Università del Salento

Dipartimento di Ingegneria dell'Innovazione

Corso di Laurea Magistrale in Ingegneria Informatica

DOCUMENTAZIONE DI PROGETTO

Progettazione di Architetture e Servizi

Realizzazione di un sistema di Waste Management per una Smart City

Docenti

Prof. L. Mainetti

Prof. R. Vergallo

Autore

Culcea Cezar Narcis

Matricola n° 20086225

ANNO ACCADEMICO 2023/2024

Indice

Capitolo 1

Introduzione

Il progetto richiede la realizzazione di un sistema di **smaltimento rifiuti intelligente** da installarsi all'interno di una Smart City. Il progetto si può dividere in due parti:

- *Lato azienda rifiuti*: bisogna creare un sistema in grado di ricevere informazioni dai cassonetti smart e permetterne la gestione da parte degli operatori ecologici. Il sistema deve essere inoltre in grado di raccogliere statistiche.
- *Lato comune*: bisogna creare un sistema in grado di permettere ai cittadini di registrarsi, controllare le proprie statistiche e pagare le tasse emesse dal comune. Allo stesso tempo il sistema deve consentire l'accesso al persona autorizzato al fine di seguire i cittadini ed erogare le tasse.

Si prevede la divisione in microservizi dei sistemi e l'utilizzo dei framework **Spring** (lato backend) ed **Angular** (lato frontend). Al termine del progetto si fornirà un prodotto interamente **dockerizzato** e deployato sui server **AWS**.

1.1 Link alle Repositories

Progetto SmartCity Waste Management - https://gitlab.com/pas_smartcitywastemanagement

CityHall Frontend - https://gitlab.com/pas_smartcitywastemanagement/cityhallfe

CityHall Backend - https://gitlab.com/pas_smartcitywastemanagement/CityHallBE

WasteAgency Frontend - https://gitlab.com/pas_smartcitywastemanagement/wastedisposalagencyfe

WasteAgency Backend - https://gitlab.com/pas_smartcitywastemanagement/wastedisposalagencybe

Login Backend - https://gitlab.com/pas_smartcitywastemanagement/LoginBE

Tax Backend - https://gitlab.com/pas_smartcitywastemanagement/TaxBE

Bin Simulato - https://gitlab.com/pas_smartcitywastemanagement/bin

Consumi Energetici - https://gitlab.com/pas_smartcitywastemanagement/powerconsumption

Capitolo 2

Analisi dei requisiti

2.1 Diagrammi UML

Secondo l'analisi dei requisiti effettuata, sono stati identificati i seguenti attori:

- cittadino non registrato
- utente del comune (cittadino registrato)
- admin del comune
- operatore dell'azienda rifiuti
- cassonetto

Sono emerse inoltre tre zone logiche differenti:

- comune
- azienda rifiuti
- cassonetto

Per ogni attore sono stati rappresentati i casi d'uso implementati e la relativa porzione di progetto che se ne occuperà.

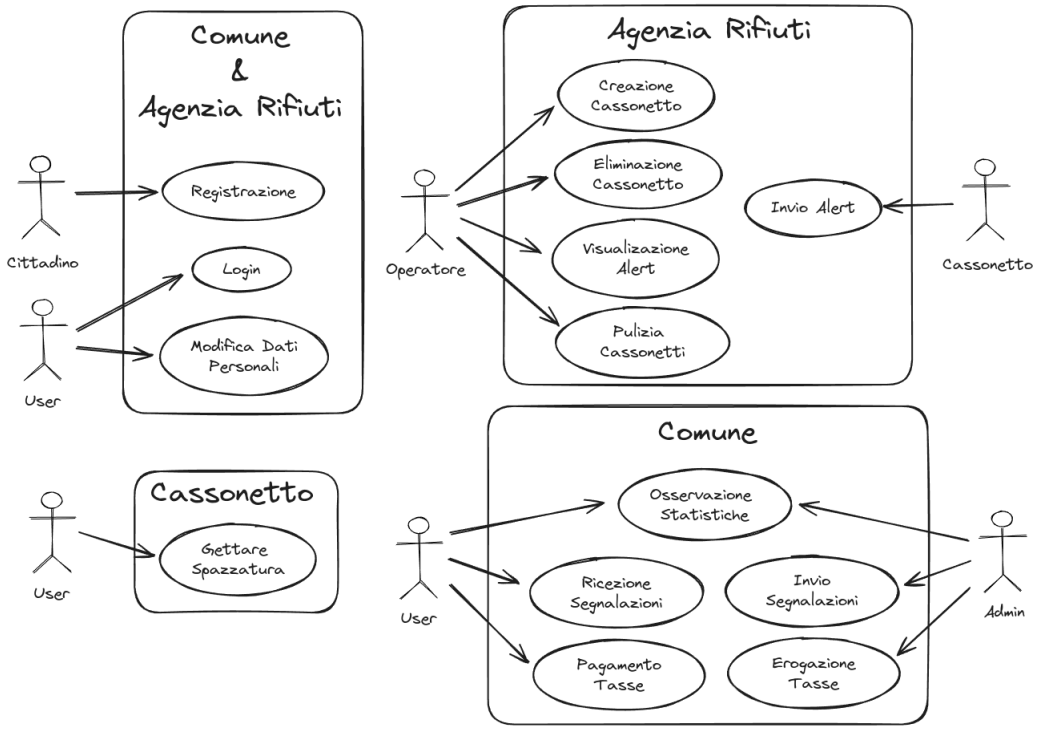

Figura 2.1: Schema UML dei casi d'uso

2.2 Casi d'Uso

2.2.1 Generali

Registrazione

Attori: Cittadino, Frontend, Servizio Login

Pretest: Cittadino non registrato, Sistemi online

1. Il cittadino visita la sezione Login del Frontend e compila i campi
2. Il frontend verifica i campi ed invia la richiesta di registrazione al servizio login
3. Il servizio login controlla i dati
4. Il servizio login memorizza l'utente ed invia la conferma al frontend
5. Il frontend invia una richiesta di login al servizio login
6. Il servizio login verifica i dati
7. Il servizio login invia i dati utente ed un JWT al frontend

8. Il frontend salva i dati nella sessione

9. Il frontend comunica al cittadino l'esito della registrazione

Login

Attori: Cittadino, Frontend, Servizio Login

Pretest: Cittadino registrato, Sistemi online

1. Il cittadino visita la sezione Login e compila i campi

2. Il frontend verifica i dati

3. Il frontend invia la richiesta di login al servizio login

4. Il servizio login verifica i dati

5. Il servizio login invia i dati utente ed un JWT associato al frontend

6. Il frontend salva le informazioni nello storage session.

7. Il frontend comunica al cittadino l'esito del login

Modifica dati personali

Attori: Utente, Interfaccia Grafica, Backend

Pretest: Utente autenticato, Sistemi online

1. L'utente visita la sezione profilo nel frontend

2. L'utente cambia i valori dei campi e conferma

3. Il frontend manda una richiesta al backend con annesso il JWT

4. Il backend autentica l'utente

5. Il backend aggiorna i dati con successo

6. Il frontend chiede i nuovi dati

2.2.2 Azienda rifiuti

Creazione cassonetto

Attori: Operatore, Frontend azienda Rifiuti, Servizio azienda Rifiuti
Pretest: Operatore autenticato, Sistemi online

1. L'operatore visita la sezione gestione cassonetti
2. L'operatore apre il menù aggiunta cassonetto
3. L'operatore seleziona sulla mappa un nuovo punto e conferma
4. Il frontend invia una richiesta di creazione cestino al servizio azienda rifiuti con il JWT
5. Il servizio azienda rifiuti autentica l'operatore e crea il nuovo cestino

Eliminazione cassonetto

Attori: Operatore, Frontend azienda Rifiuti, Servizio azienda Rifiuti
Pretest: Operatore autenticato, Sistemi online

1. L'operatore visita la sezione gestione cassonetti
2. L'operatore apre il menù eliminazione cassonetto
3. L'operatore seleziona sulla mappa un cestino
4. Il frontend invia una richiesta di eliminazione cestino al servizio azienda rifiuti con il JWT
5. Il servizio azienda rifiuti autentica l'operatore ed elimina il cestino

Gettare la spazzatura

Attori: Cittadino, Cassonetto simulato, Servizio azienda Rifiuti, Broker RabbitMQ
Pretest: Cittadino in possesso dell'ID, Sistemi online

1. Il cittadino inserisce l'id nel software di simulazione
2. Il cassonetto si assicura che l'id sia valido localmente

3. Il cittadino seleziona il cassonetto e la quantità di spazzatura differenziata e non da gettare
4. Il cassonetto invia la notifica al Broker MQTT
5. Il broker invia la notifica all'azienda Rifiuti

Invio Alert

Attori: Cassonetto, Broker RabbitMQ, Servizio azienda Rifiuti
Pretest: Sistemi online

1. Il cassonetto ha superato una soglia di capienza
2. Il cassonetto invia l'alert al broker Rabbit
3. Il broker Rabbit invia l'alert al servizio azienda rifiuti
4. Il servizio azienda rifiuti memorizza l'alert

Visualizzazione alert

Attori: Operatore, Frontend azienda Rifiuti, Servizio azienda Rifiuti
Pretest: Operatore autenticato, Sistemi online

1. L'operatore visita la sezione alerts
2. Il frontend richiede al servizio azienda rifiuti la lista alert
3. Il servizio azienda rifiuti invia la lista alert al frontend
4. Il frontend espone i dati in forma tabellare

Pulizia dei cassonetti

Attori: Operatore, Frontend azienda Rifiuti, Servizio azienda Rifiuti
Pretest: Operatore autenticato, Sistemi online

1. L'operatore visita la sezione dashboard
2. L'operatore avvia la procedura di pulizia bidoni
3. Il frontend raccoglie le coordinate relative ai bidoni pieni

4. Il frontend invia le coordinare ad un servizio di routing esterno
5. Il servizio di routing esterno invia al frontend il percorso ottimale per visitare i bidoni
6. Il frontend visualizza la lista bidoni ed il percorso ottimale
7. L'operatore consegna la lista al camion rifiuti e conferma la pulizia dei cassonetti

2.2.3 Comune

Osservazione statistiche

Attori: Utente, Frontend Comune, Servizio azienda Rifiuti, Servizio Tasse
Pretest: Utente autenticato, Sistemi online

1. L'utente visita la sezione dashboard sul frontend
2. Il frontend chiede le statistiche spazzatura al servizio azienda rifiuti
3. Il servizio azienda rifiuti risponde alla richiesta del frontend
4. Il frontend chiede le statistiche pagamenti al servizio tasse
5. Il servizio tasse risponde alla richiesta del frontend
6. Il frontend aggiorna i grafici

Erogazione Tasse

Attori: Admin, Frontend Comune, Servizio comune, Servizio azienda Rifiuti, Servizio Tasse
Pretest: Admin autenticato, Sistemi online

1. L'admin visita la sezione erogazione tasse
2. L'admin seleziona l'anno per le erogazioni delle tasse
3. Il frontend recupera la lista degli utenti dal servizio comune
4. Il servizio comune invia la lista utenti al frontend
5. Il frontend invia la lista utenti al servizio azienda rifiuti
6. Il servizio azienda rifiuti invia lo storico della spazzatura al frontend
7. Il frontend invia lo storico della spazzatura al servizio tasse

8. Il servizio tasse verifica se ci sono quantità di spazzatura nuove da pagare per ogni utente ed eroga dei ticket

9. Il servizio tasse invia la lista dei ticket erogati al frontend

Pagamento Tasse

Attori: Utente, Frontend Comune, Servizio Tasse
Pretest: Utente autenticato, Sistemi online

1. L'utente visita la sezione pagamenti

2. Il frontend chiede i pagamenti al servizio tasse per l'utente autenticato

3. Il servizio tasse invia la lista dei ticket

4. Il frontend visualizza la lista ticket

5. L'utente seleziona il tasto "Paga" del ticket che è intenzionato a pagare

6. Il frontend invia il pagamento al servizio tasse

7. Il servizio tasse conferma il pagamento al frontend

8. Il frontend visualizza il pagamento

Invio segnalazioni

Attori: Admin, Frontend Comune, Servizio azienda Rifiuti, Servizio Comune
Pretest: Admin autenticato, Sistemi online

1. L'admin visita la sezione Warnings

2. Il frontend chiede al servizio comune la lista degli utenti

3. Il servizio comune invia la lista utenti

4. Il frontend invia la lista utenti al servizio azienda rifiuti

5. Il servizio azienda rifiuti invia una lista degli utenti con le quantità di spazzatura differenziata e non, gettata da ogni utente

