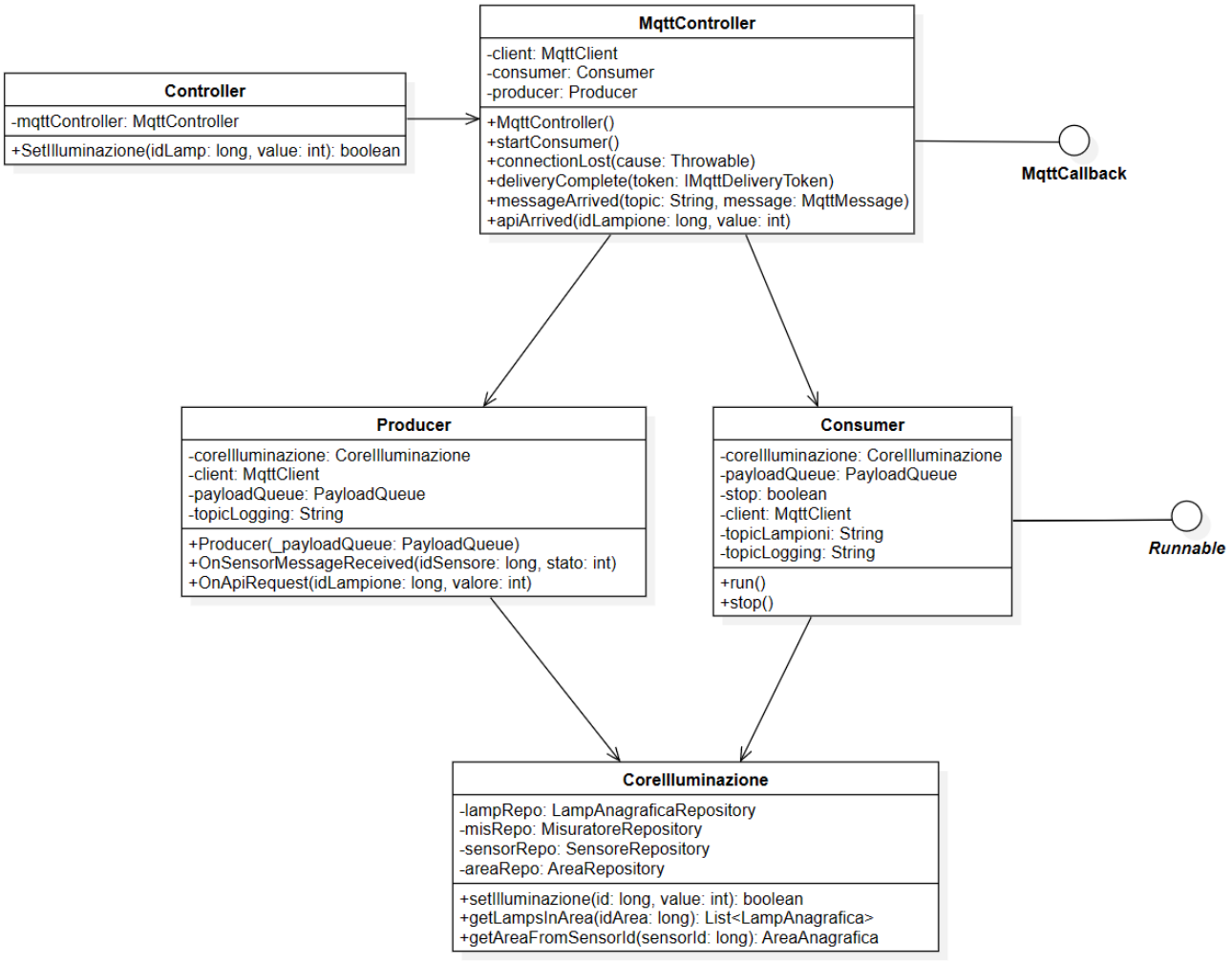



Figura 5.2: Diagramma delle classi relative alla parte mqtt del sistema di coordinazione

