



Specifica Tecnica

swellfish14@gmail.com

Informazioni

| | |
|---------------------|--|
| <i>Redattori</i> | [Davide Porporati, Elena Marchioro, Francesco Naletto] |
| <i>Revisori</i> | [Jude Vensil Braceross] |
| <i>Responsabili</i> | [Andrea Veronese] |
| <i>Uso</i> | [Esterno] |

Descrizione

File contenente la specifica tecnica necessaria per la realizzazione del progetto.

| Versione | Data | Redattore | Verificatore | Descrizione |
|-----------------|-------------|-----------------------------------|---------------------|---|
| 1.0.1 | 18/09/2023 | Davide Porporati, Elena Marchioro | Francesco Naletto | Aggiornati i diagrammi delle classi e aggiunte sezioni mancanti |
| 1.0.0 | 09/09/2023 | Davide Porporati, Elena Marchioro | Francesco Naletto | Aggiornati i diagrammi delle classi |
| 0.0.3 | 09/09/2023 | Davide Porporati, Elena Marchioro | Claudio Giarretta | Aggiornati i design pattern e revisionato il documento |
| 0.0.2 | 04/09/2023 | Davide Porporati, Elena Marchioro | Francesco Naletto | Aggiornati i design pattern e caricato diagramma delle classi |
| 0.0.1 | 01/09/2023 | Davide Porporati, Elena Marchioro | Francesco Naletto | Modificata tabella requisiti e informazioni principali |
| 0.0.0 | 09/08/2023 | Elena Marchioro | Davide Porporati | Creata struttura di base del documento |

Contents

| | | |
|----------|---|-----------|
| 1 | Introduzione | 4 |
| 1.1 | Scopo del documento | 4 |
| 1.2 | Scopo del prodotto | 4 |
| 1.3 | Riferimenti | 4 |
| 1.3.1 | Riferimenti normativi | 4 |
| 1.3.2 | Riferimenti informativi | 4 |
| 2 | Tecnologie Utilizzate | 5 |
| 2.1 | Front-end | 5 |
| 2.2 | Back-end | 5 |
| 2.3 | Database | 5 |
| 2.4 | Interfacciamento Lampioni/Sensori | 5 |
| 3 | Architettura del prodotto | 6 |
| 3.1 | Diagramma delle classi | 6 |
| 3.1.1 | Back-End | 6 |
| 3.1.2 | Front-End | 7 |
| 3.2 | Design Pattern | 9 |
| 3.2.1 | Back-end | 9 |
| 3.2.2 | Front-end | 9 |
| 3.3 | Interfacciamento con lampioni e sensori | 11 |
| 3.4 | Persistenza dei dati | 11 |
| 3.5 | Autenticazione | 12 |
| 4 | Requisiti soddisfatti | 13 |
| 4.1 | Tabella requisiti soddisfatti | 13 |
| 4.2 | Qualità | 16 |
| 4.3 | Dati copertura test | 16 |
| 4.3.1 | Percentuali test | 17 |
| 4.3.2 | Test Unità | 17 |
| 4.3.3 | Test Integrazione | 17 |

1 Introduzione

1.1 Scopo del documento

Nel seguente documento vengono illustrate e motivate le scelte architetture decise. Vengono riportati i diagrammi delle classi per l'architettura e le funzionalità principali, il diagramma ER della base di dati e infine una sezione dalla quale si può verificare lo stato di avanzamento del prodotto grazie a una tabella che illustra i requisiti soddisfatti.

1.2 Scopo del prodotto

L'obiettivo di SWellfish e dell'azienda ImolaInformatica S.p.A. è lo sviluppo di un sistema per l'ottimizzazione dell'illuminazione, attraverso la realizzazione di una WebApp che permetta a degli utenti registrati di gestire l'impianto di illuminazione di un'area in modo manuale e automatico. Nel documento viene riportata l'architettura del sistema per i vari servizi e i design pattern utilizzati.