6. L'admin sceglie l'utente a cui inviare un messaggio

7. Il frontend invia al servizio comune il messaggio

8. Il frontend salva il messaggio

Ricezione segnalazioni

Attori: Utente, Frontend Comune, Servizio comune

Pretest: Utente autenticato, Sistemi online

1. L'utente visita la sezione Warnings

2. Il frontend chiede al servizio comune gli avvisi per l'utente autenticato

3. Il servizio comune invia gli avvisi al frontend

4. Il frontend mostra gli avvisi

2.3 Diagrammi di Sequenza

2.3.1 Generali

Registrazione

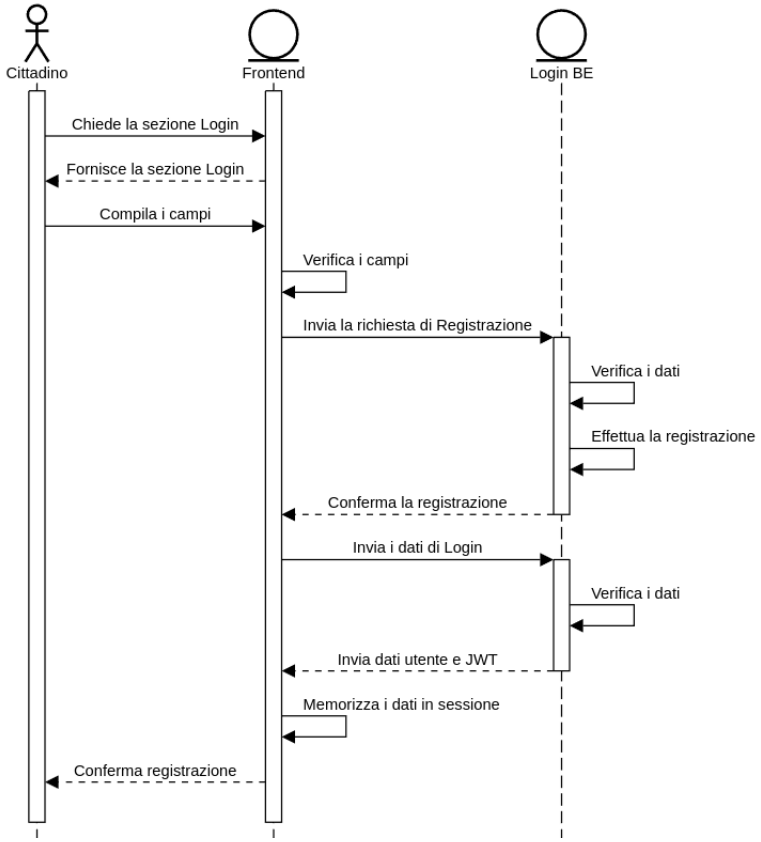

Figura 2.2: Diagramma di sequenza del caso d'uso Registrazione

Login

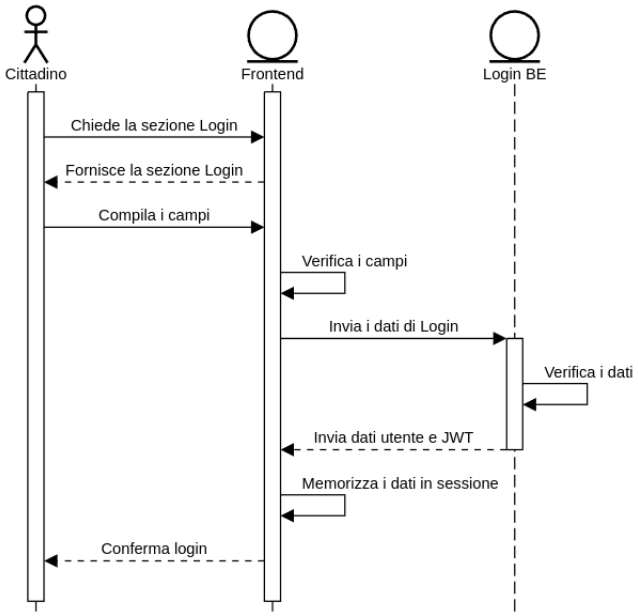

Figura 2.3: Diagramma di sequenza del caso d'uso Login

Modifica dati personali

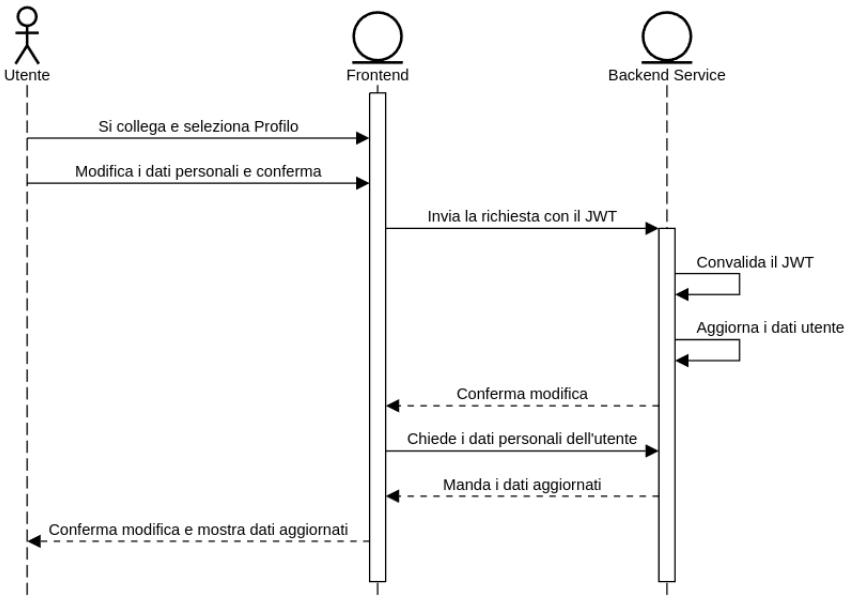

Figura 2.4: Diagramma di sequenza del caso d'uso Modifica dati personali

2.3.2 Azienda Rifiuti

Creazione cassonetto

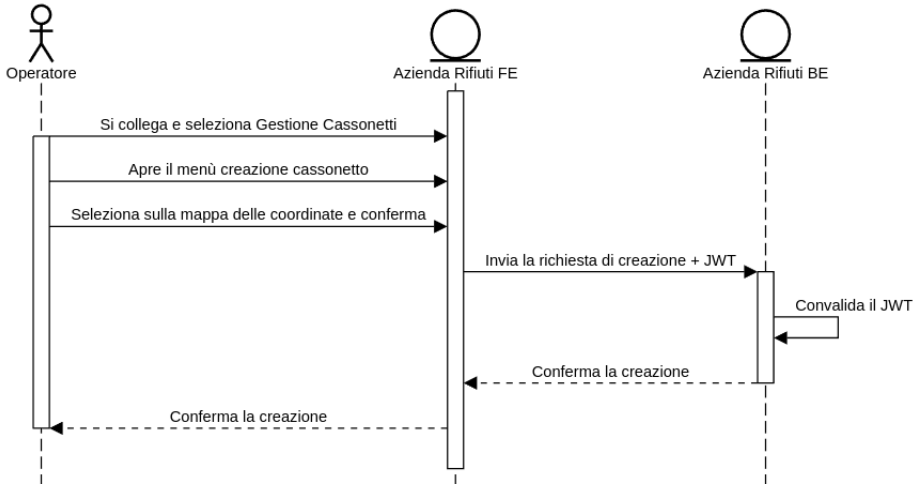

Figura 2.5: Diagramma di sequenza del caso d'uso Creazione cassonetto

Eliminazione cassonetto

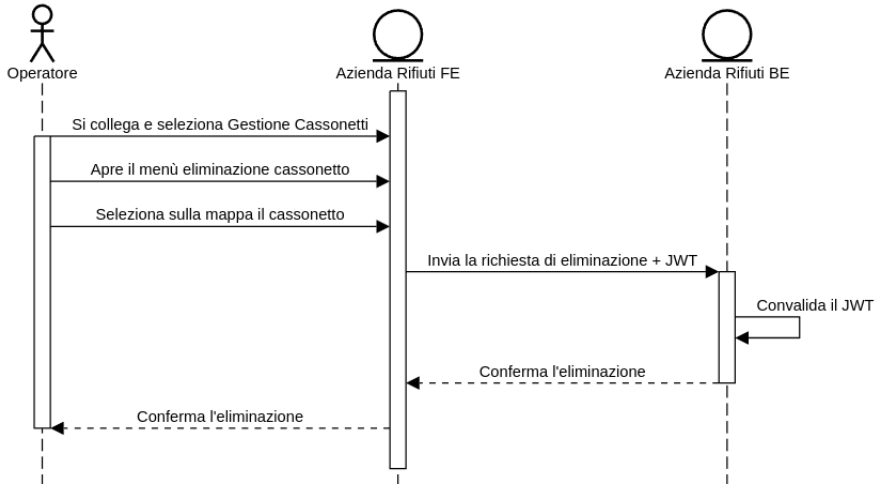

Figura 2.6: Diagramma di sequenza del caso d'uso Eliminazione cassonetto

Gettare la spazzatura

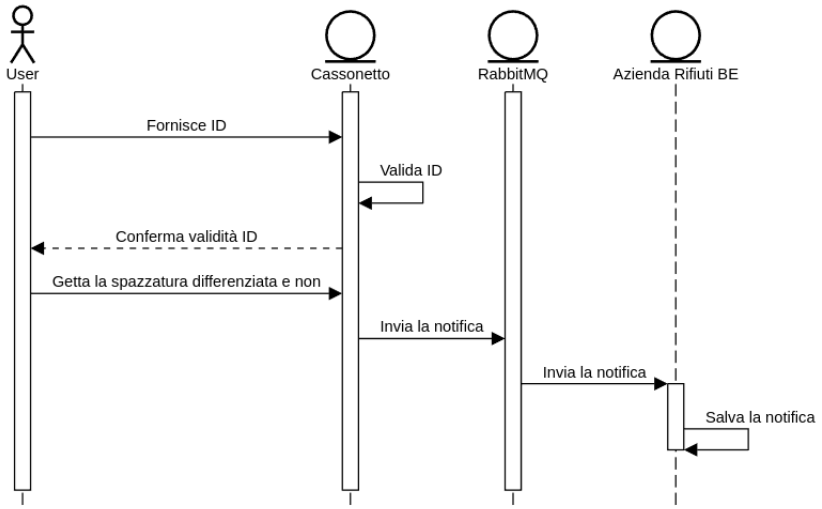

Figura 2.7: Diagramma di sequenza del caso d'uso Gettare la spazzatura

Invio Alert

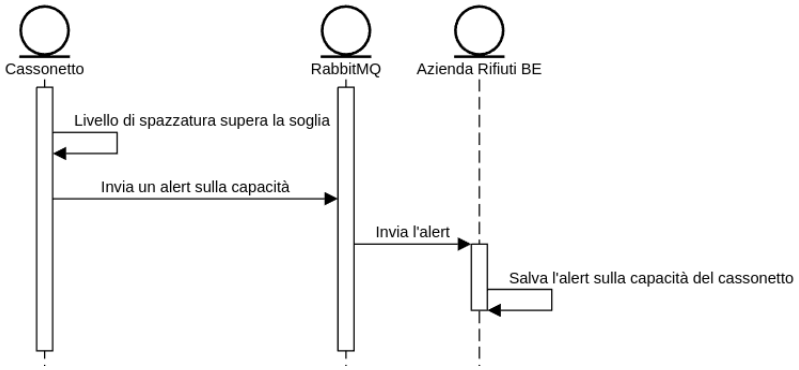

Figura 2.8: Diagramma di sequenza del caso d'uso Invio alert

Visualizzazione Alert

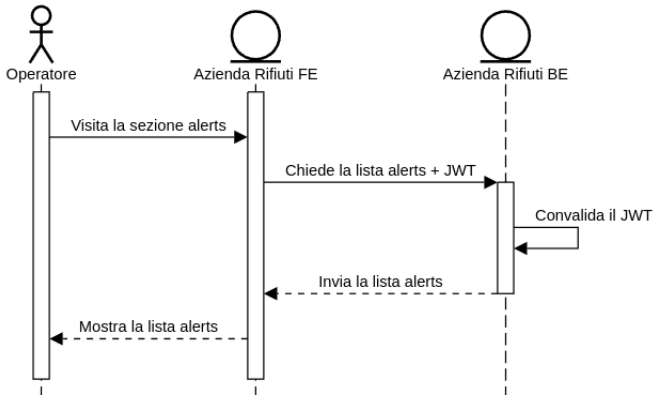

Figura 2.9: Diagramma di sequenza del caso d'uso Visualizzazione alert

Pulizia dei cassonetti

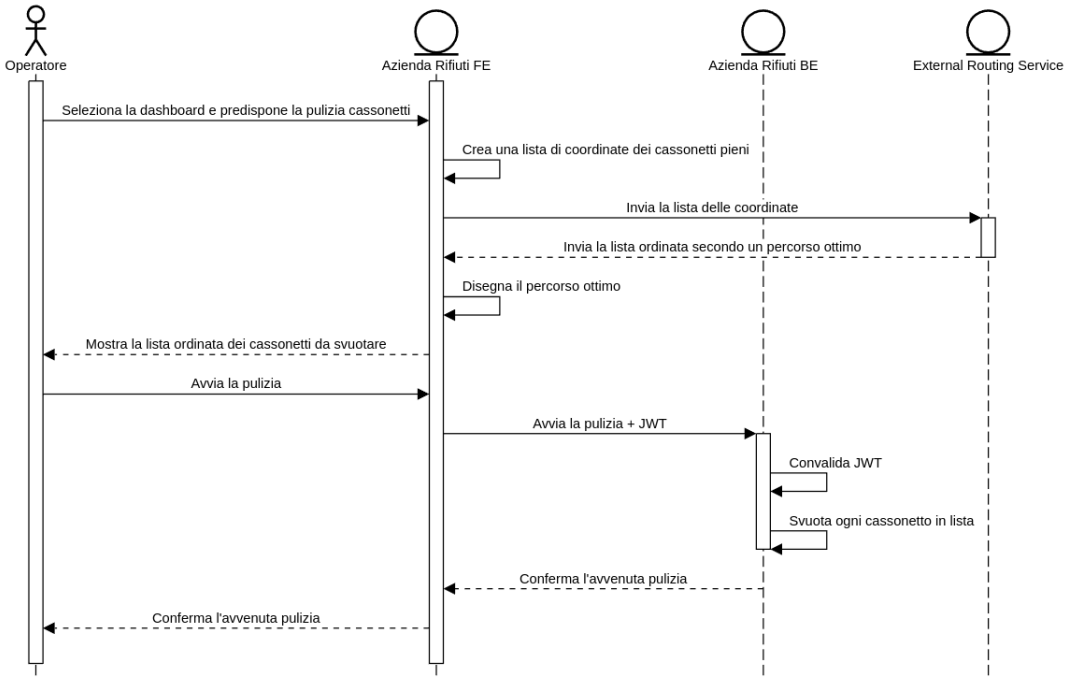

Figura 2.10: Diagramma di sequenza del caso d'uso Pulizia dei Cassonetti

2.3.3 Comune

Osservazione statistiche

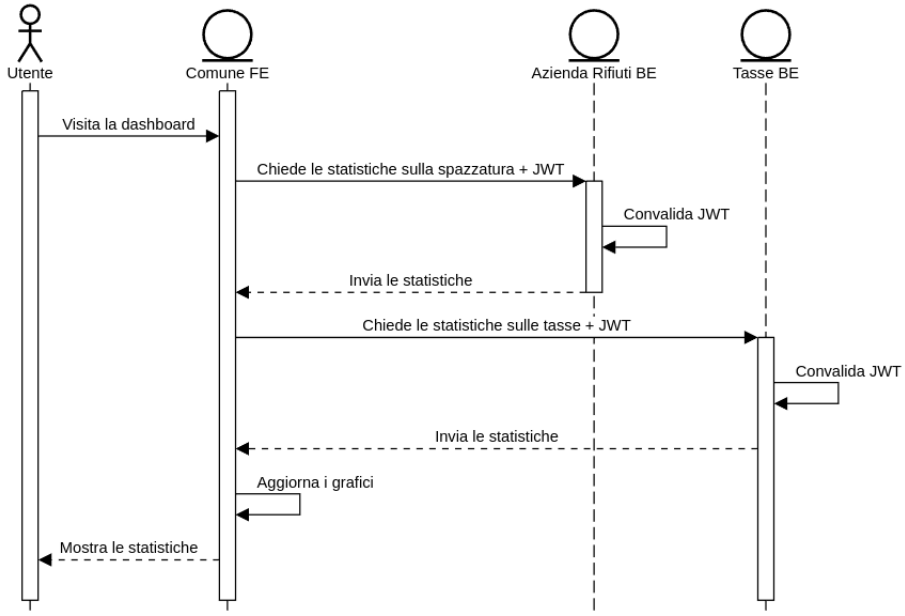

Figura 2.11: Diagramma di sequenza del caso d'uso Osservazione statistiche

Erogazione tasse

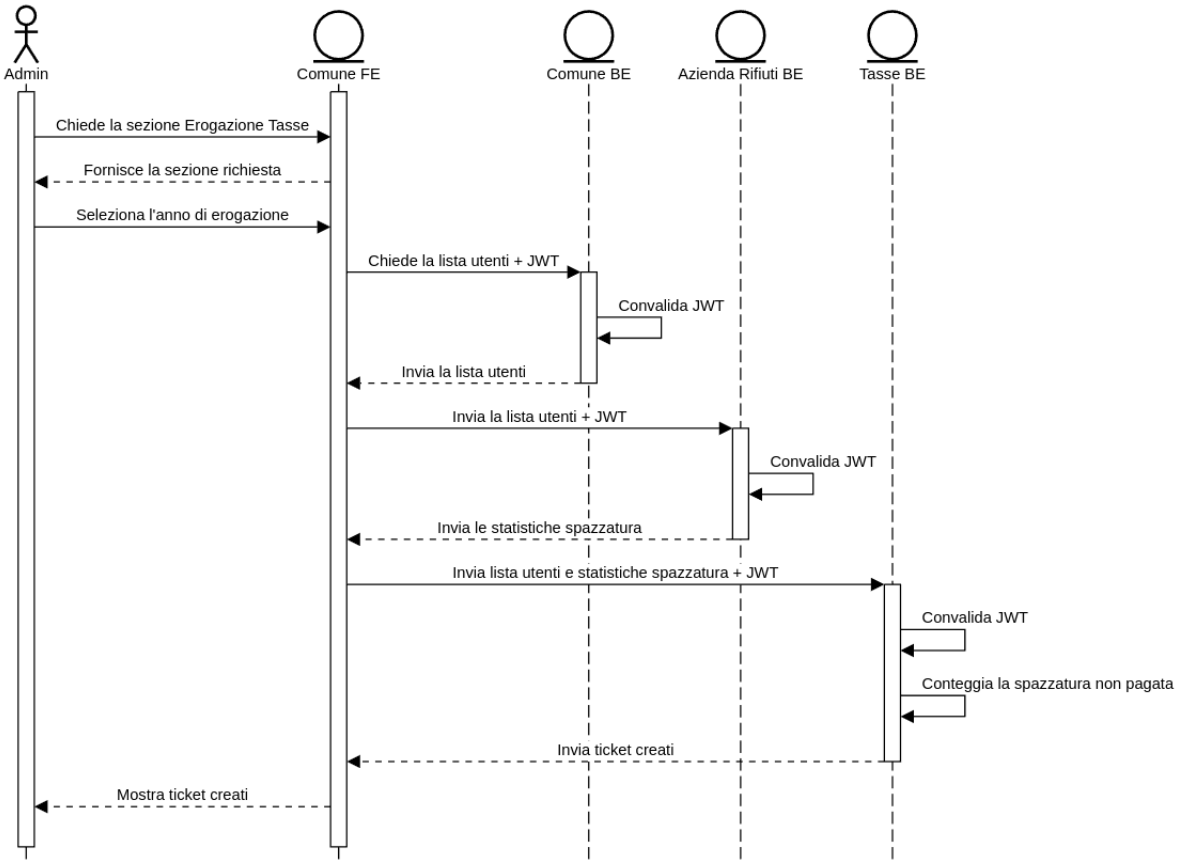

Figura 2.12: Diagramma di sequenza del caso d'uso Erogazione tasse