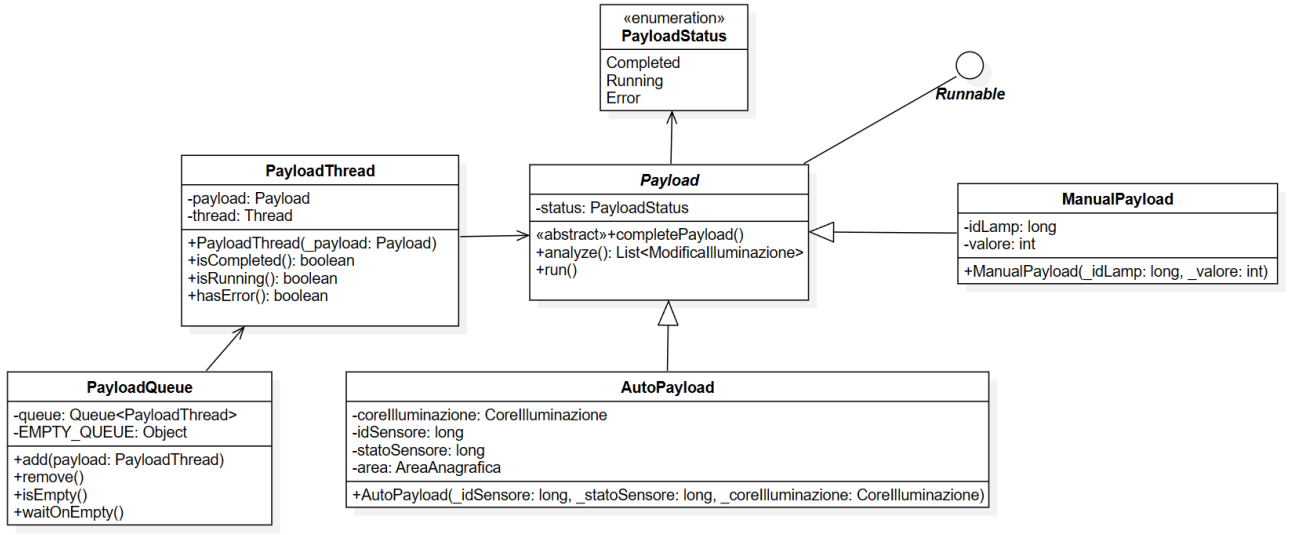



Figura 5.3: Diagramma delle classi relativa ai payloads

# Capitolo 6

# Sistema di autenticazione

## 6.1 Scopo del sistema

Il sistema di autenticazione ha lo scopo di gestire le richieste di accesso al macro-sistema generale da parte degli utenti[1] e di fornire un mezzo per rendere autenticato l'utente nell'usufruire delle risorse protette nel resto delle componenti del sistema generale.

### 6.1.1 Requsiti coperti dal sistema

### 6.1.2 RF_01

RF_01: Il sistema deve mostrare un messaggio d'errore esplicativo all'utente in caso di errore di autenticazione.

Questo microservizio, copre il requisito inviando un messaggio di errore in caso di autenticazione non valida.

### 6.1.3 RF_07

RF_07: l'utente deve effettuare l'accesso per poter utilizzare le funzionalità del sistema.

Questo microservizio, copre il requisito permettendo solo a chi è autenticato di utilizzare le funzionalità dei vari microservizi.

## 6.2 Descrizione del sistema

Per autenticare gli utenti, questi devono fornire le proprie credenziali di accesso, ovvero il proprio username e la propria password.

La sessione viene quindi mantenuta consegnando all'utente un token di sessione, che dovrà essere utilizzato per ogni richiesta successiva. Il token di sessione ha una durata limitata nel tempo, e quando scade l'utente deve richiedere un nuovo token di sessione, utilizzando il token di refresh che gli è stato consegnato insieme al token di sessione. Il token di refresh ha una durata maggiore rispetto al token di sessione, e può essere utilizzato solo per richiedere un nuovo token di sessione al servizio in oggetto, ogni volta che scade quello attualmente in uso.

Questo servizio offre le funzionalità sopra descritte tramite un'interfaccia REST.

I token di sessione utilizzati sono del tipo JWT (JSON Web Token) e sono composti da tre parti separate da un punto. La prima parte contiene le informazioni relative all'algoritmo di hashing utilizzato per la firma del token, la seconda parte contiene le informazioni relative all'utente autenticato e la terza parte contiene la firma del token.

La scelta di utilizzare un token di sessione di tipo JWT rende il sistema **stateless**, ovvero non è necessario memorizzare alcuna informazione relativa alla sessione dell'utente, in quanto tutte le informazioni necessarie sono contenute nel token stesso.

[1] Utenti umani oppure digitali

L'essere senza stato rende quindi molto semplice scalare orizzontalmente il sistema, in quanto non è necessario sincronizzare lo stato tra più istanze del servizio.

### 6.2.1 Riferimenti esterni

- Codice sorgente del sistema di autenticazione;

## 6.3 Architettura del sistema

Il sistema utilizza il paradigma dell'architettura esagonale, e più in dettaglio si attiene alla clean architecture.

Il core del sistema si occupa della business logic ed offre:

- Generazione dei JWT;
- Refresh dei JWT;
- Autenticazione degli utenti.

Le ports del sistema di autenticazione sono:

- **REST**: si occupa di esporre le funzionalità del sistema di autenticazione tramite un'interfaccia REST;
- **DB**: si occupa di interfacciarsi con il database per la memorizzazione dei dati;

Questi due sottosistemi architetturali sono uniti tramite degli adapter, che si occupano di interfacciare le ports con il core del sistema. Nel nostro glossario questi adapter sono chiamati **Service**.

## 6.4 Core del sistema

Il core del sistema è composto da nove classi, che sono rappresentate nel diagramma delle classi alla figura 6.1. Queste si occupano di implementare la business logic del sistema.

- **Authenticator:** si occupa dell'autenticazione dell'utente, verificando che l'username e la password fornite siano corrette.
- **RefreshJwtBlacklister:** si occupa di inserire in una *blacklist* i jwt di refresh che si vogliono rendere inutilizzabili.
- **JwtExtractor:** si occupa di estrarre la mappa chiave-valore dal jwt. In particolare nel nostro caso serve estrarre l'username dell'utente.
- **JwtIssuer:** si occupa della creazione e della rigenerazione dei jwt (access o refresh), inserendo al loro interno l'username dell'utente.
- **JwtPackager:** si occupa della creazione e della firma di un generico jwt. Ha un tipo, una mappa chiave-valore e una scadenza.
- **JwtVerifier:** si occupa di verificare che un jwt ricevuto sia valido, ovvero che abbia un determinato tipo (access o refresh), non sia ancora scaduto e che abbia la mappa chiave-valore.
- **JwtAuthority:** si occupa di esporre le funzionalità dei jwt utilizzando le relative classi.