1.3 Riferimenti

1.3.1 Riferimenti normativi

- Norme di progetto
- Capitolato d'appalto C2 - Lumos Minima

1.3.2 Riferimenti informativi

- Analisi dei requisiti
- Slide P2 del corso di ingegneria del software - Diagrammi delle classi
- Slide P4 del corso di ingegneria del software - Progettazione: il pattern Model-View-Controller e derivati

2 Tecnologie Utilizzate

2.1 Front-end

Per realizzare il frontend, ovvero la GUI del sistema, le seguenti tecnologie sono state impiegate:

- React: libreria JavaScript per creare GUI
- Typescript: linguaggio basato su JavaScript, offre migliore scalabilità rispetto a JS
- Bulma: framework CSS, reponsible e modulare, basato su Flexbox.

2.2 Back-end

- Node.JS: runtime di tipo JavaScript
- Express: framework per Node.JS
- Axios: client HTTP per Node.JS di tipo "promise-based"
- Sequelize: ORM tool per MariaDB, utilizzato per modellare i dati ed effettuare associazioni
- Cron: modulo di node, funge da scheduler e viene impiegato per creare task ad esecuzione automatica

2.3 Database

Il database implementato è di tipo relazionale, ed è stato implementato utilizzando MariaDB e HeidiSQL.

2.4 Interfacciamento Lampioni/Sensori

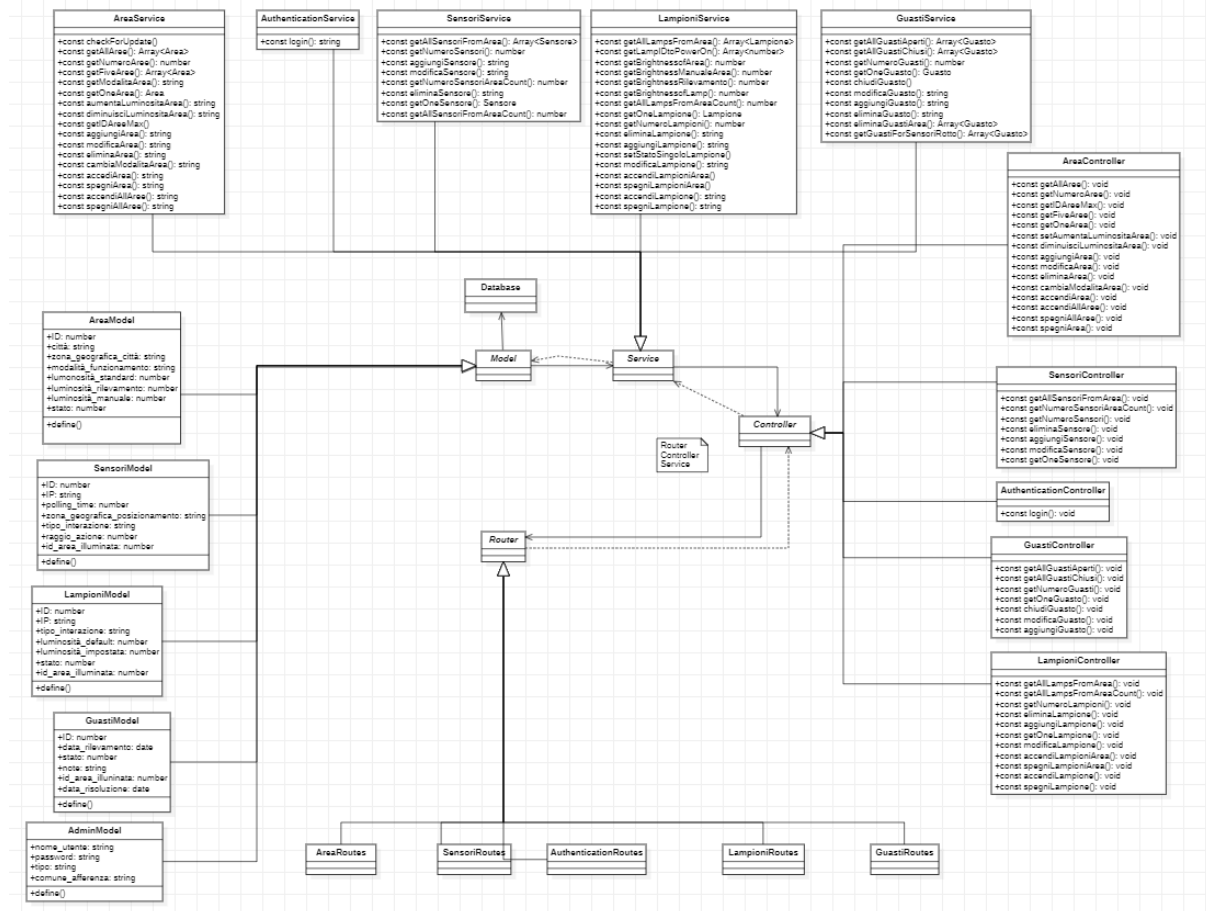
Per realizzare l'interfacciamento con i sensori e i lampioni a sistema le seguenti tecnologie sono state impiegate:

- Python
- API-rest

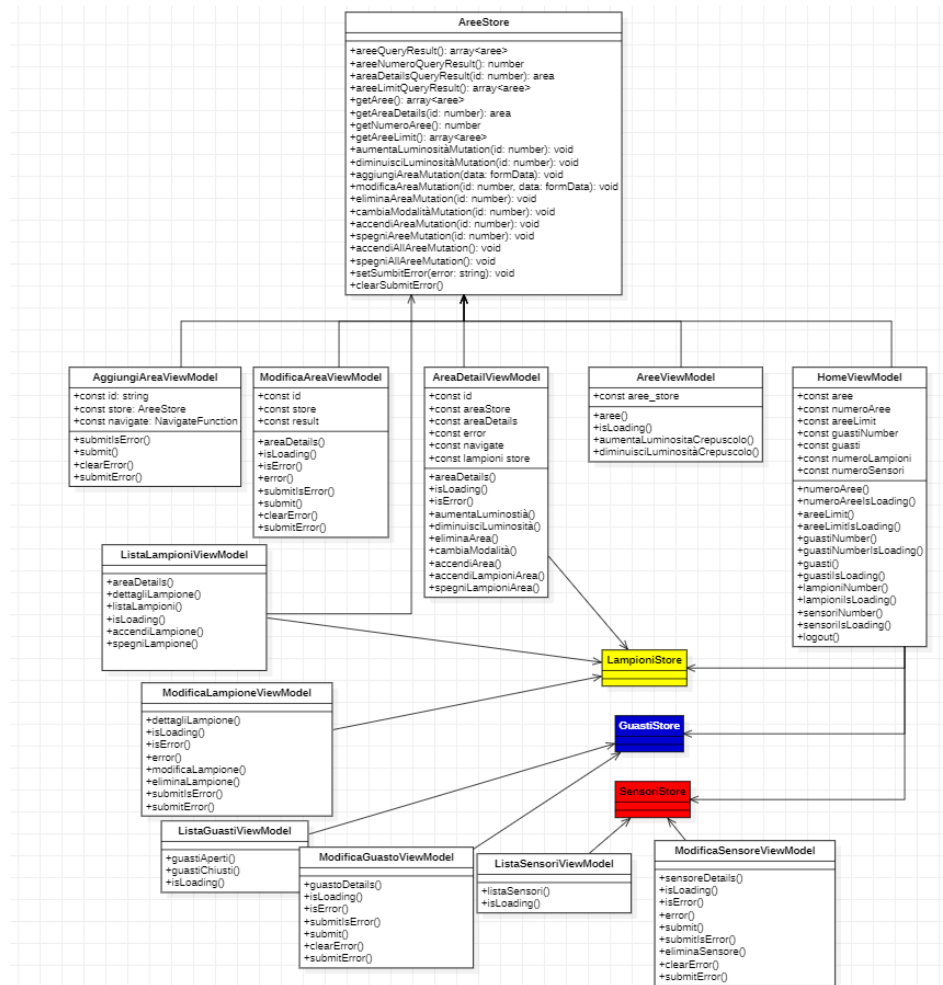
3 Architettura del prodotto