Pagamento tasse

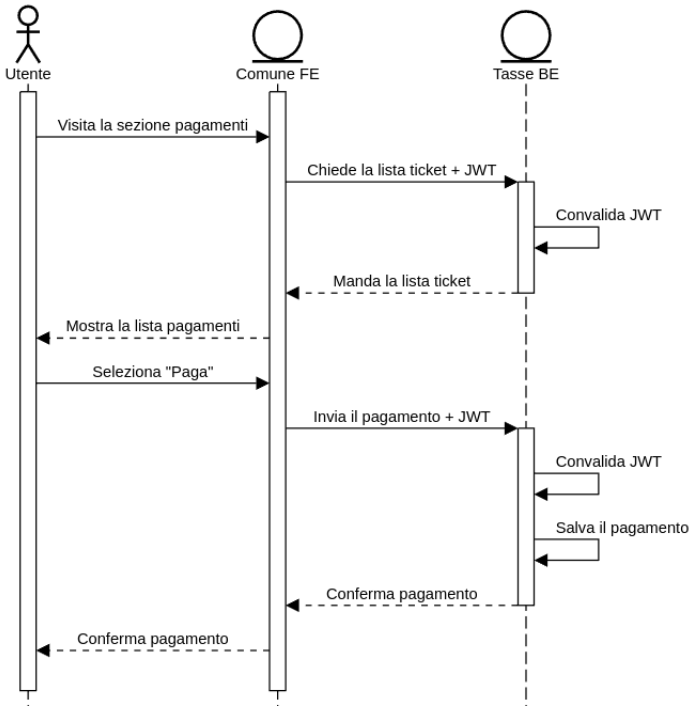

Figura 2.13: Diagramma di sequenza del caso d'uso Pagamento tasse

Invio segnalazione

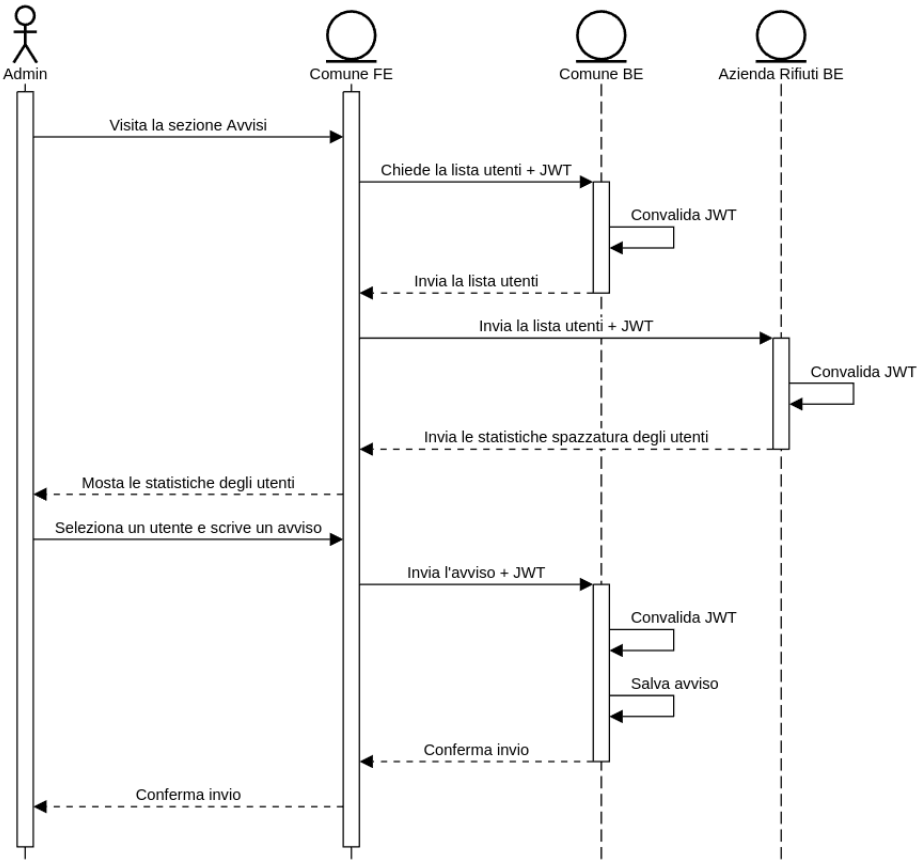

Figura 2.14: Diagramma di sequenza del caso d'uso Invio avviso

Ricezione segnalazione

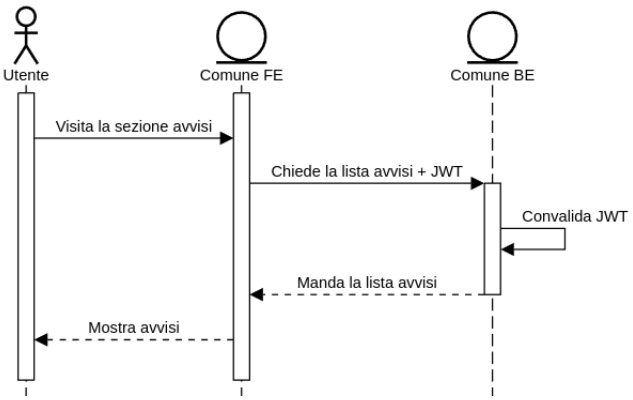

Figura 2.15: Diagramma di sequenza del caso d'uso Ricezione avviso

Capitolo 3

Architetture

3.1 Architettura Logica

L'architettura pensata per il sistema è divisa in vari servizi in base alle macrofunzioni.

Si prevedono due **Web Application** con lo scopo di fornire agli utenti delle interfacce grafiche semplici ed intuitive al fine di permettere un utilizzo adeguato del sistema. Le interfacce grafiche serviranno come punti di accesso principali al sistema azienda Rifiuti ed al sistema Comune.

Si prevede la costruzione di quattro servizi backend:

- *Servizio azienda Rifiuti*: gestione di cassonetti ed operatori ed esposizione statistiche rifiuti
- *Servizio Comune*: gestione dei cittadini e delle segnalazioni
- *Servizio Login*: registrazione ed autenticazione degli utenti
- *Servizio Tasse*: erogazione delle tasse ed esposizione delle statistiche pagamenti

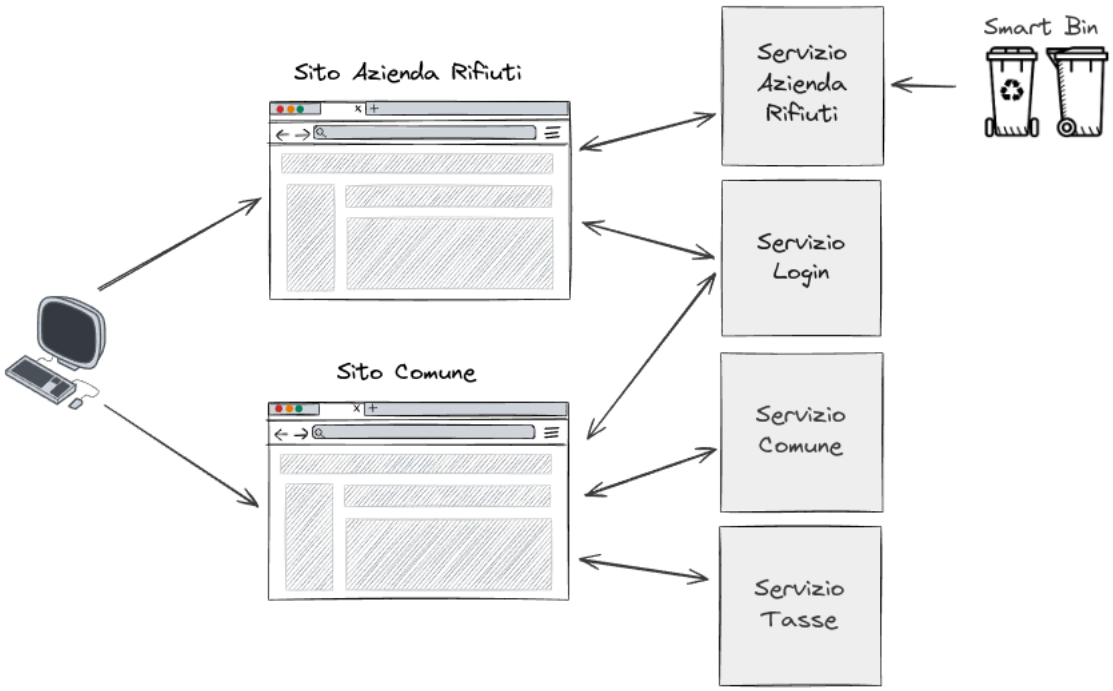

Figura 3.1: Schema dell'Architettura Logica del sistema

3.2 Architettura Fisica

Per componente del sistema è stato realizzato un container **Docker** eseguito su una macchina **EC2**. I componenti di Frontend sono realizzati grazie ai framework **AngularJS** (script per le chiamate API), **ChartJS** (costruzione di grafici real time) e **Bootstrap** (elementi HTML).

Il Frontend comunica con il Backend attraverso il servizio **API Gateway** fornito dalla piattaforma AWS.

I componenti di Backend sono realizzati grazie al framework **SpringBoot** ed ognuno di essi è dotato di un database personale **MongoDB**. A gestire la comunicazione Azienda Rifiuti e Smart Bin è **RabbitMQ** (broker MQTT). Ogni smart bin è simulato da un client Java eseguito da terminale.