### 6.4.1 I JWT

Il sistema generale è pensato per utilizzare due tipologie di Token. Viene utilizzato un token di autorizzazione, che viene utilizzato per l'autenticazione e per l'accesso alle risorse protette degli altri microservizi e un token di refresh, che viene utilizzato per la generazione di un nuovo token di autorizzazione.

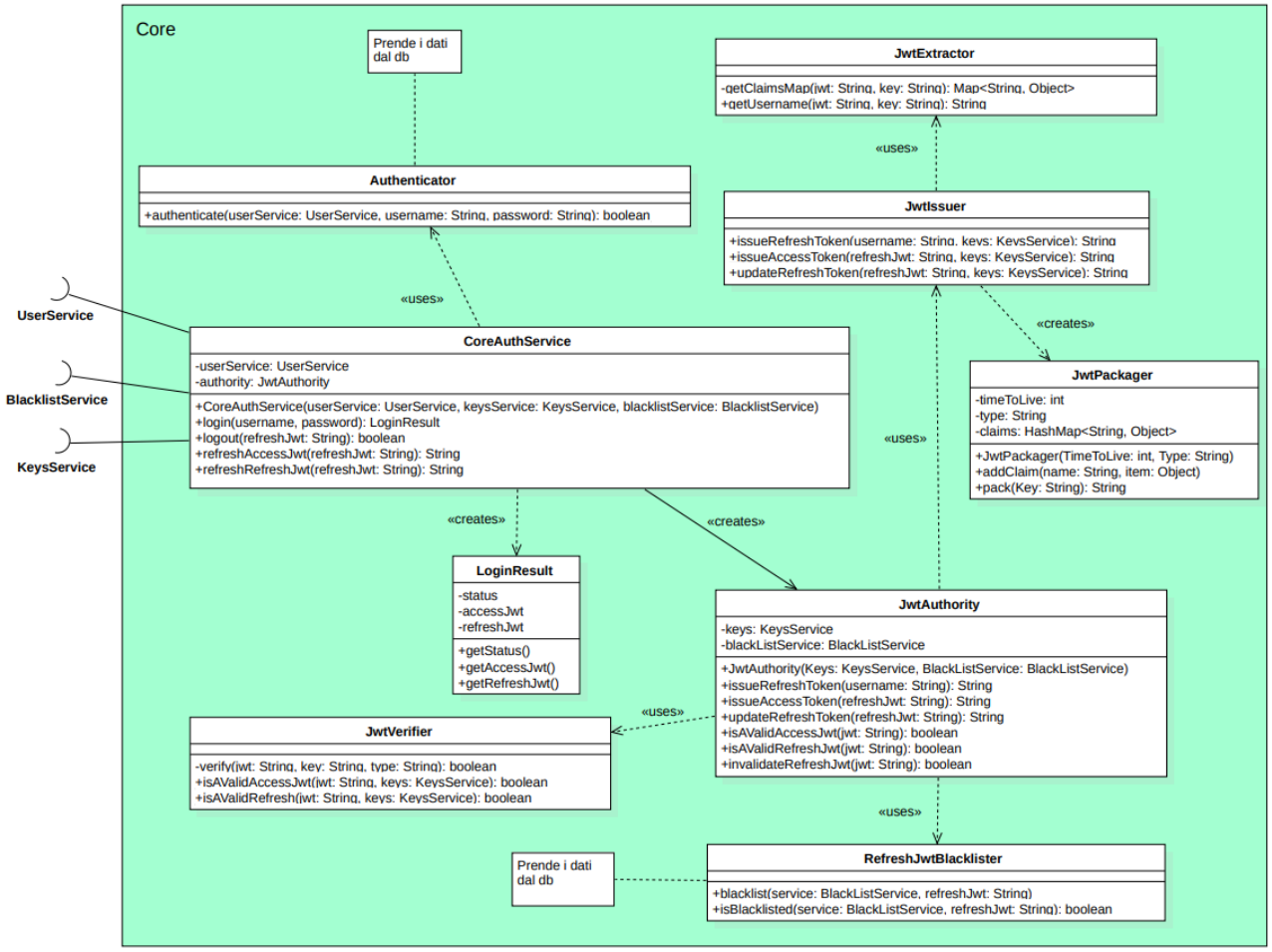



Figura 6.1: Diagramma delle classi del core del microservizio di autenticazione

Ognuno dei microservizi del sistema accetta come valido un token generato da questo microservizio. Non accetterà però come valido un token di refresh.

Il token di refresh viene infatti riconosciuto solamente da questo sistema.

Il token di refresh ha durata di 2h, mentre il token di autorizzazione ha durata di 10 minuti.

## 6.5 Ports

Le ports del sistema di autenticazione sono due, e si occupano di interfacciare il core del sistema con l'esterno.

È presente un'interfaccia REST, che espone le funzionalità del sistema di autenticazione tramite un'interfaccia REST, e un'interfaccia DB, che si occupa di interfacciarsi con il database per la memorizzazione dei dati.

## 6.6 Port Interfaccia REST

La porta REST si interfaccia al core del sistema tramite un'interfaccia chiamata **CoreAuthService**, che espone le funzionalità di autenticazione del core del microservizio, come si può vedere nel diagramma delle classi alla figura 6.2.

L'interfaccia REST viene esposta all'esterno e permette di effettuare le operazioni discusse nelle sotto-sezioni successive.