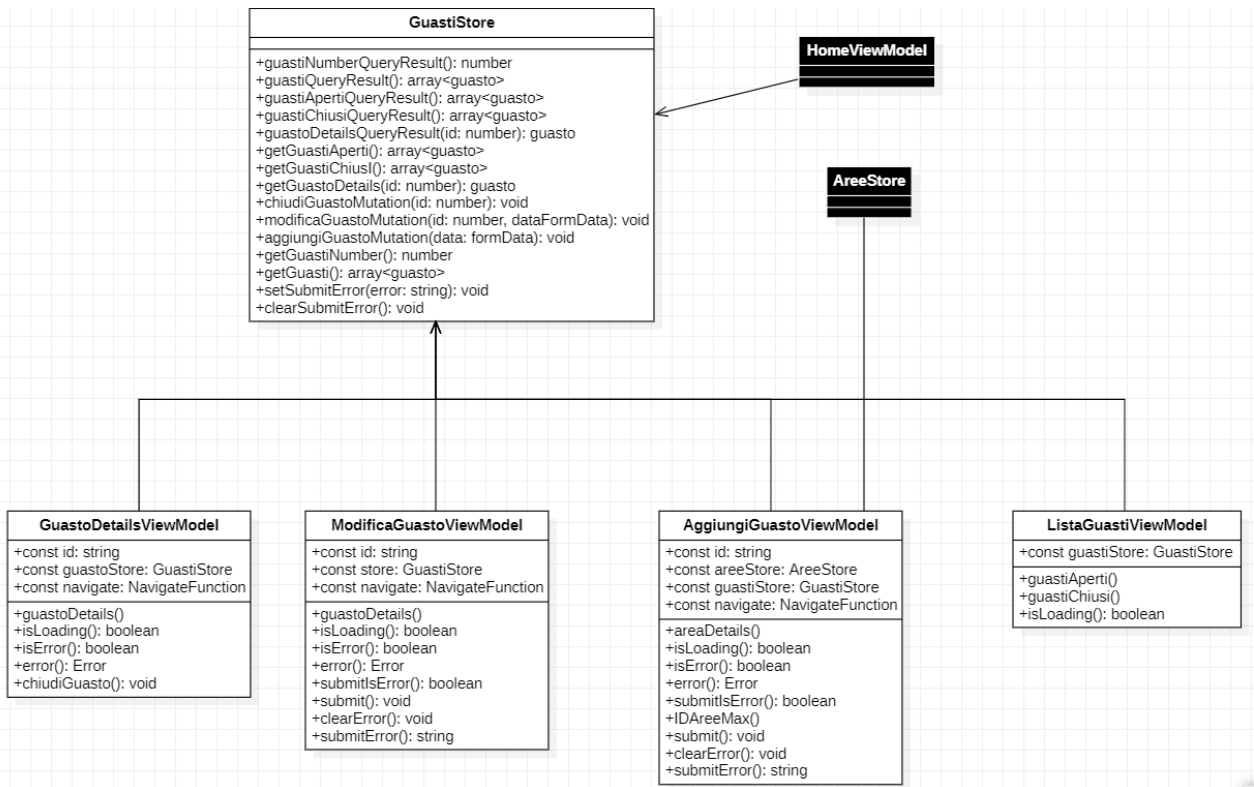
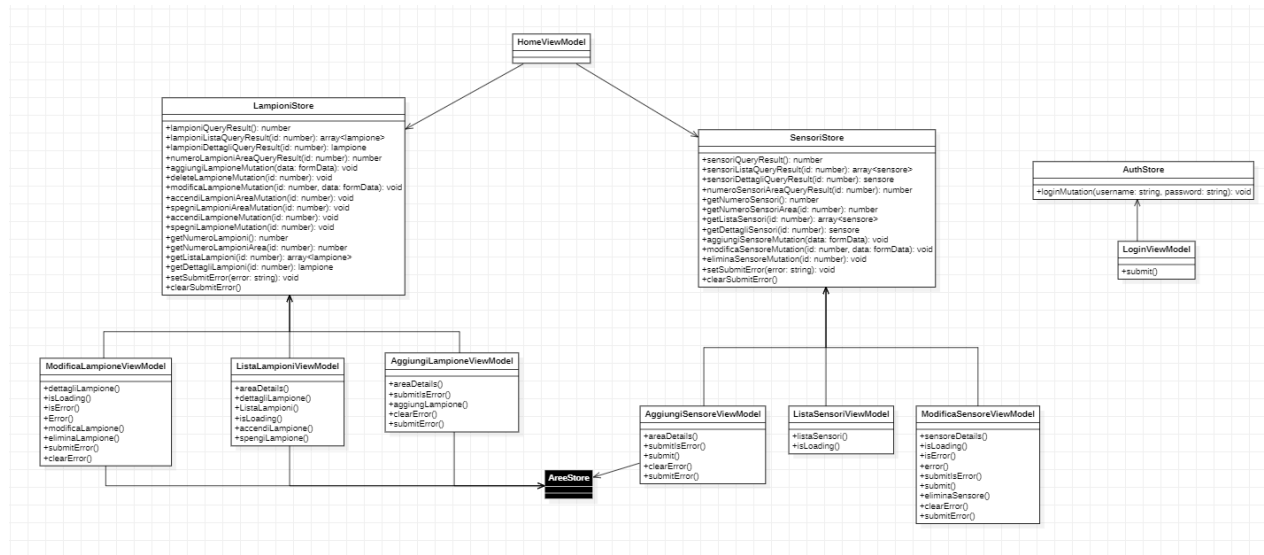
3.1 Diagramma delle classi