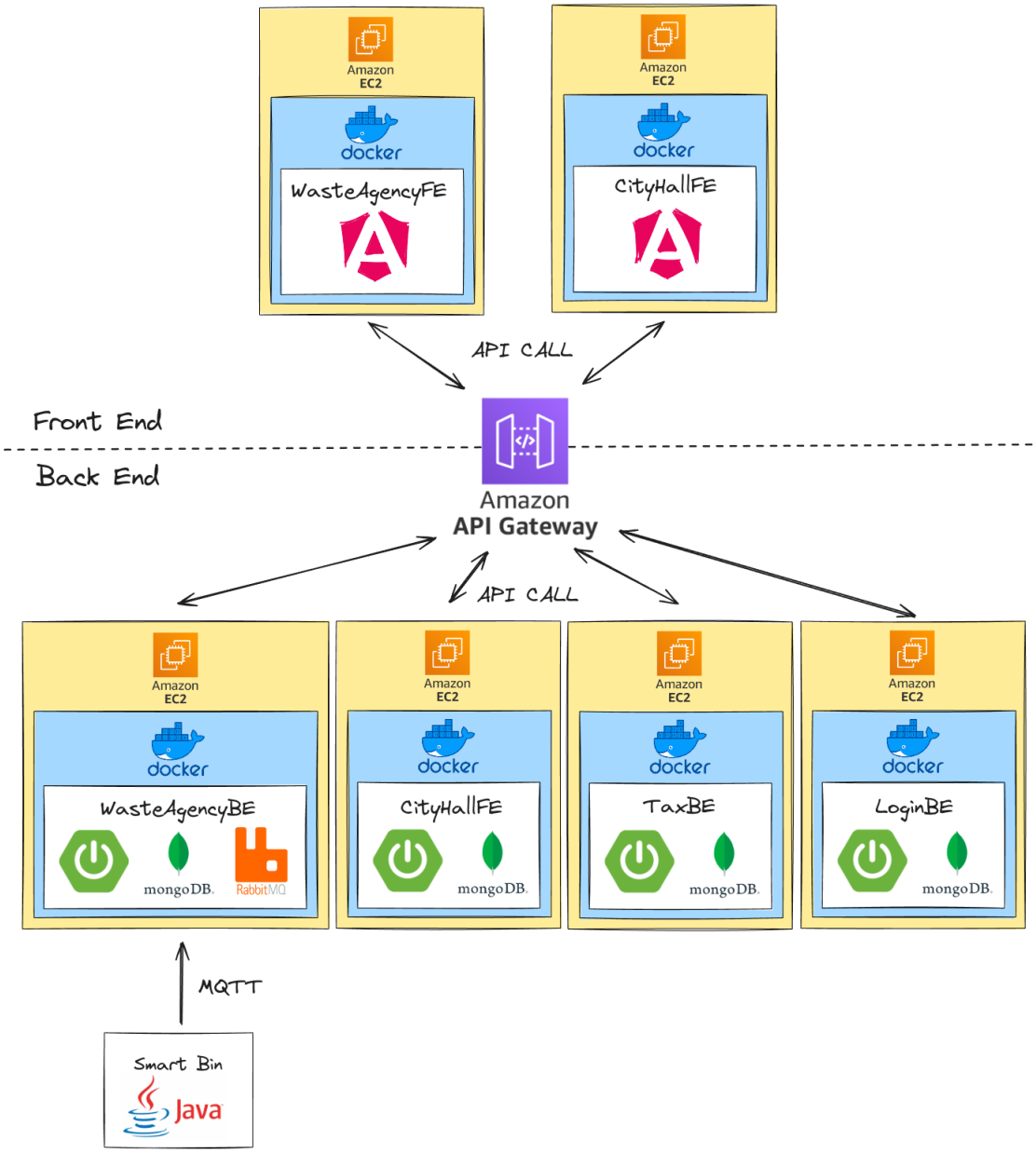

Figura 3.2: Schema dell'Architettura Fisica del sistema

Capitolo 4

Componenti e Microservizi

4.1 Front-End

Il frontend è sviluppato con il framework AngularJS ed eseguito in un container docker su due macchine EC2.

4.1.1 City Hall

Il frontend del comune è accessibile all'indirizzo:
http://ec2-34-192-139-225.compute-1.amazonaws.com
In fase di registrazione è possibile selezionare il ruolo utente (Utente o Admin), in base alla quale cambieranno le funzionalità concesse.

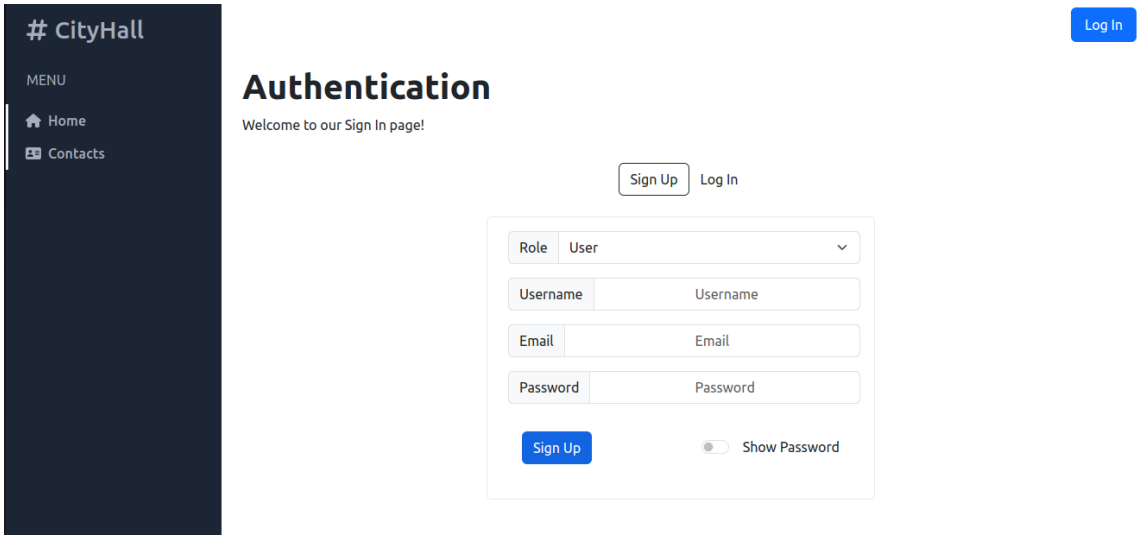

Figura 4.1: Screenshot del Login

Area Utente

La sezione utente predispone diverse aree.

La dashboard consente l'osservazione delle statistiche riguardanti la quantità di spazzatura gettata e le tasse da pagare.

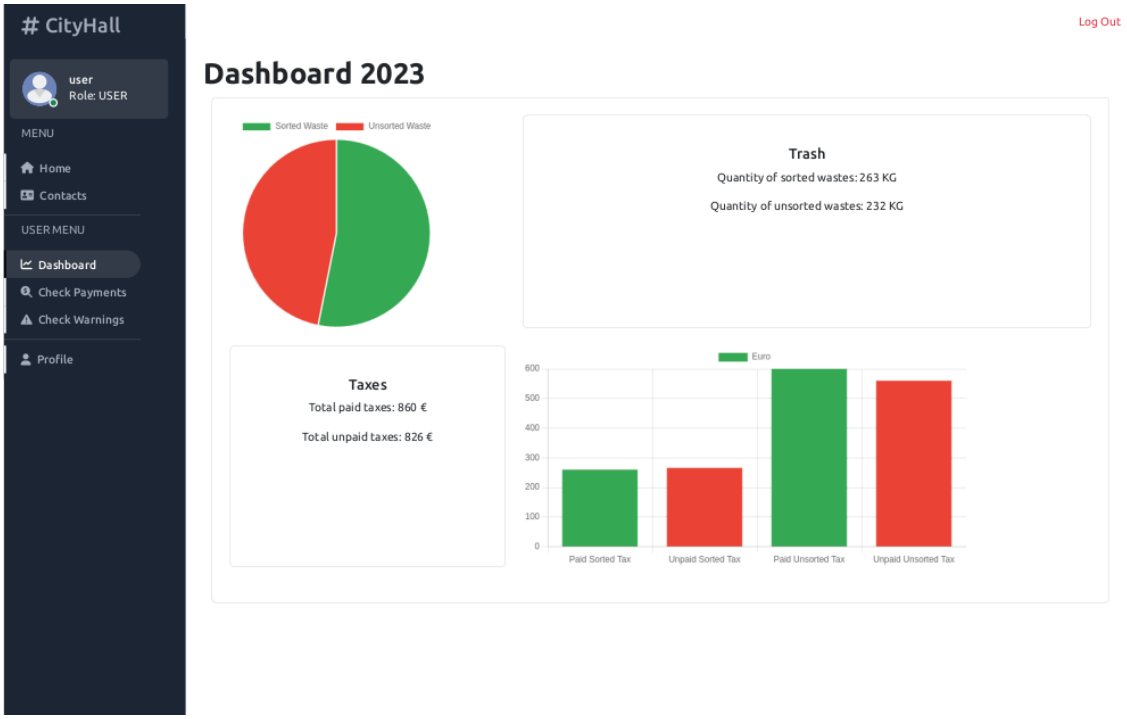

Figura 4.2: Screenshot della Dashboard

La sezione pagamenti consente la visualizzazione delle tasse erogate e ne consente il pagamento.

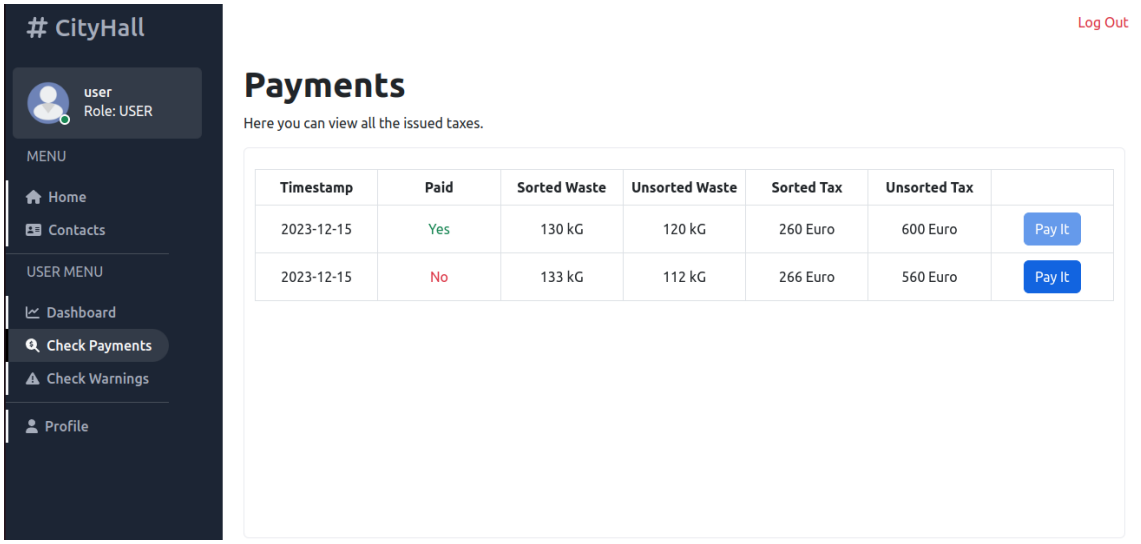

Figura 4.3: Screenshot sezione Pagamenti

La sezione Warning consente la visualizzazione di eventuali avvisi inviati dagli amministratori del comune.

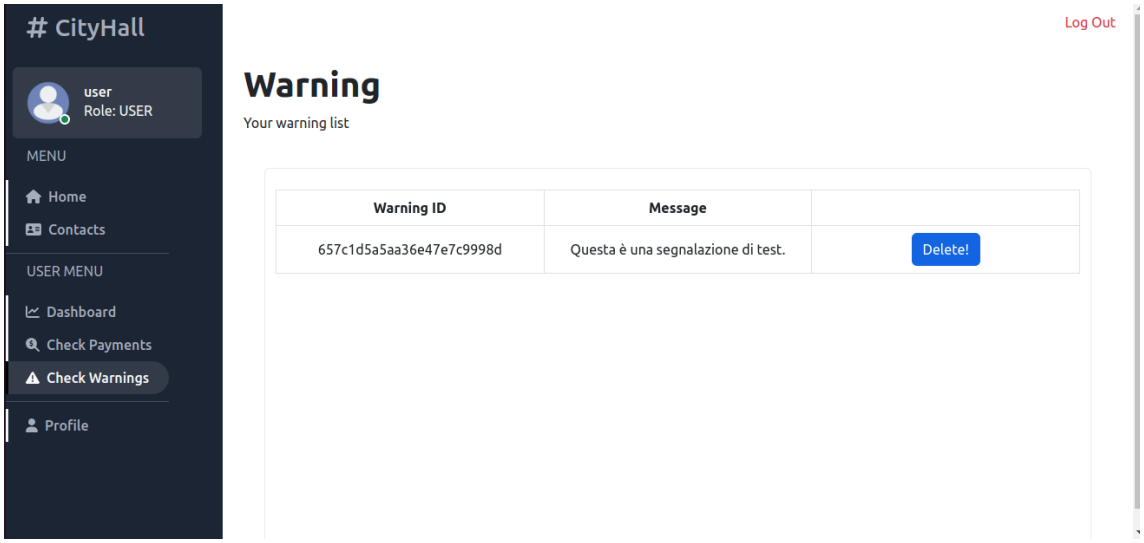

Figura 4.4: Screenshot sezione Warnings

La sezione Profilo consente la modifica dei dati personali. Inoltre mostra il codice QR personale di ogni utente. Questo codice ha scopo identificativo e permette agli utenti registrati di gettare la spazzatura nei cassonetti.