- POST /login

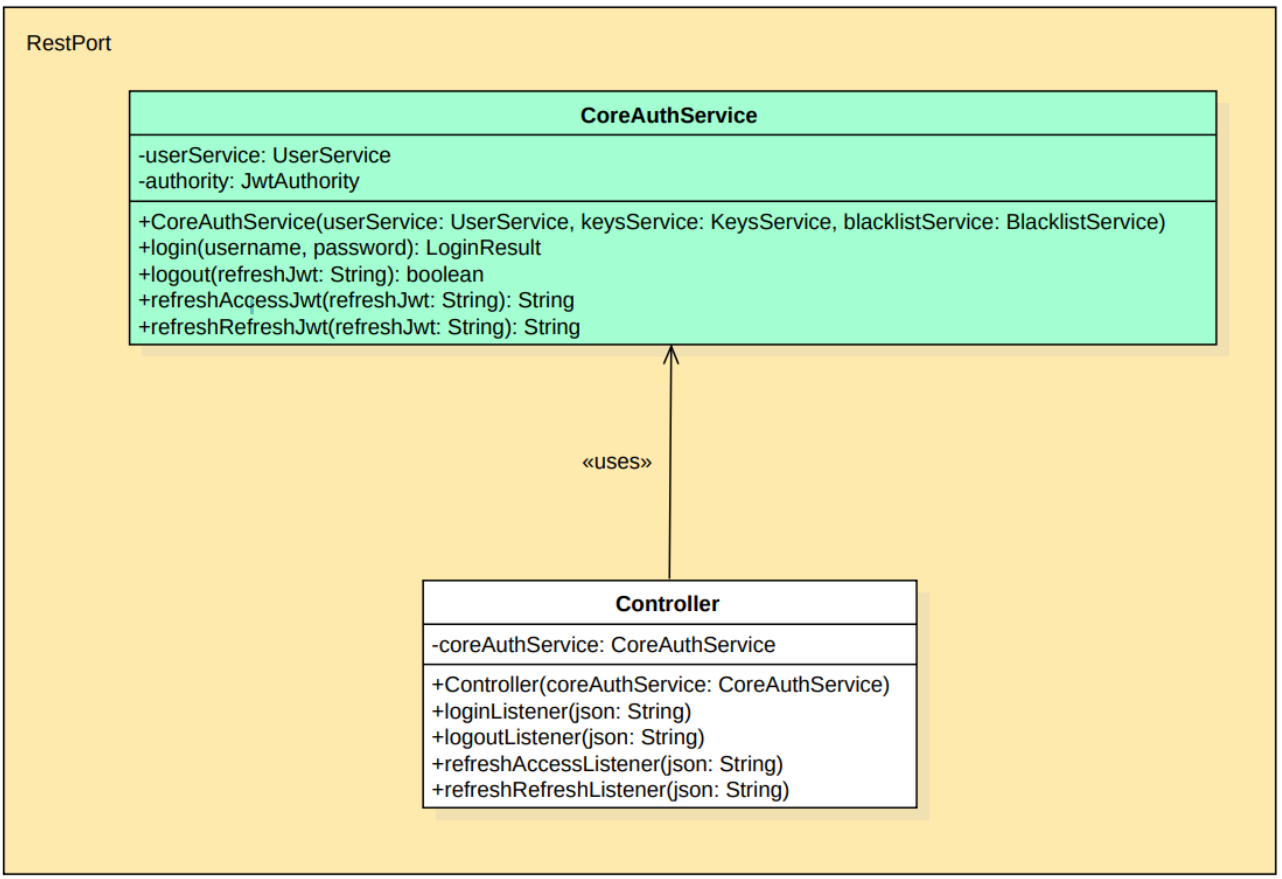



Figura 6.2: Diagramma delle classi della porta rest del microservizio di autenticazione

- POST /logout
- POST /refresh/refreshtoken
- POST /refresh/accesstoken

### 6.6.1 POST /login

Login permette di effettuare il login di un utente. L'utente deve fornire le credenziali di accesso, e il sistema risponderà con un token di autorizzazione e un token di refresh.

Le credenziali vengono fornite sotto formato JSON nel body della richiesta, e devono essere composte da username e password.

### 6.6.2 POST /logout

Logout permette di effettuare il logout di un utente. L'utente deve fornire il token di refresh, e il sistema risponderà con un messaggio di conferma.

Il sistema invalida il token di refresh fornito, e l'utente non potrà più utilizzarlo per generare nuovi token di autorizzazione.

Per invalidarlo, il sistema lo inserisce in una blacklist, e non lo accetterà più come valido.

Quando il token di refresh scade, il sistema lo rimuove dalla blacklist.

### 6.6.3 POST /refresh/refreshtoken

Refresh refreshtoken permette di generare un nuovo token di refresh. L'utente deve fornire il token di refresh, e il sistema risponderà con un nuovo token di refresh.

### 6.6.4 POST /refresh/accesstoken

Refresh accesstoken permette di generare un nuovo token di autorizzazione. L'utente deve fornire il token di refresh, e il sistema risponderà con un nuovo token di autorizzazione.

## 6.7 Port Database

Il sistema utilizza un database per memorizzare i JWT di accesso in blackilist.

Si immagina inoltre che il sistema utilizzi un database per memorizzare le credenziali degli utenti. Nella nostra versione attuale esistono però solamente due utenti e questi sono definiti all'interno del codice.