3.1.1 Back-End



3.1.2 Front-End





3.2 Design Pattern

3.2.1 Back-end

Per il backend è stato utilizzato il seguente pattern:

- Router Controller Service Pattern: Il design pattern dell'API Router-Controller-Service è un modello di architettura del software comunemente utilizzato nelle applicazioni web per strutturare e organizzare il codice responsabile della gestione delle richieste e delle risposte HTTP. Questo pattern aiuta a mantenere la separazione delle responsabilità e migliora la modularità e la manutenibilità dell'applicazione.

Il pattern è composto dai seguenti componenti:

- Router: componente che si occupa di effettuare il routing verso il controller adatto
- Controller: componente che elabora la richiesta ricevuta dal Router. Si appoggia alla classe Service per eseguire le operazioni.
- Service: componente che esegue la logica dell'applicazione.

Ecco come funziona il pattern in pratica:

- Un client invia una richiesta HTTP alla tua applicazione;
- Il componente Router riceve la richiesta e determina quale Controller deve gestirla in base all'URL e al metodo HTTP;
- Il Controller selezionato elabora la richiesta. Se necessario, chiama i metodi del livello di Servizio per eseguire la logica aziendale e le operazioni sui dati.
- Il Controller costruisce una risposta HTTP, che viene inviata al client.

3.2.2 Front-end

Per il frontend si sono utilizzati i pattern:

- Observer Pattern:
 - Scopo: definire una dipendenza fra oggetti, riflettendo la modifica di un oggetto sui dipendenti.

- Motivazione: mantenere la consistenza fra oggetti e definire come implementare la relazione di dipendenza.
- Dependency Injection: le dipendenze sono tracciate e passate agli oggetti tramite costruttore. Questo pattern è stato impiegato perchè facilita il tracciamento delle dipendenze e agevola la fase di testing, rendendo più semplice il mocking.
- Model View ViewModel (MVVM): è un modello di architettura del software che facilita la separazione dello sviluppo dell'interfaccia grafica, ovvero la GUI, sia tramite un linguaggio di markup o un codice GUI, dallo sviluppo del business logic o logica back-end in modo tale che la vista non dipenda da alcuna piattaforma di modello specifica. I componenti del modello MVVM:
 - Model: nel nostro caso è rappresentato nel file "api-types.ts"
 - View: viene definita tramite un template HTML accessibile nella cartella "Public". Per ogni view la parte root del template viene sostituita con la vista corrispondente
 - ViewModel: viene rappresentato dalle classi TypeScript utilizzate per gestire gli eventi della vista e aggiornare il modello di conseguenza. Ne è un esempio la classe "AreaViewModel", che gestisce la visualizzazione della lista delle aree presenti a sistema.

Il pattern implementato dal gruppo si appoggia ad una classe Service, che si occupa di fungere da classe di appoggio per richiamare le operazioni del backend.

- IoC(Inversion of Control)
 - Pattern che si concentra sulla gestione delle dipendenze e sul controllo dell'istanziamento degli oggetti. Questo pattern inverte il controllo tradizionalmente detenuto dalla componente chiamante, mettendo il controllo nelle mani di un framework o di un contenitore di gestione delle dipendenze. L'uso di questo pattern favorisce il disaccoppiamento e semplifica i test poiché le dipendenze sono dichiarate e possono essere facilmente sostituite con versioni mock durante i test.