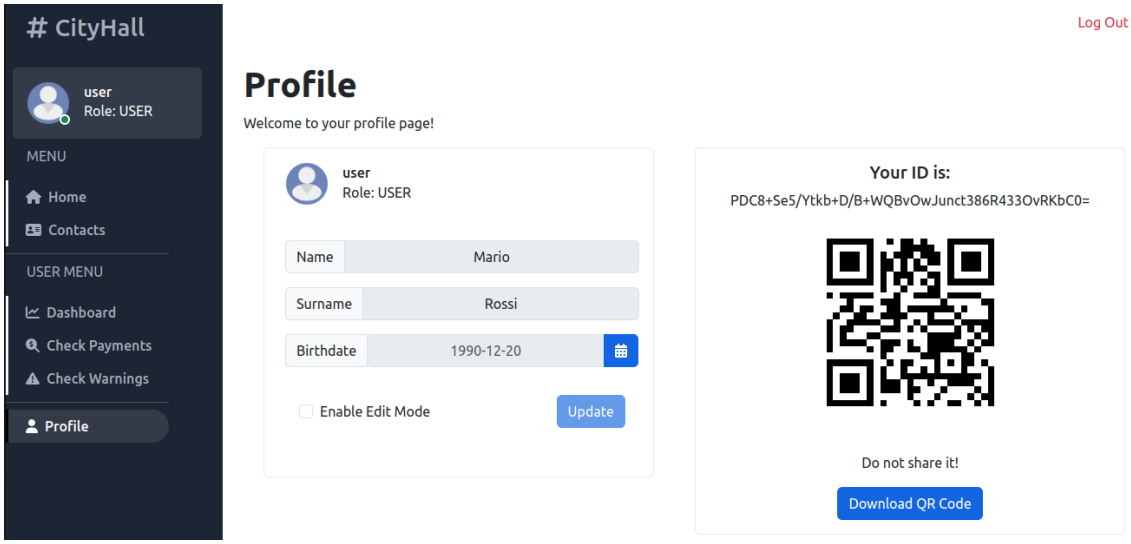

Figura 4.5: Screenshot sezione Profilo

Il QR code contiene un codice formato da due pezzi:

- keyword segreta
- ID utente del database

La stringa viene successivamente criptata con un algoritmo a chiave simmetrica, presente anche localmente sui cassonetti smart. In questo modo, se la stringa decriptata localmente dal cassonetto non contiene la keyword segreta, esso non considererà valido l'ID utente e non consentirà di gettare la spazzatura.

```
1  import * as CryptoJS from 'crypto-js';
2
   You, 4 months ago | 1 author (You)
3  export default class Encrypter {
4      private symmetricSecretKey = "mcMREavkBsiA3gYHxHSlOsvAVdNxjM6O";
5      private secretKeyword = "TANGO";
6
7      constructor() { }
8        You, 4 months ago • creazione funzioni di crittografia
9      encrypt(word: string){
10       var key = CryptoJS.enc.Base64.parse(this.symmetricSecretKey);
11       var srcs = CryptoJS.enc.Utf8.parse(this.secretKeyword + word);
12       var encrypted = CryptoJS.AES.encrypt(srcs, key, {mode:CryptoJS.mode.ECB,padding: CryptoJS.pad.Pkcs7});
13       return encrypted.toString();
14   }
15
16   /**
17       * Decrypt
18    * @param word
19    * @returns {*}
20    */
21   decrypt(word: string){
22       var key = CryptoJS.enc.Base64.parse(this.symmetricSecretKey);
23       var decrypt = CryptoJS.AES.decrypt(word, key, {mode:CryptoJS.mode.ECB, padding: CryptoJS.pad.Pkcs7});
24       return CryptoJS.enc.Utf8.stringify(decrypt).toString();
25   }
26 }
```

Figura 4.6: Snippet di codice che esegue l'encryption

Area Admin

La sezione Dashboard degli Admin è molto simile alla sezione degli Utenti, tuttavia le statistiche sono riferite alla somma delle statistiche di tutti gli utenti.

La sezione Tax Issuing consente all'amministratore di erogare le tasse per l'anno selezionato. Il sistema si occuperà automaticamente di crearle e mostrarle.

La sezione Payments mostra tutte le tasse erogate e lo stato del pagamento.

La sezione Warnings mostra una lista di utenti e le quantità di spazzatura differenziata e non, gettata da ogni utente. Nel caso in cui l'utente faccia poca raccolta differenziata, il sistema lo evidenzia come Cattivo Riciclatore.

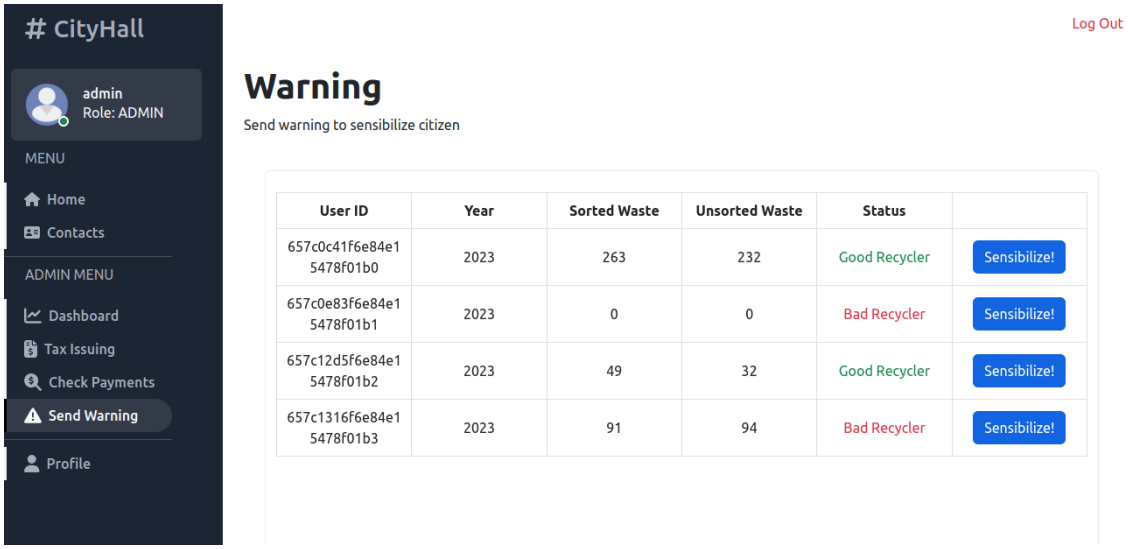

Figura 4.7: Screenshot sezione Warning

4.1.2 Waste Disposal Agency

Il frontend dell'azienda rifiuti è accessibile all'indirizzo:
http://ec2-3-227-223-40.compute-1.amazonaws.com
In fase di registrazione viene assegnato automaticamente il ruolo di Operator. Il sito è dedicato esclusivamente ai dipendenti dell'azienda.

La sezione dashboard consente di visualizzare lo stato di tutti i cassonetti. In base al livello di spazzatura che contengono, i cassonetti hanno icone di colore diverso:

- verde: livello tra 0% e 50%
- giallo: livello tra 50% 75%
- rosso: livello tra 75% e 100%

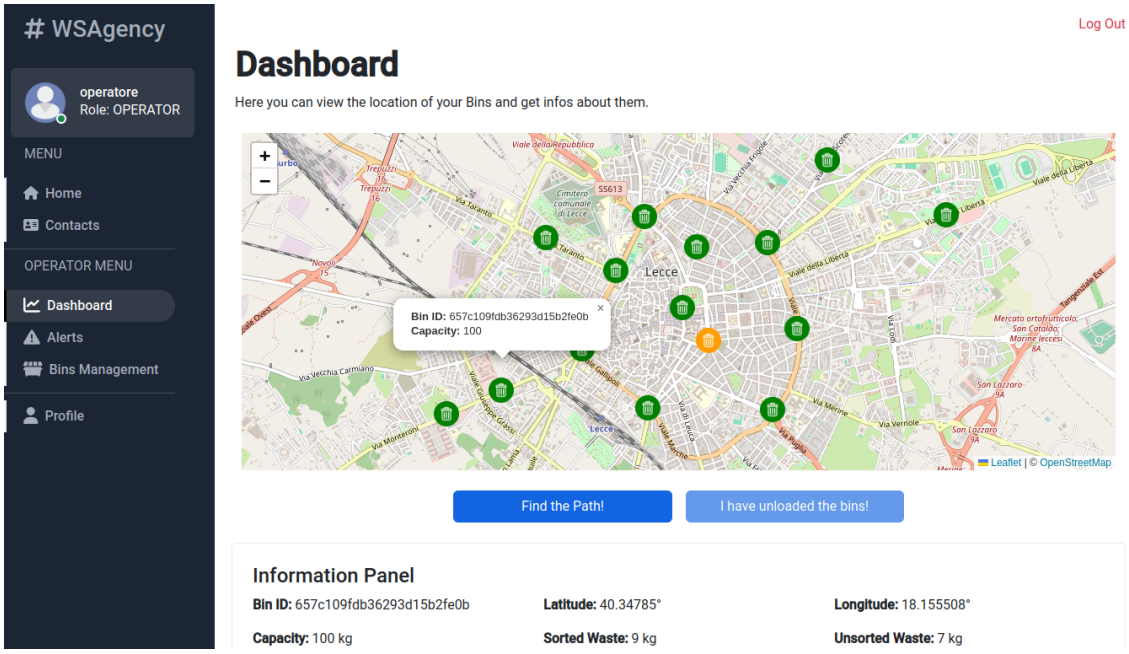

Figura 4.8: Screenshot sezione Dashboard

La sezione dashboard consente anche di istanziare i percorsi di pulizia dei cassonetti. Il sistema seleziona in automatico i cassonetti che contengono oltre il 50% di spazzatura ed invia le coordinate ad un servizio di routing esterno (**Open Route Service** che risponde con una matrice dei costi.

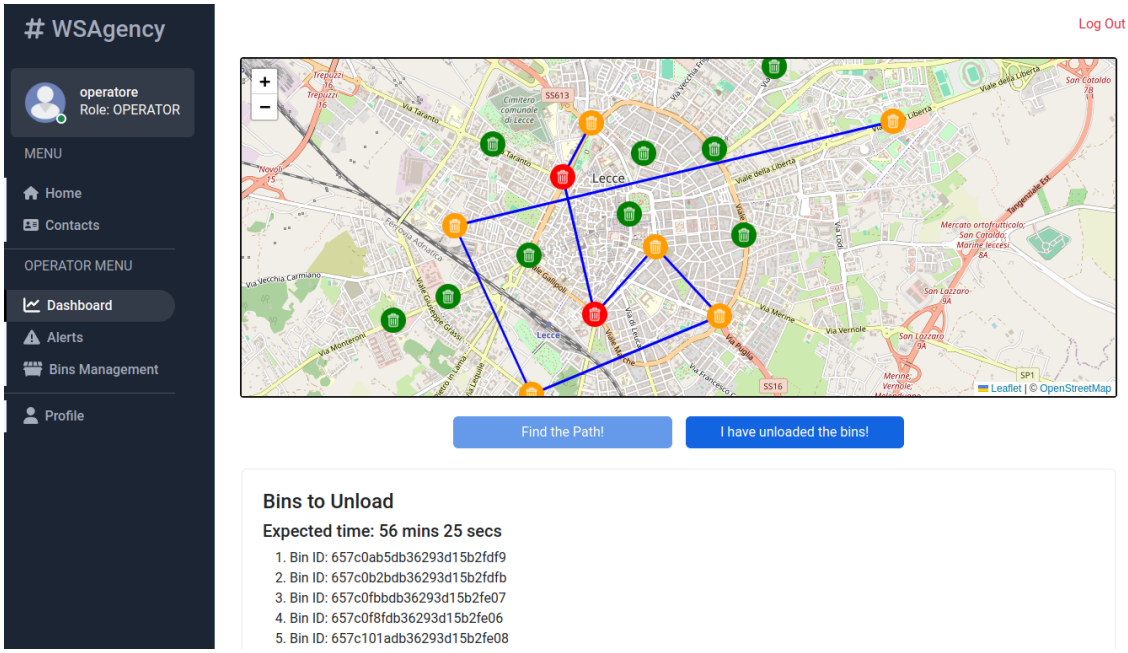

Figura 4.9: Screenshot di un percorso di pulizia

Il sistema a questo punto trova un percorso ottimo usando l'algoritmo Nearest Neighbor e fornisce l'ordine di pulizia dei cassonetti.

```typescript
import { TspSolver } from "./TspSolverBridge";

You, 18 hours ago | 1 author (You)
export default class TspNearestNeighbor implements TspSolver {
  findOptimalSolution(distances: number[][]): any {
    const numLocations = distances.length;
    const visitedNodes: number[] = [];
    let totalDistance = 0;

    // Start from the first location (node 0)
    let currentNode = 0;
    visitedNodes.push(currentNode);

    while (visitedNodes.length < numLocations) {
      let nearestNode = -1;
      let minDistance = Number.MAX_VALUE;

      // Find the nearest unvisited node
      for (let nextNode = 0; nextNode < numLocations; nextNode++) {
        if (!visitedNodes.includes(nextNode) && distances[currentNode][nextNode] < minDistance) {
          nearestNode = nextNode;
          minDistance = distances[currentNode][nextNode];
        }
      }

      if (nearestNode !== -1) {
        visitedNodes.push(nearestNode);
        totalDistance += minDistance;
        currentNode = nearestNode;
      }

    }

    // aggiungi tempo per svuotare ogni cestino
    totalDistance = totalDistance + distances.length * 160;
    return { path: visitedNodes, distance: totalDistance };
  }
}      You, 3 months ago • implementazioni calcolo routing sulla matrice di …
```

Figura 4.10: Snippet dell'implementazione del TSP Nearest Neighbor

La sezione avvisi mostra l'elenco degli alert sulla capienza dei cassonetti.

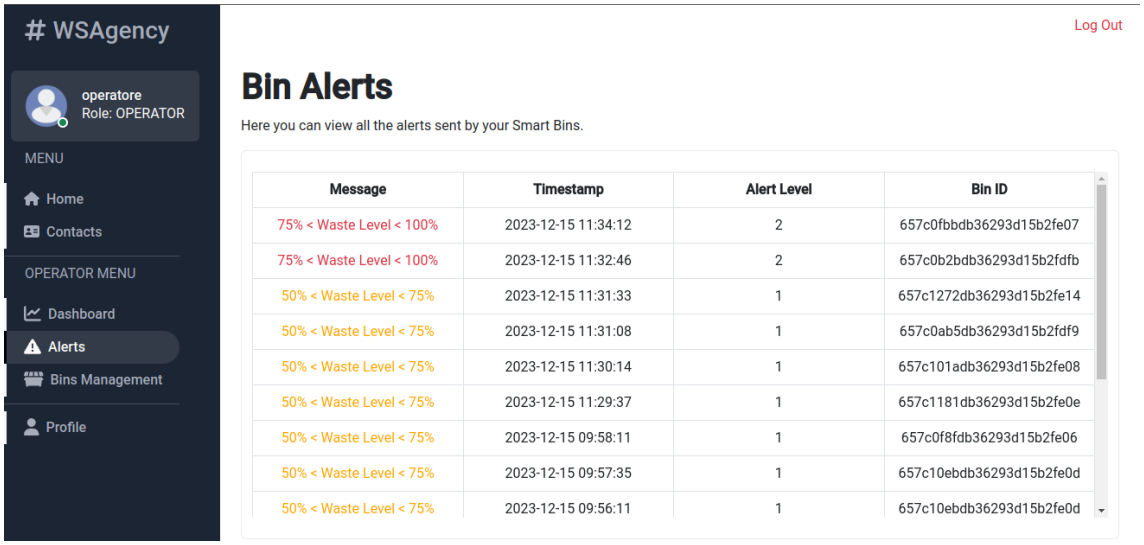

Figura 4.11: Screenshot sezione Alert

4.2 Back-end

4.2.1 City Hall Service

Il backend del comune si occupa di gestire i dati degli utenti del comune e gli avvisi inviati dagli amministratori.