## 6.8 Adapters e Service in particolare

Gli adapters si occupano di interfacciare le ports con il core del sistema. Nel nostro glossario questi adapter sono chiamati **Service**. Questi sono i servizi che vengono esposti all'esterno oppure che vengono utilizzati all'interno del microservizio.

## 6.9 UserService

UserService si occupa di ottenere gli utenti e le loro credenziali. In particolare si occupa di ottenere le credenziali di un utente a partire dal suo username e di verificare che le credenziali fornite siano corrette.

Al momento è presente solo un LocalUserService, che si occupa di ottenere le credenziali degli utenti dal codice. In futuro potrebbe essere implementato un RemoteUserService, che si occupa di ottenere le credenziali degli utenti da un altro microservizio oppure da un database.

Questo risponde al caso d'uso **UC01: Autenticazione**.

## 6.10 BlacklistService

BlacklistService si occupa di inserire un token di refresh in blacklist e di verificare che un token di refresh non sia in blacklist.

È presente un RemoteBlacklistService che utilizza la porta DB per gestire i token di refresh in blacklist. Si occupa quindi di inserire un token di refresh in blacklist e di verificare che un token di refresh non sia in blacklist.

## 6.11 KeysService

Il KeysService si occupa di fornire le chiavi utili a firmare e verificare i JWT. In particolare fornisce la chiave privata e la chiave pubblica.

È presente un LocalKeysService che utilizza le chiavi presenti nel codice. In futuro potrebbe essere implementato un RemoteKeysService, che si occupa di ottenere le chiavi da un altro microservizio oppure da un database o qualsiasi altra implementazione.

# Capitolo 7

# Sistema di logging

## 7.1 Scopo del sistema

Il sistema di logging serve per mantenere uno storico dei cambiamenti di stato e valore dei vari dispositivi, in modo da poterli consultare in un secondo momento.

## 7.2 Requisiti del sistema

### 7.2.1 RF_23

RF_23: L'utente deve essere in grado di visualizzare gli stati dei dispositivi in un certo periodo di tempo.

Questo microservizio, copre il requisito memorizzando gli stati dei dispositivi in un database.

## 7.3 Descrizione del sistema

Il sistema permette di leggere i dati storici degli stati dei dispositivi tramite un interfaccia rest. Rende quindi possibile la visione di un insieme di log con diverse opzioni, ad esempio in un certo periodo di tempo o gli ultimi n log.

## 7.4 Architettura del sistema

Il sistema è composto da un'interfaccia REST e un insieme di classi che si occupano di leggere i dati dal database. Il servizio nasce molto semplice e per questo motivo non è stato ritenuto necessario l'utilizzo di un'architettura più complessa e strutturata.

## 7.5 Repository

Il LogRepository è un'interfaccia repository JPA ed esegue fisicamente le operazioni sul database.

## 7.6 Servizi

I servizi sono in generale le interfacce ad alto livello delle funzionalità del sistema. Sono composti da classi che si occupano di gestire le richieste e di fornire i dati richiesti.

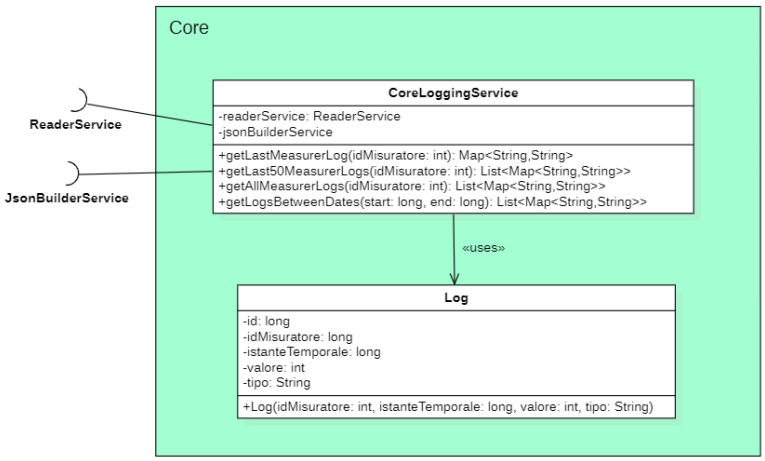



Figura 7.1: Vista generale del sistema di logging

### 7.6.1 ReaderService

ReaderService si occupa di leggere e fornire i dati dei log dal database in base alla richiesta.

ReaderService viene realizzata in localReaderService per i test. Invece per connettersi al db viene utilizzata la classe DatabaseReaderService.

## 7.7 Core

Il core è il nucleo del sistema ed in generale si pone come facade per tutti i servizi che si trovano all'interno del microservizio. Questa classe si occupa di gestire le richieste e di fornire i dati richiesti. Principalmente questa classe viene utilizzata dall'interfaccia REST.

## 7.8 Log

La classe Log viene utilizzata per trasferire le informazioni tra le varie classi ed è costruito da logrepository quando fa operazioni sul database e deve ritornarne uno.

## 7.9 JsonBuilderService

Si occupa di creare i JSON da ritornare all'interfaccia REST a partire dai log.

## 7.10 Config

La classe Config genera i bean che servono alle altre classi per funzionare.