3.3 Interfacciamento con lampioni e sensori

Per simulare i lampioni presenti in un dato momento nel Database, è stato modificato lo script fornito da Imola Informatica per simulare i lampioni. Lo script modificato, realizzato in python, utilizza l'export in formato JSON della relativa tabella dei lampioni presente nel DB per simulare in maniera automatica tutti i lampioni a sistema. Per realizzare ciò, è stato aggiunto un `argparser` allo script, e il suo utilizzo ha permesso di simulare N istanze dei lampioni, con porte diverse in base all' ID del singolo apparecchio luminoso.

Un approccio molto simile è stato applicato per simulare i sensori presenti nel DB.

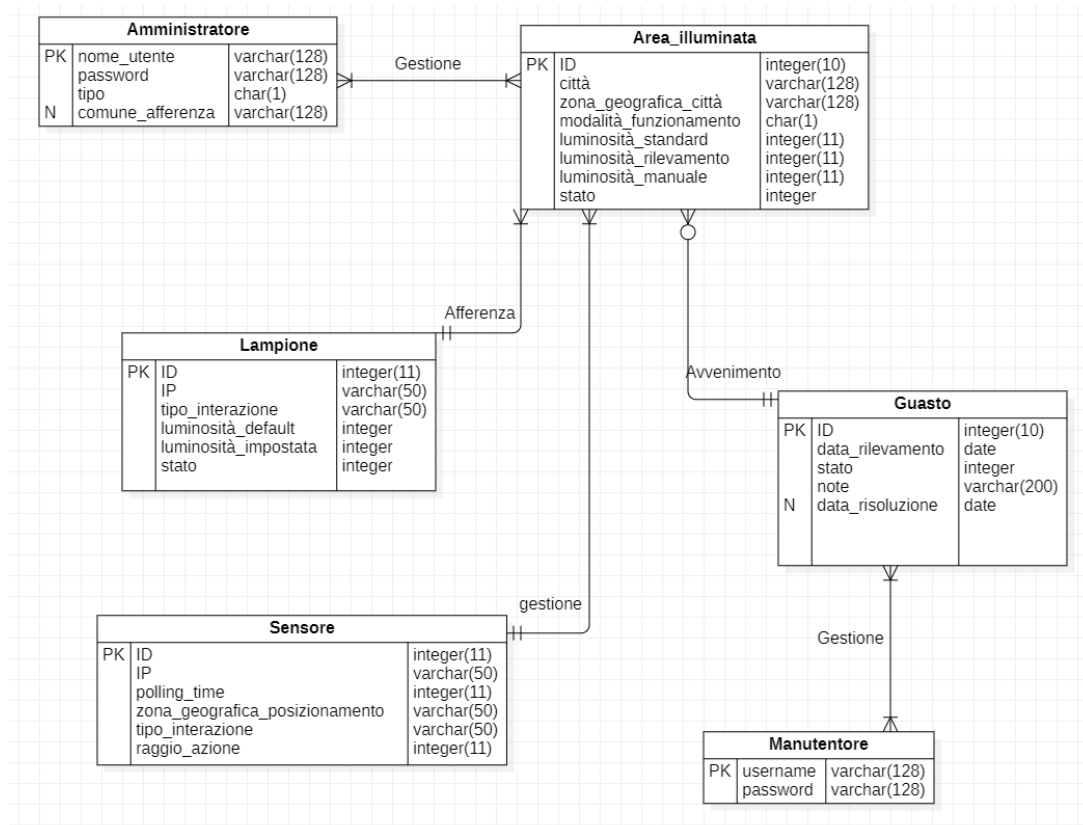
In questo modo, dopo aver fatto partire i due simulatori è possibile visualizzare:

- tutti i lampioni, raggiungibili dalla porta $4000 + \text{l'id del singolo lampione}$. Dato ad esempio il lampione con ID 1, esso è raggiungibile alla porta 4001.
- tutti i sensori, raggiungibili con lo stesso meccanismo dei lampioni, ma sulle porte 5000.