API User

Endpoint	**Descrizione**	**Dettagli**
GET /api/user/exist/{userID}	Verifica se un utente esiste o meno.	Input: userID (ID dell'utente da verificare).
POST /api/user/create	Crea un nuovo utente con i dati forniti.	Input: userDTO (dati dell'utente da creare).
POST /api/user/update	Aggiorna un utente esistente con i dati forniti.	Input: userDTO (nuovi dati dell'utente).
DELETE /api/user/delete/{userID}	Elimina un utente in base all'ID specificato.	Input: userID (ID dell'utente da eliminare).
GET /api/user/get/{userID}	Ottiene i dati di un utente in base all'ID specificato.	Input: userID (ID dell'utente da recuperare).
GET /api/user/id/all	Ottiene una lista di tutti gli ID utente.	Nessun input richiesto.

API Warning

Endpoint	**Descrizione**	**Dettagli**
POST /api/warning/create	Crea un nuovo avviso (warning) con i dati forniti.	Input: warningDTO (dati dell'avviso da creare).
DELETE /api/warning/delete/{warningId}	Elimina un avviso (warning) in base all'ID specificato.	Input: warningId (ID dell'avviso da eliminare).
GET /api/warning/get/user/{userId}	Ottiene tutti gli avvisi (warnings) associati a un utente specifico.	Input: userId (ID dell'utente di cui si desiderano gli avvisi).

4.2.2 Waste Disposal Agency Service

Il backend dell'agenzia rifiuti si occupa di:

- gestire gli operatori
- gestire i cassonetti
- gestire le notifiche e le statistiche della spazzatura
- gestire gli alert sulla capienza
- gestire la configurazione del Broker MQTT che comunica con i cassonetti

API User

Endpoint	**Descrizione**	**Dettagli**
GET /api/user/exist/{userID}	Verifica l'esistenza di un utente con l'ID specificato.	Input: userID (ID dell'utente da verificare).
POST /api/user/create	Crea un nuovo utente con i dati forniti.	Input: userDTO (dati dell'utente da creare).
POST /api/user/update	Aggiorna un utente esistente con i dati forniti.	Input: userDTO (nuovi dati dell'utente).
DELETE /api/user/delete/{userID}	Elimina un utente con l'ID specificato.	Input: userID (ID dell'utente da eliminare).
GET /api/user/get/{userID}	Ottiene un utente con l'ID specificato.	Input: userID (ID dell'utente da ottenere).

API Bin

Endpoint	Descrizione	Dettagli
POST /api/bin/create	Crea un nuovo contenitore (bin) con i dati forniti.	Input: binDTO (dati del contenitore da creare).
POST /api/bin/update	Aggiorna un contenitore (bin) esistente con i dati forniti.	Input: binDTO (dati del contenitore da aggiornare).
DELETE /api/bin/delete/{binID}	Elimina un contenitore (bin) con l'ID specificato.	Input: binID (ID del contenitore da eliminare).
GET /api/bin/get/{binID}	Ottiene un contenitore (bin) con l'ID specificato.	Input: binID (ID del contenitore da ottenere).
GET /api/bin/get/all	Ottiene tutti i contenitori (bin) presenti nel sistema.	Nessun input richiesto.

API Trash

Endpoint	Descrizione	Dettagli
GET /api/trash/notifications/user/{userID}	Ottiene le notifiche dei rifiuti per un utente specifico.	Input: userID (ID dell'utente per le notifiche dei rifiuti).
GET /api/trash/statistics /user/{userID}/{year}	Ottiene le statistiche dei rifiuti per un utente in un dato anno.	Input: userID (ID dell'utente), year (anno delle statistiche).
POST /api/trash/statistics/user/all/{year}	Ottiene le statistiche dei rifiuti per una lista di utenti in un dato anno.	Input: idList (lista di ID utenti), year (anno delle statistiche).
GET /api/trash/statistics/city/{year}	Ottiene le statistiche dei rifiuti per l'intera città in un dato anno.	Input: year (anno delle statistiche).

API Alert

Endpoint	Descrizione	Dettagli
GET /api/alert/get/all	Ottiene tutte le notifiche di allarme di capacità.	Nessun input richiesto.

4.2.3 Il broker RabbitMQ

Accanto al servizio dell'agenzia rifiuti, viene deployato anche il servizio del broker **RabbitMQ**. Quest'ultimo avrà il ruolo di intermediario con i cassonetti installati nella città.

Il broker RabbitMQ si rende disponbile per ricevere le notifiche sulla porta 5672. Viene inoltre lasciata attiva la porta 15672 per collegarsi all'interfaccia grafica.

Il file di configurazione del broker definisce due code con rispettive chiavi:

- TRASH_QUEUE e TRASH_NOTIFICATION_ROUTING_KEY: coda dedicata alle notifiche degli utenti che gettano la spazzatura
- ALERT_QUEUE e CAPACITY_ALERT_ROUTING_KEY: coda dedicata alle notifiche di capacità dei cassonetti

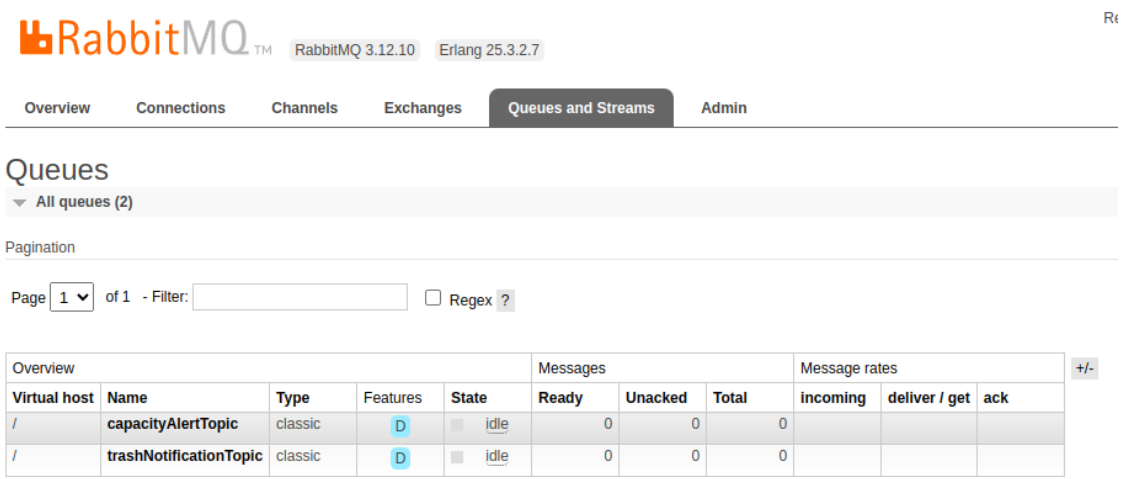

Figura 4.12: Interfaccia di RabbitMQ

Le chiavi vengono utilizzate dal topic exchange *wasteDisposalAgencyTopic* per lo smistamento dei pacchetti.

4.2.4 Login Service

Il servizio di Login ha lo scopo di:

- gestire le credenziali di accesso dei cittadini
- gestire le credenziali di accesso degli operatori
- fornire dei JWT validi per l'accesso alle API post autenticazione

API Auth

Endpoint	**Descrizione**	**Dettagli**
POST /api/auth/signin	Autenticazione di un utente.	Input: username, password.
POST /api/auth/signup	Registrazione di un nuovo utente.	Input: username, email, password, ruolo.
DELETE /api/auth/delete/{id}	Elimina un utente esistente.	Input: id dell'utente da eliminare.

4.2.5 Tax Service

Il servizio delle tasse ha lo scopo di:

- erogare le tasse per gli utenti
- permettere il pagamento delle tasse
- fornire le statistiche sul pagamento delle tasse

API Fee

Endpoint	**Descrizione**	**Dettagli**
POST /api/fee/create/all	Crea tasse basate sulle statistiche dei rifiuti fornite.	Input: Lista di statistiche dei rifiuti in formato DTO.
DELETE /api/fee/delete/{feeId}	Elimina una tassa in base all'ID.	Input: ID della tassa da eliminare.
POST /api/fee/pay/{feeId}	Registra il pagamento di una tassa in base all'ID.	Input: ID della tassa da pagare.
GET /api/fee/get/user/{userId}	Ottiene tutte le tasse di un utente in base all'ID dell'utente.	Input: ID dell'utente.
GET /api/fee/get/all	Ottiene tutte le tasse erogate.	Nessun input.

API Statistics

Endpoint	**Descrizione**	**Dettagli**
GET /api/stats/all/{year} /{paidStatus}	Ottiene le statistiche di tutte le tariffe in base allo stato di pagamento e all'anno	Input: year (anno delle tariffe), paidStatus (1 per tariffe pagate, 0 per tariffe non pagate)
GET /api/stats/user/{userId}/{year} /{paidStatus}	Ottiene le statistiche di tutte le tariffe di un utente specifico in base allo stato di pagamento, all'anno e all'ID dell'utente	Input: userId (ID dell'utente), year (anno delle tariffe), paidStatus (1 per tariffe pagate, 0 per tariffe non pagate)

4.3 Gestione dei Database

Ogni servizio di backend ha la propria istanza di MongoDB, nella quale salva dati di propria competenza. Si è scelto un database non relazionale per garantire maggiore flessibilità al sistema.

Data models servizio Login

Figura 4.13: Schema di un utente del servizio Login

Data models servizio Comune

Figura 4.14: Schema di un utente del servizio Comune

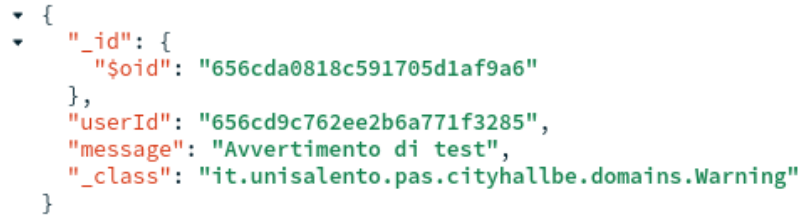

Figura 4.15: Schema di un warning del servizio Comune

Data models servizio Agenzia Rifiuti

Figura 4.16: Schema di un operatore del servizio Agenzia Rifiuti

```
{
  "_id": {
    "$oid": "657c0a4bdb36293d15b2fdf8"
  },
  "latitude": 40.35426330566406,
  "longitude": 18.173919677734375,
  "capacity": 100,
  "sortedWaste": 10,
  "unsortedWaste": 15,
  "alertLevel": 0,
  "_class": "it.unisalento.pas.wastedisposalagencybe.domains.Bin"
}
```


Figura 4.17: Schema di un cassonetto del servizio Agenzia Rifiuti


```
{
  "_id": {
    "$oid": "657c0cdcdb36293d15b2fdfe"
  },
  "timestamp": "2023-12-15T09:22:49.655978173",
  "userId": "657c0c41f6e84e15478f01b0",
  "binId": "657c0ab5db36293d15b2fdf9",
  "sortedWaste": 60,
  "unsortedWaste": 10,
  "_class": "it.unisalento.pas.wastedisposalagencybe.domains.Trash"
}
```


Figura 4.18: Schema di una notifica spazzatura del servizio Agenzia Rifiuti


```
{
  "_id": {
    "$oid": "657c0cdadb36293d15b2fdfd"
  },
  "timestamp": "2023-12-15T09:22:49.665218790",
  "binId": "657c0ab5db36293d15b2fdf9",
  "alertLevel": 1,
  "_class": "it.unisalento.pas.wastedisposalagencybe.domains.Alert"
}
```


Figura 4.19: Schema di una notifica alert sulla capacità cassonetto del servizio Agenzia Rifiuti

Data models servizio Tasse


```
{
  "_id": {
    "$oid": "657c0ea6f4dc334c7735cac6"
  },
  "timestamp": "2023-12-15",
  "userId": "657c0c41f6e84e15478f01b0",
  "paid": 1,
  "sortedWaste": 130,
  "unsortedWaste": 120,
  "sortedTax": 260,
  "unsortedTax": 600,
  "_class": "it.unisalento.pas.taxbe.domains.Fee"
}
```


Figura 4.20: Schema di una tassa del servizio Tasse

4.4 Client del cassonetto

Il client del cassonetto nasce con lo scopo di simulare il comportamento di eventuali cassonetti fisici. Non possiede interfaccia grafica ed è comune sia agli operatori che agli utenti.

Il client possiede un database locale di testo sul quale tiene traccia dello status dei cassonetti, in modo che non si perdano ad ogni riavvio. Il suo utilizzo è diviso in due grandi macro funzionalità:

- simulazione di gestione dei bidoni: area dedicata agli operatori
- lancio di rifiuti: area dedicata agli utenti

La zona operatore consente di:

- installare un bidone: azione da eseguire in contemporanea con la creazione nel database dell'agenzia rifiuti
- rimuovere un bidone: azione da eseguire in contemporanea con l'eliminazione dal database dell'agenzia rifiuti
- scaricare i bidoni: consente di svuotare i cassonetti secondo l'ordine comunicato dall'agenzia rifiuti
- mostra i bidoni: stampa un'immagine del database locale

La zona utente consente di:

- lanciare manualmente i rifiuti: viene creata una notifica in cui si sceglie il cassonetto e la quantità di spazzatura
- lanciare casualmente i rifiuti: una volta inserito l'id utente, il sistema seleziona un cassonetto ed una quantità di spazzatura random per la notifica

Il client contiene una classe `UserVerifier` che è in possesso della chiave simmetrica del servizio comune. Essa serve a decodificare l'ID utente e garantirne l'attendibilità in locale.