## 7.11 Interfaccia REST

- GET /log/last/idMisuratore
- GET /log/last50/idMisuratore
- GET /log/all/idMisuratore
- GET /log/dates/start/end

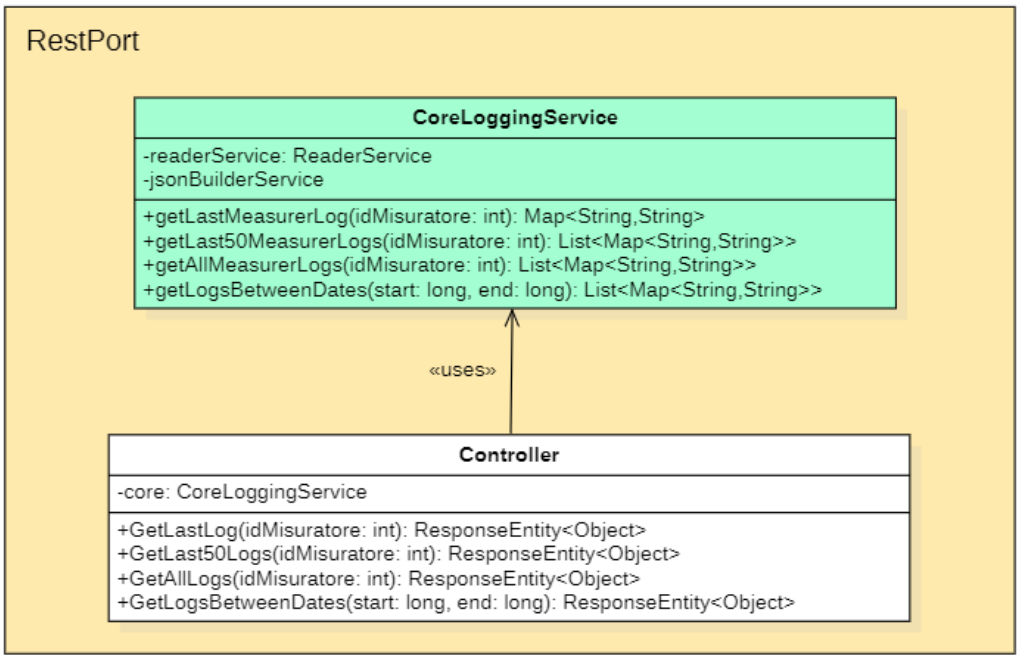



Figura 7.2: Vista delle classi che si occupano dell'interfaccia rest

### 7.11.1 GET /log/last/idMisuratore

Get /log/last/idMisuratore ritorna l'ultimo log del dispositivo con id idMisuratore.

### 7.11.2 GET /log/last50/idMisuratore

Get /log/last50/idMisuratore ritorna gli ultimi 50 log del dispositivo con id idMisuratore.

### 7.11.3 GET /log/all/idMisuratore

Get /log/all/idMisuratore ritorna tutti i log del dispositivo con id idMisuratore.

### 7.11.4 GET /log/dates/start/end

Get /log/dates/start/end ritorna tutti i log compresi tra start e end.

# Appendice A

# Specifica delle basi di dati

## A.1 Base di dati sistema anagrafe

### A.1.1 Abstract

L'obiettivo è sviluppare un servizio che tenga ordinati e renda sempre disponibili tutte le informazioni relative a nomi, gruppi, peculiarità di ognuno dei sensori, delle aree, e dei lampioni.

Queste informazioni sono particolarmente utili per contestualizzare ognuna delle componenti *hardware* del sistema.

### A.1.2 Analisi dei requisiti

**Descrizione testuale**

Il microservizio di anagrafe ha bisogno di memorizzare, per poter funzionare correttamente, i seguenti dati:

- le aree, in particolare le informazioni;
- i lampioni e le loro informazioni, ovvero posizione e consumo energetico;
- i sensori e le loro informazioni, ovvero posizione e raggio d'azione.

Inoltre per i misuratori, ovvero lampioni e sensori, viene salvata anche l'area di appartenenza.

**Glossario dei termini**

Per evitare ambiguità relative alle terminologie utilizzate è stato creato un documento denominato *Glossario*.

Questo documento contiene tutti i termini specifici di settore utilizzati nei documenti, con le relative definizioni.

**Operazioni tipiche**

Le operazioni tipiche che ci si aspetta di avere sono:

| OPERAZIONI TIPICHE | |
|---|---|
| **DESCRIZIONE** | **FREQUENZA D'USO** |
| Quanti lampioni si trovano all'interno di un'area | Poche volte |
| Quali lampioni si trovano all'interno di un'area | Molte volte |
| Il sensore X che lampioni controlla? | Molte volte |
| Aggiunta di un'area | Poche volte |
| Aggiunta di un lampione ad un'area | Medie volte |
| Aggiunta di un sensore ad un'area | Medie volte |

### A.1.3 Progettazione concettuale

**Analisi delle entità**

**Se non specificato l'attributo è NOT NULL**

| AREA | | | |
|---|---|---|---|
| id | INTEGER | Identifica univocamente un'area all'interno del sistema | Chiave |
| nome | VARCHAR(50) | Il nome dell'area | |
| autoMode | BOOLEAN | Specifica se l'area viene gestita in modalità automatica o no (manuale) | |
| lvlInf | VARCHAR(50) | In area gestita in modalità automatica, questo è il livello di luminosità a cui vengono posti i lampioni se non vengono rilevate persone all'interno dell'area | |
| lvlSup | VARCHAR(20) | In area gestita in modalità automatica, questo è il livello di luminosità a cui vengono posti i lampioni se vengono rilevate persone all'interno dell'area | |

| MISURATORE | | | |
|---|---|---|---|
| id | SERIAL | Identifica univocamente un misuratore del livello di luminosità | Chiave |
| tipo | VARCHAR(10) | La tipologia del misuratore | |
| latitudine | REAL | Latitudine delle coordinate geografiche in cui viene posto il misuratore | |
| longitudine | REAL | Longitudine delle coordinate geografiche in cui viene posto il misuratore | |
| idArea | INTEGER | Identifica l'area a cui fa riferimento il misuratore | Chiave esterna: Area(id) |

| SENSORE | | |
|---|---|---|
| raggio | INTEGER | Il raggio d'azione entro il quale il sensore è in grado di rilevare persone |

| LAMPIONE | | |
|---|---|---|
| luminosita | INTEGER | Livello di luminosità in cui si trova il lampione |
| wattaggio | INTEGER | Consumo energetico del lampione |