Avendo tutte le istanze necessarie, il sistema permette di effettuare tutte le operazioni previste dal capitolato, come l'accensione e lo spegnimento dei lampioni di un'area in modalità manuale o automatica, e per avere un effettivo riscontro sullo stato di queste operazioni, accedendo ad un qualsiasi indirizzo di un lampione è possibile visualizzarne lo stato aggiornato. Tale stato è riportato anche dall'interfaccia grafica. I sensori vengono invece comandati tramite l'utilizzo di un'API tester ed eseguendo un'operazione di tipo POST è possibile modificarne lo stato, comandando ad esempio un rilevamento di un utente stradale.

3.4 Persistenza dei dati

Per realizzare la persistenza dei dati, è stato utilizzato un DB relazionale, fornito da HeidiSQL. L'immagine seguente riporta lo schema Entity-Relationship della base di dati, dopo la ristrutturazione.



Per poter usare efficacemente i dati salvati nel DB, il backend dell'applicazione utilizza un'apposita classe model, che tramite l'uso di Sequelize permette di creare degli oggetti di tipo lampione, area e sensore.

3.5 Autenticazione

Il sistema di autenticazione permette di accedere al sistema e alle Routes protette, accessibili solamente dall'amministratore. L'autenticazione avviene tramite il check delle credenziali salvate nel database per ogni amministratore. Dopo aver appurato che i dati inseriti siano corretti, viene generato un token di tipo JWT, a durata predeterminata. Alla scadenza del tempo prefissato tale token viene rinnovato, altrimenti se si effettua il logout, questo viene cancellato e all'accesso successivo è necessario fornire nuovamente le credenziali per l'accesso.

4 Requisiti soddisfatti

4.1 Tabella requisiti soddisfatti

| Requisito | Descrizione | Classificazione | Stato |
|-----------|---|-----------------|-------------|
| RF1 | L'utente deve poter fare il login al sistema | Obbligatorio | Soddisfatto |
| RF2 | L'utente visualizza lo stato del sistema | Obbligatorio | Soddisfatto |
| RF3 | L'utente deve poter aumentare la luminosità di un'area | Obbligatorio | Soddisfatto |
| RF4 | Il sistema deve visualizzare un messaggio d'errore se non si è potuto aumentare la luminosità | Obbligatorio | Soddisfatto |
| RF5 | L'utente deve poter vedere l'elenco delle aree illuminate | Obbligatorio | Soddisfatto |
| RF6 | L'utente deve poter vedere l'elenco delle aree | Obbligatorio | Soddisfatto |
| RF7 | L'utente deve poter selezionare le aree su cui operare | Obbligatorio | Soddisfatto |
| RF8 | L'utente deve poter diminuire la luminosità di un'area | Obbligatorio | Soddisfatto |
| RF10 | L'utente deve poter accedere alla dashboard | Obbligatorio | Soddisfatto |
| RF11 | Il sistema deve visualizzare un messaggio d'errore nel caso l'operazione di diminuzione della luminosità non fosse andata a buon fine | Obbligatorio | Soddisfatto |
| RF12 | L'utente deve poter diminuire la luminosità | Obbligatorio | Soddisfatto |

| Requisito | Descrizione | Classificazione | Stato |
|-----------|---|-----------------|-------------|
| RF13 | L'utente deve poter inserire una nuova area illuminata | Obbligatorio | Soddisfatto |
| RF14 | L'utente deve poter rimuovere un area illuminata | Obbligatorio | Soddisfatto |
| RF15 | L'utente deve poter accedere alla lista delle aree gestite | Obbligatorio | Soddisfatto |
| RF16 | L'utente deve poter modificare le informazioni di un'area illuminata | Obbligatorio | Soddisfatto |
| RF17 | Il sistema mostra un messaggio di notifica una volta effettuata la modifica ad un area illuminata | Obbligatorio | Soddisfatto |
| RF18 | L'utente deve poter inserire un nuovo sensore in una area illuminata | Obbligatorio | Soddisfatto |
| RF19 | L'utente deve poter accedere all'area illuminata | Obbligatorio | Soddisfatto |
| RF20 | L'utente deve poter rimuovere un sensore da un'area illuminata | Obbligatorio | Soddisfatto |
| RF21 | L'utente deve poter fare il logout dal sistema | Obbligatorio | Soddisfatto |
| RF22 | L'utente deve poter inserire un impianto nell'elenco dei guasti | Obbligatorio | Soddisfatto |
| RF23 | L'utente deve poter rimuovere un impianto dall'elenco dei guasti | Obbligatorio | Soddisfatto |
| RF24 | L'utente deve poter visualizzare i dettagli di un'area | Obbligatorio | Soddisfatto |
| RF25 | L'utente deve poter selezionare un lampione | Obbligatorio | Soddisfatto |
| RF26 | L'utente deve poter visualizzare i dettagli di un lampione | Obbligatorio | Soddisfatto |
| RF27 | L'utente deve poter inserire un nuovo lampione all'interno di un'area illuminata | Obbligatorio | Soddisfatto |
| RF28 | L'utente deve poter rimuovere un lampione all'interno di un'area illuminata | Obbligatorio | Soddisfatto |