Capitolo 5

Design Pattern

5.1 IoC

Il design pattern **Inversion of Control** (IoC) è un concetto chiave nello sviluppo del software che sposta il controllo dell'esecuzione del programma da parte dell'applicazione stessa a un'esterna entità di gestione. Nel contesto di **Java**, il framework SpringBoot fa ampio uso di questo principio. **SpringBoot** implementa IoC attraverso il concetto di **Dependency Injection** (DI), dove le dipendenze di un oggetto non sono più gestite dall'oggetto stesso, ma sono fornite da un esterno, spesso tramite configurazioni. Questo approccio permette di separare la logica dell'applicazione dalle sue dipendenze, migliorando la modularità, la manutenibilità e la testabilità del codice. Nel contesto di SpringBoot, le dipendenze vengono definite tramite annotazioni come **@Autowired**, permettendo al framework di gestire l'iniezione delle dipendenze durante l'esecuzione dell'applicazione. Questo modello permette agli sviluppatori di concentrarsi sulla logica dell'applicazione, lasciando a SpringBoot la gestione delle dipendenze e delle configurazioni, facilitando così lo sviluppo di applicazioni robuste e scalabili.

Il pattern è ampiamente utilizzato all'interno di ogni *controller* e *servizio* dei vari backend.

5.2 Publish & Subscribe

Il design pattern Publish & Subscribe (pub/sub) è ampiamente utilizzato per consentire la comunicazione asincrona tra diversi componenti di un sistema software. Quando imple-

mentato in **Java** con un broker come **RabbitMQ**, questo pattern permette ai vari moduli dell'applicazione di scambiare messaggi in modo efficiente e scalabile.

Utilizzando Java insieme a RabbitMQ, i publisher (cassonetti) inviano i messaggi a specifici "**exchanges**" all'interno del broker, mentre i subscriber (agenzia rifiuti) si collegano a "**queues**" associate a tali exchanges. Il broker gestisce la distribuzione dei messaggi ai subscriber interessati, consentendo una comunicazione disaccoppiata e altamente affidabile tra i diversi componenti del sistema.

Nel contesto di AngularJS, le operazioni subscribe, async e await sono utilizzate per gestire la comunicazione asincrona lato client. Le chiamate asincrone permettono di elaborare richieste al server senza bloccare l'interfaccia utente. Le funzioni subscribe vengono utilizzate per osservare i cambiamenti di stato o i flussi di dati e agire di conseguenza quando i dati sono disponibili o quando avviene un evento specifico.

```
27  async signUp() {
28    if (this.isValidForm()) {
29      try {
30        const response = await this.authService.signUp(this.username, this.email, this.password, this.role).toPromise();
31        if (response) {
32          const signUpResponse: SignUpResponse = response as SignUpResponse;
33          const user: User ={
34            id: signUpResponse.id,
35            username: this.username,
36            email: this.email,
37            role: this.role,
38            name: '',
39            surname: '',
40            bdate: ''
41          };
42
43          await this.logIn();
44
45          this.userService.createUser(user).subscribe(
46            () => {
47              console.log("User created successfully");
48            },
49            (error) => {
50              console.error("Error creating user: ", error);
51              this.toastService.showErrorToast(error.error['message']);
52            }
53          );
```

Figura 5.1: Esempio di utilizzo delle funzioni async e await nel auth component

5.3 Bridge

Il design pattern Bridge è una struttura che favorisce la separazione tra l'astrazione e l'implementazione di un sistema. Questo pattern permette di gestire entrambi questi aspetti in maniera indipendente, consentendo variazioni e modifiche separate senza che l'una influenzi l'altra. L'idea di base è quella di avere un'**interfaccia** che agisce come un ponte tra l'astrazione e le varie implementazioni, permettendo così di sostituire o estendere sia

l'astrazione che le implementazioni in modo flessibile. Questo approccio favorisce la modularità, la manutenibilità e la scalabilità del sistema, consentendo di adattare più facilmente il software a nuove esigenze senza dover riscrivere grandi porzioni di codice.

Il design pattern Bridge è stato usato nella WebApplication dell'agenzia rifiuti. Era necessario risolvere un problema di tipo **TSP**. Per questo si è resa disponibile un'interfaccia che ne consente la risoluzione.

src > app > utils > TS TspSolverBridge.ts > ...
You, 4 days ago | 1 author (You)

```
1  export interface TspSolver {
2      findOptimalSolution(distances: number[][]): any;
3  }
```

Figura 5.2: Interfaccia del TspSolver

Successivamente è stata creata una classe che implementa una versione Nearest Neighbor. L'utilizzo dell'implementazione dell'interfaccia avviene attraverso una classe service che la istanzia.

```
8  export class TspSolverService {
9
10   getSolver(): TspSolver {
11     return new TspNearestNeighbor(); // Return the specific implementation
12   }
13
14   constructor() { }
15 }
```

Figura 5.3: Classe che istanza la versione desiderata dell'interfaccia TspSolver

Infine il service viene utilizzato all'interno del componente che ne ha bisogno, senza istanziare direttamente il solver.

```
160   // Trova il percorso ottimale sulla mappa
161   findOptimalPath(): Promise<any> {
162     var alertBins: Bin[] | undefined;
163     // seleziono i bidoni con capienza minore del 50%
164     alertBins = this.bins.filter(bin => bin.alertLevel === 1 || bin.alertLevel === 2);
165
166     // estraggo le coordinate dei bidoni
167     const locations: number[][] = alertBins.map(bin => [bin.longitude, bin.latitude]);
168
169     return new Promise((resolve, reject) => {
170       if (alertBins!.length < 1) {
171         this.toastService.showErrorToast("Not enought full bins to create a path");
172         reject('There are not enough bins with alert level 1 or 2 to calculate a path.');
173
174       } else {
175         // ottengo la matrice dei costi dal servizio
176         this.mapService.calculateDistanceMatrix(locations).subscribe((response: any) => {
177           const durationsMatrix = response.durations;
178
179           // calcolo la soluzione ottimale
180           const tspSolver: TspSolver = this.tspSolverService.getSolver();      You, 4 days ago
181           const routingSolution = tspSolver.findOptimalSolution(durationsMatrix);
182
183           // riordino i cestini secondo la soluzione ottimale
184           for (var i = 0; i < routingSolution.path.length; i++) {
185             this.binsToUnload.push(alertBins![routingSolution.path[i]]);
186           }
187
188           this.showPath();
189           resolve({ "path": this.binsToUnload, "time": routingSolution.distance });
190         });
191       }
192     })
193   }
```

Figura 5.4: Alla riga 180 viene utilizzato il design pattern Bridge

5.4 Factory

Il design pattern Factory in Java è uno dei più utilizzati e si concentra sulla creazione di oggetti. Si basa sull'idea di avere una classe dedicata alla creazione di istanze di oggetti di diverse classi, senza esporre la logica specifica di creazione direttamente al chiamante. In pratica, si utilizza un metodo di fabbrica che restituisce un'istanza di oggetto basata su determinati parametri o condizioni. Questo pattern favorisce la modularità e l'estensibilità del codice, permettendo di creare oggetti senza dover specificare esplicitamente la classe esatta dell'oggetto che si desidera istanziare. Ciò consente anche di gestire facilmente eventuali modifiche o aggiunte alle classi di oggetti senza dover modificare il codice in più parti dell'applicazione.

Il design pattern è stato utilizzato prevalentemente nel backend dell'agenzia rifiuti, per istanziare in maniera modulare il tipo di spazzatura gettato ed il tipo di cassonet-

to da creare. Al momento non esistono varianti, ma si predispone il sistema per futuri aggiornamenti.

```java
package it.unisalento.pas.wastedisposalagencybe.dto;

2 usages  1 implementation  Cesare
public interface IWasteFactory {
    1 usage  1 implementation  Cesare
    Waste getWasteType(WasteType wasteType);
}
```

Figura 5.5: Interfaccia della factory di rifiuti

```java
package it.unisalento.pas.wastedisposalagencybe.dto;

1 usage  Cesare
public class WasteFactory implements IWasteFactory{  Complexity is 5 Everything is cool!

    1 usage  Cesare
    @Override
    public Waste getWasteType(WasteType wasteType) {  Complexity is 4 Everything is cool!
        switch (wasteType){
            case SORTED_UNSORTED -> {
                return new TrashDTO();
            }
            default -> {
                return null;
            }
        }
    }
}
```

Figura 5.6: Implementazione della factory

```java
130 @   private TrashDTO fromTrashToTrashDTO(Trash trash) {
131         IWasteFactory wasteFactory = new WasteFactory();
132         TrashDTO trashDTO = (TrashDTO) wasteFactory.getWasteType(WasteType.SORTED_UNSORTED);
133
134         trashDTO.setId(trash.getId());
135         trashDTO.setTimestamp(trash.getTimestamp());
136         trashDTO.setBinId(trash.getBinId());
137         trashDTO.setUserId(trashDTO.getUserId());
138         trashDTO.setSortedWaste(trash.getSortedWaste());
139         trashDTO.setUnsortedWaste(trash.getUnsortedWaste());
140
141         return trashDTO;
142     }
```

Figura 5.7: Esempio di utilizzo del WasteFactory

5.5 Iterator

Il design pattern Iterator in Java è un modello comportamentale che offre un modo standardizzato per accedere sequenzialmente agli elementi di una collezione di oggetti senza esporre la struttura interna della collezione stessa. Questo pattern definisce un'interfaccia per navigare attraverso gli elementi di una collezione e un'implementazione specifica di questa interfaccia per ogni tipo di collezione. L'uso dell'Iterator semplifica l'iterazione attraverso gli elementi di una collezione, consentendo ai clienti di accedere agli elementi senza preoccuparsi della struttura sottostante. Inoltre, l'Iterator fornisce un modo sicuro e standardizzato per eseguire operazioni come l'accesso, l'aggiunta e la rimozione di elementi da una collezione durante l'iterazione, senza causare problemi di concorrenza o errori. In Java, l'Iterator è ampiamente utilizzato nelle collezioni come ArrayList, LinkedList e altre implementazioni di Collection Framework.

Il design pattern è stato ampiamente utilizzato nei vari backend per scorrere liste di elementi.

```java
private WasteStatistics sumWastes(List<Trash> trashList) {  Complexity is 5 Everything is cool!
    WasteStatistics wasteStatistics = new WasteStatistics();
    wasteStatistics.setTotalSortedWaste(0);
    wasteStatistics.setTotalUnsortedWaste(0);

    Iterator<Trash> iterator = trashList.iterator();

    while(iterator.hasNext()){
        Trash trash = iterator.next();
        wasteStatistics.setTotalUnsortedWaste(wasteStatistics.getTotalUnsortedWaste() + trash.getUnsortedWaste());
        wasteStatistics.setTotalSortedWaste(wasteStatistics.getTotalSortedWaste() + trash.getSortedWaste());
    }

    return wasteStatistics;
}
```

Figura 5.8: Esempio di Iterator all'interno del servizio Trash nel backend dell'agenzia rifiuti

Capitolo 6

Unit Test

Ogni SpringBoot application realizzato implementa i propri Unit Test. Essi sono scritti per verificare il corretto funzionamento degli endpoint implementati nei vari controller.

Gli unit test sono realizzati mediante i framework **Mockito** e **JUnit**.

```
43      @Test
44      void createUserTest() throws Exception {
45          UserDTO userDTO = new UserDTO();
46          userDTO.setName("John");
47          userDTO.setSurname("Doe");
48          userDTO.setEmail("john.doe@example.com");
49          userDTO.setBdate("1990-01-01");
50
51          Gson gson = new Gson();
52          String json = gson.toJson(userDTO);
53
54          when(userService.createUser(any())).thenReturn( value: 0);
55
56          mockMvc.perform(post( urlTemplate: "/api/user/create")
57                          .contentType(MediaType.APPLICATION_JSON)
58                          .content(json)
59                          .with(user( username: "operator").authorities(new SimpleGrantedAuthority( role: "ROLE_OPERATOR"))))
60                  .andExpect(status().isOk())
61                  .andExpect(content().string( expectedContent: "{\"message\": \"User created successfully\"}"));
62      }
```

Figura 6.1: Snippet di esempio di un Test

Le funzioni di test scritte vengono utilizzate all'interno delle pipeline di CICD. All'interno dello stage *Test* vengono eseguite e ne viene verificato l'esito. In caso di fallimento viene bloccata l'intera pipeline.

Capitolo 7

C.I.C.D

L'intero codice del progetto è reperibile pubblicamente sulla piattaforma **GitLab**. La piattaforma fornisce alcune importanti funzionalità:

- storico delle versioni del codice
- possibilità di creazione di pipeline CICD

In particolare, ogni repository ha la propria pipeline CICD. Nel complesso le pipeline sono uguali tra loro, le uniche differenze sono tra frontend e backend.

Per ogni repository sono definite delle variabili in modo che non venga reso pubblico l'ip della macchina EC2 e la chiave privata di accesso ssh.

7.1 Pipeline Frontend

La pipeline si compone di soli due stage:

- build: si verifica che l'applicativo sia costruibile
- deploy: si aggiorna il codice sulla macchina EC2 e si ricostruisce l'immagine docker

Ad ogni pull, GitLab si preoccupa di eseguire il comando `sudo docker compose build -no-cache` al fine di ricostruire la nuova immagine docker.

```
...
stages:
  - build
  - deploy

build_procedure:
  stage: build
  image: docker:20.10.16
  services:
    - docker:20.10.16-dind
  script:
    - docker compose build
  cache:
    key: "$CI_COMMIT_REF_NAME"
    policy: pull
    paths:
      - build


deploy_procedure:
  stage: deploy
  before_script:
    - chmod 400 $SSH_KEY
  script:
    - ssh -o StrictHostKeyChecking=no -i $SSH_KEY ubuntu@$SSH_IP "
        cd /home/ubuntu/cityhallfe/ &&
        git pull origin &&
        sudo docker compose down &&
        sudo docker compose build --no-cache &&
        sudo docker compose up -d"
```

Figura 7.1: Snippet della Pipeline del Backend

7.2 Pipeline Backend

La pipeline si compone di soli due stage:

- build: si verifica che l'applicativo sia costruibile
- test: si eseguono gli unit test per controllare che il codice funzioni
- deploy: si aggiorna il codice sulla macchina EC2 e si avvia il nuovo container

```yaml
stages:
  - build
  - test
  - deploy

build_procedure:
  stage: build
  image: docker:20.10.16
  services:
    - docker:20.10.16-dind
  script:
    - docker compose build

test_procedure:
  stage: test
  image: gradle:jdk17
  script:
    - ./gradlew test

deploy_procedure:
  stage: deploy
  before_script:
    - chmod 400 $SSH_KEY
  script:
    - ssh -o StrictHostKeyChecking=no -i $SSH_KEY ubuntu@$SSH_IP "
      cd /home/ubuntu/CityHallBE/ &&
      sudo git pull origin &&
      sudo docker compose down &&
      sudo docker compose up -d"
```

Figura 7.2: Snippet della Pipeline del Backend

Capitolo 8

Green Software

8.1 Calcolo delle Metriche

Per il calcolo delle metriche si è fatto uso del plugin **MetricsTree**. Esso permette vari tipi di analisi:

- livello di progetto
- livello di classe
- livello di metodo

Le metriche sono state valutate a livello di progetto e quelle selezionate sono le seguenti:

- **Halstead Difficulty**: La metrica valuta la complessità di un programma basandosi sul numero di operatori e operandi utilizzati, offrendo un'indicazione della sua comprensibilità e manutenibilità.
- **Halstead Errors**: Questa metrica stima il numero di errori presenti in un programma in base alla sua dimensione e complessità, fornendo un'indicazione approssimativa dei potenziali difetti all'interno del software.
- **Coupling Factor**: Valuta il grado di dipendenza tra le diverse parti di un sistema software, misurando quanto le componenti siano interconnesse e quanto un cambiamento in una componente possa influenzare le altre.
- **Polymorphism Factor**: Questa metrica misura l'estensione in cui il polimorfismo, un concetto chiave nell'orientamento agli oggetti, viene utilizzato nel codice. Indica

la capacità di vari oggetti di essere trattati in modo simile attraverso una singola interfaccia comune.