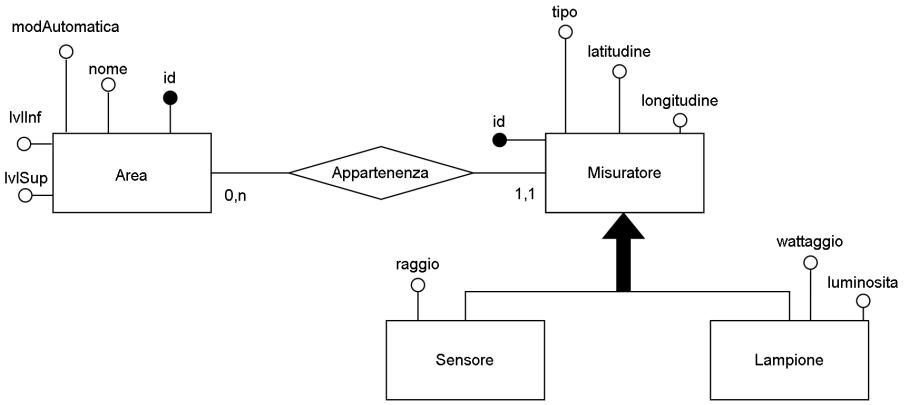
**Analisi delle relazioni e delle cardinalità**

- Area - Misuratore: **Appartenenza**
  - Un misuratore è contenuto in una ed una sola area (1,1);
  - Un'area contiene da 0 a N misuratori (0,N);

**Generalizzazioni**

- Lampione è generalizzazione totale esclusiva di Misuratore;
- Sensore è generalizzazione totale esclusiva di Misuratore;

**Schema ER concettuale**




## A.1.4 Progettazione logica

**Eliminazione delle generalizzazioni**

**Misuratore** Per evitare di accorpare le entità Sensore e Lampione nell'entità padre Misuratore, creando così dei campi NULL, è stato deciso di risolvere la generalizzazione mantenendo le tre entità inserendo le relazioni tra le entità figlie e l'entità padre.

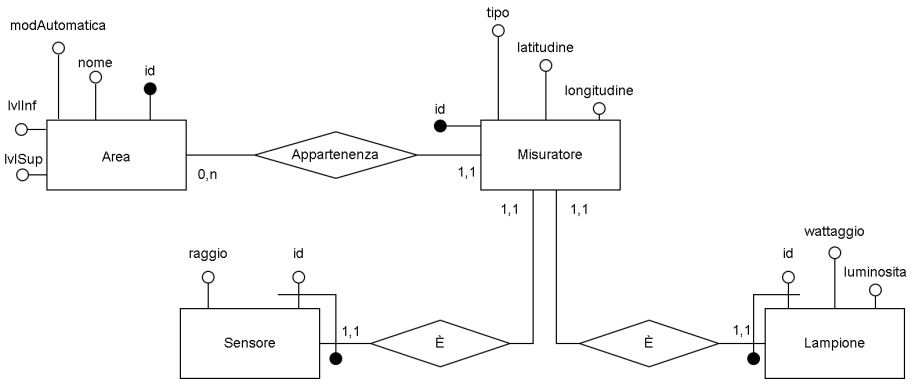
**Modifiche, aggiunte e chiarimenti alle chiavi**

Tutte le chiavi primarie sono definite utilizzando le chiavi della progettazione concettuale, tranne nei seguenti casi.

**Sensore** Diventando il Sensore un'entità a sé stante, viene aggiunta una chiave esterna, che fungerà anche da chiave primaria, che indica il Misuratore a cui fa riferimento.

**Lampione** Diventando il Lampione un'entità a sé stante, viene aggiunta una chiave esterna, che fungerà anche da chiave primaria, che indica il Misuratore a cui fa riferimento.

**Schema concettuale ristrutturato - Schema Logico**




**Descrizione schema relazionale**

Per questione di compatibilità con il DBMS alcuni nomi di attributi entità e relazioni sono stati normalizzati, utilizzando il camelCase, togliendo gli accenti, accorciando i nomi molto lunghi e con altre piccole accortezze. La chiave primaria è indicata in **grassetto**, le chiavi esterne sono indicate con la sottolineatura.

*Area*(**id**, nome, autoMode, lvlInf, lvlSup)
*Misuratore*(**id**, idArea, tipo, latitudine, longitudine)
*Sensore*(**idMisuratore**, raggio)
*Lampione*(**idMisuratore**, luminosita, wattaggio)

#### Vincoli di integrità referenziali

**Sensore**.idMisuratore -> *Misuratore*.id
**Lampione**.idMisuratore -> *Misuratore*.id

#### Check e constraint

**Area** In **Area** è attivato un check che controlla che il valore del livello inferiore sia sempre minore del valore del livello superiore.