| Requisito | Descrizione | Classificazione | Stato |
|-----------|---|-----------------|-----------------|
| RF29 | L'utente deve poter visualizzare l'elenco delle aree illuminate con dei malfunzionamenti | Obbligatorio | Soddisfatto |
| RF30 | L'amministratore deve poter aprire una nuova segnalazione di un guasto tramite un ticket | Obbligatorio | Soddisfatto |
| RF31 | L'amministratore deve poter chiudere il ticket dopo aver fatto la dovuta manutenzione | Obbligatorio | Soddisfatto |
| RF32 | Il manutentore deve poter visualizzare i dettagli aggiuntivi di un guasto forniti dal ticket | Desiderabile | Soddisfatto |
| RF33 | L'utente non amministratore riceve le credenziali da amministratore da un superamministratore | Desiderabile | Non Soddisfatto |
| RF34 | L'utente consulta il manuale Lumos Minima | Desiderabile | Soddisfatto |
| RF35 | Le nuove aree illuminate appena inserite hanno un setup standard | Desiderabile | Soddisfatto |

Numero di requisiti obbligatori soddisfatti: 30/30

Numero di requisiti desiderabili soddisfatti: 3/4

4.2 Qualità

| Requisito | Descrizione | Classificazione | Stato |
|-----------|---|-----------------|-----------------|
| RQ1 | La webapp deve essere sviluppata seguendo le regole descritte nel documento Norme di progetto | Obbligatorio | Soddisfatto |
| RQ2 | Devono essere sviluppati dei test con una copertura minima dell'80% e correlati di report | Obbligatorio | Soddisfatto |
| RQ3 | Deve essere prodotto un documento sulle scelte implementative e progettuali | Obbligatorio | Soddisfatto |
| RQ4 | Deve essere prodotto un documento sui problemi aperti e sulle eventuali soluzioni da esplorare | Obbligatorio | Non soddisfatto |
| RQ5 | Fornire un'analisi rispetto al carico massimo supportato in numero di dispositivi e di quale sarebbe il servizio cloud più adatto per supportarlo analizzando prezzo, stabilità del servizio ed assistenza. | Facoltativo | Non soddisfatto |

Numero di requisiti qualitativi obbligatori soddisfatti: 3/4.

Numero di requisiti qualitativi facoltativi soddisfatti: 0/1.

Il RQ4 non è stato completato poichè non sono state rilevate particolari criticità, come confermato da Imola Informatica.

4.3 Dati copertura test

La piattaforma utilizzata per il testing è Jest, ed è stata utilizzata sia per i test di unità che per i test di integrazione, concordando con il committente una percentuale minima di copertura dell'80%.

Tali dati sono riproducibili eseguendo il comando "npm test" sia su frontend che su backend. I valori forniti sono le percentuali medie riscontrate, visibili nella prima riga delle percentuali del report fornito da jest.

4.3.1 Percentuali test

Dopo aver completato un'accurata fase di testing, i risultati sono i seguenti:

4.3.2 Test Unità

- Statement Coverage: 87%
- Branch Coverage: 81%
- Function Coverage: 95%
- Line Coverage: 87%

4.3.3 Test Integrazione

- Statement Coverage: 99%
- Branch Coverage: 94%
- Function Coverage: 92%
- Line Coverage: 99%