- **Maintainability Index**: Questa metrica valuta la facilità con cui un software può essere modificato o corretto. Incorpora diverse misure di complessità, coesione e accoppiamento per fornire un punteggio che riflette la facilità di manutenzione del codice nel tempo.

	CityHallBE	WasteAgencyBE	LoginBE	TaxBE
Halstead Difficulty	219,0927	455,9598	179,7609	348,1521
Halstead Errors	1,4755	3,1032	1,1903	2,2789
Coupling Factor	11,76%	7,08%	16,91%	14,74%
Polymorphism Factor	122,22%	8,58%	375,00%	125,00%
Maintainability Index	15,6045	6,4071	21,0165	13,4905
Lines of code	495	1042	297	574

Figura 8.1: Statistiche dei servizi backend

Complessità ciclomatica

La complessità ciclomatica è una metrica utilizzata per valutare la complessità di un programma misurando il numero di percorsi distinti all'interno del codice sorgente. Essenzialmente, indica il numero di decisioni indipendenti all'interno di una funzione o di un metodo. Valori più alti di complessità ciclomatica possono indicare una maggiore complessità nel controllo del flusso del programma, suggerendo la potenziale necessità di semplificare o ristrutturare il codice per renderlo più comprensibile e più facile da testare.

Essa è stata analizzata con il plugin **CodeMetrics** e per ogni metodo o classe è presente un valore di complessità ciclomatica. Vengono riportate solo le classe con complessità ciclomatica superiore o uguale a 7.

Classi comuni a tutti i servizi sono:

- SecurityConfig (8): la classe contiene le definizioni della sicurezza di SpringBoot
- JwtUtils (8): sono presenti i metodi di validazione del JWT con conseguenti *try catch* che aumentano la complessità

Seguono ulteriori classi con il rispettivo valore di complessità ciclomatica:

- WasteAgencyBE -> BinController (7): sono presenti vari metodi che iterano su liste

- LoginBE -> AuthController (7): la classe contiene le API di login e autenticazione, oltre ad alcuni controlli per validare i dati utente
- TaxBE -> FeeUtils (8): la classe contiene molti metodi di utilità che lavorano con liste di fee al fine di generare statistiche

8.2 Calcolo dei Consumi

La software carbon intensity (SCI) rappresenta la quantità di emissioni di carbonio generate durante lo sviluppo, l'esecuzione e l'uso del software. Questo concetto è cruciale nell'ambito della sostenibilità ambientale, poiché mira a valutare e ridurre l'impatto ambientale delle tecnologie digitali, considerando fattori come l'energia utilizzata dai data center, il consumo di risorse e la gestione dei rifiuti elettronici. Misurare e ottimizzare la carbon intensity del software è un passo significativo verso la creazione di soluzioni digitali più sostenibili.

8.2.1 Calcolo SCI

$$\text{SCI} = ((E * I) * M)R$$

I parametri utilizzati sono i seguenti:

- E: l'energia consumata durante lo sviluppo e l'esecuzione del software
- I: l'intensità di carbonio dell'energia utilizzata
- M: fattori di emissione di carbonio non direttamente legati all'hardware usato
- R: unità funzionale legata all'utilizzo del software

Calcolo E

Per ogni componente di backend è stato misurato il consumo energetico in fase di esecuzione in **Joule**, convertito successivamente in **kWh**.

Servizio	Consumo (J)	Consumo (kWh)
CityHallBE	83,38	2,31611E-05
WasteAgencyBE	100,05	2,77917E-05
LoginBE	82,02	2,27833E-05
TaxBE	62,36	1,73222E-05

Figura 8.2: Consumi energetici dei software

I valori sono stati ottenuti eseguendo i vari **jar** del backend attraverso il software **JoularJX**, un agente basato su Java progettato per il monitoraggio dell'energia direttamente dal codice sorgente, che consente agli sviluppatori di valutare e analizzare il consumo di energia delle loro applicazioni Java durante l'esecuzione.

```
2023-12-19T09:10:40.326+01:00  INFO 74831 --- [JX Agent Thread] i.u.p.cityhallbe.CityHallBeApplication
 Started CityHallBeApplication in 8.029 seconds (process running for 10.497)
^C19/12/2023 09:10:45.256 - [INFO] - JoularJX finished monitoring application with ID 74831
19/12/2023 09:10:45.256 - [INFO] - Program consumed 37,39 joules
19/12/2023 09:10:45.316 - [INFO] - Energy consumption of methods and filtered methods written to files
```

Figura 8.3: Risultato esempio dell'output di JoularJX

Calcolo I

Per ottenere il valore si è fatto uso del portale **ElectricityMap**, facendo riferimento alla zona di deploy delle macchine EC2.

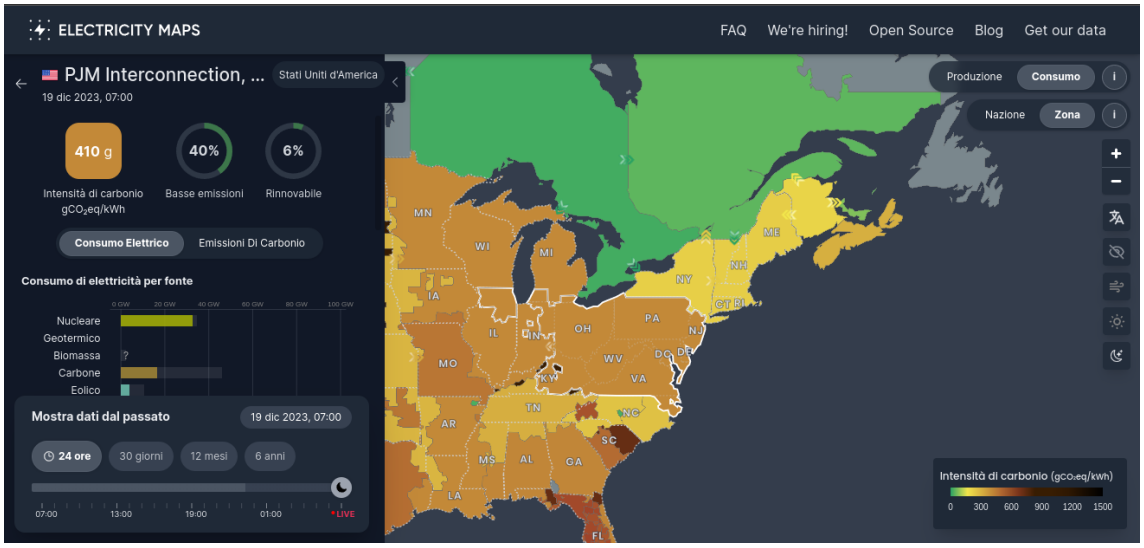

Figura 8.4: Carbon Intensity in data 19/12/2023 09:23 (Ora Italiana)

Calcolo M

La formula per calcolare M è la seguente:

$$M = \text{TotalEmbodied} * \frac{\text{TimeReserved}}{\text{TotalLifeSpan}} * \frac{\text{ReservedResources}}{\text{TotalResources}}$$

Si considerano i seguenti valori:

- TotalEmbodied: per molti server del tipo e2-standard-4 vale 1230.3 kgCO2eq

- TimeReserver: tempo di attività del server, ipotizziamo un mese (720 ore)
- TotalLifeSpan: tempo di vita totale di un server, mediamente 35040 ore
- ReservedResources: numero di CPU dedicate al nostro server, nel nostro caso 2
- TotalResources: numero di CPU totali del nostro server, nel nostro caso 32

Otteniamo un valore associato ad M pari a **1,58 kgCO2eq**.

Calcolo R

Per i vari software sono stati stimati i seguenti numeri di API considerando la frequenza con cui queste vengono chiamate dal frontend:

- CityHallFE: 10 mila (solo chiamate in fase di login, modifica profilo e avvisi)
- WasteAgencyBE: 30 mila (chiamate in fase di login, modifica profilo, statistiche, gestione cassonetto, allarmi e notifiche)
- LoginBE: 5 mila (solo in fase di login)
- TaxBE: 15 mila (erogazione tasse annuali e statistiche)

Risultato

Inserendo i valori nella formula per il calcolo dell'SCI otteniamo i valori presenti in tabella.

Servizio	E (kWh)	I (kgCO2eq/kWh)	M (kgCO2eq)	1/R	SCI (kgCO2eq)
CityHallBE	2,31611E-05	410	1,58	10000	1,5895E-04
WasteAgencyBE	2,77917E-05	410	1,58	30000	5,3047E-05
LoginBE	2,27833E-05	410	1,58	5000	3,1787E-04
TaxBE	1,73222E-05	410	1,58	15000	1,0581E-04

Figura 8.5: Calcolo dell'SCI

Capitolo 9

Conclusioni

9.1 Sprint Backlog

Attività	Ore stimate	Ore effettive	Errore
Analisi casi d'uso	1	1	0
Analisi data model ed entità	1	1	0
Sviluppo CityHallFE	10	14	-4
Sviluppo CityHallBE	15	20	-5
Creazione CityHallDB	1	1	0
Sviluppo WasteAgencyFE	10	10	0
Sviluppo WasteAgencyBE	15	25	-10
Configurazione RabbitMQ	2	2	0
Creazione WasteAgencyDB	1	1	0
Sviluppo LoginBE	10	27	-17
Creazione LoginDB	1	1	0
Sviluppo TaxBE	15	12	3
Creazione TaxDB	1	1	0
Sviluppo TestCase backend	10	16	-6
Debug	25	20	5
Configurazione EC2	8	5	3
Configurazione Pipeline CICD	4	2	2
Configurazione API Gateway	4	3	1
Misure per il GreenSoftware	3	1	2
Documentazione	7	7	0
Totale	**144**	**170**	**-26**

Figura 9.1: Tabella Spring Backlog

9.2 Burndown Chart

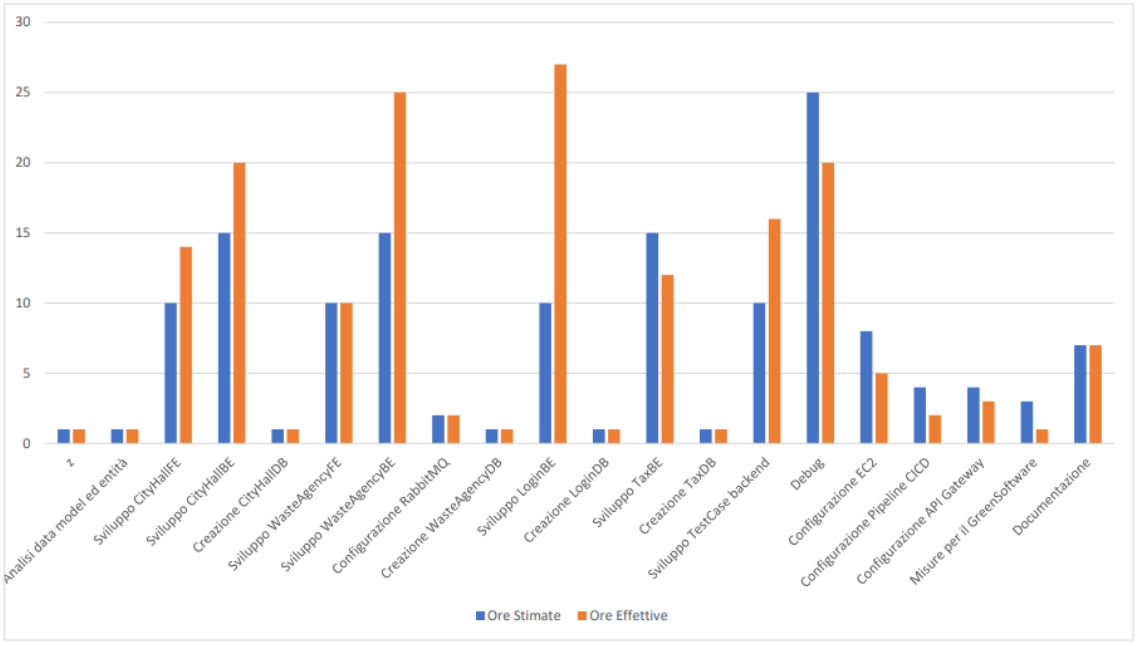

Figura 9.2: Burndown Chart

9.3 Sviluppi Futuri

Si rimandano ad aggiornamenti futuri le seguenti funzionalità:

- API del servizio di Login per registrare una nuova applicazione e collezioni utenti dinamiche
- API del servizio di Login per il recupero della password
- invio dei ticket erogati tramite mail
- invio delle segnalazioni tramite mail
- possibilità di inserire una propria foto profilo
- una gestione dei rifiuti più dettagliata, non solo differenziata ed indifferenziata
- eventuale raggruppamento di logica per la creazione di un ulteriore microservizio dedito alle statistiche
- eventuale implementazione della logica di erogazione ticket in una lambda function