## A.2 Base di dati sistema coordinazione

### A.2.1 Abstract

L'obiettivo è quello di sviluppare un sistema che coordini e decida quando illuminare o non illuminare specifiche zone. Questo sistema dovrà tenere traccia di tutte le informazioni relative allo stato attuale dei lampioni.

Il sistema si collegherà ad altre componenti esterne, utilizzando algoritmi proprietari, e deciderà quando effettuare cambiamenti di stato. Si consiglia la visione del capitolo 5.

### A.2.2 Analisi dei requisiti

#### Descrizione testuale

Il microservizio per funzionare correttamente avrà bisogno di salvare lo stato dei lampioni per controllare che non vengano fatte modifiche con tempistiche troppo ravvicinate, e inoltre vengono memorizzati la luminosità e l'istante dell'ultima modifica di un lampione.

#### Glossario dei termini

Per evitare ambiguità relative alle terminologie utilizzate è stato creato un documento denominato *Glossario*.

Questo documento contiene tutti i termini specifici di settore utilizzati nei documenti, con le relative definizioni.

#### Operazioni tipiche

Le operazioni tipiche che ci si aspetta di avere sono:

| OPERAZIONI TIPICHE | |
|---|---|
| **DESCRIZIONE** | **FREQUENZA D'USO** |
| Lettura dello stato in cui si trova il lampione | Molte volte |
| Scrittura dello stato di un lampione | Molte volte |

### A.2.3 Progettazione concettuale e progettazione logica

In seguito alla riunione dei progettisti del 30/07[1], il sistema di coordinazione non avrà un database "indipendente" ma utilizzerà, a seconda dell'operazione richiesta, il database del sistema di anagrafe e quello del sistema di logging.

## A.3 Base di dati sistema logging

### A.3.1 Abstract

L'obiettivo è sviluppare un servizio che tenga traccia in maniera intelligente dei log. Questo servizio offre numerosi benefici legati alla retrospettiva. Il miglioramento delle prestazioni del sistema, la risoluzione rapida dei problemi, il rilevamento delle minacce alla sicurezza e la generazione di informazioni utili per l'analisi e il miglioramento continuo sono solo alcuni dei benefici che questo servizio permette.

### A.3.2 Analisi dei requisiti

#### Descrizione testuale

Il microservizio per funzionare correttamente avrà bisogno di memorizzare ogni cambiamento, denominato *log*, a lungo termine. Dei *log* ci interessa:

- l'id del log;
- l'id del misuratore che ha cambiato stato;
- l'istante in cui la modifica è stata effettuata;
- il valore del log, che cambia a seconda che la modifica avvenga ad un sensore oppure ad un lampione;
- la tipologia di log.

#### Glossario dei termini

Per evitare ambiguità relative alle terminologie utilizzate è stato creato un documento denominato *Glossario*.

Questo documento contiene tutti i termini specifici di settore utilizzati nei documenti, con le relative definizioni.

#### Operazioni tipiche

Le operazioni tipiche che ci si aspetta di avere sono:

| OPERAZIONI TIPICHE | |
|---|---|
| **DESCRIZIONE** | **FREQUENZA D'USO** |
| Lettura di molteplici log aggregati per l'estrazione dei trend | Molte volte |
| Scrittura del log | Molte volte |

### A.3.3 Progettazione concettuale

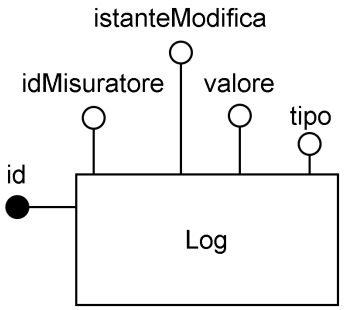
#### Analisi delle entità

**Se non specificato l'attributo è NOT NULL**

[1] Vedasi verbale VIN_20230730

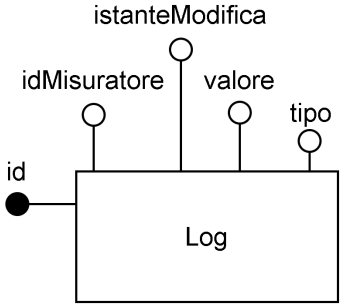
| LOG | | | |
|---|---|---|---|
| id | INTEGER | Identifica univocamente un log | Chiave |
| idMisuratore | INTEGER | Identifica il misuratore di riferimento | |
| istanteModifica | LONG | Indica quando c'è stata una variazione ad un lampione o ad un sensore | |
| valore | INTEGER | Nel caso di un sensore, indica se è in grado di rilevare persone oppure no<br>Nel caso di un lampione, indica il suo valore di luminosità | |
| tipo | VARCHAR(50) | Indica se il sistema sta facendo riferimento ad un sensore oppure ad un lampione | |

**Schema ER concettuale**




### A.3.4 Progettazione logica

**Schema concettuale ristrutturato - Schema Logico**




**Descrizione schema relazionale**

Per questione di compatibilità con il DBMS alcuni nomi di attributi entità e relazioni sono stati normalizzati, utilizzando il camelCase, togliendo gli accenti, accorciando i nomi molto lunghi e con altre piccole accortezze. La chiave primaria è indicata in **grassetto**, le chiavi esterne sono indicate con la sottolineatura.

*Log*(**id**, idMisuratore, istanteModifica, valore, tipo)

# Elenco delle figure